Apuntes Web

Profesor: J.A. Sampedro

IES Virgen del Carmen (Jaén)

Índice

UD 3: Gestión de módulos y configuraciones

UD 4: Host virtuales

Parte VI: Plataformas

UD 1: Virtualbox

UD 2: Instalar Wordpress

Parte VII: Herramientas

UD 1: Git

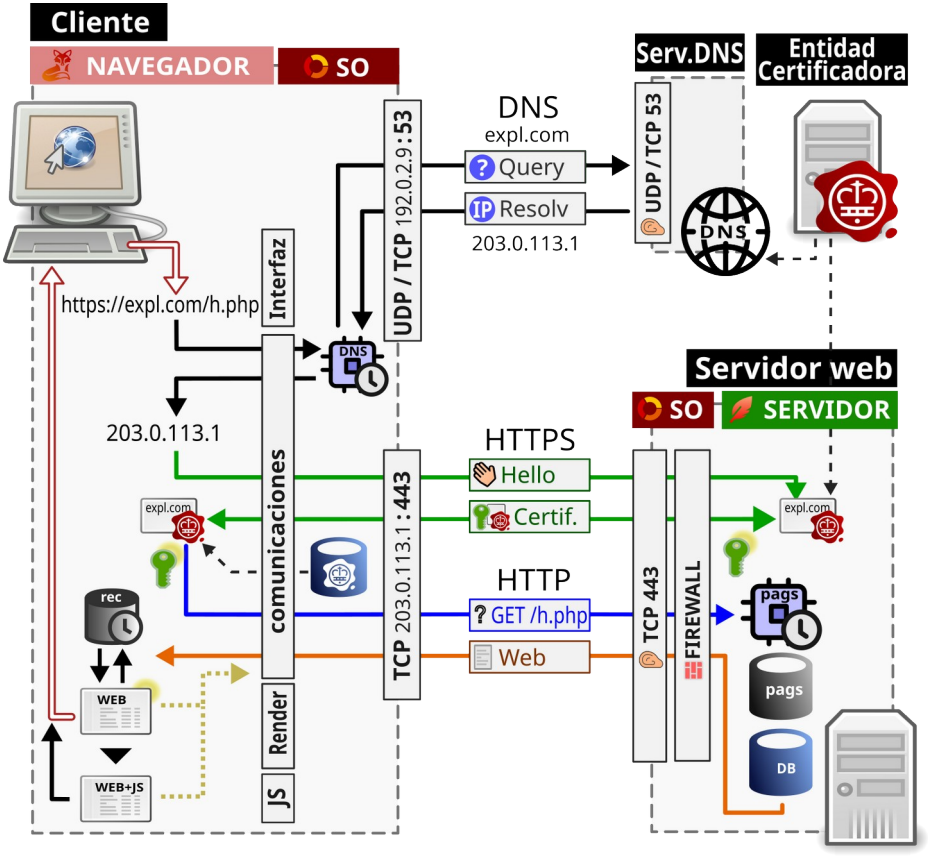
Parte I:

HTML

Unidad Didáctica 1:

Tecnologías web

1. Petición web



Los entornos web hacen uso de una enorme cantidad de tecnologías, normas y protocolos con el fin de asegurar la seguridad, la estabilidad y la privacidad.

En el siguiente diagrama se ilustra, de forma simplificada, los pasos necesarios para acceder a una página web, empezando por el momento en que el usuario introduce una URL en el navegador o pulsa un enlace.

2. La red TCP/IP

Las tecnologías web, al igual que otros muchos campos del área de las telecomunicaciones, hacen uso de la red Internet y resto de subredes así como de la enorme cantidad de tecnologías usadas en ellas. Dichas tecnologías, sin embargo, no entran en la competencia del desarrollo web, por lo que no serán de objeto de estudio aquí, aunque conviene conocer algunos aspectos:

- Toda máquina conectada a Internet tiene una IP pública.
- Estas máquinas se encuentran escuchando en uno o más puertos. Los servidores web, típicamente, escuchan en los puertos 80 y 443, aunque también hay puertos típicos como el 8080.
- Cuando queremos comunicarnos con un servidor web, primero debemos realizar una conexión TCP especificando IP y puerto. Si el servidor no está escuchando en ese puerto, la conexión no será posible.
- Si el servidor está escuchando, y el firewall no lo impide, entonces se establece la conexión y se podrá producir la comunicación (en ambas direcciones) entre el servidor y el cliente, empleando los protocolos HTTPS y HTTP.

Ej. 1: En una terminal, ejecuta el siguiente comando, el cual crea un servidor sencillo que escucha en el puerto 5000:

```
printf '%s\n' '#!/bin/sh' 'while IFS= read -r l; do l=$(printf "%s" "$l"|tr -d "\r"); case "$l" in HELLO) echo Hola;; DATE) date;; EXIT) exit 0;; *) echo "Error";; esac; done' >/tmp/t.sh; chmod +x /tmp/t.sh; socat -v TCP-LISTEN:5000,reuseaddr,fork EXEC:/tmp/t.sh
```

Ahora, en una terminal distinta, realiza una conexión de red con el servidor con el comando `nc 127.0.0.1 5000`. Prueba a poner los comandos HELLO, DATE, EXIT u otro inventado (en cuyo caso debería dar error), para ver el resultado.

Ej. 2: Averigua la IP de tu equipo con el comando `ifconfig`. Realiza la conexión de red con el equipo de un compañero y vuelve a probar los comandos antes mencionados.

Máquinas con varias IPs

Algunas máquinas poseen varias IPs, lo que puede tener varios objetivos, como es el aumentar el caudal de tráfico entre la máquina e internet (balanceo de carga), el ofrecer dominios webs con IPs exclusivas (IPs dedicadas), aumentar la confiabilidad de la conexión a Internet (conexiones múltiples), conectar con redes privadas (LANs) etcétera.

UDP

En vez de TCP, en algunos servicios de red también se puede usar UDP, que no está orientado a la conexión y, por tanto, no garantiza que los datos lleguen a su destino ni en el mismo orden en que fueron enviados. A cambio, es más rápido y ligero, empleándose en aplicaciones donde la velocidad es más importante, como en streaming, videollamadas, juegos en línea o el servicio de DNS. UDP también necesita la IP y el puerto en el que la máquina a la que queremos enviar la información está escuchando.

3.

4. Direcciones web

Lo primero que debemos tener para conseguir un recurso web es su dirección web, llamada URL (Uniform Resource Locator). La URL se encarga de dos cosas:

1) Identifica un recurso web, que puede ser un fichero, una página web, etcétera.
2) Indica cómo puede conseguirse dicho recurso web.

La URL no debe confundirse con URI: éste último puede identificar otros tipos de recursos y, a diferencia de las URLs, no obliga a especificar el cómo conseguirlos. Ejemplos de URIs que no identifican dónde acceder al recurso pueden ser:

- ***mailto:someone@example.com*** Identifica un correo electrónico, pero no cómo acceder a dicho correo.
- ***Isbn:978-3-16-148410-0*** identifica un libro, pero no cómo acceder a él.

Estructura básica de una URL

Respecto a la estructura básica de una URL, aunque existen URLs sin el componente "máquina", en la práctica tiene la siguiente estructura mínima:

esquema://máquina/ruta

Así por ejemplo, en las siguientes direcciones web, identificamos en rojo el esquema, en naranja la máquina y en verde la ruta:

- http://www.phy.mtu.edu/faculty/Nemiroff.html
- https://wikimedia.org/static/images/project-logos/enwiki-1.5x.png
- `http://duckduckgo.com/` (suele servir el index.html de ese directorio)
- https://en.wikipedia.org/wiki/Subdomain
- ftp://ftp.ubuntu.com/ubuntu/dists/xenial/Release
- file:///home/samar/Escritorio/client.html (no tiene máquina)

Es común que los navegadores completen las direcciones que se introduzcan en sus barras de direcciones. Así, por ejemplo, `http://duckduckgo.com` realmente NO es una dirección web válida (la ruta no puede ser vacía), pero el navegador añade la barra final, para formar la dirección `http://duckduckgo.com/` que sí es válida.

Ej. 3: ¿Cuál de éstas URLs es válida?

- google.com/url
- http://nasa.gov/img.mkv
- https://nasa.gov/img.mkv/
- ftp://nasa.gov/
- nasa.gov

Dominio, subdominios y DNS

Los nombres de máquina se componen de una o más etiquetas, separadas por puntos. Cada etiqueta tiene un mínimo de 1 carácter y un máximo de 63, que pueden ser letras inglesas, números o guiones, no habiendo diferencia entre letras mayúscula y minúsculas. Además, no se permiten guiones seguidos ni guiones al inicio ni al final.

- hi.hola_.com ← Incorrecto: termina en guion
- ho_la.org ← Incorrecto: guiones seguidos
- _hi.uk ← Incorrecto: no puede empezar por guion
- hola..net ← Incorrecto: min. 1 carácter
- www.hola.es ← Correcto

Cada una de esas etiquetas se llama subdominio, excepto la etiqueta a la derecha, que es un dominio de alto nivel (por ejemplo com, org, net, es, uk, etćetera).

Ej. 4: Indica los nombres de máquina son incorrectos y el porqué:

llllll.net wey_ _wey.jp enseñar.es br.br.br MAYUSCULAS.ORG 1234.com guión.ar uno1.fr hablar_.fi ecuador1.ec hola que tal.com _ap.com a..a.com

5. Opciones a la URL

Respecto a la estructura básica anterior, es posible añadir una, varias, todas o ninguna de las opciones explicadas a continuación (puerto, usuario, contraseña, consulta y/o sección).

Especificar un puerto

Se puede especificar un puerto de la máquina a la que nos dirigimos, añadiendo dos puntos y un número tras la máquina. Si no se especifica puerto, se usa el puerto por defecto para el esquema indicado (80 para http, 443 para https, 23 para ftp, etc.):

https://en.wikipedia.org**:80**/wiki/Subdomain

Puedes encontrar los puerto asignados, tanto de forma oficial como de facto, para cada esquema (también llamado protocolo) en `https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers` .

Ej. 5: Prueba a cambiar el puerto del ejemplo anterior a 8080: verás que dicha URL no identifica ningún recurso existente y, por tanto, el navegador da el error "No se puede conectar", porque ni tan siquiera se habrá podido establecer la conexión.

Ej. 6: Queremos acceder, usando el protocolo http seguro, al recurso situado en /hola.html en un servidor llamado example.com, el cual está escuchando en el puerto 8080 ¿cuál es la URL necesaria para ello?

Usuario y/o contraseña

Para aquellos recursos web que están bajo el control de un usuario de la máquina destino, podemos especificar un usuario y, si es necesario, también una contraseña de acceso para dicho usuario: ftp://usuario:contraseña@mirrowindftp.es/profiles

Consultas

Algo muy común en la web es especificar una consulta, que se añadirá tras la ruta, empezando por una interrogación. Esta consulta podrá ser usada por el servidor web para mostrar una información u otra, o realizar otras tareas. Un ejemplo muy común está en los buscadores: `https://www.google.com/search?q=url` .

Las búsquedas están siempre precedidas por una interrogación inicial, tras la cual se indica un parámetro y su valor unidos con un igual. Cada parámetro nuevo estará precedido de un ampersand:

?prm1**=**val1**&**prm2**=**val2**&**prm3**=**val3

Ej. 7: Prueba a cambiar la parte a la derecha del igual (cambia "url" por otra cosa) y ponlo en la barra de direcciones del navegador: verás que los resultados de la búsqueda difieren. Observa que la página es la misma: es una página que recoge la parte de búsqueda de la dirección web y proporciona los resultados relacionados con ese texto.

Ej. 8: Identifica los parámetros y los valores correspondientes de la siguiente URL: `https://www.google.com/search?q=site:bbc.com+climate+change&lr=lang_es&safe=active&tbs=qdr:y`

Ej. 9: Crea una URL para acceder, vía http puerto 80, al servidor "youtube.com", y el recurso a buscar es un fichero llamado "watch". Se le ha de pasar una consulta con el párametro "v" valiendo "NomVW1mm4BY y un parámetro llamado "t" con el valor "550s" ¿Es posible no especificar el puerto?

Nota: la URL resultante debe llevarte a un video de Baitybait.

Sección

Finalmente, se puede especificar una sección dentro del recurso web, de forma que dirigimos al usuario a esa parte específica dentro del recurso web. Se añade al final, empezando con una almohadilla (el carácter "#"): https://en.wikipedia.org/wiki/URL#Notes

El uso de secciones es posible en todo tipo de ficheros, pero es especialmente usado en archivos html.

Ej. 10: Pon la dirección anterior en la barra de direcciones del navegador. Luego, cambia "Notes" por "See_also" o por "Syntax" y verás que la posición del navegador cambia.

Cuando la sección indicada en la dirección no existe, entonces no producirá ningún cambio (si se estaba ya en la web) o el navegador se situaría al inicio de la página: cierra el navegador y, tras reabrirlo, pon https://en.wikipedia.org/wiki/URL#Blah .

Sintaxis completa

La sintaxis completa para una URL sería la siguiente, siendo obligatorios solo el esquema y la ruta. Además, si se especifica usuario, contraseña o puerto, también debe especificarse la máquina, la cual se especifica en la inmensa mayoría de URLs:

esquema://usuario:contraseña@máquina:puerto/ruta?
consulta#sección

Ej. 11: Crea una URL para acceder, vía ftp puerto 20, al servidor "mylastserver.it" con un usuario "admin", un password "abcdef", y el recurso a buscar es un fichero llamado "a.png" dentro del directorio "img". Se le ha de pasar una consulta con el parámetro "sec" valiendo "a54" y un parámetro llamado "par" con el valor "32" ¿Es posible no especificar el puerto?

Nota: La URL resultante no es real.

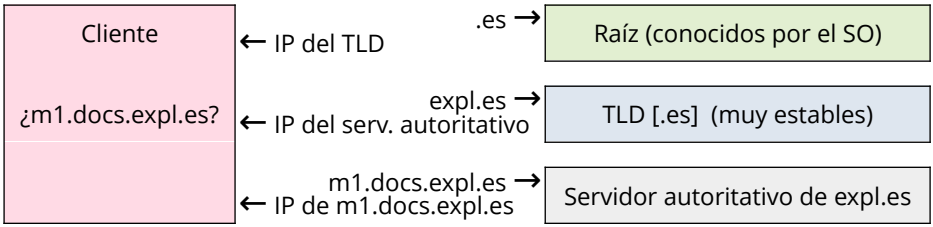
Ej. 12: Construye una URL cuya máquina sea "dragonslite.es", con esquema https, y que apunte al archivo pass.php que está en el directorio "pass" que, a su vez, está en el directorio "docencia". Además, la URL tendrá:

- Puerto: 443,
- Búsqueda: parámetro alfa igual a `abd`, y beta igual a `-20`.
- Sección: exito
- Usuario admin, contraseña: `HOLA%2B%23.hola`
  (la contraseña real es "HOLA + #.hola" pero debemos codificarla).

Nota: La página resultante tiene varios párrafos de texto y una sección llamada “Éxito”.

6. DNS

Como hemos visto, para realizar una conexión de red con una máquina conectada a Internet necesitamos la IP de dicha máquina. Sin embargo, en la web real, rara vez se usan directamente IPs, sino nombres de dominio, como google.com o amazon.es. El servicio DNS transforma estos nombres de dominio por su correspondiente IP. El proceso para una consulta DNS es la siguiente:



- El cliente consulta a uno de los servidores raíz (estos servidores son muy estables y sus direcciones son conocidas por el sistema operativo) para saber qué servidores *Top Level Domain* (TLD) tienen información sobre el dominio de mayor nivel del dominio a consultar (.com, .es, etc.).
- El cliente consulta a uno de esos servidores TLD para saber qué servidores autoritativos poseen el dominio a buscar.
- Consulta a uno de esos servidores autoritativos para saber qué IP corresponde al dominio consultado.

Un servidor autoritativo contiene zonas DNS. Una zona DNS contiene toda la información sobre un dominio concreto. Observa

Servidor autoritativo en el propio dominio

En este caso, el servidor autoritativo será el propio dominio (en el ejmplo de arriba, el servidor gris sería el propio expl.es).

Un ejemplo de los registros existentes de una zona DNS (la cual estará en un servidor autoritativo), donde el servidor autoritativo coincide con el servidor que sirve el dominio en cuestión es el siguiente:

```
dominio.com.  IN SOA   dominio.com. admin.dominio.com. (
              2025091501  ;serial: Nº serie de la zona de dominio
              7200  ;refresh: cada cuanto los sec. revisan si hay cambios
              3600  ;retry: lo que un sec. espera ante fallo para reintentar
              1209600  ;expire: con fallo, cuando la info de sec. expira
              3600 )   ;minimum TTL: tiempo resolutores guardan respuestas
                       ;negativas

dominio.com.  IN NS    dominio.com.  ;serv. primario definido en SOA
                                     ;En este caso, dominio.com coincide
;Si los servidores de nombres estuvieran en subdominios (ej ns1.dominio.com
; y ns2.dominio.com, entonces deberían definirse las IPs de esos subdominios
; con registro A/AAAA.

dominio.com.  IN A     203.0.113.25  ;IPv4 del dominio
ejemplo.com.  IN AAAA  2001:db8::8a2e:370:7334;  IPv6 del dominio
www           IN CNAME dominio.com.  ;alias
sub           IN A    198.51.100.25  ;subdominio gestionado en la misma
                                     ; zona, pero en otro servidor.

mail          IN A     203.0.113.26  ;IPv4 del subdominio mail
ejemplo.com.  IN AAAA  2001:db8::8a2e:370:7629;  IPv6 del subdominio mail
dominio.com.  IN MX    10 mail.dominio.com. ;Definición del mail (MX)

dominio.com.  IN TXT   "v=spf1 include:_spf.hostinger.com ~all"
                       ;Validación SPF del correo
```

La zona DNS tiene que tener, como mínimo:

- Un registro SOA
- Un registro A para IPv4 o uno AAA para IPv6 (o ambos) para el dominio que se quiere definir.
- Un registro NS que especifica el servidor autoritativo para el dominio.

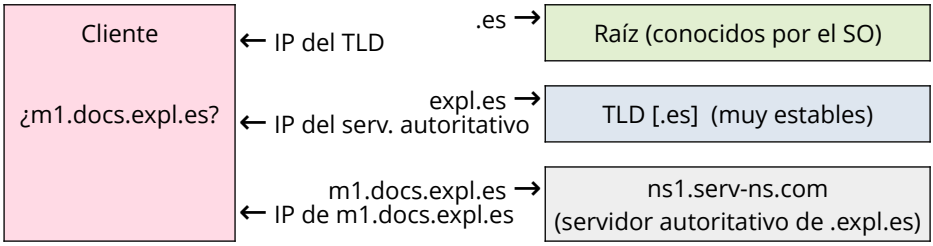
Otros campos son posibles, como registros A/AAA para subdominios, alias (CNAME), mail (MX) o campos de texto (TXT). Los campos de texto se emplean, entre otras cosas, para implementar opciones seguras del servidor de correo (SPF) o publicar la clave pública del dominio.

Ej. 13: Escribe los registros de una zona DNS en donde el servidor vdc.es es el servidor autoritativo de vdc. Además:

- El SOA tendrá valores similares, pero un serial de 7320597714 y que un servidor secundario esperará 2400 segundos antes de reintentar un fallo.
- Existirá un alias para que daw.vdc.es apunte al mismo dominio vdc.es.
- La IPv6 será 2001:db8:85a3::8a2e:370:7334, y la IPv4 será 103.3.3.144.

Delegación a un servidor de nombres

Sin embargo, es muy frecuente delegar en un servidor de nombres. En ese caso, el servidor autoritativo es ese servidor de nombres, que suele proporcionar, como mínimo, un servidor secundario.



En ese caso cambian los siguientes registros:

```
dominio.com.  IN SOA   ns1.serv-ns.com. admin.midominio.com. ( ...
dominio.com.  IN NS    ns1.serv-ns.com.  ;serv. primario definido en SOA
dominio.com.  IN NS    ns2.serv-ns.com.  ;servidor secundario
```

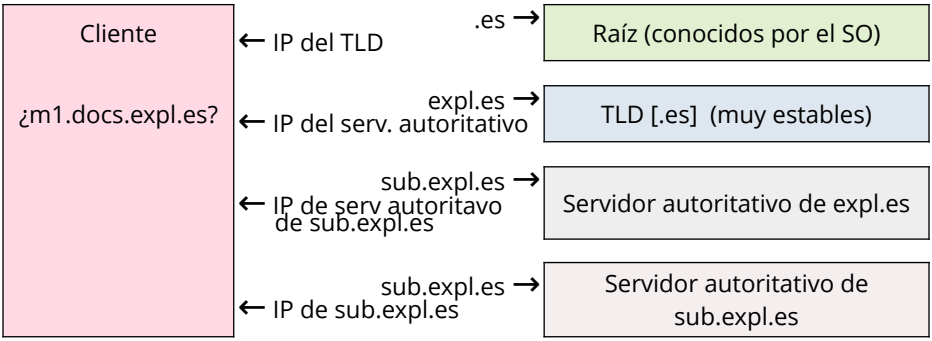
Estos registros habrá que modificarlos en el servidor de nombres (en nuestro ejemplo: ns1.serv-ns.com), ya que ahora nuestro servidor no servirá peticiones DNS.

Ej. 14: Escribe los registros de una zona DNS en donde el servidor de nombres mi-ns1.com es el servidor autoritativo de vdc.es. También existirán:

- Un mail cuya IPv4 será 103.3.3.145 (sin IPV6).
- un servidor DNS secundario: mi-ns2.com
- El resto de datos serán similares al ejercicio anterior.

Delegación de subdominios

En este caso, tenemos un subdominio que está gestionado en otro servidor. En ese caso, en el servidor autoritativo tendremos, en vez de alias (CNAME), el nombre del subdominio con registros NS que apuntarán al servidor autoritativo que gestiona ese subdominio.



En este caso, pondremos que los servidores de nombres del subdominio están gestionados por el propio subdominio:

```
sub        IN NS ns1.sub.ejemplo.com.
sub        IN NS ns2.sub.ejemplo.com.
;También estarían los SOA y demás registros del dominio principal.
```

En el servidor del subdominio tendremos otro SOA, de forma similar a dominio más:

```
$ORIGIN ejemplo.com. ;directiva privada: establece dominio raíz de la zona.

sub.ejemplo.com.  IN SOA  ns1.sub.ejemplo.com. admin.ejemplo.com. (
                      2025101701  3600  1800  604800  86400 )

sub.ejemplo.com.  IN NS  ns1.sub.ejemplo.com.
sub.ejemplo.com.  IN NS  ns2.sub.ejemplo.com.

sub.ejemplo.com.  IN A    203.0.113.50
sub.ejemplo.com.  IN AAAA 2001:db8::50
ns1.sub           IN A    203.0.113.10  ;Los servidores de nombres se
ns1.sub           IN AAAA 2001:db8::10  ;  gestionan en otros servidores
ns2.sub           IN A    203.0.113.11  ;  físicos; hay que iondicar IPs
ns2.sub           IN AAAA 2001:db8::11
```

Ej. 15: Escribe los registros de una zona DNS de el subdomino delegado de daw.vdc.es que posee:

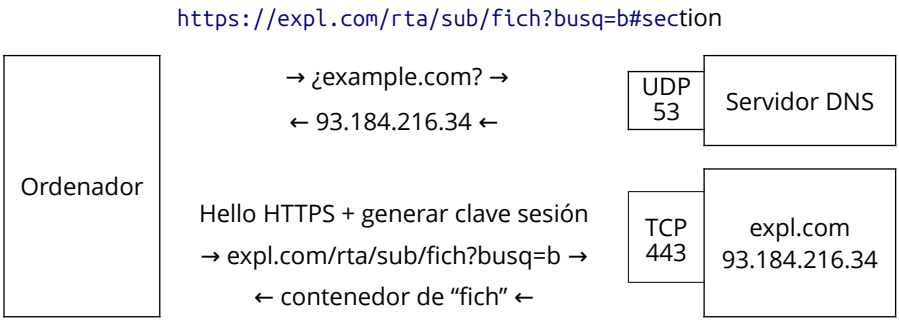
- Un único servidor DNS gestionado en el mismo servidor físico que el subdominio daw.vdc.es.
- Un SOA con datos similares al ejemplo anterior.
- La IP de daw.vdc.es es 183.200.1.5 (sin IPv6).

Resolutores de nombres

El proceso de resolver un DNS es costoso, por lo que los resolutores de nombres tienen cachés que almacenan, temporalmente, los pares dominio-IP. Además, los resolutores se establecen de forma jerárquica, para mejorar el rendimiento.

7. Obtener un recurso web

Toca ahora dirigirnos al servidor web, esto se hará realizando una conexión de red al servidor web, usando la IP obtenida del proceso DNS y el puerto indicado en la URL. Si la URL es válida, y el servidor esta escuchando en el puerto empleado, se establecerá la conexión de red.



HTTPS

Abierta la conexión web, lo primero será utilizar el protocolo HTTPS para implementar la seguridad y privacidad de la comunicación, con los siguientes pasos:

- ClientHello: El cliente envía al servidor las opciones de encriptación de las que dispone.
- ServerHello: El servidor escoge una y le envía su certificado digital Su certificado digital X.509, que incluye su clave pública y que estará firmado por una CA (Autoridad de Certificación).
- El cliente comprueba que el certificado es válido. El navegador tendrá una lista de entidades certificadores con sus correspondientes claves públicas, para poder comprobar la autenticidad del servidor web.
- El servidor y el cliente acuerdan una clave de sesión, con RSA o ECDHE. A partir de ahora, ambos pueden comunicarse de forma encriptada, ya que el resto de la comunicación se cifra con esa clave de sesión.

Ej. 16: ¿Qué pasará si cliente y servidor no tienen ninguna opción de encriptación en común?

Nota: es muy raro hoy día salvo cuando se intente conectar con un servidor obsoleto que solo soporte SSLv3, TLS 1.0 o suites débiles como RC4, y el navegador sea moderno, que soporta TLS 1.2/1.3.

Ej. 17: Busca, en tu navegador, el listado de entidades certificadoras necesarias para poder comprobar que el servidor es lícito.

Ej. 18: Si el navegador, al comprobar el certificado del servidor, detectara que está caducado o no si está firmado por una entidad certificadora que el navegador no tiene en su listado, o si el certificado no es válido ¿qué debería suceder?

HTTP

Ahora el cliente, empleando el protocolo HTTP, empleará la URL para indicarle al servidor qué fichero desea recibir. Entonces, dicho servidor recibirá la petición y devolverá el fichero solicitado. Es posible que, para generar, calcular o seleccionar el fichero a devolver, tenga en cuanta los parámetros de búsqueda especificados en la URL recibida, o que se realicen ciertas operaciones gracias a lenguajes como PHP.

Finalmente el ordenador recibirá el fichero y hará lo que corresponda, por ejemplo, si el programa que realiza el proceso es un navegador en busca de un archivo HTML, mostrará la web en pantalla. La mayoría de navegadores evaluará el contenido de dicha página web y, usando la misma conexión o abriendo otras si es necesario, solicitan otros recursos referenciados por la página web. Estas peticiones a otros recursos emplearán la misma conexión de red si es posible, que será cuando conecten al mismo dominio que la página web original.

Ej. 19: En el proceso anterior, si especificamos el puerto en la URL, ¿qué debe cambiar en el esquema para que todo funcione?

Ej. 20: Cuando el navegador pide y recibe un archivo HTML, necesitará bajar otros archivos del mismo servidor (imágenes, CSS, JS, etc.) ¿deberá

realizar una conexión de red nueva para cada uno de ellos? ¿y si una ejecución de JS de la página realiza un fetch?

Ej. 21: ¿Que pasará si la URL es errónea?

Una prueba con netcat

En una terminal ejecuta el siguiente comando de netcat para realizar una conexión a www.google.com en el puerto 80:

nc www.google.com 80

A continuación, introduce lo siguiente (tendrás que usar el retorno de carro entre una línea y otra, y dos retornos de carro tras la segunda línea). Verás que te devuelve la página web base de Google:

GET /search HTTP/1.0

Host: www.google.com

La primera línea especifica un comando GET, es decir, le pide al servidor que le envíe un fichero, es decir le especificamos la ruta. El fichero solicitado es el que se especifica justo después, en este caso se solicita el fichero en la ruta "/search" (como es un directorio, el servidor devolverá un fichero concreto dentro de ese directorio). La segunda línea especifica el host que, en los casos que nosotros realizaremos, coincidirá con el servidor al que hemos realizado la conexión (el que hay tras el comando "nc").

La resuesta de Google será un código 302, indicando que el fichero que estaba en esa dirección ha sido movido.

Ej. 22: Realiza una petición a la nueva dirección indicada por el mensaje 302, que será la página de inicio de Google (te saldrán un código HTML bastante largo.
Nota: Puedes comprobar que la web recién obtenida es la correcta cargando la misma dirección en un navegador, dirigiéndote al menú "Herramientas → Herramientas del desarrollador → Código fuente de la página" en Firefox, o una ruta similar en otros navegadores.

Para HTTPS, puedes usar el siguiente comando, pero deberás instalar el paquete “ncap” (puedes hacerlo a través de apt):

```
echo -en "GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n" | ncat --ssl google.com 443
```

Ej. 23: Obtén el archivo “prueba.html” dentro del directorio “docencia”, usando “HTTP/1.1” en el host “dragonslite.es”. Debes pasarle un parámetro llamado “asignatura” con el valor “web” y otro parámetro llamado turno con el valor “tarde”.

Unidad Didáctica 2:

HTML 5

1. Lenguaje de marcas

HTML5 (Hyper Text Markup Language, versión 5) es un lenguaje de marcas. Con HTML5, se puede describir tanto el contenido como la estructura de cualquier web, usando dichas marcas, también llamadas etiquetas.

Estructura de un nodo

En HTML 5, un elemento o nodo está compuesto de una etiqueta de apertura, un contenido y una etiqueta cierre:

```
<div>
  Contenido del div
</div>
```

La etiqueta de apertura puede tener ***parámetros***. Cada parámetro podrá tener asociado un valor, añadiendo, a la derecha del parámetro, un igual y el valor del parámetro entre comillas (aunque HTML 5 permitiría no usar las comillas, éstas son obligatorias para seguir la guía de estilo).

```
<div class="mi-clase" id="mi-id" hidden>
  Contenido del div
</div>
```

Cuando una etiqueta no posee nada en su interior, puede usarse la forma ***autoconclusiva***, que consiste en eliminar la etiqueta de cierre, pero añadiendo una / al final de la de apertura (esta barra al final puede omitirse en HTML 5, pero no si se quiere ser compatible con XHTML:

```
<img src="imagen.jpg" alt="Descripción de la imagen" />
```

Las etiquetas autoconclusivas de HTML 5 son: area, base, br, col, embed, hr, img, input, link, meta, param, source, track y wbr.

Anidar nodos

Para construir la estructura del documento HTML, vamos añadiendo nodos uno detrás de otro. Sin embargo, un nodo puede contener a otro:

```
<div class="mi-clase" id="mi-id" hidden>
  <p>Esto es un div</p>
</div>
```

Lo siguiente sería incorrecto, pues se anidan las etiquetas de forma incorrecta.

```
<b><i>Texto</b></i>    <!--Mal: etiquetas cruzadas-->
```

2. Guía de estilo

Según la guía de estilo de HTML[1], deben escribirse en minúsculas las etiquetas, los atributos y los nombres de fichero. También deben usarse las comillas en los atributos, sin espacios alrededor del símbolo de igual. Aunque HTML permite no cerras ciertas etiquetas (<p>, <body>, <head>, etc.), éstas deberían cerrarse.

Cada una de las siguientes líneas representa un ejemplo independiente que, aunque el lenguaje HTML lo permite, no debe escribirse para obtener un buen estilo:

```
<a href = "foro.html"> <!--Sobran espacios-->
<a href=foro.html>     <!--Faltan comillas en atributo-->
<a HREF="foro.html">   <!--Atributo debe ir en minúscula-->
<A href="foro.html">   <!--Etiqueta debe ir en minúscula-->
<a href="FORO.HTML">   <!--Atributo debe ir en minúscula-->
<div><p>Texto</div>    <!--Etiqueta <p> debe tener cierre-->
```

La guía de estilo también establece el uso de ciertas etiqueta <meta>, así como el especificar siempre el título y el lenguaje de la página. Todas esas recomendaciones se incluyen en la estructura básica de un documento HTML que se propone más adelante.

Identado y espaciado

Cuando los parámetros de una etiqueta son demasiado largos, se parten en varias líneas:

```
<a href="https://www.ejemplo.com"
   title="Enlace a Ejemplo"
   target="_blank">Enlace a Ejemplo</a>
```

En general, se establece que un nuevo elemento se establece en una nueva línea:

```
<div>
    <p>
        Este es un párrafo relativamente largo dentro de un div.
    </p>
</div>
```

1 https://www.w3schools.com/html/html5_syntax.asp

Los elementos que son cortos pueden establecerse en una sola línea

```
<p> Párrafo <em>muy</em> corto.</p>
```

Respectos a las separaciones entre líneas, se añaden para separar secciones.

```
    <header>
        <h1>Bienvenidos a mi página</h1>
    </header>

    <main>
        <section>
            <h2>Sección principal</h2>
            <p>Ejemplo de contenido en un párrafo.</p>
        </section>

        <section>
            <h2>Otra sección</h2>
            <p>Texto adicional en la página.</p>
        </section>
    </main>

    <footer>
        <p>Derechos reservados 2024</p>
    </footer>
```

3. Estructura básica

El siguiente código es la base para una página HTML 5 básica, a partir de la cual podremos añadir el contenido necesario.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <title>Título de la página web</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
                initial-scale=1.0">
  </head>
  <body>
     <!-- Contenido de la página web -->
  </body>
</html>
```

`<!DOCTYPE html>` le dice al navegador que el presente documento es una web escrita con HTML5.

`<html>` debe poseer el atributo `lang`, cuyo valor especifica el idioma del documento, en nuestro caso `es`, relativo al idioma español.

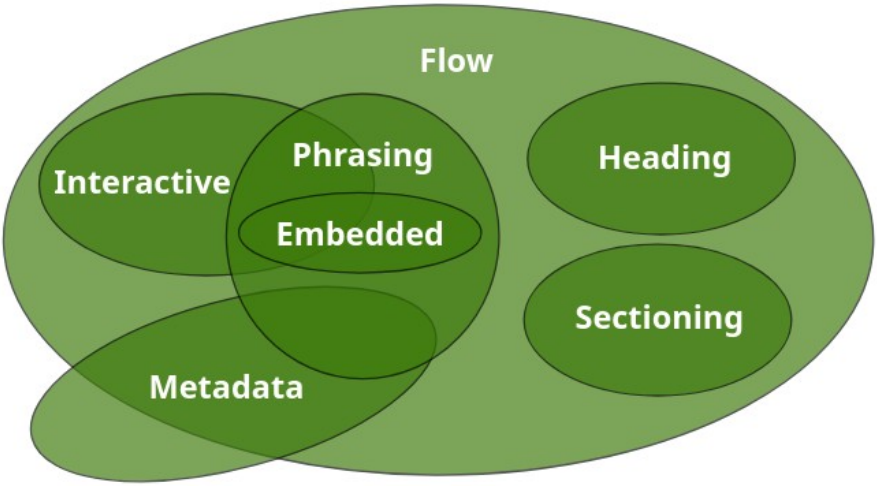
En `<head>` se definen los metadatos del documento, como son el título (obligatorio en todo documento HTML5), palabras clave, estilos, etcétera. Los metadatos no se muestran.

En `<body>` estará todo el contenido del documento: textos, imágenes, enlaces, etcétera, estructurados debidamente con etiquetas.

Es posible validar una página en `https://validator.w3.org/` .

4. Elementos básicos

En HTML 5 existen distintas categorías de elementos:



Phrasing: a, abbr, area (como descendiente de map), b, bdi, bdo, br, button, cite, code, data, datalist, del, dfn, em, i, input, ins, kbd,label, link (en los casos permitidos dentro de body), map, mark, meta (con el atributo itemprop), meter, noscript, output, progress, q, ruby, s, samp, script, select, slot, small, span, strong, sub, sup, template, textarea, time, u, var, wbr. Además, pertenecen a esta categoría los elementos personalizados y el texto plano.

Phrasing y embedded: audio, canvas, embed, iframe, img, math, object, picture, svg y video.

Heading: h1, h2, h3, h4, h5, h6 y hgroup.

Sectioning: article, aside, nav y section.

Solo flow: address, block, quote, details, dialog, div, dl, field, set, figure, footer, form, header, hr, p, pre, ol, search, table.

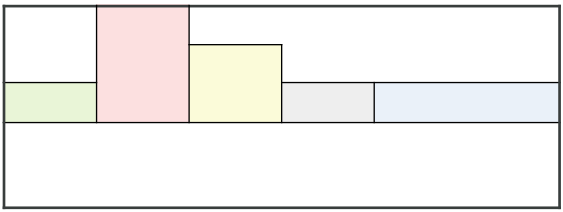
Metadata: base, link, meta, noscript (también pertenece a la categoría phrasing), script, style, template y title.

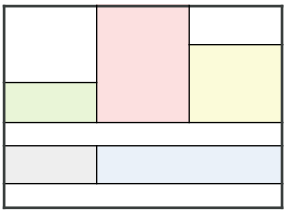
Solo Interactive: select.

https://html.spec.whatwg.org/multipage/dom.html#kinds-of-content

5. Elementos phrasing

Los elementos de la categoría phrasing se asemejan a los elementos inline de HTML 4, en cuanto a su comportamiento por defecto es renderizarse uno tras otro, coincidiendo en su línea base, hasta el final del ínea. Si un elemento es más alto que el resto en la misma línea, los elementos más pequeños dejan un espacio para que en esa línea quepan todos los elementos en altura:





A continuación se comentan los elementos phrasing de HTML 5 y su uso. También entrarían en esta categoría la etiqueta link en ciertos casos concretos y la etiqueta meta cuando tiene la propiedad itemprop.

Elementos semánticos puros

Estos elementos le proporcionan un significado semántico a la parte de texto que rodean.

Cita <cite>. Indica el título de una obra, como un libro, película, o artículo.

```
<cite>Orgullo y prejuicio</cite> es una novela de Jane Austen.
```

Código <code>. Representa un fragmento de código fuente.

```
<code>console.log("Hola, mundo!");</code>
```

Texto eliminado <del>. Indica que el texto ha sido eliminado de un documento. Por defecto, se representa como texto tachado.

```
<p>Este es un texto <del>eliminado</del>.</p>
```

Definición <dfn>. Define un término que se está introduciendo.

```
<dfn>HTML</dfn> es el lenguaje de marcado de hipertexto.
```

Énfasis . Indica énfasis en el texto, generalmente renderizado en cursiva.

```
<em>Este texto es enfatizado.</em>
```

Insertado <ins>. Indica que el texto ha sido insertado en un documento. Puede utilizarse para indicar correcciones en un texto o para indicar contenido nuevo en una página web. Ejemplo:

```
<p>Este es un texto <ins>agregado</ins>.</p>
```

Teclado <kbd>. Representa un texto, comando o combinaciones de teclas que habrán ser introducidas en un teclado por el usuario para realizar la tarea que se esté describiendo.

Texto importante <mark>. Resalta texto que es relevante para el contenido.

```
<p>Este es un <mark>texto destacado</mark> en el párrafo.</p>
<p>Presiona <kbd>Ctrl</kbd> + <kbd>C</kbd> para copiar.</p>
```

Resultado <output>. Representa el resultado de un cálculo o una acción del usuario.

```
<input id="num1" type="number" value="0"
    oninput="document.getElementById('res').value = this.value">
<output id="res">0</output>
```

Cita <q>. Representa una cita corta en línea.

```
<p>Ella dijo: <q>Esto es un ejemplo de cita.</q></p>
```

No relevante <s>. Representa texto que ya no es relevante o preciso (tachado).

```
<p>Este texto está <s>eliminado</s>.</p>
```

Salida <samp>. Representa el resultado de un programa o una salida de muestra.

```
<p>El resultado es: <samp>42</samp>.</p>
```

Texto pequeño <small>. Representa texto menos importante o más accesorio, puede que aclaratorio, un aviso legal o similar. Por defecto, aparece más pequeño que el texto circundante.

```
<p>Texto normal <small>Texto más pequeño</small>.</p>
```

Gran énfasis . Indica un énfasis fuerte, generalmente renderizado en negrita.

```
<strong>Texto enfatizado</strong>
```

Subíndice <sub>. Representa texto en subíndice.

```
H<sub>2</sub>O
```

Superíndice <sup>. Representa texto en superíndice.

```
E = mc<sup>2</sup>
```

Ej. 24: Crea un archivo HTML 5 que tenga el siguiente texto:

El uso de IA permite la técnica del *Upscaling*, de forma que el juego genere sus frames a una resolución y la tarjeta gráfica reescale cada uno de esos frames a una resolución mayor. ~~Esta resolución puede ser hasta el triple en cada una de las dimensiones.~~ Por ejemplo:

[640px x 360px $]_{x3}$ → 1920px x 1080px

También es posible el *Frame Generation*, que permite *inventarse* ~~varios frames~~ un frame a partir del frame anterior generado por el juego, aumentando los FPS.

En palabras de Tom's Hardware: "DLSS [] adds visual details that may surpass native rendering".

La recientes recnologías de Upscaling y Frame generation son DLSS™ de Nvidia, FSR™ de AMD y XeSS™ de Intel. Estas tecnologías son marcas registradas de sus respectivas compañías.

Importante: estas técnicas han de ser soportadas por el juego, y suponen una mejora muy importante en el rendimiento.

Para ver las tarjetas gráficas de tu sistema, puedes introducir en una terminal: glxinfo | grep "OpenGL. Saldrá un texto del estilo:

OpenGL vendor string: AMD
OpenGL renderer string: AMD Radeon Graphics (radeonsi, renoir, LLVM 17.0.6, DRM 3.57, 6.8.0-45-generic)

- El texto entre comillas del cuarto párrafo es una cita.
- “Upscaling” y Frame Generation son definiciones.
- “ ~~Varios frames~~” es texto erróneo a eliminar.
- “inventarse” es una palabra con énfasis.
- El párrafo “Las recientes ...” ha sido añadido recientemente.
- El párrafo “Importante … “ es importante que ha de ser resaltado.
- “Estas tecnologías...” hasta el final de párrafo, es texto aclaratorio.
- “1920px x 1080px” es un resultado
- “glxinfo | grep "OpenGL” es un comando de teclado.
- El último párrafo es la salida de un comando.
- Los índices y superíndies deben ser etiquetados con html.
- “Esta res … dimensiones” es texto superfluo a eliminar.

Elementos de valor

Abreviatura: <abbr>. Representa una abreviatura o acrónimo.

```
<abbr title="Hypertext Markup Language">HTML</abbr>
```

Valor con formato <data>. Asociado a un valor con un formato específico, útil para máquinas.

```
<data value="2023-09-22">22 de septiembre de 2023</data>
```

Medida <meter>. Representa una medida dentro de un rango conocido, como una escala de progreso.

```
<meter value="0.6">60%</meter>
<bdi>Texto en otro idioma</bdi>
```

Progreso <progress>. Indica el progreso de una tarea en curso.

```
<progress value="70" max="100">70%</progress>
```

Hora/fecha <time>. Representa una fecha, hora o intervalo de tiempo. Ejemplo:

```
<time datetime="2023-09-22">22 de septiembre de 2023</time>
```

Variable <var>. Representa una variable en una expresión matemática o lógica.

```
<p>La ecuación es: <var>x</var> + 2 = 10</p>
```

Componentes

Área <area> (solo como descendiente de map). Define una zona activa dentro de un mapa de imagen.

```
<img src="imagen.jpg" usemap="#miMapa" alt="Mapa">
<map name="miMapa">
  <area shape="rect" coords="34,44,270,350" href="pagina.html" alt="Página
de ejemplo">
</map>
```

Botón <button>. Define un botón que puede ser presionado para realizar una acción.

```
<button type="button">Haz clic aquí</button>
```

Lista de opciones <datalist>. Proporciona una lista de opciones predefinidas para un campo de entrada.

```
<input list="frutas">
<datalist id="frutas">
  <option value="Manzana">
  <option value="Banana">
  <option value="Naranja">
</datalist>
```

Campo de entrada <input>. Define un campo de entrada interactivo.

```
<input type="text" placeholder="Escribe tu nombre">
```

Etiqueta <label>. Define una etiqueta para un elemento de entrada.

```
<label for="nombre">Nombre:</label>
<input type="text" id="nombre">
```

Mapa de imagen <map>. Define un mapa de imagen que asocia áreas activas a enlaces.

```
<map name="miMapa">
<area shape="rect" coords="34,44,270,350" href="pagina.html" alt="Página de
ejemplo">
</map>
```

Menú desplegable <select>. Crea un menú desplegable para seleccionar una opción.

```
<select>
  <option value="opcion1">Opción 1</option>
  <option value="opcion2">Opción 2</option>
</select>
```

Área de texto <textarea>. Define un área de texto multilínea para la entrada del usuario. Ejemplo:

```
<textarea rows="4" cols="50">Texto aquí...</textarea>
```

Componente personalizado <slot>. Se utiliza en Web Components para definir un lugar donde se puede insertar contenido.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de Slot</title>
  <script>
    class MyComponent extends HTMLElement {
      constructor() {
        super();
        const shadow = this.attachShadow({ mode: 'open' });
        shadow.innerHTML = `
          <style>
              div {
                border: 1px solid #ccc;
                padding: 10px;
                margin: 10px 0;
              }
          </style>
          <div>
            <h2>Mi Componente</h2>
            <slot></slot>
          </div>
        `;
      }
    }

    customElements.define('my-component', MyComponent);
  </script>
</head>
<body>
  <my-component>
    <p>Este contenido se inserta en el slot del componente.</p>
```

```
  </my-component>
</body>
</html>
```

Plantilla <template>. Define contenido que no se renderiza en la página, pero puede ser instanciado con JavaScript.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo de Template</title>
</head>
<body>
    <template id="myTemplate">
        <div class="item">
            <h3>Título del Item</h3>
            <p>Descripción del item aquí.</p>
        </div>
    </template>

    <div id="container"></div>

    <button id="addItem">Agregar Item</button>

    <script>
        const button = document.getElementById('addItem');
        const container = document.getElementById('container');
        const template = document.getElementById('myTemplate');

        button.addEventListener('click', () => {
            const clone = document.importNode(template.content, true);
            container.appendChild(clone);
        });
    </script>
</body>
</html>
```

Idiomas

Aislado bidireccional <bdi>. Aísla una parte del texto que puede estar en un idioma diferente, evitando que afecte a la dirección del texto circundante.

Sobreescritura bidireccional <bdo>. Controla la dirección del texto (izquierda a derecha o derecha a izquierda).

```
<bdo dir="rtl">Texto en dirección de derecha a izquierda</bdo>
```

Anotación <ruby>. Se utiliza para realizar pequeñas anociones que son renderizadas arriba, abajo o cerca del texto base, usualmente usado para la pronuciación de caracteres del este asiático.

```
<ruby>
  漢 <rt>かん</rt>
</ruby>
```

Salto de línea

***Salto de línea
.*** Inserta un salto de línea en el texto.

```
<p>Primera línea<br>Segunda línea</p>
```

Ruptura de palabra <wbr>. Sugiere un posible punto de ruptura de línea.

```
<p>
  https://this<wbr />.is<wbr />.a<wbr />.really<wbr />
  .long<wbr />.example<wbr />.com/With<wbr />/deeper<wbr />
  /level<wbr />/pages<wbr />/deeper<wbr />/level<wbr />
  /pages<wbr />/deeper<wbr />/level<wbr />/pages
</p>
```

Comportamientos adicionales

Enlace: <a>. Define un hipervínculo que permite navegar a otra página o recurso.

```
<a href="https://www.ejemplo.com">Visitar Ejemplo</a>
```

Texto alternativo a script <noscript>. Proporciona contenido alternativo si el script está deshabilitado o no soportado.

```
<noscript>JavaScript no está habilitado en tu navegador.</noscript>
```

Script <script>. Define un script de programación, como JavaScript.

```
<script>
  console.log("Hola, mundo!");
</script>
```

Contenedor de texto . Un contenedor en línea para aplicar estilos o scripts.

```
<span style="color: red;">Texto en rojo</span>
```

Elementos de formato

Los siguientes elementos solo aplican formato, sin implicar ningún énfasis ni significado semántico. Por lo general, es preferible usar CSS en vez de estos elementos para aplicar formato:

Negrita <b>. Hace que el texto se muestre en negrita, sin implicar énfasis.

```
<b>Texto en negrita</b>
```

Itálica <i>. Representa un texto en cursiva, sin énfasis adicional.

```
<i>Texto en cursiva</i>
```

Subrayado <u>. Subraya el texto, sin implicar énfasis adicional.

```
<u>Texto subrayado</u>
```

Parte II:

Programación básica

Unidad Didáctica 1:

Instalación del entorno

1. Instalación en Linux

En una distribución de Linux, el software se divide en paquetes. Cada paquete aporta una aplicación, una librería, plugin u otro tipo de software. Para instalar un paquete en Ubuntu, podemos usar apt-get:

```
sudo apt-get install «paquete»  #Instala el paquete indicado con todas sus
                                # dependencias
```

Los paquetes pueden tener ***dependencias***, es decir, que necesitan que otros paquetes estén instalados para poder funcionar. En el caso de la derecha, tenemos dos aplicaciones, `vim` y `vim-tiny`, que ambas requieren varias librerías para funcionar. A su vez, el paquete `vim-julia`, necesita `vim` para funcionar. Aunque existen otros muchos, en Ubuntu, tenemos los siguientes comandos para ver información sobre los paquetes y sus dependencias:

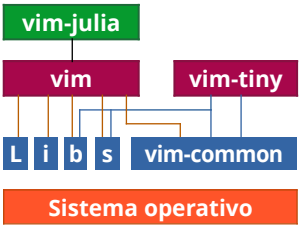


Fig. 1: Paquetes y dependencias

```
sudo apt-cache search «texto» #Busca paquetes que contenga, en su nombre,
                              # descripción, etc. a «texto»
apt-cache show «paquete»   #Da información del paquete, incluidas
dependencias
apt-rdepends -r «paquete»  #Indica los paquetes que dependen de «paquete»
```

Cada paquete tiene tiene una ***versión***. Cuando se arregla algún fallo o se añaden funcionalidades a algunos componentes de un paquete, lo que ocurre es que se publica ese paquete con una versión superior. Es muy normal que un paquete necesite que la versión de su dependencia sea de una versión concreta o superior.

Cuando se lanza una distribución (por ejemplo Ubuntu 23.010) ésta se “congela”, de forma que todos los paquetes funcionen bien entre sí, cada paquete en sus respectivas versiones.

Con este sistema, solo es posible consultar, instalar y desinstalar los paquetes que están en los repositorios del sistema, tanto los que trae por defecto el sistema operativo como los que se añadan por parte del administrador (en Ubuntu, a través del sistema de PPA) que veremos más adelante.

Ej. 25: Consulta los paquetes que dependen de `neovim`, y los paquetes de los que `neovim` depende.

Ej. 26: Busca, con `apt-cache`, el paquete `steam`. Di la descripción corta del paquete.

Centro de aplicaciones

La forma más fácil de instalar un paquete, en nuestro caso el de la aplicación VSCode, es abrir la aplicación gráfica de `Centro de Aplicaciones` y, una vez allí, buscar `vscode`. Aparecerá la aplicación y un botón para instalar el IDE.

El problema de esta solución es que no podemos controlar cómo se instala la aplicación. En este caso, para VSCode, lo que realmente está haciendo es instalarla como una aplicación `snap`.

Repositorio de terceros

En caso de programas no soportados por el sistema operativo, como es el caso de VSCode, podemos añadir un repositorio de terceros: en Ubuntu tenemos los Personal Package Archive, abreviado como `PPA`.

En ciertos casos, como con VSCode, debemos primero añadir las ***claves públicas*** del proveedor de la aplicación, que normalmente estarán disponibles en su página web. El siguiente comando descarga, usando `curl`, un fichero con las claves de los productos de Microsoft desde su página web. Luego, usando `gpg --dearmor`, lo convierte a formato texto, guardando el resultado en microsoft.gpg:

```
sudo apt install curl
curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor >
microsoft.gpg
```

Una vez tenemos las claves públicas, copiamos el fichero `microsoft.gpg` al directorio `/etc/apt/trusted.gpg.d/`, de forma que el fichero pertenezca a root, su grupo sea root y tenga permisos `644` (lectura para todos, escritura solo para root). Hay muchas formas de hacer esto, una de ellas es:

```
sudo install -o root -g root -m 644 microsoft.gpg /etc/apt/trusted.gpg.d/
```

Todo lo anterior, insistemos, es solo necesario para ciertos paquetes. Lo que si es necesario siempre es añadir el repositorio al sistema, usando el comando `add-apt-repository`:

```
sudo add-apt-repository "deb [arch=amd64]
https://packages.microsoft.com/repos/vscode stable main"
```

Con esto, Ubuntu refrescará la base de datos de paquetes disponibles, y el paquete en cuestión ya estará disponible, y podremos realizar una instalación típica, como en el apartado anterior:

```
sudo apt-get install code
```

Podemos ver los repositorios activos de múltiples formas. Una de ellas es con la aplicación gráfica Actualización de Software, pulsando el botón de Configuración, donde veremos varias pestañas, siendo las dos primeras sobre los repositorios instalados y activos:

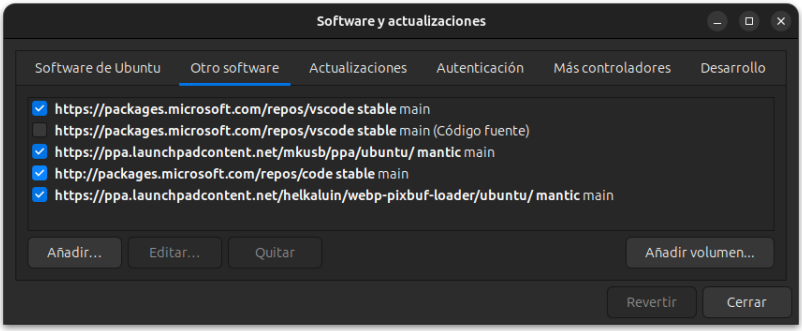


Fig. 2: Actualización de Software, repositorios no oficiales

Ej. 27: Busca otra forma (seguramente con varios comandos), para mover el fichero `microsoft.gpg` al directorio de claves, estableciendo el mismo propietario, grupo y permisos que en el comando de más arriba.

Ej. 28: Busca e instala la aplicación mkusb usando un repositorio PPA.

Snap

Desarrollado por Canonical, Snap es un sistema en el que el paquete en cuestión se instala en un entrono aislado[2]. En dicho entorno, la aplicación instala las librerías que necesite, independientemente de que dichas librerías estén o no instaladas en el sistema base. Esto tiene varias ventajas, pero la principal es que las librerías instaladas en el entorno pueden ser distintas versiones que las instaladas en el sistema, y permite a los desarrolladores publicar un software para múltiples distribuciones Linux. También es posible impedir que el paquete (en definitiva, las aplicaciones que contiene) modifiquen el sistema base. Por otro lado, se desperdicia espacio y rendimiento. Puedes conocer un poco más sobre esto en `https://www.youtube.com/watch?v=F6qbDQo4nrM` .

Snap es bastante sencillo de instalar, destacando la opción `--classic`: si incluimos dicha opción, el paquete tendrá acceso al sistema base, en caso contrario no lo tendrá. En caso de VScode, podemos emplear los siguientes comandos para instalarlo y desinstalarlo:

```
sudo snap install --classic code  #Instala VSCode con acceso al sistema
base
sudo snap --remove code  #Desinstala VSCode
```

Ej. 29: Cuando instalamos VSCode con snap, éste tiene acceso a los directorios y ficheros de este sistema ¿por qué es así?

Ej. 30: instala otro programa con snap (telegram, steam, gimp, etc.), pero sin acceso a ficheros del sistema.

Paquetes Deb

Los paquetes deb son ficheros que podemos descargar, y conforman una extensión del sistema de paquetes y dependencias del sistema operativo. En el paquete deb está la aplicación en cuestión que queremos instalar, e información sobre las librerías necesarias para su funcionamiento.

Para VSCode nos descargamos el fichero deb desde `https://code.visualstudio.com/`, y obtendremos un fichero con extensión `.deb`.

[2] Existen alternativas a Snap, como Flatpak, AppImage y otros.

Podemos Instalar la aplicación de forma gráfica usando Gdebi, o con el comando dpkg:

```
sudo dpkg -i «fichero»  #Instala el paquete contenido en el fichero .deb
sudo dpkg -r «paquete»  #borra el paquete en cuestión, en cuestro caso:
code
```

Observa que, al desinstalar, debemos especificar el nombre del paquete, no del fichero que nos descargamos en primer lugar.

Los paquetes deb, bien actualizados, permiten instalar aplicaciones de forma muy cómoda para el usuario. Sin embargo, tienen una serie de desventajas, como la mayor dificultad de creación y mantenimiento comparados con los paquetes Snap: un paquete deb es específico para una distribución y, posiblemente, versión.

Puedes ver una rápida comparativa en https://www.youtube.com/watch?v=dIdamacw_CE

Ej. 31: Los paquetes deb pueden ocupar desde algunos Kbytes a Mbytes, ¿a qué se debe esto?

Código fuente

También puedes instalar aplicaciones a partir del código fuente. Esta solución es la más compleja, pero es la que el resultado se ajustará a las librerías de nuestro sistema, creando un ejecutable específico para nuestro sistema.

Lo primero que hay que hacer es instalar las herramientas necesarias para compilar programas, y las librerías de compilación que el programa emplea, aparte de los paquetes para el funcionamiento de la aplicación. Para compilar VSCode en un entorno Debian reciente, las librerías necesarias son:

```
sudo apt-get install build-essential g++ libx11-dev libxkbfile-dev
libsecret-1-dev python3 npm
```

Los paquetes `build-essential` y `g++` son herramientas de compilación, los paquetes terminación `-dev` son paquetes con cabeceras de compilación. El paquete `python3` es un lenguaje de script usado por VSCode y, en cuanto a `npm`, lo usamos para obtener `yarn`, que lo usaremos para crear el ejecutable de VSCode.

El siguiente comando de `git` bajará el código de VSCode en una carpeta. Lo siguiente será entrar en la carpeta e instalar la última herramienta de compilación:

```
git clone https://github.com/microsoft/vscode.git
cd vscode
npm config set legacy-peer-deps true
npm install yarn
npm install ternary-stream gulp-merge-json esbuild jsonc-parser parse-
semver
```

Finalmente, con `yarn`, creamos el ejecutable. Recuerda que estamos ya dentro de la carpeta vscode:

```
./node_modules/yarn/bin/yarn
./node_modules/yarn/bin/yarn run gulp vscode-linux-x64
./node_modules/yarn/bin/yarn run gulp vscode-linux-x64-build-deb
```

Podemos configurar el sistema para evitar la telemetría:

```
mkdir -p ~/.config/Code\ -\ OSS/User
echo -e "{\n   \"telemetry.enableCrashReporter\": false,\n
\"telemetry.enableTelemetry\": false\n}" > ~/.config/Code\ -\
OSS/User/settings.json
```

Podemos instalar el sistema para que sea accesible para todos los usuarios:

```
sudo mv ../VSCode-linux-x64 /opt/
sudo chown -R root:root /opt/VSCode-linux-x64
sudo ln -s /opt/VSCode-linux-x64/bin/code-oss /usr/local/bin/vscode
```

Ej. 32: ¿Qué hace el paquete build-essential?

2. Entorno VSCode

El entorno gráfico de VSCode es similar a muchos otros IDEs, con un explorador a la izquierda, unos editor de texto separador por pestañas a laizquierda, y un entorno de consola, terminal, depuración, etcétera en la parte inferior (oculto haasta no emplear alguno de ellos). Puedes echar un vistazo rápido al entorno de VSCode en `https://www.youtube.com/watch?v=CxF3ykWP1H4`, y de manera un poco más detallada en `https://www.youtube.com/watch?v=FzRGSlhQIyY`.

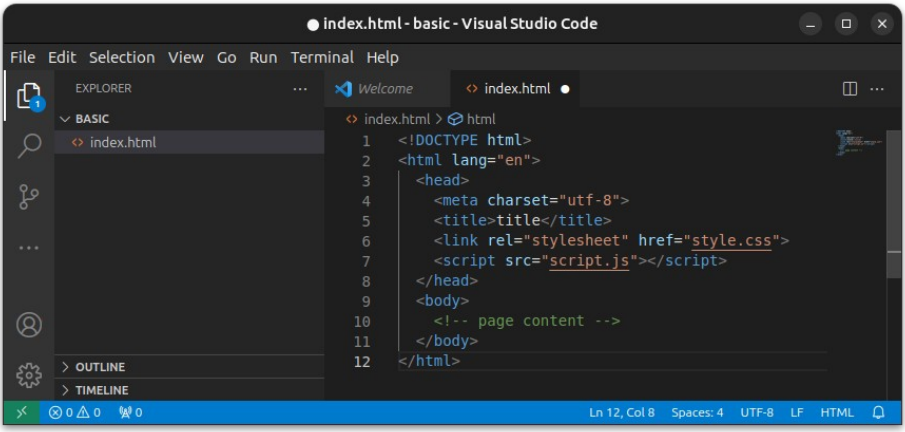


Fig. 3: VSCode

Barra lateral

A la izquierda vemos la barra lateral, que contiene el explorador, el gestor de extensiones y otros, al cual podemos acceder con los iconos más a la derecha. La barra lateral puede abrirse y cerrarse con `Ctrl+B`.

El primer icono de la barra lateral pertenece al ***Explorador***, el cual podemos seleccionar con `Ctrl+Shift+E`. Aquí aparecerán los ficheros y directorio de nuestro proyecto. Pulsando con el botón derecho sobre cualquier fichero o carpeta podremos borrarlo, renombrarlo, abrirlo en el editor (también es posible hacerlo con doble click), etcétera. Cuando aparezca un círculo azul con un número, significará que hay uno o varios ficheros sin guardar.

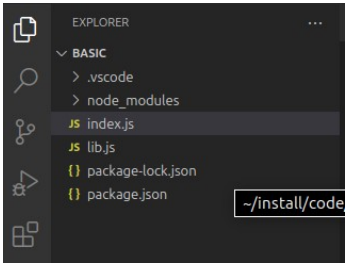


Figura 4: Explorador

Cuando tienes el cursor sobre el explorador, arriba aparecerán 4 iconos. De izquierda a derecha, te permiten (1) crear un nuevo fichero, (2) Crear un nuevo directorio (3) Refrescar, para el improbable caso de que haya habido un cambio externo que aún no se ha reflejado y (4) Colapsar el árbol de directorios.

El segundo icono, la lupa, es el ***buscador***. Rellenando campo de texto, buscará lo indicado en todos los archivos del proyecto (pero no en node_modules), o en ciertos ficheros, si pinchamos en los tres puntos y rellenamos los campos. También podemos hacer búsquedas sensibles a mayúsculas y minúsculas, que coincidan solo con palabras enteras, o buscar ficheros que se ajusten a una expresión regular, pulsando los iconos a la derecha. Finalmente, también es posible hacer la función de reemplazar.

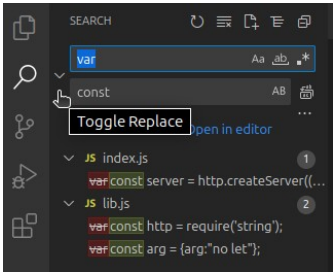


Fig. 5: Buscador

Los iconos sobre ***repositorios*** y sobre la ***ejecución de programas*** se explicarán más adelante.

El último icono a la derecha es el ***gestor de extensiones***. Aparecerán las extensiones instaladas, así como algunas extensiones del marketplace populares y recomendadas. Podemos usar el campo de texto para buscar extensiones. Pulsando sobre una extensión veremos los detalles y podremos instalarla y desinstalarla.

Ej. 33: Realiza los ejercicios de expresiones regulares en VSCode.

En primer lugar, descarga el archivo y sitúa las 3 carpetas de textos de ejercicios en un proyecto/directorio abierto por VSCode. Luego, dirígete a la búsqueda y realiza los ejercicios de `regex_ejercicios`.

Ej. 34: Instala la extensión Key Promoter.

Editor de código

El editor de código está separado en pestañas, como es usual en los IDEs, pudiendo cerrar un fichero con solo pulsar en la “x” al lado del nombre del

fichero. Puedes dividir la zona de edición en varios editores pulsando el icono que hay arriba a la derecha, y también puedes mover los ficheros abiertos de una zona a otra arrastrando la pestaña que contiene el nombre.

El editor de código posee ***resaltado de sintaxis***, de forma que el código aparece con distintos colores para facilitar su lectura y edición. Si un código se presenta todo de un mismo color, es porque aún no se ha detectado el lenguaje que usa, o bien VSCode no posee el plugin adecuado para hacerlo. Si VSCode no posee por defecto la capacidad para resaltar un lenguaje, posiblemente haya algún plugin que podamos buscar en el el gestor de extensiones.

Una de las funcionalidades más interesantes del editor de código es la de ***autocompletar***. Ésta puede usarse con el tabulador. También necesitará saber el lenguaje usado y poseer un plugin para ese lenguaje.

Otra funcionalidad es el ***formateo de código***, que ajusta las tabulaciones y los espacios del código a un formato estándar. En Linux, el atajo de teclado es `Ctrl+May+I`.

Ej. 35: Crea un fichero con un programa de Python (la extensión de los ficheros de Python es `.py`). Instala un plugin para añadir el resaltado de código para Python.

Ej. 36: Visualiza un mismo fichero de código en dos editores, uno al lado de otro (visibles ambos a la vez).

Ejecución JS con LiveServer

Una vez instalado, deberemos iniciarlo y nos aparecerá un mensaje de bienvenida, con un botón que nos pedirá o bien crear/abrir un proyecto, y otro botón para abrir un directorio. Pulsaremos sobre este segundo botón y seleccionaremos una carpeta vacía donde, más adelante, iremos creando los ficheros de ejercicios. En el menú anterior también es posible crear carpetas con el botón que hay arriba a la derecha.

Tendremos ahora un panel lateral, a la izquierda, con varios botones. Pulsando el quinto y último de ellos (un icono de cuatro bloques), deberemos buscar la extensión `Live Server (Five Server)` e instalarla.

Posteriormente, podemos pinchar sobre el primero de los iconos (un icono de dos papeles) y nos aparecerá el explorador de ficheros. Ahí podremos, ya sea pulsando el botón derecho y seleccionando Nuevo Fichero (New File), o bien pulsando los iconos de arriba, crear nuevos ficheros, como por ejemplo, un nuevo fichero html.

Finalmente, pulsando sobre los nombres de los ficheros en el explorador, abriremos el contenido del fichero, que estará vacío si acabamos de crearlo.

3. ¿Qué es un programa?

Un programa es un conjunto de instrucciones que son ejecutadas por el ordenador para realizar una tarea. Por ejemplo, lo siguiente podría ser un programa genérico compuesto por 4 instrucciones:

```
Pide al usuario que introduzca un número
Pide al usuario que introduzca otro número
Suma los dos números
Muestra el resultado de la suma
```

El ordenador ejecuta las instrucciones una a una, de forma secuencial, empezando por la primera y continuando hacia abajo. Por tanto, en el ejemplo anterior, empezaría pidiendo un dato, luego otro, luego realizaría la suma y finalmente mostraría el resultado.

Además, las instrucciones se ejecutan de derecha a izquierda, de forma que, si la instrucción tiene dos o más partes, se evalúa primero la más a la derecha, y luego se continúa hacia la izquierda. Veremos más sobre esto en el apartado de variables.

Áreas de sistemas y de desarrollo

Mientras está claro que los conceptos de programación son fundamentales para el área de desarrollo de software, también son necesarios en el rea de sistema.

A la hora de mantener o implantar un sistema informático, es muy frecuente tener que realizar una misma tarea muchas veces. También es frecuente que debamos estar preparados para realizar una tarea previamente prevista lo más rápido posible, con el fin de minimizar molestias e inconvenientes a los usuarios. La creación y uso de programas en un sistema informático nos permite conseguir ambos objetivos.

4. El primer programa

Usando el explorador de ficheros, crea un nuevo fichero y ponle el nombre de `index.html`. Insértale el siguiente contenido:

```
<!doctype html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
          initial-scale=1">

    <title>Hola mundo </title>
    <link rel="shortcut icon" href="#" />
    <link rel="icon" href="images/favicon.png"
          type="image/png" />
</head>

<body>

    <script>
        document.write("¡Mi primer programa!");
        console.log("Script ejecutado con éxito");
    </script>

</body>

</html>
```

Reformatear y grabar código

Cuando escribimos el código en las ventanas de contenido, podemos usar la combinación de teclas `Ctrl-Mayús-I` para reformatear el código de forma correcta. También podemos usar `Ctrl-S` para grabar las modificaciones.

5. Ejecutar el programa

Abajo a la derecha podemos ver el texto `Go Live`, establecido por la extensión que instalamos anteriormente. Si pulsamos sobre él, se iniciará un navegador en el que podremos ver el resultado de nuestra página web.

Si modificamos el fichero y lo guardamos (`Ctrl-S`), la página web se autorecargará, mostrando en el navegador el resultado de la ahora modificada web.

Ej. 1: Modifica el programa anterior para que, en vez del mensaje actual, muestre el siguiente mensaje: ¡Hola mundo!. Bastará con que hagas los cambios y los guardes para que automáticamente se muestren dichos cambios en el navegador.

Unidad Didáctica 2:

Variables y operaciones

1. Variables y tipos

En JS podemos definir variables. Cada variable posee un nombre y un valor. Por ejemplo, con la siguiente instrucción definimos la variable llamada `x`, y le asignamos el valor de 10.

```
let x = 10;
```

Vamos a examinar esta instrucción. Recuerda que las instrucciones son interpretadas por el ordenador de derecha a izquierda. En este caso, JS evalúa la parte izquierda, que tiene el valor de 10. Entonces, ese valor se lo asigna a la variable `x`, con lo que `x` es una variable que, tras ejecutarse la instrucción, termina valiendo `10`.

Tipos de datos básicos

Las variables pueden contener distintos tipos de datos. Los tipos básicos son los siguientes:

- Número: `let x = 15;` `let pesoTotal = -1.3;`
  Los números (number) engloban todo tipo de dato numérico, ya sea entero o con decimales.

- Cadenas: `let mensaje = “Hola”` `let busqueda = ”a23”;`
  Las cadenas (String) pueden contener números, letras y otros símbolos, y se diferencian de los números gracias a las comillas de inicio y de fin.

- Booleanos: `let hoyLlovera = true;` `let alarma = false;`
  El tipo de dato booleano tan solo tiene dos posibles valores, `true` o `false`.

- Undefined: `let tema = Undefined;` `let kms = Undefined;`
  El tipo de dato Undefined (No definido) significa que el valor de la variable no está definido.

En JS también existen otros tipos de datos (los tipos de datos Enteros largos, Null, Symbos y Object, además de los Arrays), que los veremos más adelante.

Literales

Se llaman literales a los valores concretos que hay en el código. Por ejemplo, en las instrucciones de más arriba, tenemos literales tales como: `15`, `-1.3`, `"¡Hola!"` y `Undefined`.

Nombres de las variables

Los nombres de las variables serán descriptivos respecto a su contenido. Estos nombres podrán tener solo:

- Dígitos (`0` – `9`).
- Letras, pudiendo JS soportar sin tildes, diéresis y otros.
- Símbolos de dólar y guion bajo (`$` o `_`).

La variable NO puede empezar por un número. Es costumbre, además que, si las variables están compuestas de varias palabras, se ponga mayúscula la primera letra de cada palabra más allá de la primera. Ejemplo de ésto serían: `resultadoMayor`, `estadoDelCalefactor`, `dniSinLetra` o `arbolDeDirectorios`.

NO serían correctos los siguientes nombres de variables, si se quieren seguir las guías de estilo:

- 1camino (empieza por número).
- numerofinal (las segundas palabras deben ser mayúsculas).
- Resultado (la primera letra es mayúscula).
- OtroResultado (la primera letra es mayúscula)..

2. Inputs y Alerts

Uno de los comando más básicos de javascript para mostrar un número, una cadena, o cualquier otro tipo de dato es `alert`.

Para introducir datos debemos ayudarnos del HTML. Deberemos crear un campo de texto para cada dato. También podemos usar un botón para "disparar" la acción que deseamos hacer. Para ello utilizaremos el elemento `onClick` llamando a una función.

En JS, podemos acceder a un elemento HTML concreto con `document.getElementById('...')`, substituyendo los puntos suspensivos por la id del elemento. Los elementos obtenidos son de tipo cadena, por lo que, para realizar operaciones con ellos, deberemos utilizar la instrucción parseInt.

```
<body>
  <input id="input1" type="text" />
  <input id="input2" type="text" />

  <input id="action" type="button" value="Sumar" />
  <label id="res">---</label>

  <script>

    document.getElementById('action').onclick = function () {
      let cad1 = document.getElementById('input1').value;
      let int1 = parseInt(cad1);
      let cad2 = document.getElementById('input2').value;
      let int2 = parseInt(cad2);
      let rslt = int1 + int2;

      document.getElementById('res').innerHTML = rslt;
    };

  </script>
</body>
```

3. Cadenas

Como hemos visto arriba, una variable puede almacenar, en vez de un número, una cadena. Por ejemplo, la instrucción siguiente, define una variable llamada mensaje, cuyo contenido es `Hola`:

```
let mensaje = "Hola";
```

Observa que, para que el contenido de la variable sea `Hola`, hemos tenido que asignarle `"Hola"`. Esto es porque, en JS, todos los literales de cadenas se escriben entre comillas, con el fin de que el ordenador pueda diferenciarlas del resto del código. En JS, para definir, podemos usar tanto comillas simples como dobles, mientras coincida el tipo inicial y el final. Además, es posible insertar unas comillas del otro tipo dentro de la cadena. Los siguientes ejemplos también son válidos:

```
let mensaje = 'Hola';
let mensaje = '"Este texto" es una cita';
let mensaje = "'Este texto' es una cita";
```

Concatenar cadenas

Es posible crear cadenas más largas uniendo otras más cortas, usando la concatenación. Para ello, simplemente usamos el símbolo `+` entre ellas. En dicha concatenación es posible combinar tanto literales de cadenas como variables de tipo cadena en la forma que deseemos:

```
let mensaje_base = "Hola.";
let numero = "3";
let mensaje_final = mensaje_base + " El número es ";
let mensaje_final = mensaje_final + numero + ".";
alert(mensaje_final);  //Imprime: Hola. El número es 3.
```

Ej. 2: Crea un programa que pida al usuario tres cadenas (esto serían tres campos de texto), y que, al pulsar un botón luego se muestre un único mensaje con el contenido de los tres campos de texto concatenados, pero separadas por un punto y un espacio. Por ejemplo, si el usuario introduce las siguientes cadenas: `Hola mundo`, `Lunes` y `Dia 13`, entonces el programa imprimirá `Hola Mundo. Lunes. Día 13`.

Longitud de una cadena

Para obtener la longitud de una cadena, se emplea:

***«cad»*.length** Devuelve un número con la longitud de de la cadena.

```
let saludo = "¡Hola!"
let longitud = saludo.length;
alert(longitud);  // Imprime: 6
```

Subcadenas

Existen dos métodos para obtener subcadenas:

***«cad»*.substr(*«inicio»*)** Devuelve una subcadena que contiene desde el carácter de inicio (incluido) hasta el fin de la cadena.

***«cad»*.substr(*«inicio»*, *«longitud»*)** Devuelve una subcadena que contiene desde el carácter de inicio (incluido) y contiene tantos caracteres como los indicados en longitud (o hasta el final de la cadena). El primer carácter de la cadena es el carácter número cero.

```
let letras = "abcdef";
let subcadena1 = letras.substr(1);
alert(subcadena1);  // Imprime: bcdef.

let subcadena2 = letras.substr(2,3);
alert(subcadena2); // Imprime: cde

let subcadena3 = letras.substr(2,10);
alert(subcadena3); // Imprime: cdef
```

También es posible utilizar números negativos en el parámetro “inicio”. Un número negativo indica el carácter i-ésimo contado desde el final.

```
let digitos = "987654321";
let subcadena = digitos.substr(-7, 4)
alert(subcadena)  //Imprime: 7654
```

Ej. 3: Crea un programa en el que el usuario escriba una cadena (asume que ésta tendrá al menos 2 caracteres) y el programa muestre la misma cadena, pero sin el primer carácter.

Ej. 4: Crea un programa en el que el usuario escriba una cadena (asume que ésta tendrá al menos 2 caracteres) y el programa muestre los 2 primeros caracteres de dicha cadena.

Ej. 5: Crea un programa en el que el usuario escriba una cadena (asume que ésta tendrá al menos 3 caracteres) y el programa muestre los 3 últimos caracteres de dicha cadena.

Ej. 6: Crea un programa en el que el usuario escriba una cadena (asume que ésta tendrá al menos 5 caracteres) y el programa muestre los 5 primeros caracteres de dicha cadena, luego un guion, y luego el resto de la cadena.

Ej. 7: Escribe otro programa en el que el usuario introduzca una cadena. Tras ello, el programa almacenará en una variable cuál es la mitad de la longitud de dicha cadena. Finalmente, usando substr y la variable anterior, el programa imprimirá la segunda mitad de la cadena.

Ejemplo: si la cadena es 12345, se imprimirá 345.

***«cad»*.indexOf(*«subcad»*)** Devuelve la primera ocurrencia de «subcad» dentro de «cad». Si no encuentra «subcad» dentro de «cad», entonces devuelve -1.

```
let cadena = "Hola mundo";
let index = cadena.indexOf("mundo");
alert(index);  //Imprime: 5
```

Como curiosidad, este código puede compactarse en el siguiente:

```
alert( "Hola mundo".indexOf("mundo") );  //Imprime: 5
```

Ej. 7b: Realiza un ejercicio en el que el usuario introduzca dos cadenas, una cadena base y una cadena de búsqueda. El programa deberá decir en qué posición se encuentra la cadena de búsqueda dentro de la cadena base. Si no se encuentra, mostrará -1.

Ej. 7b: (corregir numeración 7b para el próximo curso).

Ej. 7c: Realiza un ejercicio en el que el usuario introduzca dos cadenas, una cadena base y una cadena de búsqueda. El programa deberá mostrar primera cadena (la cadena base), pero solo desde el punto en que se encuentra la segunda cadena (la cadena búsqueda).

No tengas en cuenta la posibilidad de que el usuario introduzca una cadena de búsqueda que no esté dentro de la cadena base.

Ejemplo: `Hola mundo, ¿qué tal?` y `mundo` devolverá la cadena: `mundo, ¿qué tal?`.

Ej. 7c: (corregir numeración 7c para el próximo curso).

4. Números

Las operaciones básicas en JS son las siguientes:

«num_1» + «num_2» suma dos números.

«num_1» - «num_2» resta dos números.

«num_1» * «num_2» Multiplica dos números. Observa que se usa el asterisco, nunca la equis (`x`), el aspa (`×`) ni ningún otro carácter.

Conversión Cadena ⇔ Número

Observa que, para realizar correctamente las operaciones numéricas, «num_1» y «num_2» deben ser variables de tipo número, no de tipo cadena. El problema aquí, es que con las instrucciones de `document.getElementById('...').value;` devuelven valores de tipo cadenas:

```
let op1 = document.getElementById('op1').value; //op1, op2 son
let op2 = document.getElementById('op2').value; // cadenas

let suma = op1 + op2  //Concatena op1 y op2
alert(suma)  //imprime op1 y, luego, op2, no la suma
```

Para solucionarlo, usamos parseInt para convertir las cadenas a datos numéricos:

```
let op1 = document.getElementById('op1').value; //op1, op2 son
let op2 = document.getElementById('op2').value; // cadenas

let num1 = parseInt(op1);
let num2 = parseInt(op2);
let suma = num1 + num2  //Suma los números

alert(suma)  //imprime la suma
```

En este código de arriba, la `suma` es una variable definida a partir de la suma de dos números, por lo que también es una variable de tipo numérico.

`Ej. 8:` Escribe un programa en el que el usuario deba introducir un número que exprese una cantidad entera de litros. Una vez introducida la cantidad, dicho programa convertirá ese valor a cc3 (centímetros cúbicos) y mostrará el resultado.

Ej. 9: Escribe otro que convierta lb (libras) a Kg (kilogramos).

Ej. 10: Escribe otro programa en que el usuario deba introducir 3 números, que representarán horas, minutos y segundos al total de segundos.

El programa calculará el número de segundos totales qu representa lo anterior. Por ejemplo, si se le introduce 1 horas, 2 minutos y 3 segundos, devolverá 3723 segundos (1 * 60*60 + 2 * 60 + 3 = 2723).

Ej. 11: Escribe otro que convierta grados centígrados a grados fahrenheit.

Ej. 12: Escribe un programa que convierta cc/s (litros por cada segundo) a l/h (litros por cada hora). Nota: 1 cc/s equivale a 3.6 l/h

Ej. 13: Escribe otro que convierta de kilómetros cuadrados a millas cuadradas.

Ej. 14: Escribe otro programa que convierta milibar (milibares) a lb/pie2 (libras por cada pie cuadrado).

«num_1» / «num_2» Divide dos números. Aún en caso de que se dividan números enteros, el resultado de la división será un número en coma fotante.

Math.floor(«num_1») Redondea al entero justo menor a «num1».

Math.ceil(«num_1») Redondea al entero justo mayor a «num1».

Math.trunc(«num_1») Remueve la parte decimal de «num1».

Math.round(«num_1») Redondea al entero más cercano a «num1». Un número con parte decimal igual a `.5` se redondeará al entero superior.

	-3.7	-3.5	-3.1	3.1	3.5	3.7
floor	-4	-4	-4	3	3	3
ceil	-3	-3	-3	4	4	4
trunc	-3	-3	-3	3	3	3
round	-4	-3	-3	3	4	4

«num_1» % «num_2» Devuelve el resto de la división. Observa que el módulo siempre será menor que el valor absoluto de num_2, y siempre igual o mayor que 0.

```
let num1 = 7;
let num2 = 3;

let div = num1 / num2;
alert(div);  //Imprime: 1.6666666666666667

let div_entera = Math.floor(num1 / num2);
alert(div_entera);  //Imprime: 2.

let modulo = num1 % num2;
alert(modulo);  //Imprime: 1 (7 entre tres es 2, y sobra 1).
```

Ej. 15: Tenemos cierta cantidad de ordenadores, y queremos asignarlos a diversas aulas, de forma que cada aula contenga exactamente el mismo número de ordenadores.

Escribe un programa en el que se pueda introducir el número de ordenadores disponibles y las aulas existentes. Con ello, calculará y mostrará cuántos ordenadores cabrán en cada aula, y cuántos ordenadores sobrarán.

Por ejemplo, si tenemos 7 ordenadores y 2 aulas, entonces cabrán 3 ordenadores por aula y sobrará un ordenador.

Ej. 16: Escribe un programa en el que el usuario indique el número de Kg totales a trasportar y el número de Kg que puede transportar el camión en un viaje. El programa debe devolver el número de veces que el camión debe viajar para trasportar toda la carga.

Haz el ejercicio de dos formas. La primera, usando Math.floor. Para evitar condicionales, asume que la última carga a transportar nunca van a ser completa. La segunda, usando Math.ceil, si que te permitirá que la última carga sea exacta.

«num_1» ** «num_2» Devuelve num_1 exponentado por num_2.

```
let num1 = 5;
let num2 = 3;
```

```
let exponente = num1 ** num2;
alert(exponente);  //Imprime: 125 (5 x 5 x 5 = 125).
```

Ej. 17: Crea un programa que calcule el número de combinaciones posibles con n bits, donde n es indicado por el usuario.

Ej. 18: Escribe un programa en el que el usuario deba introducir dos números, tras lo cual el programa calculará y mostrará todos los siguientes valores, cada apartado en una línea distinta:

a) La suma de ambos números

b) La resta del primero menos el segundo.

c) La multiplicación de ambos números.

d) La división del primero entre el segundo.

e) La división entera entre ambos y, justo después, entre paréntesis, que muestre el resto de la división.

f) El primer número elevado al segundo.

Ej. 19: Pon a prueba el programa introduciendo distintos valores, como por ejemplo, que el primer valor sea menor que el segundo (¿qué sucede en la resta y en la división entera?) o que el segundo número sea cero (¿qué sucede en la división?).

Ej. 20: Una empresa de papelería tiene distintos productos, que vienen siempre en cajas. Todas las cajas contienen el mismo número de productos.

Escribe un programa que pregunte el número de cajas y los productos por caja. Con ello calculará y mostrará el número de productos totales.

Ej. 21: Escribe otro programa, parecido al anterior, en el que se pregunte el número de productos en cada caja, el número de cajas en un palé, y el número de palés. El programa calculará el número total de productos.

5. Booleanos

Los booleanos (tipo boolean) son un tipo de dato que solo puede tomar dos valores: verdadero (`true`) o falso (`false`).

! «bool» devuelve el valor contrario al contenido por «bool».

«bool_1» && «bool_2» devuelve true solo si tanto «bool_1» como «bool_2 son true.

«bool_1» || «bool_2» devuelve true si cualquiera, «bool_1» o «bool_2», son true.

```
let hoyLlueve = false;
let mananaLlueve = true;
alert( !hoyLlueve ) );  //Imprime: true
alert( hoyLlueve && mananaLlueve );  //false
alert( hoyLlueve || mananaLlueve );  //true
```

Para convertir una cadena a booleano se hace de forma especial, con una comparación estricta con la cadena `true`. En este ejemplo se usa también la integración del código JavaScript con inputs y etiquetas HTML, en vez de usar alerts:

```
let op = document.getElementById('op').value; //cadena

let bool1 = (op === 'true');  //bool1 es booleano
document.getElementByID('r').innerHTML = !bool1;
                              //imprime lo contrario de bool1
```

Comparadores

Nos permiten comparar números entre si. El resultado de todos los comparadores es un valor de True o bien False.

«num_1» == «num_2» Devuelve `true` si son iguales, o `false` si son distintos.

«num_1» > «num_2» Devuelve `true` si «num_1» es mayor que «num_2», o `false` en caso contrario.

«num_1» >= «num_2» Devuelve `true` si «num_1» es mayor o igual que «num_2», o `false` en caso contrario.

«num_1» < «num_2» Devuelve `true` si «num_1» es menor que «num_2», o `false` en caso contrario.

«num_1» <= «num_2» Devuelve `true` si «num_1» es menor o igual que «num_2», o `false` en caso contrario.

«num_1» != «num_2» Devuelve `true` si «num_1» y «num_2» son distintos, o `false` en caso contrario.

Además, podemos combinar los comparadores entre si y con las operaciones booleanas básicas para crear expresiones más complejas:

```
let inputVol = document.getElementById('volumen').value;
let inputBrillo = document.getElementById('brillo').value;

let vol = parseInt(inputVol);  //Convierte a entero
let brillo = parseInt(inputBrillo);  //Convierte a entero

let ok = (vol>=20 && vol<=75 && brillo>20 && brillo<80);
document.getElementByID('r').innerHTML = "¿Se ve ok?: " + ok;
  //Imprime true si volumen está entre 20 y 75 y además el
  // brillo está entre 21 y 79, o false en caso contrario.
```

Ej. 22: Escribe un programa en el que el usuario introduzca dos números, e indique si el primero es menor o igual que el segundo.

Ej. 23: Escribe otro programa en el que el usuario introduzca tres números, y el programa indique si están ordenados de mayor a menor.

Estarán ordenados de mayor a menor si el primero es mayor que el segundo, y además, el segundo es mayor que el tercero.

Ej. 24: Escribe otro programa en el que el usuario introduzca tres números, e indique si existen números repetidos.

Habrá números repetidos si, y solo si, el primero es igual al segundo, o bien el primero es igual al tercero, o bien el segundo es igual al tercero.

Ejemplo: si el usuario introduce 8, 8 y 9, devolverá true. También devolverá true si el usuario introduce 9, 9 y 9.

Nota: Deberás hacer dos (y solo dos) comparaciones.

Ej. 25: Escribe otro programa en el que el usuario introduzca tres números, y el programa indique si están ordenados. Es decir, devolverá true tanto si están ordenados de forma creciente como si están ordenados de forma decreciente.

Ej. 26: Escribe otro programa en el que el usuario introduce la longitud en centímetros de una estantería. Además, indicará el grosor de los libros que van a ir en ella y el número de libros. El programa indicará si la estantería puede contener esos libros.

6. Ejercicios adicionales

Ej. 26b: Crea un programa en la que el usuario introducirá una hora en formato HH/MM Usando solo indexOf y substr, el programa mostrará "La hora es HH, y los minutos MM" (subtituyendo HH y MM por los introducidos por el usuario).

Ej. 26c: Crea un programa en la que el usuario introducirá una fecha en formato DD/MM/AA. El programa mostrará "El día es DD, el mes MM y el año AA" (subtituyendo DD, MM y AA por los introducidos por el usuario).

Ej. 26d: Usando solo substr e indexOf, crea un programa en la que el usuario introducirá una cadena que tendrá al menos un carácter `&`. Devolverá solo lo que hasta ese carácter, éste no incluido.

Ejemplo: si se introduce `abc&de`, se devuelve `abc`.

Ej. 26e: Usando solo substr, crea un programa en la que el usuario introducirá una cadena (que tendrá al menos 3 caracteres), y devuelva una cadena igual pero con el primero ni el último carácter.

Ejemplo: si se introduce `abcde`, se devuelve `bcd`.

Ej. 27: Escribe un programa en el que el usuario introduzca una cadena que contenga un paréntesis de apertura y otro de cierre. El programa imprimirá la cadena que hay en medio. Por ejemplo, si el usuario introduce `ab(cde)fg`, el programa sacará `'cdf'`.

Ej. 28: Escribe un programa en el que el usuario introduzca una cadena que contenga dos (y solo dos) guiones. Devolverá la cadena entre los guiones. Por ejemplo, si el usuario introduce `ab-cde-fg`, el programa sacará `cde`. Usa tan solo substr e indexOf para resover el ejercicio.

Pista: busca el primer guion y crea una subcadena que vaya desde el primer guion hasta el final (en nuestro ejemplo `cde-fg`). Luego, en esa cadena resultante, crea una subcadena que vaya desde el principio hasta el segundo guion.

Ej. 28b: Usando solo indexof y substr, analiza una cadena introducida por el usuario, que tendrá una o dos barras (`/`).

Pista: busca la primera barra y crea una subcadena que vaya desde justo después de la primera barra hasta el final (por ejemplo, si el usuario introduce `ab/cd/ef` , la nueva cadena será `cd/ef`). Luego, en esa cadena resultante, busca de nuevo si hay barra usando indexOf: si la hay, entonces hay dos barras en la cadena inicial, si no la hay (indexOf devuelve -1) entonces había una sola barra en la cadena original.

Ej. 29: Crea un programa como el anterior. Sin embargo, en vez de devolver la cadena entre los guiones, devolverá el número de caracteres que hay en cada uno de los segmentos. Por ejemplo, para la cadena `ab-cde-fg` , el programa devolverá `2` , `3` y `2` .

Ej. 30: Crea un programa en el que el usuario indique la longitud que hay que construir de un bordillo de una calle, el cual está formado por una hilera de adoquines. Además, el usuario especificará la longitud de cada adoquín. El programa indicará la longitud del último adoquin, que habrá que acortar para ajustarse a la longitud total.

Ej. 31: Escribe otro en el que el usuario introduzca dos las dimensiones de una parcela rectangular (longitud y anchura expresado en metros). Luego, deberá introducir el espacio que ocupa cada planta que quiere plantar en dicha parcela, expresado también en metros (cada planta ocupa un cuadrado perfecto de lado al indicado por el usuario). El programa debe calcular la cantidad de plantas que caben en la parcela.

Ej. 31b: Crea un programa que convierta el formato decimal de horas a formato horario. Redondea al minuto más cercano.

Ejemplo: 6,75h se transformará en 6:45h

Ej. 31c: Crea un programa que transforme una coordenada 2D hacia la coordenada más cercana que tenga valores enteros.

Ejemplo: la coordenada (2.1,7.1) se transforma a (2,7) .

Ej. 31d: Crea un programa que transforme una coordenada 2D hacia la coordenada que tenga valores enteros más cercana al origen.

Ejemplo: la coordenada (2.1, 0.1) se transforma a (2,0). y la coordenada (-3.5,-7.3) se convierte en (-3,-7).

Ej. 32: Escribe otro en el que el usuario introduzca también longitud y anchura de la parcela (expresados en metros), y el espacio por planta (también expresado en metros). Deberá calcular los metros cuadrados sobrantes.

Ej. 33: Tenemos un palé el cual el usuario indicará cuántos centímetros tendrá de ancho, profundo y alto. También introducirá el ancho, profundo y alto de cada caja de producto.

El programa indicará cuántas cajas de producto caben en un palé. Observa que una pequeña parte del espacio del palé puede que se quede vacía.

Unidad Didáctica 3:

Condicionales

1. Condicional simple

En el condicional simple, una o varias instrucciones se ejecutan dependiendo de si se satisface una condición o no. La condición se pone después de la palabra reservada `if`, y la(s) instrucción(es) a ejecutar solo en el caso de que la condición sea cierta, se ponen justo debajo, con una sangría de 4 espacios:

```
let inputNumero = document.getElementById('volumen').value;
let numero = parseInt(inputNumero);
alert("Inicio")  //Estas 3 líneas se ejecutan siempre

if(numero>3){   //Condición
    let mensaje = "Es mayor que tres";  //Se ejecutan solo
    alert(mensaje);                     //  si numero>3.
}

alert("Fin de programa")  //Se ejecuta siempre
```

En el programa de arriba, tras el mensaje de `Inicio`, se comprobará si `número` es mayor que 3, en cuyo caso se imprimirá el mensaje `Es mayor que tres`. Luego, en otra línea, se imprimirá `Fin`.

En caso de que `numero` sea menor o igual que tres, tan solo se mostrarán los mensajes de `Inicio` y de `Fin del programa`.

Observa que, en este caso, tras la sentencia de `if` existen dos instrucciones (las dos que están entre las llaves) que se ejecutarán tan solo si la condición es cierta. En otros programas, podría haber una cantidad distinta de instrucciones (todas ellas entre las llaves).

Ej. 1: Escribe un programa que pida dos números al usuario, tras lo cual, el programa imprimirá el mensaje `'Inicio de programa'`. Tras ello, mostrará `'Son distintos'` pero solo si los números son distintos entre si (si son iguales, no imprimirá dicho mensaje). Luego, y en todo caso, mostrará `'Fin de programa'`. Solo para este ejercicio, usa alert en vez de escribir en una etiqueta.

Ej. 2: Escribe otro programa que pida al usuario un número y, si es divisible entre 3, indique `'Divisible entre 3'`.

Ej. 3: Escribe otro que también pida una cadena, tras lo cual el programa imprimirá la misma cadena. Tras ello, si la cadena tiene 10 caracteres o más, entonces el programa imprimirá la cadena al revés.

Ej. 4: Escribe otro que pida una cadena al usuario. Si la cadena mide mas de 5 caracteres, el programa sobrescribirá el contenido de la cadena con `error`. Finalmente, mostrará el contenido de la cadena.

Ejemplos: Si el usuario introduce `abcde`, el programa devolverá `error`. Si la cadena introducida es `1234`, se devolverá `1234`.

Ej. 5: Escribe otro programa, muy similar al anterior. Pedirá una cadena al usuario y, si la cadena mide menos de 5 caracteres, el programa imprimirá la cadena concatenada consigo misma. En caso contrario, mostrará tan solo la cadena de entrada.

Ejemplos: Si el usuario introduce `abc`, el programa devolverá `abcabc`. Si la cadena es `12345`, se devolverá `12345`.

Ej. 6: Escribe otro que pida dos cadenas. El programa indicará la posición de la segunda cadena en la primera (para ello, puedes usar la función indexOf). Sin embargo, si la segunda cadena no está en la primera, también se imprimirá en mensaje `Cadena no contenida`.

Nota: Cuando una cadena no está contenida en otra, la función indexOf devolverá -1.

Ej. 7: Escribe otro que pida dos cadenas. El programa indicará la posición de la segunda cadena en la primera (para ello, puedes usar la función indexOf). Tras ello, si dicha posición está en la primera mitad de la cadena, el mensaje incluirá también `, Está en la primera mitad`.

Ejemplos: La primera cadena es `abcdef` y la segunda `cd`. La función index devolverá `2 Está en la primera mitad` (ya que la posición de `b` es 2.

Condiciones complejas

Observa que la condición puede ser una expresión tan simple o tan compleja como sea necesario. El único requisito es que dicha condición debe ser una

expresión lógica cuyo resultado, al ser la expresión evaluada por Python, sea `True` o `False`.

```
let inputNumero = document.getElementById('volumen').value;
let numero = parseInt(inputNumero);  //Se ejecutan siempre

if(3<numero && numero<5){  //Condición lógica
    let mensaje = “Entre tres y cinco”  // Se ejecutan solo
    alert(mensaje)                      //  si 5>numero>3.
}

alert(“Fin del programa”)  //Se ejecuta siempre
```

También es posible utilizar una variable de tipo booleano como condición. El programa anterior y el siguiente se comportan de forma idéntica:

```
let inputNumero = document.getElementById('volumen').value;
let numero = parseInt(inputNumero);  //Se ejecutan siempre

let enRango = numero>3 && numero<5;
if(enRango){   //condición
    mensaje = “Entre tres y cinco”;  //Se ejecutan solo si
    print(mensaje);                  // enRango == true.
}

alert(“fin del programa”);  //Se ejecuta siempre
```

Ej. 8: Escribe un programa que pida al usuario el volumen (de 0 a 100). Si el volumen está entre 20 y 80 (ambos inclusive), el programa mostrará `Volumen adecuado`. Luego, en todo caso, mostrará `Fin de programa`.

Ej. 9: Escribe un programa que pida al usuario el volumen (de 0 a 100). Si el volumen es menor que 20 o mayor que 80, el programa mostrará `Volumen inadecuado`. Luego, en todo caso, mostrará `Fin de programa`.

Ej. 10: Escribe un programa que pida al usuario tres números. El programa imprimirá los tres números. Tras ello, imprimirá `Son iguales` si los tres números son iguales.

Ej. 11: Escribe un programa que pida al usuario tres números. El programa imprimirá los tres números. Tras ello, imprimirá `Son distintos` si los cualquiera de los tres números es distinto.

Ej. 12: Escribe un programa que pida un número al usuario y que imprima uno de los siguientes mensajes: `Es mayor que 7` o bien `Es menor que 7`, dependiendo de si el número introducido es mayor o menor que 7.

Pista: necesitarás dos sentencias `if` simples para hacerlo (esto puede hacerse de mejor forma usando condicionales más complejos que veremos más adelante).

2. Anidar condicionales

Es posible introducir unos condicionales dentro de otros:

```
let carta =  = document.getElementById('carta').value;
let palo =  = document.getElementById('palo').value;
if(palo==”corazones”){
    if(carta==”Q”){
        document.getElementById('rslt').innerHTML =
            "Es la Reina de Corazones";
    }
}
```

Observa que programa anterior puede simplificarse, usando un único condicional.

```
let carta = document.getElementById('carta').value;
let palo = document.getElementById('palo').value;
if(palo==”corazones” && carta==”Q”){
        document.getElementById('rslt').innerHTML =
            "Es la Reina de Corazones";
}
```

Existen otros casos en los que si que es necesario anidar condicionales. En el ejemplo siguiente se calculan los puntos de la carta introducida por el usuario (si es de corazones, vale un punto, y si es la reina de corazones, vale 3 puntos).

```
let carta = document.getElementById('carta').value;
let palo = document.getElementById('palo').value;
let puntos = 0;
if(palo==”corazones”){  //Solo corazones vale puntos
    puntos = 1;
    if(carta == ”Q”){   //La única reina que vale puntos
        puntos = 3;     // es la reina de corazones
    }
}
document.getElementById('rslt').innerHTML = puntos;
```

Observa que, en el programa anterior, solo se introduce en el bucle interior si la carta es de corazones y una reina, puesto que para que se evalúe la condición `carta==”Q”`, antes ha tenido que ser cierta la condición `palo == ”corazones”`.

Ej. 13: Escribe un programa en el que el usuario introduce una cadena. Si dicha cadena posee más de 10 caracteres, imprimirá el mensaje `‘Es una`

`cadena larga'`. Si posee más de 20 caracteres, imprimirá, además, otro mensaje `'Demasiado'`.

Ej. 14: Escribe un programa que haga lo mismo que el anterior, pero que, en caso de que la cadena tenga 30 caracteres, ponga otro mensaje `'Te has colado'`.

Ej. 15: Escribe un programa que haga lo mismo que el anterior, pero que muestre los mensajes en la misma línea. **Pista:** concatena los mensajes.

Ej. 16: Escribe un programa en el que el usuario introduce los dos números de una ficha de dominó. El programa calculará los puntos que vale la ficha: Cada ficha vale un punto, si es un doble, vale 2 puntos, y la ficha doble seis vale 5 puntos.

3. Estructura else if

Un condicional puede ampliarse con una cláusula `elif` (también llamada cláusula SINO).

```
let numero = document.getElementById('numero').value;

if (numero>5){   //condición
    mensaje = "Mayor que 5";     //Se ejecutan solo si numer>5
    document.getElementById('rslt').innerHTML = mensaje;

} else if(numero>=3){
    mensaje = "Entre 3 y 5";  //Solo si numero>=3 pero no >5
    document.getElementById('rslt').innerHTML = mensaje;
}
```

Una clausula else if es una concatenación de condicionales if, por lo que no debe encontrase sola. Si es necesario, también pueden acumularse varios else if:

```
let numero = document.getElementById('numero').value;

if(numero>1000){
    document.getElementById('rslt').innerHTML = "Pos. grande";
} else(if numero>0){
    document.getElementById('rslt').innerHTML = "Positivo";
} else if(numero==0){
    document.getElementById('rslt').innerHTML = "Cero";
} else if(numero<0){
    document.getElementById('rslt').innerHTML = "Negativo";
}
```

Ej. 17: Realiza el ejercicio 12 de los condicionales con condiciones complejas usando tan solo un condicional más su cláusula SINO. **Nota:** Observa que (tanto en este programa como en ej_12.html) si el número introducido por el usuario es igual a 7, el programa no imprime nada.

Ej. 18: Crea el menú de una calculadora básica en el que el usuario introducirá dos números en dos campos de texto, y también seleccionará una operación entre suma, resta, multiplicación y división.

El select tiene una sintaxis HTML como la siguiente:

```
<select id="op">
```

```
        <option value="suma">+</option>
        . . .
    </select>
```

Para saber qué opción está seleccionada, se emplea una instrucción igual que las ya empleadas:

```
let operacion = document.getElementById('op').value;
```

El programa, al pulsar un botón, realizará la operación seleccionada imprimiendo el resultado en una etiqueta.

4. Cláusula else

Cuando un condicional incluye una cláusula else, ésta se ejecuta si y solo si no se ejecuta ninguna de las cláusulas anteriores.

```
let cad = document.getElementById('cad').value;
let longitud = cad.length;

if(longitud<=1){
    document.getElementById('rslt').innerHTML = "Muy Corta";
else{
    document.getElementById('rslt').innerHTML =
        cad.substr(0,2);
}
```

La cláusula if puede combinarse con una o varias cláusulas else if:

```
numero = int( input("Número: ") )  #Se ejecuta siempre

if(numero>5){   //condición
    mensaje = "Mayor que 5"  //Se ejecutan solo si numero>5
    document.getElementById('rslt').innerHTML = mensaje;
}else if(numero>=3){
    mensaje = "Entre 3 y 5"  //Solo si numero>=3 pero no >5
    document.getElementById('rslt').innerHTML = mensaje;
}else{
    mensaje = "Menor que 3"  //Solo si ninguna otra se cumple
    document.getElementById('rslt').innerHTML = mensaje;
}
```

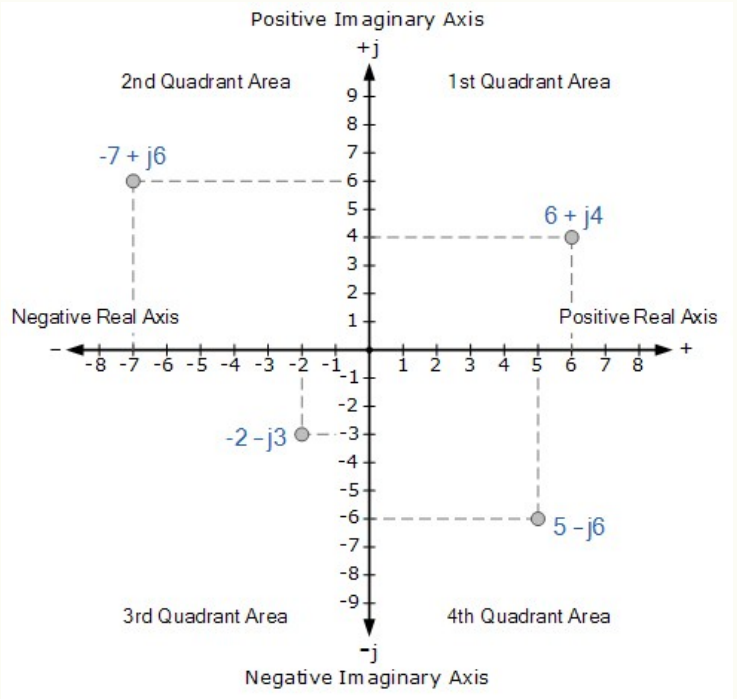
Una clausula else es siempre parte de un condicional if, por lo que no debe encontrase sola.

Ej. 19: Escribe un programa que pida al usuario un número. Si es par, el programa indicará `'Es par'`, y si es impar, el programa indicará `'Es impar'`. Utiliza un único if con cláusula else para hacerlo.

Ej. 20: Escribe un programa que indique si una cadena tiene más de 10 caracteres, con `'Tiene más de 10 caracteres'` o no `'Tiene 10 o menos caracteres'`.

Ej. 21: Escribe un programa que pida al usuario un número, y muestre si es positivo (`'Es positivo'`), negativo (`'Es negativo'`) o cero (`'Es cero'`).

Ej. 22: Escribe un programa que pida al usuario dos números, que representarán la parte real e imaginaria de un número complejo. El programa indicará el cuadrante (`'Cuadrante 1'`, etc.) o eje (`'Eje imaginario positivo'`, `'Eje real negativo'`, etc.) o bien cero (`'Cero'`) en que se sitúa el número:



5. Ejercicios adicionales

Ej. 23: ~~Escribe un programa en el que el usuario escriba un número con decimales, y el programa redondee hacia el entero más próximo a 0. **Ejemplos:** `3.7` es redondeado a `3`. `-3,7` es redondeado a `-3`.~~

~~**Pista 1:** Para redondear un número a su entero más bajo, puedes usar la división entera. Por ejemplo `3.7 // 1 = 3.0`, `-3.7 // 1 = -4`, y `-3.0 // 1 = 3.0`.~~

~~**Pista 2:** Con lo anterior, vemos que, solo con incrementar los números negativos con, podemos dar el resultado correcto. Los números que no tienen decimales (`3.0`, `-4.0`, etc.) cumplen lo siguiente: `numero == numero // 1`.~~

Ej. 24: Escribe un programa en el que el usuario introduce los dos números de una ficha de dominó. El programa calculará los puntos que vale la ficha: Cada ficha vale igual a la suma de sus dos números. Si la suma es mayor que 6, los puntos valen el doble y si además de ello (que la suma sea mayor que 6), es una ficha doble, los puntos vale cuatro veces la suma de sus números.

Ej. 25: Crea un programa en el que el usuario introduzca una palabra en singular y en minúsculas. Si la palabra la palabra termina en `a`, `d`, `ón`, `z`, `is`, `ie` o `umbre`, el probrama dirá `'probable género femenino'`. Eso si, si terminase en `l`, `o`, `n`, `e`, `r`, `s`, `ma` o `ta`, el programa dirá `'probable género masculino'`. Si no termina en ninguno de los anteriores, mostrará `'Incierto'`.

6. Ejercicios de refuerzo

Ej. 26: Escribe un programa que pida dos cadenas, de forma que la segunda estará contenida en la primera. El programa indicará la posición de la segunda cadena en la primera. Tras ello, si la posición es igual a 0, el programa añadirá un segundo mensaje diciendo `Al principio`.

Ejemplo: si las cadenas son `12345` y `123`, el programa mostrará `'0'`, y luego otro mensaje `Al principio`.

```
let cad1 = prompt("Cadena 1: ");
let cad2 = prompt("Cadena 2: ");
let posicion = cad1.indexOf(cad2);

print( posicion );
if(posicion1 == 0){
    alert("Al principio")
}
```

Ej. 27: Escribe un programa que pida dos cadenas, de forma que la segunda estará contenida en la primera. El programa indicará la posición de la segunda cadena en la primera. Tras ello, si la cadena está al final, el programa añadirá un segundo mensaje diciendo `Al final`.

Ejemplo: si las cadenas son 123457 y 4567, el programa mostrará `2`, y luego otro mensaje `Al final`.

Pista: Suma de la posición obtenida con `index()` a la longitud de la segunda cadena, y comparala con la longitud de la primera.

Esta vez, debemos de mostrar si la cadena está al final. Sin embargo, en el ejercicio anterior, la variable posición almacena la posición del primer carácter de cad2 en cad1 (por ejemplo, con cad1: `abcd` y cad2: `bc`, la variable `posicion` almacenaba el valor `1` (que es la posición del carácter `b` en cad1). Para averiguar la posición del último carácter (en este ejemplo, la posición de `c` en cad1), debemos sumar la longitud de cad2. Tras ello, comparamos posición con la longitud de cad1.

```
let cad1 = prompt("Cadena 1: ");
let cad2 = prompt("Cadena 2: ");

longitud1 = cad1.length;
```

```
longitud2 = cad2.length;
let posicion = cad1.indexOf(cad2) + longitud;

print( cad1.indexOf(cad2) );

if(posicion == longitud1){
    alert("Al final")
}
```

Ej. 28: Combina los dos ejercicios anteriores en otro programa que te muestre la posición y luego te muestre, si es el caso, `Al final` o `Al final`.

Nota: observa que, si la segunda cadena es igual a la primera, se deberán mostrar los dos mensajes: ej: si las cadenas son `12345` y `12345`, se mostrarán ambos mensajes.

Simplemente debemos combinar los dos condicionales anteriores (cambiando levemente algunos nombres de variable). Puesto que ambas condiciones son independientes (una cadena puede estar al principio, al final, en ambas, o en ninguna), se tratará de condicionales separados:

```
let cad1 = prompt("Cadena 1: ");
let cad2 = prompt("Cadena 2: ");

let longitud1 = cad1.length
let longitud2 = cad2.length
let posicion1 = cad1.indexOf(cad2);
let posicion2 = cad1.indexOf(cad2) + longitud2;

print( posicion1 );
if(posicion1 == 0){
    alert("Al principio");
}
if(posicion2 == longitud1){
    alert("Al final");
}
```

Ej. 29: Escribe un programa en el que el usuario introduzca una cadena. Si la cadena tiene un número de caracteres par, el programa devolverá la misma cadena, pero en mayúsculas. Si el número de caracteres es impar, devolverá la cadena en minúsculas.

Ejemplo: Si se introduce la cadena `Hola007`, al tener 7 caracteres (impar), devolverá `hola007`.

Nota: para pasar a mayúsculas, usa «cad».toUpperCase(), y para las minúsculas, «cad».toLowerCase().

En este caso, según una condición (que la longitud sea par o no), deberemos hacer una cosa u otra. Es decir, son dos casos excluyentes, y siempre se va a dar uno u otro. Por tanto es un caso de if con un else.

```
let cad = tnput("Cadena: ");
let longitud = cad.length;
if(longitud%2 == 0){  //longitud par
    alert(cad.toUpperCase());
}else{  //longitud impar
    alert(cad.toLowerCase());
}
```

Ej. 30: Escribe un programa en el que el usuario introduzca una cantidad entera de euros, y luego indique si es socio (`si`) o no (`no`). Si la cantidad de euros es mayor o igual que 1000, se le aplicará un 5% si no es socio, o 7% si es socio. El programa mostrará la cantidad final.

Ejemplos: el usuario introduce `1000` y `no`, entonces el programa sacará `950.0 €`. Si el usuario introduce `400` y `si`, el programa sacará `400.0 €`.

```
let euros = parseInt( prompt("Euros: ") );
let socio = parseInt( prompt("Socio: ") );

if(euros>=1000) {
    if(socio == "no"){
        euros = euros * 0.95;  #Descuento del 5%
    }else{
        euros = euros * 0.93;  #Descuento del 7%
    }
}
alert(euros + "€" );
```

Ej. 31: Copia y modifica el programa anterior para que haga lo mismo, con la única diferencia de que, si la cantidad es negativa, muestre un error (`Error, cantidad negativa`). En este caso, el usuario ni siquiera tendrá la oportunidad de decir si es socio o no.

Este ejercicio es igual que el anterior, tan solo debemos rodear el programa entero, a excepción de la instrucción input relativa a euros, con un if. Es decir, si la variable euros es positiva, haz todo igual que antes, sino imprime el mensaje de

error. De nuevo, tenemos un caso en el que hacemos una o la otra (nunca las dos o ninguna), lo que se traduce en un if con su else.

```
let euros = parseInt( prompt("Euros: ") );
let socio = parseInt( prompt("Socio: ") );

if(euros>=0){
    if(euros>=1000) {
        if(socio == "no"){
            euros = euros * 0.95;  #Descuento del 5%
        }else{
            euros = euros * 0.93;  #Descuento del 7%
        }
    }
}else{
    alert("Error, cantidad negativa");
}

alert(euros + "€" );
```

Ej. 32: Escribe un programa que pida una cantidad en watios (W), y calcule su equivalente en caballos de fuerza (CV). Sin embargo, si la cantidad es negativa, deberá mostrar, en su lugar, el error `'El número de watios debe ser positivo'`.

Nota: Recuerda usar `float()` en vez de `int()`.

Una vez más, dos condiciones excluyentes, en la que se dará una o la otra (pero no las dos o ninguna), por lo que usamos un if con su else.

```
let watios = parseFloat( prompt("Watios: ") );
if(watios>=0){
    let cv = watios * 0.00134102;  //Tb: cv = watios/745.7
    alert(cv + " W")
}else{  //Variable watiox es negativo → Error
    alert("El número de watios debe ser positivo");
}
```

Ej. 33: Escribe un programa en el que el usuario escriba una cadena y éste diga si la cadena está entrecomillada con comillas simples (mostrará `'Entrecomillada'`) o no lo está (`'No entrecomillada'`).

Pista: una cadena entrecomillada cumplirá que su primer carácter sea igual a `'` y también que su último carácter sea igual a `'` (utiiza subcadenas para

extraer el primer o el último carácter de la cadena, y usa el comparador `==` para compararlas con la cadena `'`).

Este caso es como los anteriores, donde tenemos dos casos excluyentes y siempre se da uno y el otro. La condición para que se ejecute una cosa (`'Entrecomillada'`) o la otra (`'No entrecomillada'`) es, en términos coloquiales, que *"El primer carácter sea igual a* `'` **Y** *que el último carácter sea igual a* `'`.

```
let cad = input("Cadena: ")
let primer_char = cad[0:1] //Coge solo la posicion 0
let ultimo_char = cad[-1:] //Coge solo la posicion -1

if(primer_char=="'" and ultimo_char=="'"){
    alert("Entrecomillada");
}else{
    alert("No entrecomillada");
}
```

Ej. 34: Escribe otro programa que indique si una cadena está debidamente encerrada por signos de exclamación o interrogación, en cuyo caso dirá `'Si'`) o no lo está (`'No'`).

Nota: si, por ejemplo, la cadena empezara por `¿` y terminase en `!`, o si empieza por `?`, la respuesta debe ser `'No'`.

Este ejercicio es exacto al anterior, solo que la condición es algo más complejo: *"Que el primer carácter sea igual a* `¡` **Y** *que el último carácter sea igual a* `!` **O BIEN** *que el primer carácter sea igual a* `¿` **Y** *que el último carácter sea igual a* `?`.

```
let cad = prompt("Cadena: ")
let primer_char = cad.substr(0,1);  //Coge solo la posicion 0
let ultimo_char = cad.substr(-1,1); //Coge solo la posicion -1

if( (primer_char=="¡" and ultimo_char=="!") ||
    (primer_char=="¿" and ultimo_char=="?") ){
    alert("Si");
}else{
    alert("No");
}
```

Nota: el carácter `\` en la línea del if se usa para indicar que la siguiente línea (la que empieza por `primer_char="¿"...` forma parte de la misma instrucción.

También es posible poner ambas líneas (desde el if hasta los dos puntos) todo en una sola línea.

Ej. 35: Escribe un programa en el que el usuario introduzca una frase. El programa indicará si la frase es `'Exclamativa'` (estará encerrada entre signos de exclamación) o `'Interrogativa'`. Mostrará el mensaje `'Otra'`, si no está en los casos anteriores (incluido si la frase tiene errores como empezar sin signos y terminar con un `!`, o empiezar con signos de cierre `?`, etcétera.

Pista: Controla los 2 primeros casos con sentencias if/elif, y la última (la del error) con un else.

En este caso, tenemos 3 casos excluyentes, con lo que deberemos realizar un if con los correspondientes elif. Además, como también se debe ejecutar siempre uno de los casos (o es exclamativa, o interrogativa u otra), en el último caso se usa un else.

```
let cad = prompt("Cadena: ");
let primer_char = cad[0:1] //Coge solo la posicion 0
let ultimo_char = cad[-1:] //Coge solo la posicion -1

if (primer_char=="¡" and ultimo_char=="!"){
    alert("Exclamativa");
}else if (primer_char=="¿" and ultimo_char=="?"){
    alert("Interrogativa");
}else:
    print("Otra");
}
```

Ej. 36: Una tienda vende, exclusivamente, ramos de flores, las cuales valen, por defecto, 5€. Si el ramo posee una banda de felicitación, el precio sube 2 €. Si dicha banda está en papel dorado, incrementa el precio en 1 € adicional

Escribe un programa en el que el se le pregunte al dependiente si el ramo posee una banda y en el que el dependiente introducirá `Si` o `No`. En caso de que se introduzca `Si`, el programa preguntará si es dorada o no, volviendo el dependiente a indicar `Si` o `No`. El programa calculará y mostrará el precio final del ramo.

Puesto que nos dan un valor por defecto, lo más fácil es definir una variable que, inicialmente, almacene ese valor por defecto, y modificar su valor después.

```
let banda = prompt("¿Banda?: ")
let precio = 5;
```

```
if(banda=="Sí"){
    precio = precio + 2;
    let dorada = prompt("¿Dorada?: ");
    if(dorada=="Sí"){
        precio = precio + 1;
    }
}
alert(precio + " €");
```

Unidad Didáctica 4:

Listas

1. Listas

Una lista es, simplemente, un conjunto de valores, uno detrás de otro. Por ejemplo:

```
const frutas = ["pera", "manzana", "mandarina"];
document.getElementById('r').innerHTML = fruits.length;  //3
```

En el ejemplo anterior, todos los elementos de la lista eran cadenas, pero, realmente, una lista puede mezclar variables de todo tipo (cadenas, booleanos, números de todo tipo, etcétera) en la forma que se desee:

```
const lista = ["pera", "3", "manzana"];
document.getElementById('r').innerHTML = lista;   // pera, 3, manzana
```

También es posible utilizar el valor de variables previas para definir el valor de las listas. En el siguiente ejemplo se usan dos variables y un literal para definir una lista:

```
let mensaje = "solución";
let valor = 3;
const lista = [mensaje, valor, "entero"];
document.getElementById('r').innerHTML = lista;   // solución, 3, entero
```

Ej. 1: Escribe un programa que pida 3 datos. El programa creará una lista con esos tres datos. Luego, imprimirá esa lista.

Ejemplo: Si el usuario introduce `hola`, `-3` y `false`, el programa creará e imprimirá la lista `['hola', -3, false]`.

Ej. 2: Escribe un programa que pida 3 datos. El programa creará una lista con esos tres datos, pero en orden inverso. Luego, imprimirá esa lista.

Ejemplo: Si el usuario introduce `1.1`, `false` y `-3`, el programa creará e imprimirá la lista `[-3, false, 1.1]`.

Tamaño de una lista

Para saber el número de elementos que tiene una lista, se utiliza la función length, de forma idéntica a como se hacía con las cadenas:

«lista».length Devuelve la longitud de 'lista'.

```
const lista = ["hola", "que", "tal"];
document.getElementById('r').innerHTML = lista.length;  //3
```

Añadir un elemento al final

«lista».push(«nuevoElem») Añade «nuevoElem» al final de la lista.

```
const lista = ["hola", "que", "tal"];
lista.push("¿cómo estás");
document.getElementById('r').innerHTML = lista;
                    //["hola", "que", "tal", "¿cómo estás" ];
```

Concatenar listas

Utilizando el operador suma, podemos concatenar listas, de forma análoga a como lo hacíamos con las cadenas:

```
const arr1 = ["Celia", "Juan"]
const arr2 = ["Sofía", "Tomás", "Lola"];
const nombres = arr1.concat(arr2);
document.getElementById('r').innerHTML = nombres;
                //Imprime:  ["Celi"a, "Juan", "Sofía", "Tomás", "Lola"]
```

Ej. 3: Escribe un programa que pida 2 datos, y en el que el programa creará una lista con esos dos datos. Luego le concatenará un elemento con el valor `0.0` al inicio de la lista, y un elemento de tipo cadena y de valor `'final'` al final de la lista. Luego, imprimirá esa lista.

Ejemplo: Si el usuario introduce `3` y `True`, el programa creará e imprimirá la lista `[0.0, 3, True, 'final']`.

Sublistas

Podemos quedarnos con solo parte de los valores de una lista, utilizando los corchetes, de la misma forma que hacíamos con las cadenas.

«arr1».sllce(Inicio) Devuelve una lista desde el elemento indicado, incluyendo éste.

«arr1».sllce(Inicio,fin) Devuelve una lista desde el elemento indicado, incluyendo éste, hasta el segundo elemento indicado, excluyendo éste.

Los valores de inicio y fin pueden indicarse con números negativos, de forma similar a como ocurre con las cadenas de caracteres, representando el -1 al último elemento, el -2 al penúltimo, y así sucesivamente.

```
const lista = [];
lista.push("haba", "pera", 3.0, true);
document.getElementById('r1').innerHTML = lista.slice(1);
                        //Imprime:  [pera, 3.0, true]
document.getElementById('r2').innerHTML = lista.slice(1,3);
                        //Imprime:  [pera, 3.0]
document.getElementById('r3').innerHTML = lista.slice(2,20);
                        //Imprime:  [3.0, true]
```

Ej. 4: Escribe un programa que pida 5 números, y cree una lista con dichos números. El programa mostrará las siguientes listas, una en cada línea (debes usar valores negativos cuando necesites determinar las posiciones última, penúltima o antepenúltima):

- Los dos primeros elementos

- Los dos últimos elementos

- El último y el penúltimo.

Ej. 5: Escribe un programa que pida 5 números, y cree una lista con dichos números. El programa devolverá una lista con los 2 primeros y luego con los dos últimos. Emplea slice para hacer las dos sublistas y concat para encadenarlas.

Modificar elementos

Podemos modificar elementos de la lista:

```
const lista = ["haba", 3.0, "cereza"];
lista[2] = 5;                                   //Modifica posición 2
```

```
document.getElementById('r').innerHTML = lista;  //Imprime:  haba, 3.0, 5
```

Ej. 6: Escribe un programa que pida 5 datos y que cree una lista con ellos. Luego, modificará el 2º y el 3º, subtituyéndolos con los valores -1 y -2. Luego, imprimirá la lista.

Borrar elementos

Puedes borrar elementos de una lista de las siguientes formas:

«lista».pop() Borra el último elemento de lista y lo devuelve.

```
Lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
let x = lista.pop();
document.getElementById('r1').innerHTML = lista
                  //Imprime:  0,1,2,3,4,5,6,7,8
document.getElementById('r2').innerHTML = x;  //Imprime:  9
```

«lista».splice(«indice», «numElem») Borra, a partir de «índice», un número de elementos seguidos iguales a «numElem».

```
const array = ["a", "b", "c", "d", "e", "f"];
array.splice(1, 2);
document.getElementById('r').innerHTML = array;   //Imprime: a,d,e,f
```

Ej. 7: Escribe un programa que pida 3 datos y cree una lista con ellos. Pedirá también un cuarto valor, que será entre 0 y 2. El programa borrará ese elemento de lista. Finalmente, imprimirá la lista.

Ejemplo: si el usuario introduce `a`, `b`, `c` y `1`, el programa imprimirá `[a, c]`.

Elementos en una lista

Puedes comprobar la posición de un elemento en la una lista utilizando indexOf, de forma similar a las cadenas:

«lista».indexOf(«elemento») Devuelve la posición de «elemento» en la «lista». Si el elemento no está en la lista, se devolverá -1.

```
const array = ["a", "b", "c", "d", "e", "f"];
document.getElementById('r').innerHTML = array.indexOf("c"); //2
document.getElementById('r').innerHTML = array.indexOf("x"); //-1
```

Ej. 8: Escribe un programa que pida 3 elementos al usuario, creando una lista de 3 elementos. Luego pedirá un cuarto número. Si dicho cuarto número está en la lista, imprimirá el mensaje `'Está en la lista'`, o en caso contrario, imprimirá `'No está en la lista'`.

Ej. 9: Escribe un programa en el que el usuario introduzca un carácter (por ejemplo `a`, `9` o `#`). El programa indicará si el carácter introducido es un dígito, mostrando `'Es un dígito'` o `'No es un dígito'`.

Pista: Crea una lista de la forma: [“0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”]. Luego, busca el carácter introducido en la lista.

Ej. 10: Escribe un programa igual al anterior, pero que imprima si si el carácter es una letra mayúscula (`'Letra mayúscula'`), letra minúscula (`'Letra minúscula'`), un dígito (`'Dígito'`) u otro carácter (`'Otro'`).

Pista: tendrás que diseñar 3 listas para comparar el carácter con cada lista. Si no es ninguna de las anteriores, automáticamente será del tipo `'Otro'`. Puedes usar una estrctura “if – else if – else if – else”.

2. Ejercicios adicionales

Ej. 11: Crea un programa en el que el usuario introduzca una palabra en singular y en minúsculas. Si la palabra termina en `l`, `o`, `n`, `e`, `r`, `s`, `ma` o `ta`, el programa dirá `'probable género masculino'`. En cambio, si la palabra termina en `a`, `d`, `ón`, `z`, `is`, `ie` o `umbre`, el probrama dirá `'probable género femenino'`. Si no termina en ninguno de los anteriores, mostrará `'Incierto'`.

Realiza el ejercicio usando dos listas, una en la que almacenarás los valores del probable género masculino, y otra para los valores del probable género femenino.

Ej. 12: Escribe un programa que pida al usuario 3 cadenas. El programa creará y mostrará una lista con solo los elementos no vacíos. Haz el programa de dos formas, usando push y sin usar push.

Ejemplo: Si el usuario introduce `hola`, una cadena vacía y `¿qué tal?`, entonces el resultado será `['hola', '¿qué tal?']`.

Pista: para hacer el ejercicio sin push, empieza definiendo una variable llamada `lista_final`, que contenga una lista vacía (`[]`). Luego, tras el primer `input()`, comprueba si la cadena introducida es distinta de la cadena vacía (`""`), en cuyo caso, crea una lista con dicha cadena y concaténala a `lista_final`. Luego repite el proceso 3 veces. Finalmente, imprime `lista_final`.

Unidad Didáctica 5:

Bucles

1. Bucles while

Bucle while básico

Los bucles `while` repiten una serie de instrucciones (el cuerpo del bucle) mientras se cumpla una condición, la cual es definida tras la palabra reservada `while`. En el siguiente programa se usa un bucle para imprimir dos números, empezando por el 2:

```
let i = 2;  //Indices usan nombres como: i,j,k
let rs = "";
while (i < 4){  //Al llegar a 5, no ejecuta más el bucle
    rs = rs + i +",";  //Añade i al resultado
    i = i + 1;  //Incrementa i en 1
}
rs = rs +"FIN";
document.getElementById('rstl').innerHTML = resultado; //Imprime:  2,3,FIN
```

A continuación se muestran todos los pasos del programa anterior. En el paso ❶ se define `i`, inicialmente con el valor de `2`. Tras ello, en el paso ❷, se comprueba si `i` es menor que cuatro. Como ello es así, el cuerpo del bucle (las instrucciones de ❸ `print` y ❹ `i=i+1`), se ejecutan, imprimiendo en pantalla un `2` e incrementando la variable `i` en uno.

❶ i:?→ 2

```
let i = 2;
let rs = "";

while (i<4){
  rs = rs + i+",";
  i = i + 1;
}
rs = rs +"FIN";
```

❷ i: 2

```
let i = 2;
let rs = "";

while (i<4){ //2<4
  rs = rs + i+",";
  i = i + 1;
}
rs = rs +"FIN";
```

❸ i:2 `2,`

```
let i = 2;
let rs = "";

while (i<4){
  rs = rs + i+",";
  i = i + 1;
}
rs = rs +"FIN";
```

❹ i:2→ 3 `2,`

```
let i = 2;
let rs = "";

while (i<4){
  rs = rs + i+",";
  i = i + 1;
```

❺ i:3 `2,`

```
let i = 2;
let rs = "";

while (i<4){ //3<4
  rs = rs + i+",";
  i = i + 1;
```

❻ i:3 `2,3,`

```
let i = 2;
let rs = "";

while (i<4){
  rs = rs + i+",";
  i = i + 1;
```

```
}
rs = rs +"FIN";
```

```
}
rs = rs +"FIN";
```

```
}
rs = rs +"FIN";
```

❼ i:3→ 4 2,3,

```
let i = 2;
let rs = "";

while (i<4){
  rs = rs + i+",";
  i = i + 1;
}
rs = rs +"FIN";
```

❽ i: 4 2,3,

```
let i = 2;
let rs = "";

while (i<4){ //4≮4
  rs = rs + i+",";
  i = i + 1;
}
rs = rs +"FIN";
```

❾ i:3 2,3,FIN

```
let i = 2;
let rs = "";

while (i<4){
  rs = rs + i+",";
  i = i + 1;
}
rs = rs + "FIN";
```

Cuando el cuerpo del bucle se ha ejecutado, volvemos a comprobar la condición (paso ❺), aunque esta vez la variable `i` ya vale `3`. Como la condición sigue siendo cierta, volvemos a ejecutar el cuerpo del bucle (los pasos ❻ y ❼), imprimiendo en pantalla un 3 e incrementando, de nuevo,la variable contador en uno, quedando con un valor de `4`.

Finalmente, volvemos a comprobar la condición del bucle (paso ❽). Como `i` ya no es menor que `4`, ya no ejecutamos más el cuerpo del bucle, siguiendo adelante con el paso ❾.

`Ej. 1:` Escribe un programa que pregunte al usuario un número. El programa imprimirá una cadena con tantas a's indicó el usuario.

Ejemplo: si el usuario introduce un `3`, el programa devolverá `aaa`.

`Ej. 2:` Escribe un programa que pregunte al usuario un número. El programa creará y luego imprimirá una cadena que empezará por 0 y terminará en el número indicado.

Ejemplo: si el usuario introduce un `3`, el programa devolverá `'0123'`.

Contador inverso

En ocasiones es útil o interesante realizar el programa con un contador que se va decrementando:

```
let lanzamientos = 10; // Número total de lanzamientos
let caras = 0; // Contador de cuántas veces sale cara
```

```
while (lanzamientos > 0) {
  // Si Math.random() < 0.5 será cara, y >= 0.5 cruz
  caras += Math.random() < 0.5; // Suma 1 si es cara
  lanzamientos--; // Queda un lanzamiento menos.
}

document.getElementById('r').innerHTML = "Nº Caras: " + caras);
```

Ej. 3: Escribe un programa que pregunte al usuario un número. El programa creará y luego imprimirá una cadena que empezará por el número y terminará en `0`.

Ejemplo: si el usuario introduce un `3`, el programa devolverá `'3210'`.

Ej. 4: Implementa un programa que sume, usando un bucle, los números 1 a n, siendo n un número especificado por el usuario.

Por ejemplo, si el usuario introduce `3`, el programa mostrará `6` (1 + 2 + 3);

Ej. 5: Implementa un programa que calcule el número de Fibonacci especificado por el usuario, que será mayor que 1. El número de Fibonnacci se calcula:

Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2);

Pista: Tendrás que usar 2 variables que almacenen Fibonacci(n-1) y Fibonacci(n), que irás actualizando en cada iteración del bucle.

Ej. 6: Calcula el factorial de un número proporcionado por el usuario. El de un número n es igual a 1 * 2 * 3 * … * n.

Ej. 7: El usuario introducirá 2 valores que estarán ambos entre 0 y 1. El programa generará números aleatorios hasta obtener un valor entre el intervalo descrito por el usuario. Escribirá los números generados separados por comas.

Nota: Math.random() genera valores aleatorios entre 0 y 1.

Ejemplo: si el usuario introduce `0.2` y `0.3`, el programa generará números aleatorios hasta que uno de ellos esté entre `0.2` y `0.3`.

Iterar cadenas

Aunque es más típico hacerlo con un bucle `for` (que veremos más adelante), un bucle `while` puede usarse para iterar sobre una cadena.

```
let cadena = "01234567";  //Cadena de longitud 8
let i = 0;  //variables índices suelen usar nombres como i,j,k
let longitud = cadena.length;
let resultado = "";
while(i<longitud){
    let c = cadena[i]  //c almacena el carácter i-ésimo
    resultado = resultado + c + ",";
    i = i + 1;
}
document.getElementById('rstl').innerHTML = resultado;
                      //Imprime:  1,2,3,4,5,6,7,
```

En el ejemplo anterior, la variable contador `i` se irá incrementando desde `0` hasta tomar un valor igual a la variable longitud (en este caso, `8`). Mientras `i` sea menor que `8`, el cuerpo del bucle se repetirá, pero, una vez que `i` valga `8`, la condición `i<longitud` ya no será cierta, por lo que se terminará el bucle.

Observa que, para obtener el carácter que haya en la posición i-ésima de la cadena, usamos la expresión `c = cadena[i]`, de forma que, si por ejemplo, `i` vale `0`, entonces a `c` se le asigna el valor `"0"`.

Nota: `c` es de tipo de cadena (aunque almacene un cero, dicho cero es un carácter), mientras que `i` es de tipo entero.

Ej. 8: Crea un programa que tome una cadena de texto ingresada por el usuario, la invierta y la muestre. Se recomienda usar un contador que vaya decrementando.

Ej. 9: Crea un programa que tome una cadena de texto ingresada por el usuario, y muestre otra con cada letra en mayúsculas y minúsculas.

Por ejemplo, si el usuario inserta `abCd`, el programa devolverá `aAbBcCdD`.

Para poner un carácter o cadena en mayúsculas o minúsculas, puedes usar «cad».toUperCase() y «cad».toLowerCase().

Ej.9b: Crea un programa que tome una cadena de texto ingresada por el usuario y cuente cuántas veces aparece cada vocal (a, e, i, o, u). Muestra el resultado indicando cuántas veces aparece cada vocal.

Nota: no uses listas, para comprobar si la letra actual es una vocasl, usa un condicional con una condición de Si letra es igual a “a” o letra es igual a “b” …

Ej.9c: Crea un programa que tome una cadena de texto ingresada por el usuario y genere otra cadena en la que cada carácter sea reemplazado por el siguiente en el alfabeto. Si el carácter es "z" o "Z", deberá reemplazarse por "a" o "A", respectivamente.

Otras condiciones de salida

En el siguiente ejemplo realizamos una condición de salida algo más compleja para evitar seguir recorriendo la cadena cuando hayamos encontrado la letra que buscamos.

```
let cad = "abcÑdeÑf";  //adena de longitud 8
let i = 0;  //Indices usan nombres como: i,j,k
let c = "";
let resultado;
while (c!="Ñ" && i<cad.length){
    c = cad[i]
    i = i + 1;
}
i = i - 1;  //En el bucle, después de poner c a 'Ñ', incrementamos i en 1,
            // y aquí deshacemos ese último incremento.
if(c=="Ñ") resultado = i;
else resultado = "no encontrado";
document.getElementById('rstl').innerHTML = resultado;
//Imprime la posición de la primera "Ñ" en cad: 3
```

Ej. 10: Escribe un programa que, con un bucle while, vaya calculando la suma de los números de 1 hasta "n". Cuando dicha suma sea justo mayor que un número proporcionado por el usuario, el programa devolverá el "n" que lo cumple.

Condicional dentro del bucle

El anterior ejemplo puede realizarse con un condicional dentro del bucle, algo bastante habitual en los bucles. Aquí usamos una variable boolena para salir una vez hayamos encontrado la `Ñ`:

```
let cad = "abcÑdeÑf";  //adena de longitud 8
let i = 0;  //Indices usan nombres como: i,j,k
let encontrado = false;
let resultado = "no encontrado";

while (!encontrado && i<cad.length){
    if(cad[i] == "Ñ"){
        encontrado = true;
        resultado = i;
    }
    i = i + 1;
}

document.getElementById('rstl').innerHTML = resultado;
      //Imprime la posición de la primera "Ñ" en cad: 3
```

En el código anterior, definimos una variable contador, llamada `i`, que valdrá inicialmente `0`. Luego, en cada iteración del bucle, examinaremos el carácter i-ésimo de la cadena llamada `cad`. Por ejemplo, en la primera iteración del bucle `i` vale `0`, y lo primero que hacemos es comprobar `cad[i] == "Ñ"`, es decir, si el primer carácter de la `cad` es igual a una `Ñ`. Como no es así (el primer carácter vale `a`), nos saltamos el cuerpo del condicional, y seguimos con la siguiente instrucción: `i= i + 1`, por lo que ahora `i` vale ahora `1`, y volvemos a la cabecera del bucle (la línea que empieza por `while`). Como la condición aún se cumple, volvemos a ejecutar el cuerpo del bucle, que tampoco entrará dentro del condicional, pero que si incrementará `i`.

Tras un par de iteraciones más, cuando `i` valga `3`, volvemos a ejecutar el bucle. En esta ocasión, `cad[i] == Ñ`, si que será cierto, por lo esta vez si que entramos

en el bucle, y la variable `encontrado` pasará a valer `True`. También imprimiremos el valor de `i`, que es la posición en la que hemos encontrado la `Ñ`.

Finalmente, salimos del condicional, ejecutando la instrucción `i = i + 1`, para volver, una vez más, a la cabecera del bucle. Pero, en esta ocasiónz, `! encontrado` es `false`, por lo que la condición ya no se cumple más, y hemos terminado con el bucle. Seguimos, pues, con las instrucciones que siguen al bucle, en este caso, la que imprime el fin de programa. Observa que el cuerpo del bucle no se ha ejecutado con `i` valiendo `8`.

Ej. 11: El usuario escribirá una cadena en un campo de texto y, en otro campo de texto, un carácter. El programa calculará cuantas veces aparece el carácter en la cadena.

Ej. 12: Escribe un programa que muestre la posición de todos los caracteres `Ñ` (no solo el primero). Utiliza el código del ejemplo de base.

Pista: No es necesario aquí usar la variable `encontrado`. El condicional irá añadiendo, a un mensaje, los índices donde encuentre las `Ñ`.

Ej. 13: Escribe un programa en que el usuario introduzca una cadena en un campo de texto, y un carácter en otro campo de texto. El programa contará cuantas veces aparece el carácter en la cadena.

Ej. 14: Comprueba si un número proporcionado por el usuario es primo. Un número “n” es primo solo si NO es divisible por todos y cada uno de los números entre 2 y n-1.

Pista: el número n NO es divisible entre x si el resto de dividir n entre x no es igual a 0).

Ej. 15: Crea un programa que vaya recorriendo, con un blucle, cada uno de los caracteres de una cadena proporcionada al usuario.

El bucle terminará justo cuando encuentre una `Ñ`. Mostrará la posición en la que se ha quedado.

Nota: este programa NO usa `indexOf`.

Ej. 16: Crea un programa parecido al anterior, pero el bucle parará cuando encuentre una `Ñ` o una `ñ`.

Nota: este programa NO usa `indexOf`.

Ej. 17: Crea un programa parecido al anterior, pero que encuentre el primer dígito en una cadena. En este caso, si será necesario el uso de `indexOf`.

Ej. 18: Crea un programa que vaya mostrando, en sucesivas líneas de un textarea, los últimos caracteres de una cadena proporcionada por el usuario.

Por ejemplo, si la cadena proporcionada por el usuario es `abcd`, el resultado será:

`abcd`
`bcd`
`cd`
`d`

Nota: Deberás usar substr dentro del bucle para generar las sucesivas cadenas.

Nota: para realizar este ejercicio, necesitarás usar un textarea, en vez de una etiqueta, por ejemplo:

```
<textarea rows="10" cols="50"> </textarea>
```

Para insertar un salto de línea en el textarea, se añade `\n`.

Ej. 19: Crea un programa similar al anterior, pero que vaya mostrando los primeros caracteres, en vez de los últimos.

Ej. 20: Crea un programa similar a los anteriores, pero que vaya mostrando los caracteres centrales. La cadena introducida por el usuario tendrá un número par de caracteres.

Por ejemplo, si la cadena proporcionada por el usuario es `abcd`, el resultado será:

`abcdef`
`bcde`
`cd`

Nota: El bucle tendrá una variable que irá desde 0 hasta la mitad de la longitud (o el redondeo de la mitad hacia arriba, según el caso).

2. Bucles for

Los bucles for iteran una lista. Definen una variable auxiliar (comúnmente llamada `e`), la cual va valiendo, en cada iteración del bucle, uno (y solo uno) de los índices de la lista. Pore ejemplo, en el siguiente programa, definimos una lista de enteros. Luego, definimos un bucle for, el cual define una variable e, que va valiendo, en cada iteración, los índices de la lista (0, 1, y finalmente, 2):

```
const ls = [0, 2, 7];       //lista de enteros
let rs = "";

for(const i in ls) {        //variable 'e' controlada por el for
      rs += ls[i] + ",";    //Imprime un elemento de la lista
}
rs += "FIN";
```

A continuación se muestran los pasos del programa. En primer lugar, se necesita una lista, en este caso de enteros (paso ❶). Luego, en el paso ❷, se evalúa el for, asignando a la variable `i` el primer valor de la lista `ls` (un `0`). Luego, en el paso ❸, se imprime ese valor.

❶ e:?

```
const ls=[0,2,7];
let rs = "";
for(const i in ls){
  rs += ls[i]+",";
}
rs += "FIN";
```

❷ e:0

```
cons nums=[0,2,7];
let rs = "";
for(const i in ls){
  rs += ls[i]+",";
}
rs += "FIN";
```

❸ e:0 `0,`

```
const nums=[0,2,7];
let rs = "";
for(const i in ls){
  rs += ls[i]+",";
}
rs += "FIN";
```

❹ e:2 `0,`

```
const nums=[0,2,7];
let rs = "";
for(const i in ls){
  rs += ls[i]+",";
}
rs += "FIN";
```

❺ e:2 `0,2,`

```
const nums=[0,2,7];
let rs = "";
for(const i in ls){
  rs += ls[i]+",";
}
rs += "FIN";
```

❻ e:7 `0,2,`

```
const nums=[0,2,7];
let rs = "";
for(const i in ls){
  rs += ls[i]+",";
}
rs += "FIN";
```

❼ e:7 `0,2,7,`

```
const nums=[0,2,7];
let rs = "";
for(const i in ls){
  rs += ls[i]+",";
```

❽ e:? `0,2,7,FIN`

```
const nums=[0,2,7];
let rs = "";
for(const i in ls){
  rs += ls[i]+",";
```

```
}
rs += "FIN";
```

```
}
rs += "FIN";
```

De igual forma se realizan los pasos ❹ y ❺, pero esta vez `e` valdrá `2`. Y una última vez, los pasos ❻ y ❼, con `i` valiendo `7`. Tras ello, el programa, que ha iterado el bucle 3 veces (las mismas veces que elementos tenía la lista `ls`), pasa a ejecutar las siguientes instrucciones, en esta caso ❽.

Cuidado: En un bucle for, no debe modificarse la lista sobre la que se está iterando en el cuerpo del bucle (en el caso anterior, no debe modificarse la lista `ls` dentro del cuerpo del bucle). Tampoco es necesaria, a priori, ninguna variable contador.

En el siguiente ejemplo, la variable de bucle `e` va tomando el valor de cada uno de los valores de lista `nums` en cada una de las iteraciones del bucle.

```
let nums = [3, 4, -1, 2, -3];
let num_neg = 0;
for(const i in nums){      //variable 'e' controlada por el for
    if(nums[i]<0){                    //Si e es menor que 0
        num_neg ++;        //Incrementa num_neg
    }
}
document.getElementById('rstl').innerHTML = num_neg;   //2
```

En cada iteración del bucle, se comprueba, mediante un condicional, si el valor de e es negativo, En caso de serlo, se incrementa la variable num_neg en uno. De esta forma, al finalizar el bucle, obtenemos la cantidad de números negativos en la lista.

Ej. 21: Escribe un programa que pida al usuario tres cadenas, y cree una lista de 3 elementos con ellas. El programa informará si al menos una de ellas es igual a la cadena vacía (`""`).

Ej. 22: Escribe un programa que pida al usuario 5 números, y cree una lista de 5 elementos con ellos. El programa imprimirá la cantidad de números positivos introducidos, la cantidad de ceros, y la cantidad de números negativos.

Bucles for y cadenas

Recuerda que una cadena es, realmente, una lista de caracteres (realmente, es una lista de cadenas de longitud 1). Por tanto, podemos usar el bucle for para iterar sobre una cadena. Por ejemplo, el siguiente programa:

```
let cadena = "027";  //Cadena

for (i in cadena){   //variable 'c' controlada por el for
    rs += cadena[i] + ",";  //Imprime un elemento de la lista
}
rs += "FIN";
```

En este caso, el bucle for comprueba, en cada iteración del bucle, si el carácter es igual a una a. En tal caso, aumentará la variable `num_de_a`.

```
Let cadena = "hasiudasuiasd";
let num_de_a = 0;
for (i in cadena){
    if(cadena[i] == "a"){
        num_de_a = num_de_a + 1;
    }
}
document.getElementById('rstl').innerHTML = num_de_a;  //3
```

Las comprobaciones del carácter pueden ser más complejas. Por ejemplo, en el siguiente ejemplo, se cuenta el número de símbolos que hay en la cadena.

```
let cadena = "hola8&que|tal_";
let simbolos = ["8", "_", "#", "&", "%", "|"];
num_simbolos = 0;
for(i in cadena){
    if(simbolos.indexOf(cadena[i]) != -1){
        num_simbolos = num_simbolos + 1;
    }
}
document.getElementById('rstl').innerHTML = num_simbolos;  //4
```

Cuidado: observa que los elementos de la lista `simbolos` están todos encerrados entre comillas, es decir, son cadenas. Esto es así porque la variable `c` es de tipo cadena, y `c in simbolos` solo devolverá True (entrando en el cuerpo del condicional) solo si existe en `simbolos` una cadena igual a `c`. Por ejemplo, si

quitamos las comillas al 8 de la lita (quedando `8` en vez de `"8"`), el resultado del programa será `3`.

Ej. 23: Escribe un programa que cuente el número de letras minúsculas que hay en una cadena introducida por el usuario.

Ej. 24: Escribe un programa en el que el usuario introduzca una cadena. El programa imprimirá la misma cadena, pero sustituyendo los dígitos por guiones.

Ejemplo: Si el usuario introduce `ab34cd5`, el resultado del programa será `ab--cd-`.

Pista: Recorre la cadena con un bucle for. Para cada iteración, si es un dígito, imprime un guion. En caso contrario, imprime el carácter original.

Ej. 25: Escribe un programa que cuente el número de caracteres que no son dígitos que hay en una cadena introducida por el usuario.

Pista: resta, a la longitud total de la cadena, el número de dígitos que hay en dicha cadena.

Ej. 26: Escribe un programa que pida una cadena al usuario. El programa imprimirá la cadena, con una coma tras cada carácter. El programa NO imprimirá coma tras el último carácter.

Ejemplo: Si el usuario introduce abcd, el programa devolverá `a,b,c,d`.

3. For tradicional

El lenguaje JS también posee el for tradicional. Este tipo de bucles es similar al while, solo que posee, entre los paréntesis, 3 instrucciones (en vez de una) separadas por dos punto y coma.

```
let cad = "012";
let mensaje = "";
for (let i=0; i<cad.length; i++){
    mensaje = mensaje + cad[i] + ",";
}
document.getElementById('rstl').innerHTML = mensaje;  //0,1,2,
```

El primer elemento, a la izquierda, es una instrucción que se realiza una sola vez al iniciar el bucle. El último elemento, a la derecha, es una instrucción que se realiza al final de cada iteración. El elemento central tiene la misma funcionalidad que en el while.

```
let cad = "012";
let mensaje = "";
let i = 0;
for (i<cad.length){
    mensaje = mensaje + cad[i] + ",";
    i++;
}
document.getElementById('rstl').innerHTML = mensaje;  //0,1,2,
```

Escribe un programa que pida una cadena al usuario. El programa imprimirá cada posición de la cadena y el carácter en dicha posición.

Ejemplo: Si el usuario introduce `abc`, el programa devolverá:

```
0 a
1 b
2 c
```

Pista: crea un bucle cuya variable vaya valiendo desde 0 hasta la longitud de la cadena menos 1. Usa esa variable para construir cada línea a imprimir.

Nota: para realizar este ejercicio, necesitarás usar un textarea, en vez de una etiqueta, por ejemplo:

<textarea name="nombreTextarea" rows="10" cols="50">

</textarea>

Para insertar un salto de línea en el textarea, se añade `\n`.

Ej. 27: Escribe un programa que use el for antes descrito, en el que el usuario proporcione un número. El programa imprimirá un mensaje que irá desde 1 hasta el número indicado por el usuario. Por ejemplo, si el usaurio introduce `3`, el programa imprimirá `123`.

Ej. 28: Realiza un programa en el que el usuario introduzca un número. El programa irá mostrando, en un textarea, los cuadrados de los números enteros, empezando por 1, hasta el número indicado por el usuario.

Por ejemplo, si el usuario introduce `3`, el programa mostrará:

```
El cuadrado de 1 es 1.
El cuadrado de 2 es 4.
El cuadrado de 3 es 9.
```

Ej. 29: Realiza un programa en el que el usuario introduzca un número. El programa irá mostrando, en un textarea, los cuadrados de los números enteros, empezando por 1, hasta que el cuadrado sea mayor que el número indicado por el usuario.

Por ejemplo, si el usuario introduce `8`, el programa mostrará:

```
El cuadrado de 1 es 1.
El cuadrado de 2 es 4.
```

4. Bucles anidados

Como sucedía con los condicionales, también es posible anidar bucles. En este caso, vamos a exponer un programa que imprime un rango de números (desde 0 hasta 2), pero lo hace, no una, sino dos veces:

```
for(let i=0; i<2; i++){  //i es controlada por este for
    for(let j=0; j<3; j++){   //j es controlada por este for
        mensaje += j;
}
document.write(mensaje);  //Imprime: 012012
```

En el paso ❶, el bucle exterior establece `i` a `0`. En este momento, `j` aún no está definida. En ❷ si que se define `j` a `0`. Ahora, en ❸, se imprime por pantallael valor de `j`.

❶ i:0 j:?

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❷ i:0 j:0

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❸ i:0 j:0 0

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❹ i:0 j:1 0

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❹ i:0 j:1 0

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❺ i:0 j:1 01

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❻ i:0 j:2 01

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❼ i:0 j:2 012

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❽ i:1 j:? 012

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❾ i:1 j:0 012

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❿ i:1 j:0 0120

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

❿ i:1 j:0 0120

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

⓬ i:1 j:1 01201

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

⓬ i:1 j:1 01201

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

⓮ i:1 j:2 012012

```
for(let i=0; i<2; i++)
  for(let j=0; j<3; j++)
    mensaje += j;
```

Tras ello, en el paso ❹, volvemos a la cabecera del bucle interno (el de la `j`), pues aún no lo hemos completado, cambiando el valor de `j` a `1` y, en ❺, volvemos a imprimir el valor de `j`. Luego, realizamos una iteración más del bucle interno en ❻ y ❼.

Completado lo anterior, puesto que el cuerpo del bucle exterior (el de la `i`), no posee más instrucciones que el bucle interno (el de la `j`), el paso ❽ vuelve a la cabecera del bucle exterior, incrementando el valor de `i` en uno, hasta un valor de `1`. Lo siguiente es realizar una nueva iteración del cuerpo del bucle exterior.

Dado que el cuerpo del bucle exterior es, justamente, ejecutar el bucle interior, en esta segunda iteración volvemos a realizar todos y cada uno de los pasos para completar las 3 iteraciones que nos impone el bucle interior. Así pues, volvemos a ejecutar ❾ y ❿ (primera iteración con `j` igual a `0`); ⓫ y ⓬ (segunda iteración con `j` igual a `1`); y ⓭ y ⓮ (tercera iteración con `j` igual a `2`). Tras ello, como no hay más instrucciones, el programa termina.

Cuidado: Las variables controladas por los bucles for anidados (`i`, `j`), no deben coincidir entre si (por ejemplo, no se debe poner dos bucles anidados usando, ambos, la variable i).

El siguiente programa ejecuta un bucle exterior (el de `i`), 3 veces (con `i` valiendo 0, 1 y 2). En cada una de estas iteraciones, se ejecuta el bucle interno (el de `j`), que ejecuta el `print` 5 veces (con `j` valiendo 0, 1, 2, 3 y 4).

Observa también que la otra instrucción `mensaje += "<br>";`, la que si implementa salto, se ejecuta tan solo 3 veces, pues está incluida en el bucle exterior, pero no en el interior.

```
for(let i=0; i<3; i++){  //i es controlada por este for
    for(let j=0; j<5; j++){   //j es controlada por este for
        mensaje += (i+j);
    }
    mensaje += "<br>";
}

document.write(mensaje);  //Imprime: 01234
                          //         12345
                          //         23456
```

Observa también que la instrucción ejecutada por el bucle más interno usa tanto `i` como `j`, lo cual es perfectamente posible.

Ej. 30: Escribe un programa que pida al usuario un número. El programa imprimirá un cuadrado relleno de asteriscos:

Ejemplo: si el usuario introduce `2`, el programa imprimirá:

```
**
**
```

Ej. 31: Escribe un programa que pida el número de filas y el número de columnas. El programa imprimirá un rectángulo relleno de asteriscos:

Ejemplo: si el usuario introduce `3` y `4`, el programa imprimirá:

```
****
****
****
```

Ej. 32: Escribe un programa que pida al usuario un número. El programa imprimirá un marco hueco de asteriscos:

Ejemplo: si el usuario introduce `3`, el programa imprimirá:

```
***
* *
***
```

Pista: Realiza el ejercicio como los anteriores, pero imprime el asterisco tan solo si cualquiera de las variables de los bucles valen igual a 0 o igual al número introducido menos uno.

Ej. 33: Escribe un programa que pida al usuario un número. El programa imprimirá esa cantidad de filas, la primera con un 0, la segunda, con un 1, etc.

Ejemplo: si el usuario introduce `3`, el programa imprimirá:

```
0
01
012
```

Pista: Realiza el ejercicio como los anteriores, pero (si la variable del bucle externo es `i`) el segundo bucle (el interno), irá desde `1` hasta `i`.

Escribe un programa que pida al usuario un número. El programa imprimirá esa misma cantidad de filas. La primera línea imprimirá tantos números consecutivos como el número introducido, empezando por cero. La segunda imprimirá uno menos, y así sucesivamente.

Ejemplo: si el usuario introduce `3`, el programa imprimirá:

```
012
01
0
```

Pista: Realiza el ejercicio como los anteriores, pero (si la variable del bucle externo es `i`) el segundo bucle (el interno), irá desde `i` hasta el número introducido.

5. Ejercicios adicionales

Ej. 34: Escribe un programa en el que el usuario escriba un número. Si el número es negativo, el programa volverá a preguntar al usuario un número, hasta que éste introduzca uno positivo. Finalmente, imprimirá ese positivo número introducido.

Ej. 35: Escribe un programa en el que el usuario escriba la parte real y la parte imaginaria de un número complejo (a través de dos instrucciones input). Si el número no está en el primer cuadrante (tanto parte real como parte compleja mayores o iguales a 0), el programa volverá a pedir el número complejo. Finalmente, imprimirá el número complejo introducido.

Ej. 36: Escribe un programa que pida al usuario una cadena y, sin usar subcadenas, imprima dicha cadena al revés.

Ejemplo: si el usuario introduce `abcd`, el programa devolverá `'dcba'`.

Pista: usa una variable índice `i` que, mediante un bucle, vaya valiendo desde longitud-1 hasta 0. En cada iteración, se imprimirá el caracter i-ésimo.

Ej. 37: Escribe un programa que cuente el número de dígitos que hay en una cadena introducida por el usuario.

Ej. 38: Escribe un programa que pida al usuario dos números, siendo el primero menor que el último. El programa imprimirá una línea empezando en el primer número y finalizando en el último

Ejemplo: Si el usuario introduce `3` y `6`, el programa imprimirá `3456`.

Nota: recuerda que la función `range(ini:fin)` empieza por `ini`, pero no incluye `fin`.

Ej. 39: Escribe un programa que pida al usuario una cadena y dos números. Sin usar subcadenas, el programa imprimirá el trozo de cadena entre las dos posiciones indicadas.

Ejemplo: si el usuario introduce `0123456`, y luego `3` y `5`, el programa devolverá `'345'`,

Ej. 40: Escribe un programa que pida al usuario 3 cadenas. El programa, usando un bucle, imprimirá la longitud de cada cadena.

Ej. 41: Escribe un programa que pida al usuario un número. El programa pedirá luego un número de cadenas igual al número indicado. Finalmente, imprimirá todas la cadenas concatenadas.

Ejemplo: si el usuario introduce un `2`, entonces el programa le pedirá 2 cadenas. Si, entonces, introduce `¡hola` y `mundo!`, el programa devolverá `'¡hola mundo!'`.

Pista: crea una variable `resultado`, que contenga una cadena vacía. En cada iteración del bucle, el programa pedirá una cadena (mediante un input), y concatenará dicha cadena `resultado`.

Ej. 42: Escribe un programa en el que, mediante un bucle y usando solo sumas, multiplique dos números introducidos por el usuario.

Ejemplo: si el usuario introduce 3 y 4, el programa devolverá 12.

Pista: crea una variable, llamada `resultado`, que empiece valiendo 0. Luego, crea un bucle que se repetirá tantas veces como indica el primer operando. En cada iteración, suma a `resultado` el valor del segundo operando. Por ejemplo, con `3` y `4`:

Bucle se repite 3 veces, sumando 4 en cada iteración: $4 + 4 + 4 = 12$

Ej. 43: Escribe un programa que muestre las tablas de multiplicar, del 1 al 10. Cada tabla se mostrará en una línea, y cada tabla separará los valores con coma y espacio:

```
1x1=1, 1x2=2, ...
2x1=2, 2x2=4, ...
...
```

Nota: este ejercicio no necesita ninguna instrucción input.

Ej. 44: Escribe un programa que pida al usuario un número mayor que 1, y compruebe si es primo. Para saber si un número es primo, dividimos el número indicado entre cada uno los números que vayan desde 2 hasta número-1. Todas esas divisiones deben tener un resto distinto de 0:

Pista: crea un bucle con una variable `i` que vaya desde 2 hasta `numero-1`. Para cada valor de `i`, comprueba que `numero%i == 0.` Si existe, aunque sea un valor de `i` en que ello suceda, entonces `numero` no es primo.

Nota (para comprobar resultados): los números primos entre 2 y 20 son 2, 3, 5, 7, 11, 13, 17 y 19.

Ej. 45: Escribe un programa que pregunte al usuario un número entero mayor que 0 , y que calcule el factorial de dicho número. Para calcular el factorial de un número, debes multiplicar todos los números que vayan desde 1 hasta dicho número;

n! = 1 * 2 * 3 * … * (n-1) * n

Ejemplo: si el número introducido es `3`, el resultado será 1*2*3 = `'6'`.

Pista: crea una variable `resultado` que inicialmente valga 1. Luego, crea un bucle que use una variable `i` que vaya desde 1 hasta el número introducido y que, en cada iteración, multiplique `resultado` con `i`.

Ej. 46: Escribe un programa que pida un números y que indique si es primo o no. Luego, repetirá el proceso dos veces más.

Pista: usa el código del ej_105, pero encapsula todo con un bucle que se ejecute 3 veces.

Ej. 47: Escribe un programa que pida 3 números mayores que 0. Luego, los introducirá en una lista. Finalmente, imprimirá cada número introducido y su factorial

Pista: Pide los 3 números y crea la lista como has hecho en otros ejercicios. Luego, rodea el código realizado en ej_106 con un bucle que itere la lista (tan solo tendrás que hacer un cambio menor en el print del ej_106).

Ej. 48: Escribe un programa en el que el usuario introduzca 2 números, siendo el primero menor que el segundo. Luego, el programa imprimirá desde el primer número hasta el segundo, pero saltando de dos en dos.

Ejemplo: si el usuario introduce `3` y `8`, el programa imprimirá `'357'`.

Pista: crea un while con una variable i que empiece valiendo el primer número. Luego, en cada iteración del bucle, incrementa i, no en 1, sino en 2.

Ej. 49: Escribe un programa que pida 10 números, y luego los almacene en una lista. Luego, usando un bucle, el programa imprimirá las posiciones pares, es decir, las posiciones 0, la 2, 4, 6 y 8. (el primer elemento, el tercero, etcétera.

Nota: observa que, en una lista de 10 números, la "posición 10" no existe (sería el elemento undécimo).

Pista: usa, como en el ejercicio anterior, un bucle cuya variable se incremente de dos en dos.

Ej. 50: Escribe un programa que pida 10 números, y luego los almacene en una lista. Luego, usando bucles, el programa creará una lista con las las posiciones pares, y otra lista con las posiciones impares. Finalmente, imprimirá esas dos listas.

Ej. 51: Escribe un programa en el que el usuario escriba 3 números y los introduzca en una lista. Para cada uno de ellos, indicará si es par o no.

Escribe un programa en el el usuario escriba un número. El programa mostrará todos los números entre 1 y el número indicado que sean divisibles entre 3.

Nota: `numero` es divisible entre 3 si si solo sí `numero%3==0`.

Ej. 52: Escribe un programa en el que el usuario escriba dos números. El programa mostrará todos los números entre 1 y el primer número indicado que sean divisibles entre el segundo número.

Nota: `num1` es divisible entre `num2` si si solo sí `num1%num2==0`.

Ej. 53: Crea un programa que vaya recorriendo una cadena proporcionada por el usuario. Para cada letra, el programa contará y mostrará, en un textarea, el número de veces que dicha letra aparece en la cadena, Por ejemplo, si el usuario introduce `aula`, el resultado será:

```
a: 2 vez(veces).
u: 1 vez(veces).
```

```
l: 1 vez(veces).
a: 2 vez(veces).
```

Ej. 54: Crea un programa que vaya recorriendo una cadena proporcionada por el usuario. El programa mostrará los caracteres repetido. Por ejemplo, si el usuario introduce `ordenador`, el resultado será: `ord`.

Ej. 55: Crea un programa que vaya recorriendo una cadena proporcionada por el usuario. El programa mostrará las posiciones y los caracteres que estén seguidos por la misma letra. Por ejemplo, si la entrada es `uuhhh`, el resultado será:

```
u: Posición 0.
h: Posición 2.
h: Posición 3.
```

Ej. 56: Crea un programa que vaya recorriendo una cadena proporcionada por el usuario. Para cada letra, el programa contará y mostrará, en un textarea, el número de veces que dicha letra aparece en la cadena. Sin embargo, no mostrará varias veces la misma letra. Por ejemplo, si el usuario introduce `aula`, el resultado será:

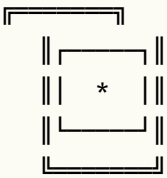
```
a: 2 vez(veces).
u: 1 vez(veces).
l: 1 vez(veces).
```

Pista: a la hora de comprobar si una letra tiene repetidas o no, comprueba antes, con un bucle, si la letra no ha aparecido antes.

6. Ejercicios avanzados

Ej. 57: Crea un programa en la que el usuario introduzca los valores de alto y ancho de un rectángulo, los cuales serán ambos impares y mayores que 4. El programa dibujará el rectángulo de bordes doble (usa los caracteres: ╔ ╗ ╚ ╝ = ║) y, en su interior, dibujará un rectángulo de borde simple (usa ┌ y ┐ └ ┘ − │). Finalmente, en el centro habrá un asterisco, estando el resto del interior vacío. Intenta hacer el código lo más compacto y eficiente posible.

Ejemplo: para un 9 y 5:



Tras realizar el ejercicio, prueba a hacerlo más compacto y eficiente, empelando `"╔" + "=".repeat(…) + …` y similares.

Ej. 58: Crea un programa en la que el usuario introduzca dos cadenas, una de origen y una de búsqueda. El programa devolverá la posición de la cadena de búsqueda dentro de la cadena de origen. En caso de no encontrarla, devolverá -1.

Ejemplo: si se introduce `abcdef` y `cd`, el programa devolverá `2`.

Realiza el ejercicio para que sea lo más eficiente posible, de forma que se salga de los bucles lo antes posible.

Ej. 59: Realiza un ejercicio cree los 100 primeros números primos. Para comprobar si un número es primo. basta con comprobar que el número no sea divisible con cualquier número menor que él.

La primera optimización sería saber que basta con comprobar que no es divisible por ningún primo menor que él (puedes emplear una lista para ir almacenando los números primos).

Puedes optimizar aún más el algoritmo teniendo en cuenta que para comprobar si un número es primo tan solo es necesario comprobar si es

divisible con los números primos menores o iguales que la raíz cuadrada del número que se está comprobando.

Ejemplo: construyendo la lista de números primos se llega hasta el el número 17, que debemos comprobar si es primo o no para añadirlo a la lista o no. En este momento tendremos en la lista los primeros números primos menores a 17 ([2, 3, 5, 7, 11, 13]). Comprobamos si 17 es divisible entre 2, entre 3, pero no es necesario comprobar más, porque 5*5 es mayor que 17.

Extra: ¿Por qué es posible hacer cada una de las optimizaciones mencionadas?

Ej. 60: Realiza un programa en el que se muestre (usa `document.write` o similar), en una tabla HTML, la posición final de la partida inmortal de Annand[(1)]. LA posición está codificada en el siguiente vector: [♖a1, ♗b1, ♖c1, ♕d1, ♔e1, ♖f1, ♙g1, ♙h1, ♙a2, ♙b2, ♙c2, ♙d2, ♙e2, ♙f2, ♙g2, ♙h2, ♙a3, ♞c3, ♕d3, ♝e3, ♙h3, ♙f4, ♞g4, ♟ b5, ♟ f5, ♟ g5, ♛h5, ♟ e6, ♟ a7, ♝ b7, ♟ g7, ♜c8, ♚g8].

Se deberá mostrar el color de fondo de las casillas, y el tablero debe estar estandarizado, con la fila 1 abajo, y la fila 8 arriba.

(1) Levon Aronian con blancas contra Viswanathan Anand en la ronda 4 del Tata Steel Chess Tournament (2013)

Unidad Didáctica 6:

Funciones

1. Definición

Para definir una función, basta con usar la palabra reservada def, seguida por el nombre de la función que deseamos definir. Tras esa primera línea, con la sangría apropiada, pondremos una o varias instrucciones, que serán la que compondrán el cuerpo de la función. En el siguiente ejemplo, definimos dos funciones:

```
function miFuncion() {
  document.getElementById('rstl').innerHTML = "hola";
}

function miOtraFuncion() {
  document.getElementById('rstl').innerHTML = "hola";
  document.getElementById('rstl').innerHTML += "mundo";
}
```

Es importante señalar que, en un programa, lo que se define en una función, por defecto no se ejecuta. Por ejemplo, el siguiente programa:

```
<script>

    function miFuncion() {
        //Este código NO se ejecuta por ser parte de una función
        document.getElementById('rstl').innerHTML = "hola";
    }

    //Código normal: SI se ejecuta.
    document.getElementById('rstl').innerHTML = "Adiós";

</script>
//Imprime: Adiós
```

Al ejecutar el anterior programa, no se ejecuta la función: solo se define. Es decir, por ahora, la función no es usada en ningún momento.

2. Invocar una función

Una vez definida, una función puede invocarse en cualquier parte. Para ello, simplemente tenemos que poner el nombre de la función seguido de paréntesis de apertura y cierre. Cuando se invoca una función, lo que sucede es que se ejecutan las instrucciones del cuerpo de la función:

```
function miFuncion() {
  document.getElementById('rstl').innerHTML += "hola!";
}

document.getElementById('rstl').innerHTML = "111";
miFuncion();
miFuncion();

//Imprime: 111hola!hola!
```

Observa que una función puede invocarse tantas veces como se desee y, en cada invocación, se ejecutarán todas las instrucciones del cuerpo de la función.

Ej. 1: Escribe un programa que defina una función, la cual imprima, únicamente, una a y una b seguidas (`ab`) El programa, invocando una función tres veces seguidas, imprimirá lo siguiente: `abababab` .

Escribe un programa que defina una función, la cual imprima, únicamente, dos aes seguidas (`aa`). El programa, invocando la función dos veces seguidas, deberá imprimirá lo siguiente:

`aa-aa`

Pista: será necesario intercalar una instrucción entre las dos invocaciones a la función.

Lugares de invocación

Una función puede invocarse desde cualquier sitio en el que pueda situarse una instrucción. Por ejemplo, puede formar parte del cuerpo de un bucle o un condicional. También puede invocarse una función en el cuerpo de otra función.

```
function miFuncion() {
  document.getElementById('rstl').innerHTML += "hola!";
}
```

```
document.getElementById('rstl').innerHTML = "Inicio";
for(let i=0; i<3; i++){
    miFuncion();

//Se imprime: Inicio hola!hola!hola!
```

Ej. 2: Escribe un programa que contenga una función, la cual imprima `¡Hola!`, sin salto de línea. Usando un bucle, haz que el programa imprima 5 holas seguidos: `'¡Hola!¡Hola!¡Hola!¡Hola!¡Hola!'`.

Ej. 3: Escribe un programa que contenga una función, la cual imprima `quién`, y otra que imprima `es` (también sin salto de línea). El programa deberá imprimir, invocando las funciones anteriores: `quién es quien`.

Ej. 4: Escribe un programa que contenga una función, la cual imprima `123`, y otra que imprima `abc`. El programa deberá imprimir, usando un bucle que invoque ambas funciones, lo siguiene: `123abc123abc123abc`.

Pista: solo es necesario un bucle for, no siendo necesarios bucles anidados.

3. Argumentos

Una función puede definir argumentos. El número y nombre de éstos argumentos se define en la primera línea de la función (la que contiene la palabra reservada `def`), entre los paréntesis.

Al invocar una función que tenga definidos argumentos, debemos suministras un número de argumentos igual al que la función define. Por ejemplo, en la función `imprimeTodo` del programa siguiente, se definen 3 argumentos. Por tanto, al invocarla, también se necesitan 3 argumentos.

Además, en el cuerpo de la función podemos usar los argumentos como si fueran una variable, su valor dependerá del valor proporcionado en los argumentos:

```
function imprimeTodo(txtini, txtmed, txtfin) {
  document.getElementById('rstl').innerHTML = "txtini";
  document.getElementById('rstl').innerHTML += "txtmed";
  document.getElementById('rstl').innerHTML += "txtfin";
}

imprimeTodo("Hola ", "¿qué tal ", "Adiós")

//Imprime: Hola ¿qué tal? Adiós
```

En el cuerpo de una función se pueden realizar todo tipo de instrucciones. Otro ejemplo de función sería:

```
function imprimeSuma(op1, op2){
    let suma = op1 + op2;
    document.getElementById('rstl').innerHTML =
        op1 + " + " + op2 + " = " + suma;
}

imprimeSuma(9,3)  //Imprime: 9 + 3 = 12
imprimeSuma(4,5)  //Imprime: 4 + 7 = 9
```

Observa que las instrucciones en el cuerpo de una función se comportan de igual forma que el resto. Por ejemplo, en el código siguiente estamos intentando sumar una cadena con un número, por lo que dará error:

```
function imprimeSuma(op1, op2){
    let suma = op1 + op2;  //Aquí, suma vale otra cosa.
    document.getElementById('rstl').innerHTML =
```

```
        op1 + " + " + op2 + " = " + suma;
}

imprime_suma("9",3)  //Error
```

Ej. 5: Escribe un programa que pida dos números. El programa empleará los dos números como argumento de una función llamada suma (que tendrá 2 argumentos). La función sumará los dos argumentos e imprimirá el resultado.

Ejemplo: Si el usuario introduce `3` y `4`, el programa llamará a la función con esos dos números, la cual imprimirá `7`.

Ej. 6: Escribe un programa que contenga una función que posea dos argumentos e imprima los operandos y su multiplicación (por ejemplo, con argumentos `3` y `4`, imprimirá `3x4=12`).

Ej. 7: Escribe otro programa que pedirá un número entre 1 y 10 y luego, invocando la función, la cual contendrá un bucle, imprimirá, en un textarea, la tabla de multiplicar de ese número.

Argumentos indeterminados

Dentro de una función, la variable `arguments` devuelve un array con todos los valores de los argumentos que se le pasen a la función.

```
function mostrarArgumentos() {
    for (let i = 0; i < arguments.length; i++) {
        console.log(`Argumento ${i}: ${arguments[i]}`);
    }
}
```

4. Return

Una función puede contener una o varias cláusulas return. Dicha cláusula devuelve el valor especificado a la derecha de la cláusula return. Ese valor puede ser usado en la instrucción que invoca a la función.

En el siguiente programa invocamos a la función sumar, con los argumentos 2 y 3. En ese momento, se ejecuta la función que, fundameentalmente, suma dos dos argumentos y devuelve la suma, que almacenamos en la variable suma.

```
function sumar(op1, op2){
    return op1 + op2;
}

let suma = sumar(2,3);
document.getElementById('rstl').innerHTML = suma;  //Imprime: 5
```

Cuando se invoca una función, no es imprescindible usar el valor devuelto por la función. La función seguirá siendo ejecutada normalmente.

```
Function imprimeYDevuelveSuma(op1, op2){
    let suma = op1 + op2;
    document.getElementById('rstl').innerHTML =
        op1 + " + " + op2 + " = " + suma;
    return suma;
}

imprimeYDevuelveSuma(2,3)  //Llama a la función, pero NO
                           //almacena ni usa el resultado.

//Imprime: 2 + 3 = 5
```

Tras ejecutar return, la función termina

Una vez que se ejecuta una cláusula return, la función deja de ejecutarse. Por tanto, todo código escrito tras dicha instrucción es código que nunca se ejecuta. En el siguiente ejemplo, el print de la función nunca se ejecuta, pues siempre se va a ejecutar un return antes que él.

```
function imprimeSuma(op1, op2){
    return op1 + op2;  //Aquí, suma vale otra cosa.
    document.getElementById('rstl').innerHTML = op1 + op2;
```

```
    //Esta última línea nunca se ejecuta
}
```

Observa que, cuando un return está dentro de un condicional o un bucle, es posible que se ejecute o no. En caso de ejecutarse, se termina la función pero, si no es ejecutado (porque la condición del condicional o bucle no se cumple), entonces el código posterior tiene posibilidad de ejecutarse. Es lo que sucede en el código siguiente:

```
function menor(op1, op2){
    if(op1<op2){
        return op1;
    } else {
        return op2;
    }
}
document.getElementById('rstl').innerHTML = menor(8,3);    //3
```

Ej. 8: Escribe una función de tres argumentos, en el que la función sume los tres argumentos y devuelva la suma. Emplea la función para sumar 3 números proporcionados por el usuario y mostrarlos en una etiqueta.

Ej. 9: Escribe una función que concatene dos argumentos pasados por parámetro. Emplea la función para concatenar dos cadenas proporcionadas por el usuario y mostrarlas en una etiqueta.

Ej. 10: Práctica general con HTML

Escribe un HTML (versión HTML 5) completo en el que haya 2 campos de texto, una etiqueta y un botón. Cuando se pulse el botón, el contenido de los dos campos de texto se sumará y el resultado se pondrá en la etiqueta. Si, al pulsar el botón, alguno de los campos está vacío, o los dos, se deberá indicar en la etiqueta, en vez de realizar la suma.

5. Funciones anónimas

En JS, también podemos definir funciones anónimas. Para ello, se utiliza también la palabra reservada `function`, pero no se incluye nombre de función.

```
const suma = function(a, b) {
    return a + b;
};

console.log( suma(1,2) );
```

Es posible invocar directamente a una función anónima empleando paréntesis:

```
(function(nombre) {
    console.log("Hola, " + nombre + "!");
})("Álvaro");

var resultado = (function(op1, op2) {
    return op1 + op2;
})(3, 4);
console.log(resultado);
```

Ej. 11: Define una función anónima que devuelva el número de letras que suman todas las cadenas pasadas por argumento. Tendrás que hacer un bucle y usar `arguments`.

Función flecha

La función flecha es, tan solo, una forma más corta de escribir funciones anónimas en JavaScript. Se omite la palabra reservada `function` antes de los argumentos y, además, se pone un igual y un mayor (imulando una flecha) entre los argumentos y el cuerpo de la función:

```
const suma = (a, b) => {
    return a + b;
};
console.log(suma2(2, 3)); // 5
```

```
const suma2 = (a, b) => {
    return a + b;
}(2,3);
console.log(suma2); // 5
```

Ej. 12: Define las funciones flecha necesarias para realizar la operación (1 + 2) * 5. Tendrás que combinar ambas expresiones.

La función flecha también tiene una versión abreviada, que es con el cuerpo de la función como una sola línea, y sin usar corchetes ni return:

```
const suma = (a, b) => a + b;
console.log(suma(2, 3)); // 5

const suma2 = ((a, b) => a + b)(2,3);
console.log(suma2); // 5
```

Ej. 13: Define una función flecha, en formato abreviado, que true si el número pasado por argumento es par, o false si es impar.

Ej. 14: Define una función flecha, en formato abreviado, que devuelva el mayor de dos números pasados por argumento. Puedes usar el operador ternario para hacerlo. Haz luego otra para el mayor de tres números.

Unidad Didáctica 7:

Typescript

Referencia:

https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

Curso y ejercicios:

https://www.learn-ts.org/

Proyecto

Crea una mini aplicación de contactos, que mostrará los contactos actuales y podrá crear nuevos contactos.

models/Contact.ts	El interface “Contact” (nombre y teléfono y, opcionalmente, descripción).
models/Contacts.ts	La clase que poseerá, fundamentalmente, un array de “Contact” y métodos para (1) añadir contacto, y (2) devolver contactos, que devolverá una COPIA del array de contactos. Inicialmente, siempre habrá un contacto, llamado “Emergencias”, con número de teléfono “911”, sin descripción.
ui/render.ts	Mostrará la lista de contactos, más un formulario (con dos campos de texto y un botón), destinado a añadir un contacto. Render solo exporta una función con un argumento de Contact[], que usa para construir la lista. Cuando no haya descripción, se mostrará “sin descripción”. Usa operador || o ?? para ello. Para iterar el array de contactos, y para iterar los campos de un contacto, emplea for...in y for...of.
main.ts	El main creará una variable de tipo Contacs, llamará a render y creará una función que conectará al botón del formulario para que se añada un nuevo contacto.

Parte III:

Programación avanzada

Unidad Didáctica 1:

Expresiones regulares

1. Busquedas literales

Las expresiones regulares tratan de buscar una expresión en un texto. Lo más sencillo que podemos hacer es buscar, por ejemplo, una palabra concreta dentro de un texto. En JS esto se hace con la función `match`:

```
const texto = "—¿Qué tengo que hacer para entrar? —volvió a preguntar " +
              "Alicia alzando la voz.\n" +
              "—Pero ¿TIENES REALMENTE QUE ENTRRAR? —dijo el lacayo—. " +
              "Esto es lo primero que hay que aclarar, sabes.\n";

const regex = /que/g;
const coincidencias = texto.match(regex); //['que', 'que', 'que']
const posiciones = [...texto.matchAll(regex)].map(match => match.index);
                   //[ 12, 152, 160 ]
```

En el ejemplo anterior, tenemos un texto en el cual vamos a buscar una expresión regular (comúnmente llamada regex). En JS, y en otros muchos lenguajes, estas expresiones regulares están encerradas entre símbolos de barra (`/`), y seguidas por alguna opción como una `g`, una `i` u otras. En el ejemplo anterior estamos buscando la palabra `que`.

Observa que el primer `Qué` no coincide, esto es porque la primera es mayúscula. Aún en caso de que no lo fuera, tampoco coincidiría, ya que ese primer `Qué` lleva tilde. El segundo `QUE`, tampoco entra porque no coincide por estar en mayúsculas.

Nota: es posible estudiar los patrones de una forma más sencilla con página web sobre patrones, tales como `https://regex101.com/`.

`Ej. 1:` En el texto anterior, haz la búsqueda de `es`.

Caracteres especiales

Algunos caracteres tienen una función especial en regex, por lo que deben ser escapados para poder ser buscados.

- `.` Cualquier carácter (excepto salto de línea).
- `^` y `$` Inicio y fin de línea. La `^`, también significa la negación en clases.

- `$` Fin de línea.
- `*`, `+`, `?`, `{` y `}` Caracteres usados para especificar repeticiones
- `\` Al usarse para escapar caracteres, para buscar el propio carácter de barra invertida deberemos usar `\\`.

Además de loa anterior, algunos caracteres no imprimibles pueden ser especificados con la barra invertida:

- `\n` Salto de línea (newline).
- `\r` Retorno de carro (carriage return).
- `\t` Tabulación horizontal (tab).
- `\f` Avance de página (form feed).
- `\v` Tabulación vertical (vertical tab).

Ej. 2: En el texto anterior, busca los caracteres de punto. Busca luego todaslas coincidencias de un punto seguido de un salto de línea.

Modificadores

Las letras del patrón de búsqueda (regex) que hay tras la segunda barra son modificadores que afectan a todo el patrón de búsqueda. Algunos de ellos son:

- `i` (insensitive): Hace la búsqueda insensible a mayúsculas y minúsculas (case-insensitive). Ejemplo: `/abc/i` encuentra `abc`, `ABC`, `AbC`, etcétera.

- `m` (multiline): Permite que el inicio (representado por `^`) y final (representado por `$`) de la expresión regular coincidan con el comienzo y final de cada línea, no solo del texto completo. Ejemplo: `/^abc/m` coincide con `abc` al principio de cada línea.

- `s` (dotall): Hace que el punto (`.`) coincida también con saltos de línea (`\n`), que normalmente no lo haría. Ejemplo: `/a.b/s` coincidiría con `a\nb`.

- `u` (unicode): Habilita el soporte completo de caracteres Unicode, permitiendo el uso de caracteres extendidos y combinados correctamente. Ejemplo: `/\p{Letter}/u` encuentra cualquier letra Unicode.

Ej. 3: En el texto anterior, haz la búsqueda de `que` de forma insensible a mayúsculas.

1

2. Clases de caracteres

En vez de realizar una búsqueda literal, carácter por carácter, es posible especificar ***clases de caracteres, usando corchetes***. Una clase de caracteres puede satisfacerse con cualquier carácter especificado entre los corchetes. Por ejemplo, Ejemplo: `[abc]` coincidirá con `a`, `b` o `c`.

El siguiente ejemplo encuentra las ocurrencias de `el` o `él` en el texto. Observa que la primera ocurrencia está dentro de una palabra mayor, pero aún así es detectada, pero no encuentra `El`, porque la `e` está en mayúscula.

```
const texto = "Y ella estaba comenzando a sentirse muy cansada de estar" +
              "sentada junto a su hermano en la orilla del río, y de no" +
              "tener nada que hacer. Una o dos veces había mirado el " +
              "libro que su hermano estaba leyendo, pero no tenía " +
              "imágenes ni conversaciones en él, '¿De qué sirve un " +
              "libro sin imágenes ni conversaciones?' pensó.\n";
              "El hermano intuyó sus pensamientos y dijo: Hermana, un ...";

const regex = /[eé]l/g;
const coincidencias = texto.match(regex); //['el', 'el', 'él']
```

Una clase de caracteres siempre representará a un único carácter.

Ej. 4: Haz que las palabras de `el` que sean inicio de frase (por lo que la `E` estará en mayúscula) también sean incluidas en la búsqueda. Busca luego todas las ocurrencias de `el`, incluyendo la mayúscula y la tilde en la `e`.

Ej. 5: Busca en el texto las ocurrencias de `hermano` o `hermana`.

Ej. 6: Encuentra todas coincidencias con una vocal minúscula.

Ejemplo: el texto `El murciélago` contendría 5 correspondencias, (`u`, `i`, `é`, `a` y `o`), cada una separada de las demás.

Ej. 7: Crea un patrón que detecte todos los nombres de fichero formados por un dígito (caracteres `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8` ó `9`) seguido por `.log`.

Nota: para especificar un punto con carácter de búsqueda, hay que poner, en el patrón de búsqueda `\.`, en vez de `.`, debido a que el punto tiene un significado especial que veremos más adelante. Así, por ejemplo, para buscar, por ejemplo, `1.1`, debemos realizar una búsqueda como `1\.1`.

Rangos

También es posible determinar un rango dentro de los corchetes, siendo los más haituales `[A-Z]` (todas la letras mayúsculas), `[A-Za-z]` (todas las letras) y `[0-9]` para todos los números. Pore ejmplo, el siguiente ejemplo encuentra todos los números de un texto:

```
const texto = "En 1789, la Revolución Francesa comenzó a cambiar el " +
              "curso de la historia. El 14 de julio, la toma de la " +
              "Bastilla fue un evento crucial. También en en '89, la " +
              "población de París era de unas 600,000 personas. A " +
              "mediados de 1793, la convención nacional proclamó la" +
              "República, mientras que en 1804 Napoleón Bonaparte se " +
              "coronaba emperador de Francia.";

const regex = /[0-9]/g;
const coincidencias = texto.match(regex); //['1','7','8','9','1','4','8'.
              //'9','6','0','0','0','0','0','1','7','9','3','1','8','0','4']
```

Ej. 8: Busca todos los años expresados de forma completa del texto anterior (`1789`, `1793` y `1804`, pero no `14`). Un año está formado por un dígito (un carácter de 0 a 9) seguido por otro dígito, luego otro y, finalmente un cuarto dígito.

Negación

Los corchetes también permiten especificar lo contrario a lo que se indica. En el siguiente ejemplo se busca una fecha que no esté seguida por una coma o bien las que no estén seguidas de una coma ni de un punto.

```
const texto = "La Revolución Francesa comenzó a cambiar el curso de la" +
              "historia en 1789. El 14 de julio, la toma de la " +
              "Bastilla fue un evento crucial. También en en '89, la " +
```

```
                "población de París era de unas 600,000 personas. A " +
                "mediados de 1793, la convención nacional proclamó la" +
                "República, mientras que en 1804 Napoleón Bonaparte se " +
                "coronaba emperador de Francia.";

const regex = /[0-9][0-9][0-9][0-9][^,]/g;
const coincidencias = texto.match(regex); //['1789','1804']
const regex2 = /[0-9][0-9][0-9][0-9][^,.]/g;
const coincidencias2 = texto.match(regex2); //[,'1804']
```

Ej. 9: Busca las expresiones en el texto del ejemplo que correspondan a un espacio, una `e` y luego otra cosa que no sea `n` ni `l`.

Ej. 10: Crea un patrón que corresponda a rutas formadas por una barra inicial, luego un directorio llamado `swap`, otra barra, y luego un nombre de fichero formado por cualquier carácter que no sea la barra, finalizado por la extensión `.txt por` (por ejmplo: `/swap/h.txt`).

Ej. 11: Crea un patrón igual que el anterior, solo que el carácter que conforma el nombre del fichero no es ni una barra ni un espacio ni un dígito.

Clases predefinidas

Algunos de los rangos anteriores pueden expresarse con clases predefinidas:

- `.` Significa cualquier carácter. Según las opciones de la búsqueda, puede incluir los saltos de línea y similares o no.
- `\d` coincide con cualquier dígito, equivalente a `[0-9]`, y `\D` Coincide con cualquier carácter que no sea un dígito.
- `\w` coincide con cualquier carácter alfanumérico (letras, números y guion bajo). Equivalente a `[a-zA-Z0-9_]`, y `\W` coincide con cualquier carácter que no sea alfanumérico.
- `\s` coincide con cualquier carácter de espacio en blanco (espacio, tabulación, salto de línea, retorno de carro, etcétera, y `\S` coincide con cualquier carácter que no sea un espacio en blanco.

Algunos de estos caracteres en blanco son el `\r` (retorno de carro, carácter ASCII 13), el `\n` (salto de línea, carácter ASCII 10), el `\t` (tabulación horizontal, carácter ASCII 9), y el `\f` (avance de página, carácter ASCII 12).

- `\p{}` coincide con cualquier carácter que pertenezca a una categoría Unicode específica. Por ejemplo: `\p{L}` coincide con cualquier letra

  Por su parte `\P{}` coincide con cualquier carácter que no sea de la categoría especificada. Por ejemplo: `\P{L}` coincide con cualquier carácter que no sea una letra.

```
const texto = "Ajustó el pequeño cuadrante de su máquina.\n" +
              "—Voy a avanzar exactamente 5 años en el futuro —dijo—. " +
              "Será un salto breve para probar que los mecanismos " +
              "funcionan correctamente.  Según mis cálculos, el reloj " +
              "interno de la máquina marcará 1.825 días cuando regrese.\n" +
              "El grupo de espectadores, formado por 7 personas, observó " +
              "con atención.  El Viajero encendió la máquina, y un " +
              "zumbido llenó la sala.\n" +
              "—¡Espera! —gritó uno de ellos—. ¿Y si avanzas 10 años " +
              "en lugar de 5?\n" +
              "El Viajero sonrió.\n";

const regex = /\d/g;
const coincidencias = texto.match(regex);
    //[ '5', '1', '8', '2', '5', '7', '1', '0', '5' ]
const regex2 = /\p{L}/gu;
const coincidencias2 = texto.match(regex2);
    //[ 'A', 'j', 'u', 's', 't', 'ó', 'e', 'l' ...]
const regex3 = /\P{L}/gu;
const coincidencias3 = texto.match(regex3);
    //[' ', ' ', ' ', ' ', ' ', ' ', '.', '—', ' ', ' ', ' ', ' ', '5' ...]
```

Ej. 12: Del texto del ejemplo, obtén los números completos (`5` , `1.825` , `7` , `10` y `5`).

Ej. 13: Crea un patrón que coincida con cualquier número expresado con puntos para miles, millones, miles de millones, etcétera.

Ejemplos: `10` , `1.456` , `14.332.333` .

Ej. 14: Crea un patrón que corresponda con todas las palabras del texto (`Ajustó`, `el`, `pequeño`, etc.). que no sean número (no se incluirá `5` ni otros números). Asegúrate de que usar búsqueda unicode.

Ej. 15: Crea un patrón que corresponda con todas las palabras del texto, incluyendo los símbolos de puntuación pegados a ellas. Por ejemplo, algunas de ellas serán `—Voy` o `—dijo—.`. Los saltos de línea y los espacios no se consideran símbolos de puntuación.

Ej. 16: Crea un patrón que coincida con un nombre de fichero que contenga uno o más dígitos seguidos.

3. Multiplicadores

Los multiplicadores afectan a lo que haya justo delante, de forma que, para coincidir, un texto debe tener cero, una o varias veces un patrón.

- `*` equivale a cero o más repeticiones. Por ejemplo, Ejemplo `a*` coincide con la vacía `a`, `aa`, `aaa`, etc. Otro ejemplo, `[ab]*` coincide con cualquier cadena que posea cualquier número de a y b (`aaaabaaa`, `bababa`, `bbbb`, etc.).

- `?` equivale a cero o una repetición (hace que el elemento anterior sea opcional). Por ejemplo: `colou?r` coincide con `color` y con `colour`.

- `+` equivale a una o más repeticiones, es decir, requiere al menos una ocurrencia para coincidir. Por ejemplo, `a+` coincide con `a`, `aa`, `aaa`, etcétera, pero no con la cadena vacía.

- `{n}` equivale a, exactamente n repeticiones. Por ejemplo, `a{3}` coincide con `aaa` (exactamente 3 veces `a`). Otro ejemplo: `[ab]{2}` coincide con `aa`, `ab`, `ba` y `bb`.

- {n,} equivale a n o más repeticiones. Por ejemplo, `a{2,}` coincide con `aa`, `aaa`, `aaaa`, `aaaaa`, etcétera.

- {n,m}: Coincide con entre n y m repeticiones, inclusive. Por ejemplo: `a{2,4}` coincide únicamente con `aa`, `aaa` y `aaaa`.

Estos multiplicadores son, por defecto, "greedy" (avariciosos), por lo que intentarán abarcar todos los caracteres posibles en su expresión regular. Por ejemplo, en un texto como `bbaaaabb`, el patrón `a{2,}`, coincidirá con la expresión aaaa, en vez de coincidir dos veces, una con `aa` y otra con la segunda `aa`.

Ej. 17: Busca todos los años del texto sobre la revolución Francesa, las fechas de año (1789, 89, 1793 y 1804), empleando esta vez, multiplicadores.

Ej. 18: Crea un patrón que se corresponda con cualquier número entre 0 y 99999.

Ej. 19: Crea un patrón que se corresponda con cualquier texto en español de entre 2 y cinco letras minúsculas seguido por un espacio (considera que las palabras pueden tener letras entre `a` y `z`, vocales con tildes o la última vocal con diéresis (`ü`), todo lo anterior mayúscula y minúscula.

Ej. 20: Crea un patrón que coincida con cualquier número que vaya desde 100 hasta 4999.

Pista: haz un patrón con una barra para contemplar dos posibilidades, una con un número de 100 a 999 y otra desde 1000 hasta 4999.

Ej. 21: Crea un patrón que coincida con cualquier dirección de correo electrónico válida. El patrón debe coincidir con lo siguiente:

1) Un nombre de usuario que contenga entre 1 y 20 caracteres, que pueden ser letras (mayúsculas o minúsculas), números, puntos (.), guiones bajos (_) y guiones (-).

2) El símbolo @.

3) Un dominio de entre 2 y 10 caracteres alfanuméricos (puede contener letras y números).

4) Un punto (.) seguido de una extensión que sea entre 2 y 4 letras.

Ej. 22: Ej. 23: Crea un patrón que coincida con cualquier número decimal positivo de al menos 1 dígito antes del punto decimal y hasta 4 dígitos después del punto.

4. Grupos de captura

Los grupos de captura tienen varios usos, uno de ellos es el poder aplicarles a un grupo de caracteres un multiplicador, o bien realizar varias posibilidades, con el operador barra.

```
const regex2 = /(la)+/g;
```

El ejemplo anterior aplica el multiplicador `+` a todo el grupo de captura, pudiendo encontrar textos como `la`, `lala`, `lalala`, etcétera.

Ej. 23: Crea una regla que corresponda a ficheros cuyo nombre esté formado por letra seguido de dígito, y así sucesivamente, terminando luego en `.txt`.

Ejemplo: coincidirá con `a0b1c8.txt` y con `z9.txt`, pero no con `0a1b.txt`.

Ej. 24: Extiende lo anterior para que los números puedan ser, en vez de dígitos, números de hasta 2 cifras.

Ejemplo: coincidirá con `a0b12.txt` y con `z90.txt`, pero no con `a123.txt`.

Operador Barra

Otra opción para buscar alternativas a la búsqueda es usar la barra. Por ejemplo, lo siguiente busca

```
const texto = "Y ella estaba comenzando a sentirse muy cansada de estar" +
              "sentada junto a su hermano en la orilla del río, y de no" +
              "tener nada que hacer. Una o dos veces había mirado el " +
              "libro que su hermano estaba leyendo, pero no tenía " +
              "imágenes ni conversaciones en él, '¿De qué sirve un " +
              "libro sin imágenes ni conversaciones?' pensó.\n";
              "El hermano intuyó sus pensamientos y dijo: Hermana, un ...";

const regex2 = /Una|Un/g;
const coincidencias2 = texto.match(regex); //[ 'Una' ]
const regex2 = /[Uu]n|[Uu]na/g;
const coincidencias2 = texto.match(regex2); //[ 'un', 'Un', 'un' ]
```

Ej. 25: busca todas las ocurrencias de `el`, `ella` y sus variantes (tildes y mayúsculas). Siempre que exista en el texto, da preferencia a encontrar `ella`.

Ej. 26: Busca todas las ocurrencias de `con` y `sin`. Hazlo teniendo en cuenta que alguno de ellos podría estar al principio de un frase.

Ej. 27: Busca todos los años del texto sobre la revolución Francesa, las fechas de año (1789, 89, 1793 y 1804). Un año está formado, o bien por 4 dígitos, o bien por un apóstrofo seguido por 2 dígitos.

Ej. 28: Crea una expresión que corresponda con todas las onomatopeyas de risa simples (`ja`, `je`, `ji`, `jo` y `ju`): `¡ja!`, `¡jajaja!`, `¡jeje!`, `¡jajejijoju!`, etcétera.

Pista: Aparte de las exclamaciones que son siempre igual, necesitarás un grupo de captura con varias opciones, y luego un multiplicador.

Ej. 29: Extiende lo anterior para que la primera letra de la risa sea en mayúscula. Añade también la posibilidad de `juas`.

5. Marcas

Las marcas identifican partes del texto sobre elque se realiza la búsqueda:

- `^` representa el inicio de línea o cadena.
- `$` representa elfin de línea o cadena.
- `\b` coincide con una frontera de palabra (un punto donde una palabra comienza o termina, como el espacio entre dos palabras).
- `\B` Coincide con cualquier posición que no sea una frontera de palabra.

```
const texto = "Ajustó el pequeño cuadrante de su máquina.\n" +
              "—Voy a avanzar exactamente 5 años en el futuro —dijo—. " +
              "Será un salto breve para probar que los mecanismos " +
              "funcionan correctamente.  Según mis cálculos, el reloj " +
              "interno de la máquina marcará 1.825 días cuando regrese.\n" +
              "El grupo de espectadores, formado por 7 personas, observó " +
              "con atención.  El Viajero encendió la máquina, y un " +
              "zumbido llenó la sala.\n" +
              "—¡Espera! —gritó uno de ellos—. ¿Y si avanzas 10 años " +
              "en lugar de 5?\n" +
              "El Viajero sonrió.\n";

const regex = /\b[Ee]n\b/gu;
const coincidencias = texto.match(regex); //[ 'en', 'en' ]
```

Las marcas representan puntos específicos del texto, pero no son obligatorias. Por ejemplo, si tenemos el texto `¡Hola que tal!`, y queremos buscar el término `a q` lo haremos con la expresión `a q`, no es necesario poner `a\b \bq`.

Ej. 30: Crea un patrón que corresponda con todas las palabras del texto. Asegúrate de que usar búsqueda unicode.

Ej. 31: Crea un patrón que corresponda con todas las palabras del texto que solo tengan palabras inglesas (`el`, `cuadrante`, etc.).

Ej. 32: Crea un patrón que coincida con un nombre de fichero que esté formado por uno o más dígitos (el nombre del fichero no tendrá nada más que dígitos).

Ej. 33: Crea un patrón que coincida con una ruta completa, empezando por barra (*/*), seguida por cualquier número de caracteres.

Ej. 34: Crea un patrón que coincida con una ruta de un fichero txt (terminado por `.txt`).

Ej. 35: Crea un patrón que coincida con los subdirectorios directos de `/var/www/`, pero no con otros directorios ni con ficheros.

Nota: los directorios terminan en `/`.

Ejemplo: Coincidirán `/var/www/html/`, `/var/www/public/`, pero no `/var/www/index.html` ni `/var/www/src/index.html` ni `/var/www/src/sam/`.

Unidad Didáctica 2:

Closures

1. Closures

Un closure es una función que recuerda el entorno en el que fue creada, es decir, puede acceder a variables fuera de su ámbito interno. Aunque hay varias formas de emplear este concepto, directamente accesible ej JS, una de la formás más típicas es algo semejante a lo siguiente:

```
function mensajeria() {
    const mensaje = "Hola";

    function muestraMensaje() {
        console.log(mensaje);  //mensaje está fuera de muestraMensaje
    }

    return muestraMensaje;  //devuelve una función
}

const miFuncion = mensajeria();  //mensajeria() devuelve una función
miFuncion();  //muestra: Hola
```

La constante `miFuncion` es una referencia a `muestraMensaje`, formado un closure, el cual almacena el valor de las variables de su entorno externo, en este caso, la constante `mensaje`. Si, en la función `muestraMensaje`, es modificada la variable externa, los cambios son conservados en la siguiente invocación a `miFuncion`. Por ejemplo:

```
function mensajeria() {
    let mensaje = "Hola";

    function muestraMensaje() {
        console.log(mensaje);  //mensaje está fuera de muestraMensaje
        mensaje = "¡" + mensaje + "!";  //Modificamos mensaje (externa)
    }

    return muestraMensaje;  //devuelve una función
}

const miFuncion = mensajeria();  //mensajeria() devuelve una función
miFuncion();  //muestra: Hola
miFuncion();  //muestra: ¡Hola!
miFuncion();  //muestra: ¡¡Hola!!
```

Ej. 36: Crea un closure que referencie a una función. La función sacará por consola (o por HTML con document.write) un cero. Si es invocado el closure

una vez más, deberá mostrar un uno, si es invocada una tercera vez, mostrará un dos, y así sucesivamente.

Los closures pueden tener argumentos

```
function mensajeria(mensInicial) {
    const mensaje = mensInicial;

    function muestraMensaje(mensAdicional) {
        mensaje = mensaje + mensAdicional
        console.log(mensaje);  //aumentamos mensaje
    }

    return muestraMensaje;  //devuelve una función
}

const miFuncion = mensajeria("hola");  //mensajeria() devuelve una función
miFuncion(" que");  //muestra: Hola que
miFuncion(" tal");  //muestra: Hola que tal
```

Ej. 37: Crea un closure como el anterior, pero en el que el valor mostrado inicialmente esté determinado por un parámetro pasado al crear el closure (la función invocada tendrá que admitir un argumento).

Ejemplo: inicializo el closure con un 3. Luego lo invoco con argumento 2, lo qu e me devuelve 5. Luego lo llamo de nuevo, con argumento 7, lo que me devuelve 12.

Ej. 38: Crea un closure que tenga un argumento que sea una tasa (un número). Luego, cada vez que se llame al closure, se realizará con un argumento, y te devolverá la tasa multiplicada por el argumento pasado.

Ejemplo: inicializo el closure con 1.08. Luego, lo invoco con 10 y me imprime 10.8. Lo invoco de nuevo con 100 y me devuelve 108. Observa que no hay valores acumulados en este closure.

Ej. 39: Haz un closure como el anterior, pero se permite invocar, tanto el argumento inicial como en cada una de las llamadas, con los argumentos que se desee.

Ejemplo: inicializo el closure con un (3,1). Luego lo invoco con argumentos (2,2,1), lo que me devuelve 9. Luego lo llamo de nuevo, sin argumentos, lo que me devuelve 9. Lo invoco otra vez, con 8, y me devuelve 17.

Varios closures simultáneos

En principio, los closures son independientes entre si, incluso aunque sean creados a partir del mismo código:

```
function contador(x) {  //x es una variable almacenada en el closure
    return function (y) {
        return x + y;
    };
}

const inicio5 = contador(5);  //closure con función "x+y", y "x" valiendo 5
const inicio8 = contador(8);  //closure con función "x+y", y "x" valiendo 8

console.log( inicio5(2) ); // Muestra 7  (5 inicial más 2)
console.log( inicio8(3) ); // Muestra 11 (8 inicial más 3)
```

Observa que cada closure almacena su propia variable `x`, independientemente de otros closures.

Ej. 40: Haz un closure que, como argumento a la hora de crearlo, establezca un número mayor o igual a cero. Cada vez que se invoque el closure, devolverá primero un cero, a la siguiente llamada un 1, y así hasta el número que se estableció al inicio. La siguiente invocación volverá a dar 0, la siguiente 1, así sucesivamente.

Crea dos closures (cada uno con un número distinto) de ese tipo y observa que funcionan indepencientemente.

Unidad Didáctica 3:

Programación funcional

1. Introducción

La programación funcional trata de transformar un flujo de datos para obtener un resultado. La programación funcional pura no altera el flujo original, creando un flujo de datos adicional.

Existen varias funciones relacionadas con la programación funcional:

- Map: devuelve un flujo de igual tamaño que el original, con cada uno de los elementos transformado en otro.
- Filter: elimina los elementos que no cumplen una condición.
- Some: Devuelve true si alguno de los elementos del flujo cumple una condición.
- Every: devuelve true si todos los elementos del flujo cumplen una condición.
- Find: devuelve el primer elemento que cumple una condición.
- Slice: devuelve una parte del flujo.
- Concat: concatena dos flujos.
- IndexOf: devuelve la posición del primer elemento que cumpla una condición
- Lastindexof: devuelve la posición del último elemento que cumpla una condición.
- Reduce: Crea un valor resultado recorriendo cada uno de los elementos del flujo.

También es posible crear flujos a partir de cadenas empleando split.

Las siguientes no se consideran programación funcional pura porque modifican, o pueden modificar, el flujo original:

- foreach: realiza una acción con cada uno de los elementos.
- sort: ordena un array
- reverse: invierte un array.

2. Transformar elementos

La primera y más básica tarea es obtener, a partir de un flujo de datos, otro en el que cada uno de los elementos se transforme en otro.

Map

Uno de los métodos básicos de la programación funcional. El método `map` recibe un flujo de entrada (un array, un String, un Set u otros), y devuelve un elemento de igual tipo pero con los elementos cambiados. El método `map` no cambia el objeto original:

```
const num = [6, 4, 1, -4, -9];

const numMas1 = num.map(num => ++num);  //debe ser ++num (no vale num++)

console.log(num);  //[ 6, 4, 1, -4, -9 ]
console.log(numMas1);  //[ 7, 5, 2, -3, -8 ]
```

`Ej. 1:` Utiliza `map` para duplicar cada uno de los elementos de un array de números.

Ejemplo: si el array original es `[2,7]`, el array devuelto será `[4,14]`.

`Ej. 2:` Utiliza `map` para convertir un array de cadenas todo a mayúsculas.

Nota: `String` posee un método, `toUpperCase()` que devuelve el mismo `String` pero en mayúsculas.

`Ej. 3:` Utiliza `map` para obtener las longitudes de cada cadena de un array.

Ejemplo: si el array inicial es `['Hola', 'mundo']`, se devolverá `[4, 5]`.

`Ej. 4:` Utiliza `map` para redondear los elementos de un array de números no enteros.

Nota: `Math.round(«número»)` devuelve el redondeo de `«número»`.

Ej. 5: Utiliza `map` para convertir un array de temperaturas expresadas en Celsius un array de las mismas tempreraturas expresadas en grados Fahrenheit.

Nota: si multiplicamos una temperatura expresada en grados Celsius por `9/5` y luego le sumamos `32`, tendremos la temperatura expresada en grados Fahrenheit.

Trabajando con objetos

Como es normal, `map` puede trabajar con objetos. Una de las operaciones más típicas es extraer uno de los elementos de un array de objetos:

```
const personas = [
  { nombre: 'Juan', edad: 25 },
  { nombre: 'Maria', edad: 30 },
  { nombre: 'Pedro', edad: 22 }
];

// Utiliza map para extraer la edad de cada objeto en el array personas
const edades = personas.map( obj => obj.edad );
console.log(edades); // [25, 30, 22]
```

A la hora de manipular arrays de objetos de forma más completa, a menudo nos veremos obligados a substituir el el argumento de la función pasada a `map` por uno que responda al patrón del array que estemos tratando. Por ejemplo, el siguiente ejemplo emplea map para devolver un array con los elementos iguales excepto la edad, que estará incrementada en uno:

```
const cargos = [
  { nombre: 'Juan', edad: 25, cargo:'Jefe' },
  { nombre: 'Maria', edad: 30 },
  { nombre: 'Pedro', edad: 22 }
];

// Utiliza map para devolver arrays de objetos con la edad incrementada
const c2 = cargos.map( ({nombre, edad}) =>
    ({nombre:nombre, edad: edad+1})
);

const c3 = cargos.map( ({nombre, edad, cargo}) =>
    ({nombre: nombre, edad: edad+1, cargo: cargo})
);
```

```
const c4 = cargos.map( obj => ({...obj, edad: obj.edad+1}) );

console.log(cargos);  // [{ nombre: 'Juan', edad: 25, cargo:'Jefe' }, ..

console.log(c2);  // [{ nombre: 'Juan', edad: 26 }, …

console.log(c3);  // [{ nombre: 'Juan', edad: 25, cargo:'Jefe' },
                  // [{ nombre: 'Maria', edad: 30, cargo:undefined }, ...

console.log(c4);  // [{ nombre: 'Juan', edad: 26, cargo:'Jefe' },
                  // [{ nombre: 'Maria', edad: 30}, ...
```

Como vemos, en `p2` se pierden los atributos del array `personas` que no especifiquemos expresamente. En `p3`, los objetos que no tengan `cargo` obtendrán dicho atributo, pues así se lo especificamos. Finalmente, en `p4`, devolvemos un objetos con todos los atributos del original (usando el operador "spread", y luego sobreescribimos el atributo de `edad`.

Esta forma de trabajar con los objetos también será de utilidad en `foreach` y en otros métodos de programación funcional.

Ej. 6: Utiliza `map` para extraer los nombres del array de `personas` usado en los ejemplos anteriores. El resultado debe ser: `[ 'Juan', 'Maria', 'Pedro' ]`.

Ej. 7: Utiliza `map` para extraer los nombres del array de `personas` usado en los ejemplos anteriores, pero cada nombre debe estar en de un objeto. El resultado debe ser: `[ { nombre: 'Juan'}, { nombre: 'Maria'}, { nombre: 'Pedro'} ]`.

Ej. 8: Utiliza `map` para devolver un array igual a personas, pero con los nombres todo en mayúsculas.

Nota: `String` posee un método, `toUpperCase()` que devuelve el mismo `String` pero en mayúsculas.

Ej. 9: Utiliza `map` para, a partir de un array de cadenas que contienen nombres, devolver un array de objetos, que contendrán, cada uno, un atributo de `nombre` y un atributo llamado `id` con valor `0`.

Por ejemplo, si el array inicial es `['Juan', 'María', 'Jose']`, devolverá `[{nombre:'Juan', id:0}, {nombre:'Juan', id:0}, {nombre:'Juan', id:0} ]`.

Ej. 10: Utiliza `map` para, a partir del a un del array de `personas` usado anteriormente, se añada, a cada uno de los objetos del array, un campo llamado `ciudad`, con valor `Jaén`. Observa que cada elemento de array debe conservar sus otros campos, sin añadir ni borrar ninguno.

map con 3 argumentos

Al igual que `foreach`, `map` tiene los argumentos de elemento, índice y array. Por ejemplo, en el siguiente código, se usa un array para crear otro que contenga tantos números naturales positivos como valores tenga el array original:

```
const nums = [ 1.2, 17, -3, 'Hola', {id: 1} ];

const numsConsecutivos = nums.map( (elem, index, array) => index+1);
console.log(numsConsecutivos);   //[ 1, 2, 3, 4, 5 ]
```

Podemos usar el índice para seleccionar qué valores modificamos. Por ejemplo, el siguiente código reemplaza tan solo ciertos valores del array original:

```
const ceroAlInicio = nums.map( (elem, index) => index==0 ? 0 : elem );
console.log(conCeroAlInicio);   [ 0, 17, -3, 'Hola', {id: 1} ]

const ceroAMitad = nums.map( (elem, index, array) =>
    index== (array.length-1)/2 ? 0 : elem
);
console.log(ceroAMitad);   //[ 1.2, 17, 0, 'Hola', {id: 1} ]
                           //Con longitud del array par, no habría cambio
```

Ej. 11: Utiliza `map` para devolver el array `nums` igual excepto que en las posiciones impares (recuerda que las posiciones empiezan por `0`: que es par: el `17` y el `'Hola'` serán sustituidos por el valor `0`).

Nota: para saber si se trata de una posición par, puede usar `index%2==1`.

Ej. 12: Utiliza `map` para, a partir de un array de cadenas que contienen nombres, devolver un array de objetos, que contendrán, cada uno, un atributo de `nombre` y un atributo llamado `id` con valor numérico consecutivo, empezando por `1`.

Por ejemplo, si el array inicial es `['Juan', 'María', 'Jose']`, devolverá `[{nombre:'Juan', id:1}, {nombre:'Juan', id:2}, {nombre:'Juan', id:3} ]`.

Ej. 13: Tenemos un array que es usa sucesión de unos y ceros, por ejemplo: [0,1,1,1,0,0,1,0]. Cada uno de los elementos del array resultante será un XOR de el elemento correspondiente con el anterior (el primer elemento queda igual).

Por ejemplo, para el segundo elemento, el valor resultante será XOR(0,1) = 1. Para el tercer elemento: XOR(1,1) = 0. El resultado final sería: [0,1,0,0,1,0,1,1].

~~Tenemos un array que es usa sucesión de unos y ceros, por ejemplo: [0,1,1,1,0,0,1,0]. Tenemos que generar un array en el que el valor inicial sea igual al valor inicial del array original. A partir de ahí, el valor del array generado cambiará si el array original es 1 o no cambiará si es 0.~~

~~Por ejemplo, para el array anterior, el array generado es [0,1,0,1,1,1,0,0]. El valor inicial es igual (cero), y como el segundo valor es 1, en el array generado se cambia desde el valor anterior (de cero a uno). El tercer elemento también contiene un uno, por lo que el array generado cambiará de valor (de uno del segundo valor a cero). El cuarto elemento es un cero, por lo que el array generado no cambiará (seguirá con el uno del tercer elemento).~~ Necesita foreach

3. filtrar elementos

El método `filter` devuelve un objeto igual que el original, pero conservando solo los elementos que cumplan una condición:

```
const nums = [ 2.1, 0, -3, "Hola", {id: 3}, "0", {} ];

const numsSinCeros = nums.filter( e => e!=0 );
const numsSinValoresCeros = nums.filter( e => e!==0 );

console.log(numsSinCeros);          //[ 2.1, -3, 'Hola', { id: 3 }, {} ]
console.log(numsSinValoresCeros);   //[ 2.1, -3, 'Hola', { id: 3 }, '0',
{} ]
```

Observa que el cuerpo de la función pasada por parámetro a `filter` se interpretará como un valor booleano:

```
const todo = nums.filter( e => 3==3 );
const nada = nums.filter( e => 3!=3 );
console.log(todo);   //[ 2.1, 0, -3, 'Hola', {id: 3}, "0", {} ];
console.log(nada);   //[]
```

Ej. 14: Utiliza `filter` para quedarte solo con los valores numéricos del array `nums` mayores que cero.

Nota: si comparamos una cadena con el operador `>` (por ejemplo `'cad'>0`) o comparamos con un objeto (por ejemplo `{id: 3}>0`), el resultado es siempre `false`.

Ej. 15: Utiliza `filter` para quedarte solo con los objetos del array `personas` con `edad` mayor a `25`.

Ej. 16: Utiliza `filter` sobre un array de cadenas para quedarte solo con las cadenas que tengan una longitud de 5 o más.

Ej. 17: Utiliza `filter` sobre el array de cargos (usado en el apartado de `map`) para quedarte solo con los objetos que no tengan definido un cargo (`undefined`).

Ej. 18: Utiliza `filter` sobre un array de cadenas para quedarte solo con las cadenas que no contengan espacio.

Nota: para saber si una cadena contiene un carácter, puede usar `«cad».includes(«caracter»)`.

Ej. 19: Obtén, en un array de cadenas, solo aquellas que tengan más de 3 vocales. Por ejemplo, del array `['elefante', 'computadora', 'agua', 'murcielago', 'oso']`, queremos conseguir `[ 'elefante', 'computadora', 'murcielago' ]`.

Nota: Para conseguir una cadena que solo tenga las vocales, puedes usar `«cad».replace(/[^aeiou]/gi, '')`.

filter con 3 argumentos

Al igual que `map`, `filter` tiene los argumentos de elemento, índice y array. Por ejemplo, en el siguiente código, se usa un array para filtrar el primer elemento:

```
const nums = [ 2.1, 0, -3, "Hola", {id: 3}, "0", {} ];
const numsMenosElPrimero = nums.filter( (e, index, array) => index==0 );
console.log(numsMenosElPrimero);  // [ 0, -3, "Hola", {id: 3}, "0", {} ];
```

Ej. 20: Utiliza `filter` para eliminar los elementos pared (el primero y el último) del array `nums`.

Ej. 21: Usa filter para eliminar todos los valores que sean iguales al primer elemento, incluido éste. Por ejemplo, para [“a”, “c”,”a”,”g”,”a”], el array resultante será [“c”,”g”].

Ej. 22: Utiliza `filter` para filtrar los números consecutivos que estén repetidos. Por ejemplo: `[ 3, 3, 3, 4, 3, 3 ]` resultará en `[ 3, 4, 3 ]`.

Nota: Para saber si un elemento está repetido, debe compararse con `array[index-1]`. Observa que está el caso especial en el que `index` sea igual a `0`, en el que siempre se debe devolver `true`, antes de comparar, para que no sea filtrado.

~~Utiliza `filter` sobre un array de números para que el array resultante no contenga elementos repetidos. Por ejemplo, `[1, 2, 3, 2, 1, 4]` resultará en `[1, 2, 3, 4]`.~~

~~**Pista:** conserva un elemento solo en caso de que su posición o índice (`index`) sea igual a la primera posición que dicho elemento aparece en el array. Para ello, `«array».indexOf(«valor»)` te da el índice de la primera aparición de un elemento en el array.~~ Necesita indexOf

Ejercicios adicionales

Los siguientes ejercicios hacen uso de filter y map:

Ej. 23: Dado un array de números, obtén otro array con los valoreas absolutos de los números negativos, descartando los que originalmente eran positivos. Por ejemplo, para el array de entrada `[ 5, -3, 8, -10, 12, -6, 7 ]`, se obtendrá `[ 3, 10, 6]`.

Ej. 24: Dado el array de cadenas de nombres, obten un array con solo los nombres compuestos, expresados en mayúsculas.

Ej. 25: Dado el array de objetos de personas con nombre y edad, obtén un array con los nombres de aquellos que tienen hasta 25 años (inclusive). Por ejemplo, para el array `personas`, el resultado será `['Juan', 'Pedro']`.

Ej. 26: Para el siguiente array:

```
const books = [
  { titulo: 'El Hobbit', pags: 300, disponible: true },
  { titulo: '1984', pags: 250, disponible: false },
  { titulo: 'Cien años de soledad', pags: 400, disponible: true },
  { titulo: 'Harry Potter', pags: 150, disponible: true }
];
```

Obtén los nombres de los libros disponibles con más de 200 páginas.

Ej. 27: En un array de números, quédate con los números que sean divisibles por 3, y luego súmale 1 a cada uno.

4. Reducir array

El método `reduce` convierte un array a un solo valor. Se especifica un valor inicial para un valor acumulado, y luego va ejecutando la función pasada por argumento para cada elemento. En cada una de esas iteraciones, va calculando el resultado de ese valor acumulado y lo va guardando para la siguiente iteración.

Por ejemplo, en el siguiente código, empezamos con un valor acumulado de `0`. Luego, se ejecuta la función para cada elemento, siendo ese resultado acumulado guardado:

```
const array = [1, 2, 3, 2, 1];
const sumaArray = array.reduce(  (acc, elem) => acc + elem,  0);
console.log(sumaArray); // Muestra: 9
```

Así, en la primera iteración, el primer argumento de la función (que es el valor acumulado), empieza valiendo `0`. Entonces, se ejecuta la función que suma ese `0` al elmento actual `1`, resultando en uno. En la siguiente iteración, el primer argumento vale ya `1`, que es sumado al elemento actual `2`, resultando en `3`, que será utilizado en la iteración siguiente, y así sucesivamente.

Ej. 28: Obtén el producto de todos los números que haya en un array de números.

Ej. 29: Partiendo de un array de cadenas, obtén una cadena que sea la concatenación de todas ellas.

Ej. 30: A partir de un array de números (tanto positivos como negativos), siendo un array con un tamaño mínimo de 1 (tiene, al menos, un elemento), obtén el número menor del array.

Nota: cuidado con el valor inicial proporcionado a `reduce`.

Reduce con 3 argumentos

Reduce tiene un tercer elemento, que es el índice. En el siguiente ejemplo calculamos el factorial de un número partiendo de un array:

```
const array = [1, 1, 1, 1];
```

```
const factorial = array.reduce(  (acc, elem, i) => acc * (i+1), 1);
console.log(factorial); // Muestra el factoria de 4: 24
```

Ej. 31: El array [2, -1, 1,-4, 7, 5, -4,-1, 10, 7,-15, 6], contiene los valores de una función:, empezando con x igual a cero. Haz que los valores que estén por encima de la bisectriz del primer cuadrante se queden como mucho en la bisectriz. Haz también que los números por debajo de la bisectriz del tercer cuadrante se queden, como mucho igual a la bisectriz.

Ej. 32: Tenemos dos arrays, t1 y t2, con las medias de un alumno en las distintas asignaturas en los dos trimestres. Realiza la nota media del alumno.

```
const t1 = [3, 9, 10];
const t2 = [1, 5, 7];
```

Trabajando con objetos

Reduce también puede trabajar con objetos. En estos casos, vamos utilizando los atributos de cada uno de los objetos del array original. Por ejemplo, el siguiente código busca el el precio más alto de entre todos los productos del array:

```
const carrito = [
  { nombre: "Lápiz", precio: 1, cantidad: 2, categoria: "A" },
  { nombre: "Goma", precio: 0.5, cantidad: 1, categoria: "B" },
  { nombre: "Sacapuntas", precio: 2, cantidad: 1, categoria: "A" }
];

const masCaro = carrito.reduce((mayor, actual) =>
    actual.precio > mayor ? actual.precio : mayor
, carrito[0].precio);

console.log(masCaro);
```

Ej. 33: Partiendo del array del `carrito`, obtén el precio total de los productos del carrito. El precio total deberá tener en cuenta, por ejemplo, que el lápiz está comprado con una cantidad de `2`.

Ej. 34: Tenemos una lista de los participantes de una carrera, ordenada de forma creciente según su tiempo en la carrera:

```
participantes = [{nombre: 'Alan' , equipo:'Kelme'  , min: 2, seg: 34},
                 {nombre: 'Nikko', equipo:'M+'     , min: 2, seg: 47},
                 {nombre: 'Fran' , equipo:'Astana' , min: 3, seg:  1},
                 {nombre: 'Elan' , equipo:'Kelme'  , min: 3, seg: 12},
                 {nombre: 'Kenny', equipo:'Astana' , min: 3, seg: 29},
                 {nombre: 'Elche', equipo:'M+'     , min: 3, seg: 29},
                  . . .  ];
```

Debemos obtener los segundos totales empleados por el equipo `Kelme`. Hay que tener en cuenta que los ganadores de la carrera son recompensados: 30 segundos de bonificación (30 segundos menos) para el primero, 20 para el segundo y 10 para el tercero.

También podemos usar un objeto como valor acumulado, de forma que vamos guardando resultados en dicho objeto. En el siguiente código se cuentan las ocurrencias de cada número en un array de números, que va almacenando en un objeto:

```
const numeros = [1, 8, 3, 4, 6, 1, 8, 3, 1, 8];

const ocurrencias = numeros.reduce( (acc, curr) =>
    ({ ...acc, [curr]: (acc[curr] || 0) + 1 })  //Si acc[curr] es falsy => 0
, {});

console.log(ocurrencias);  //Muestra:  {1: 3, 3: 2, 4: 1, 6: 1, 8: 3}
```

Observa que debemos proporcionar, como objeto inicial, un objeto vacío: `{}`.

Ej. 35: A partir de un array de objetos, siendo todos los atributos distintos, como por ejemplo `[ { a: 1, b: 2 },  { c: 3 },  { d: 4 } ]`, combina todos los objetos en uno: `[ { a: 1, b: 2, c: 3, d: 4 } ]`.

Nota: en la función del `reduce`, tendrás que devolver un objeto con todos los atributos del objeto acumulado (usa el operador spread), más todos los atributos del elemento actual (usa también el operador spread).

Ej. 36: Partiendo del array del `carrito` anterior, obtén un objeto que contenga todos los nombres separados por comas, el precio total y la cantidad total. En este caso, deberemos conseguir `{nombre: 'Lapiz, Goma, Sacapunta, ', precio: 4.5, cantidad: 4 }`. Observa, para el precio total, que el artículo `Lápiz` tiene una cantidad de 2.

Ej. 37: Partiendo del array del `carrito` anterior, calcula el total de precio de cada categoría: `{'A':4, 'B':0.5 }`.

También es posible que, al ir construyendo el objeto acumador (que en principo es `{}`), queramos añadir un campo cuyo nombre sea igual al valor de una variable. Para ello se utilizan corchetes:

```
const jugadores = [
  { nombre: 'José', eq: 'Kelme' },
  { nombre: 'Ana', eq: 'Nike' },
  { nombre: 'Luis', eq: 'Kelme' },
  { nombre: 'Sofía', eq: 'Adidas' },
  { nombre: 'Mario', eq: 'Nike' },
];

const conteoEquipos = jugadores.reduce(
  (acc, elem) => ({ ...acc, [elem.eq]: (acc[elem.eq] || 0) + 1 })
, {});
```

En el ejemplo anterior, [equipo] inserta un campo que se llamará igual al

Ej. 38: Partiendo del array de participantes de un ejercicio anterior, obtén un objeto que tenga como campos el nombre de cada equipo como y su tiempo empleado como los valores de dichos campos.

Realízalo inicialmente sin tener en cuenta la bonificación por los primeros puestos, y luego añade dicha bonificación.

Generando arrays

El siguiente código genera un array igual al que se le proporciona.

```
const nums = [3, -1, 0, 4, 5, -3];
const r = nums.reduce((a,e) => {a.push(e); return a}, []);
```

Observa que el valor inicial es `[]`. Además, observa que debemos devolver el acumulador con un return, ya que estamos empleando una función con llaves, y no la versión más compacta de la función flecha. Si no se usase el return, el array resultante estaría lleno de valores `Undefined`.

Ej. 39: Partiendo de un array de números, emplea sobre dicho array el método reduce (y solo el método reduce) para que devuelva un array solo con los números mayores que cero.

Ej. 40: Realiza el código de ejemplo y el ejercicio anterior empleando concat, sin usar return.

Ej. 41: Partiendo de un array de números, usa reduce para crear un array de valores acumulados, de forma que el valor de un array es igual a sumar todos los valores anteriores. Realízalo también con un map (sin usar reduce).

Ejemplo: `[3, -1, 0, 4, 5, -3]`, resultaría en `[3, 2, 2, 6, 11, 8]`.

Ej. 42: Usa reduce para generar un array que devuelva el mismo array de origen, pero en sentido inverso. Realízalo también con un map (sin usar reduce).

Ejemplo: `[-5, true, 'hi, {a:1} ]`, resultaría en `[{a:1}, 'hi', true, -5]`

Ej. 43: Usa reduce para generar un array que devuelva el array original, pero sin elementos repetidos. Hazlo luego usando un filter, sin reduce.

Pista: para el reduce, puedes usar el método `«array».includes(«elemento»)`. Para el filter, tendrás que usar un código parecido, pero necesitarás un slice (para el filter, también puede hacerse con solo indexOf, sin includes ni slice).

Combinado reduce y otros métodos

Ej. 44: A partir del array de objetos `carrito`, calcula el precio total de la categoría A. Hazlo de dos formas, usando un filter y sin usar un filter.

Ej. 45: Calcula el precio total del objeto `carrito` pero, esta vez, todos los objetos de la categoría `B` tendrá un 10% de descuento. Hazlo usando un map y un reduce. Luego, vuelve a hacerlo solo con un recude (sin usar un map).

Ej. 46: Obtén el nombre de los alumnos y su nota media, que pueden hacer la FCT. Para hacer dicha FCT, un alumno debe de aprobar cada una de las asignaturas y, para aprobar una asignatura concreta, la nota media entre el trimestre 1 y el 2 debe ser mayor o igual a 5 (en el array propuesto, solo aprueba Sofía):

```
const estudiantes = [
  { nombre: 'Ana',   t1: [3, 9, 10],  t2: [1, 5, 7] },
  { nombre: 'Luis',  t1: [6, 2, 5],   t2: [6, 5, 6] },
  { nombre: 'Sofía', t1: [9, 8, 10],  t2: [1, 5, 1] },
];
```

El código debe ser válido para escenarios donde el número de asignaturas sea distinto a 3. Puedes usar un reduce y, dentro de éste, otro reduce para fusionar el `t1` y el `t2` a una nota media. También es posible con un map (con un reduce dentro) y luego un filter, pero en este caso hay que establecer como 0 la nota media si cualquier asignatura está suspensa, aunque la nota media sea mayor que 5.

Ej. 47: Dado un array de proyectos que representa empleados con su historial de proyectos

```
const proyectos =
[  { nombre: 'Juan',  proyectos: [ { nombre: 'A', meses: 12 },
                                   { nombre: 'B', meses: 20 }, ] },
   { nombre: 'Lucía', proyectos: [ { nombre: 'A', meses:  8 },
                                   { nombre: 'C', meses:  5 },
                                   { nombre: 'D', meses: 10 }, ] },
   { nombre: 'María', proyectos: [ { nombre: 'B', meses:  6 },
                                   { nombre: 'C', meses:  9 },
                                   { nombre: 'F', meses: 14 }, ] }  ];
```

Obtén un array con los nombres de los empleados que han trabajado en al menos 3 proyectos y 2 años o más: `['María']`.

Ej. 48: A partir del array de proyectos anterior, obtén un array con los proyectos en los que se haya trabajado 20 meses o más (el A y B).

Ej. 49: A partir del array de estudiantes anterior, obtén un array con la nota media de cada asignatura (para una asignatura concreta, sería la suma de

todas las notas media de todos los estudiantes dividida por el número de estudiantes).

Ej. 50: [solo necesita reduce] A partir del array de proyectos anterior, crea un array que contenga un objeto para cada uno de los proyectos, cada uno con un campo `nombre`, con el nombre de proyecto, un campo `meses`, con el número de meses totales trabajados por todos los trabajadores, y un campo `trabajadores`, con un array de los trabajadores que han trabajado en el proyecto:

```
[ { nombre: 'A', meses: 12, trabajadores: ['Juan', 'Lucía'] }, ...
```

Pista: puedes usar un reduce para tener todos los meses trabajados en un único array: `[ { nombre: 'A', meses: 12, tr: 'Juan' }, { nombre: 'B', meses: 20, tr: 'Juan' }, { nombre: 'A', meses: 8, tr: 'Lucía' }, … ]` y luego usar un reduce para fusionar proyectos.

Ej. 51: Calcula, a partir de un array de notas (`[ {nombre: 'John', nota: 3}, {nombre: 'Kim', nota: 5}, {nombre: 'Jem', nota: 8} ]`), un array con la diferencia de cada una de las notas con respecto a la nota media.

Ej. 52: Dado un array con objetos heterogéneos, devuelve un array donde estén los mismos objetos, pero solo con los campos que estén en todos.

Ejemplo: en el array de cargos, donde tenemos que todos los objetos tienen nombre y edad, y solo algunos objetos tienen cargo, devuelve un objeto con todos los trabajadores con solo los campos nombre y edad.

Ej. 53: A partir del array de notas de ejercicios anteriores, calcula la nota media de los alumnos que están aprobados.

Ej. 53b: Obtén, partiendo del siguiente array:

```
const alumnos = [
  { nombre: "Ana", edad: 15,
    notas: [ { mod: "Mates", nota: 8 }, { mod: "Lengua", nota: 6 } ] },
  { nombre: "Luis", edad: 16,
    notas: [ { mod: "Tech", nota: 5 }, { mod: "Historia", nota: 9 } ] },
  { nombre: "Marta", edad: 14,
    notas: [ { mod: "Arte", nota: 6 }, { mod: "Ciencias", nota: 2 },
             { mod: "Dibujo", nota: 7 } ] ];

- Un array con las notas medias de cada alumno.
     [{alumno:'Ana', media:7} ...]
- La nota media total ((8+6)/2 + (5+9)/2 + (6+2+7)/3 ) /3
- La nota más baja de todas: 2
- La nota más baja de todas {nombre:"Marta", mod: "Ciencias", nota:2}
  (en caso de haber varias notas iguales, quedarse con una de ellas).
- La nota media total de todas las asignaturas aprobadas.
```

Ej. 53c: Obtén, partiendo del siguiente array:

```
const libros = [
 {nombre: "1984",    data: { nivel: "alto",  tema: "PLTC"},  ventas: 80},
 {nombre: "Alice",   data: { nivel: "medio", tema: "FANT"},  ventas: 50},
 {nombre: "Aladino", data: { nivel: "base",  tema: "FANT"},  ventas: 20},
 {nombre: "Marx",    data: { nivel: "medio", tema: "PLTC"},  ventas: 30},
 {nombre: "Platón",  data: { nivel: "base",  tema: "PLTC"},  ventas: 40},
];
```

- La suma de todos los libros de fantasía (`70`), usando y no usando filter.
- La suma de libros de cada nivel: `{alto: 1, base:2, medio:2}`.
- La suma de las ventas de todos los libros, según la temática:
  `{política: 150, fantasía: 70}`
- La cantidad de libros de cada temática: `{política: 3, fantasía: 2}`
- **[NO HACER]** La cantidad de libros de cada nivel según temática:
  `{  FANT: {alto: 1, base:2, medio:2},`
  `PLTC: {alto: 1, base:2, medio:2}  }`

5. Detectar y buscar elementos

Los métodos a continuación buscan o examinan los elementos de un flujo de datos.

Every

Devuelve `true` si todos los elementos del array cumplen con la condición especificada, y `false` en caso contrario. El siguiente código detecta si todos los números del array son pares y, luego, si todos los números son menores que su índice.

```
const nums = [14, -6, 0, -2];
const sonPares = nums.every( n => n % 2 === 0); //true
const sonPares = nums.every( (e, i, arr) => e > i ); //false
```

Ej. 54: ¿Todos los participantes de la carrera de uno de los ejercicios anteriores tardaron más de 2 minutos y medio, sin tener en cuenta bonificaciones? ¿Y teniendo en cuenta las bonificaciones por ser 1º, 2º o 3º?

Ej. 55: Tenemos un array `[15, 5, 5, 15, 20]` que indica el número de litros de agua que entran en un depósito cada día. Al final de cada día se necesitan extraer 10 litros. Averigua si tendremos suministro durante todos los días.

Some

Devuelve `true` si al menos un elemento del array cumple con la condición especificada. Devuelve `false` si ninguno la cumple.

```
const nums = [-1, -3, 1, -5];
const sonPares = nums.some( n => n % 2 === 0); //false
const sonPares = nums.every( (e, i, arr) => e > i ); //true
```

Ej. 56: Tenemos un array `[15, 5, 5, 15, 20]` que indica el número de litros de agua que entran en un depósito cada día. Al final de cada día se

necesitan extraer 10 litros. Averigua si tendremos suministro durante todos los días. Emplea `some`, en vez de `every`, para realizar el ejercicio.

Find

Devuelve el primer elemento del array que cumple la condición especificada. Si no encuentra ningún elemento que cumpla la condición, devuelve `Undefined`.

```
const nums = [9, -3, 10, -5];
const sonPares = nums.find( n => n % 2 === 0); //10
const sonPares = nums.find( (e, i, arr) => e > i ); //9
const sonPares = nums.find( e => e === 0 ); //Undefined
```

Ej. 57: Devuelve el nombre del primer artículo del carrito (de ejemplos y ejercicios anteriores) en el que la cantidad por el precio de dicho artículo valga 2.

Ej. 58: Tenemos un array de ganancia-fecha. Devuelve la fecha de la primera ganancia (ganancia mayor que cero);

```
const ventas = [  {ganancia: 10, fecha: '10/10/24'},
                  {ganancia:  0, fecha: '21/11/24'},
                  {ganancia: -8, fecha: '21/11/24'},
                  {ganancia:  9, fecha: '21/11/24'},  ];
```

Ej. 59: Tenemos un array de ganancia-fecha. Devuelve la primera pérdida (ganancia negativa);

```
const ventas = [10, '10/10/24',  0, '21/11/24',
                -8, '21/11/24',  9, '21/11/24'];
```

indexOf y lastIndexOf

`IndexOf` devuelve la posición de la primera aparición de un elemento en el array, o bien `-1` si el elemento no está en el array. Recuerda que las posiciones empiezan por cero.

`LastIndexOf` devuelve la posición de la última aparición, o bien `-1` si el elemento no está en el array.

En ambos métodos podemos utilizar el segundo parámetro, `fromindex`, que indica la posición a partir de la que se desea buscar, ignorando las posiciones anteriores (en caso de `indexOf`) o posteriores (en caso de `lastIndexOf`), aunque la posición indicada en `fromindex` sí que formará parte de la búsqueda.

```
const nums = [13, 10, -3, 7, -5, 10];

const primero = nums.indexOf( 10 );  //1
const primeroDesde2 = nums.indexOf( 10, 1 );  //1
const primeroDesde3 = nums.indexOf( 10, 3 );  //5

const ultimo = nums.lastIndexOf( 10 );  //5
const ultimoDesdeUlt = nums.lastIndexOf( 10, -1 );  //5
const ultimoDesdePenult = nums.lastIndexOf( 10, 4 );  //1
```

Ej. 60: Tenemos una cadena de texto que contiene varias palabras separadas por entre si por un espacio (no hay espacios adicionales ni dobles espacios). Sin emplear `split` ni similar, sino usando solo un bucle e `indexOf`, crea un array que tenga las palabras separadas.

Ejemplo: con una cadena como `Every light casts a shadow`, se genera un array como `['Every', 'light', 'casts', a', 'shadow']`.

6. Crear flujo desde cadena

El método `split` se aplica normalmente a una cadena, y especifica un separador. Lo que hace es partir dicha cadena empleando dicho separador. Por ejemplo, el siguiente código trocea una ruta en sus distintas partes (observa que el separador NO aparece en las partes resultantes):

```
const ruta = "carpeta1/carpeta2/carpeta3/archivo.txt";
const partes = ruta.split("/");
console.log(partes); //["carpeta1", "carpeta2", "archivo.txt"]
```

split con límite

También es posible especificar un límite al número de partes que se desean obtener, añadiendo un segundo argumento en el `split`. Por ejemplo, lo siguiente obtiene la primera palabra del texto:

```
const texto = "Este es un ejemplo de texto. Este texto tiene varias
palabras.";
const palabra = texto.split(" ", 1);
console.log(palabra-length); //1
console.log(palabra[0]); //"Este"
```

Ej. 61: Convierte una cadena con todas las vocales (`"aeiou"`) en un array con cada una de las letras (`['a','e','i','o','u']`). Observa que la separación entre una letra y otra es la cadena vacía.

Ej. 62: A partir de una cadena de fecha completa (una cadena del tipo `"19/02/2024 15:30:45"`, obtén el año usando split.

Pista: Ttendrás que partir la cadena por el espacio quedándote solo con la primera parte. Luego deberás partir el resultado (con otro split) separando por la barra, para, finalmente, quedarte con el tercer elemento.

Split y regex

También es posible realizar el split con una expresión regular. El siguiente códig separa las palabras con cualquier espaciado (espacios, tabuladores, saltos de líneas, etcétera) o grupo de espaciados).

```
const texto = "Este es un ejemplo de texto con múltiples palabras";
const palabras = texto.split(/\s+/);
```

Ej. 63: A partir de una cadena de fecha completa (una cadena del tipo `"19/02/2024 15:30:45"`, obtén el año usando un único split.

Ej. 64: Extrae los números de un texto. Observa que los números de un texto está separados por uno o más caracteres que no son números (representados por `\D`).

Ej. 65: Empleando dos splits, extrae una lista de elementos a partir de lo siguiente: `Los alumnos son Ana, Luis y Jose` (debe conseguir `["Ana", "Luis", "Jose"]`.

Ej. 66: Usa filter y split para extraer las palabras con tilde del texto: `Julia salió rápido, hacia la tienda junto al árbol, y compró café.`";

Combinando split con otros métodos

Es muy común emplear split con otros métodos para conseguir una gran cantidad de resultados. Por ejemplo, el siguiente código parte un texto en palabras (`split`), para luego reemplazar cada palabra a una que no tenga puntos ni comas (con `map`, convertimos `ustedes,` en `ustedes`, sin la coma final), para finalmente realizar un conteo de cada una de las palabras usando `reduce`.

```
const txt = "—¡Vaya!—se dijo Alicia—. He visto muchísimas veces un gato " +
            "sin sonrisa, ¡pero una sonrisa sin gato!¡Es la cosa más " +
            "rara que he visto en toda mi vida!  -  Alicia en el país de " +
            "las maravillas (L. Carroll)";

const cnt = txt.toLowerCase()
               .split(' ')
               .map(palabra => palabra.replace(/[^\p{L}\p{M}]/gu, ''))
```

```
                .reduce( (acc, p) =>  ({ ...acc, [p]: (acc[p] || 0) + 1 }),
{});
console.log(cnt); //{vaya: 1, se: 1, dijo: 1, alicia: 2, he: 2, ...}
```

Ej. 67: Mejora el ejemplo anterior. Habrás visto que, debido a que hay dobles espacios, se genera un campo indeseado en el array final representando a la cadena vacía. Además, la separación debería hacerse no solo con el espacio, sino con todos los símbolos de puntuación (`"'.,:;!¡¿?‘’«»“”\-_—()[]`) más los espacios. Realiza el patrón regex de forma global y unicode.

Ej. 68: Escribe un programa que, usando split, cuente el número de palabras que tiene.

Ej. 69: En base al ejercicio más arriba, cuenta el número de palabra únicas de un texto.

Ej. 70: Tenemos un array de cadenas con los partidos disputados en los play offs de la última Super Bowl:

```
const superbowl = [ "Eagles 24 - 31 Chiefs", "Chiefs - bye", "Texans
14 - 12 Chargers", "Chiefs 23 - 14 Texans", "Bills 29 - 32 Chiefs",
"Eagles 28 - 22 Rams", "Commanders 45 - 31 Lions", "Eagles 55 - 23
Commanders", "Chiefs 32 - 29 Bills"];
```

Obtén un objeto que obtenga la ronda en a la que cada equipo llegó tal como {“Bills”: 4, ...} (observa que, quien quedara en 1ª ronda tan solo tendrá un partido, y así sucesivamente).

Ej. 71: Obtén ahora un objeto con los partidos jugados, los ganados, los perdidos, puntuación total a favor y puntuación total en contra

7. Concaternar flujos

Concat también es un método funcional, porque no altera los operandos. Concat puede tener uno o varios argumentos, de forma que se pueden concatenar varias listas simultáneamente:

```
const arr1 = ["Celia", "Juan"]
const arr2 = ["Sofía", "Tomás", "Lola"];
const arr3 = ["Marta", "Antonio"];
const nombres = arr1.concat(arr2, arr3);
          //["Celi"a, "Juan", "Sofía", "Tomás", "Lola", "Marta", "Antonio"]
```

Ej. 72: Usando concat y usando una única vez length, calcula el número de elementos totales que hay en los dos siguientes arrays:

```
const a1 = Math.floor(Math.random() * 10);
const a2 = Math.floor(Math.random() * 10);
```

8. foreach

El método ***foreach*** realiza una operación por cada uno de los elementos de un array. Al método `foreach` se le pasa como argumento una función. Dicha función se ejecutará tantas veces como el tamaño del array. En cada una de esas ejecuciones, el argumento de la función será cumplementando con el valor de uno de los elementos del array.

Por ejemplo, podemos usar un `foreach` para mostrar cada uno de los elementos del array con el siguiente código:

```
const numeros = [6, 4, 1, -4, -9];

// Utiliza forEach para imprimir el doble de cada número en la consola
numeros.forEach(num => console.log(num));
```

Aquí, la función pasada por parámetro (`num => console.log(num)`) es ejecutada cinco veces. En la primera ejecución `num` valdrá `6`, la segunda `4`, hasta la quinta y última vez, en la que `num` valdrá `9`.

Ej. 73: El ejemplo anterior muestra los números en distintas líneas. En vez de usar `console.log`, usa `process.stdout.write`. Luego, separa los números entre si con una coma y un espacio tras cada número;

Empleando variables externas

Aunque lo siguiente se realiza mejor con ciertas funciones de programación funcional, vamos a emplear una variable externa para realizar ciertas tareas con el `foreach`. En el siguiente código calculamos el tamaño de un array usando una variable `tam` externa al `foreach`, que vamos incrementando en cada iteración:

```
const numeros = [6, 4, 1, -4, -9];
let tam = 0;

// Utiliza forEach para ir sumando uno a tam en cada iteración
numeros.forEach(num => tam++);

console.log('El tamaño del array es ' + tam);
```

Ej. 74: Usando un array a tu elección, emplea una variable al inicio del código, llamada `suma`, inicializada a `0`. Haz que en cada iteración se sume el valor del elemento array a `suma`. Después del `foreach`, imprime el valor de `suma`.

Ej. 75: Realiza otro código, a partir de un array a tu elección, que imprima tantos números naturales consecutivos como elementos tenga el array. Por ejemplo, si el array inicial es `[4,5, 'aa']`, el programa mostrará `1,2,3,`. Usa una cadena, en principio vacía, que al final se imprimirá.

Ej. 76: Realiza otro código, a partir de un array a tu elección, que obtenga la suma de los valores absolutos del array. Por ejemplo, si el array es `[4,-5]`, el programa mostrará `9`.

Foreach con 3 argumentos

Aunque la anterior forma de foreach es una de la más usadas, realmente la función que se le pasa al foreach tiene tres argumentos. El primer argumento es, como hemos vista antes, el elemento en cuestión. El segundo es el índice del array que estamos usando en ese momento. El tercero es el array en sí.

Por ejemplo, el código siguiente escribe en el array y pone todos los elementos a `1`. En cada iteración va escribiendo el elemento i-ésimo, estableciéndolo a 1:

```
const personas = [   { nombre: 'Juan' },  { nombre: 'Maria', edad: 30 }  ];

// Establece todos los elementos del array a 1: [1, 1]
personas.forEach( (e,index,array) => array[index] = 1);
```

Ej. 77: SIN utilizar una variable externa, a partir de un array a tu elección, haz que imprima tantos números naturales consecutivos como elementos tenga el array. Por ejemplo, si el array es `[4,5, 'aa']`, el programa mostrará `1,2,3,`. ~~Utiliza `process.stdout.write` para lograrlo.~~

Ej. 78: Genera una lista con 10 elementos (puedes usar `Array(10).fill(0)`). Luego, empleando foreach, genera en dicho array los primeros 10 números de Fibonnachi.

Nota: Los números de Fibonnaci son:
Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(n) = Fibonacci(n-2) + Fibonacci(n-1).

Foreach y objetos

Cuando trabajamos con objetos, podemos usar el primer parámetro para modificar el objeto en si utilizando el primer parámetro.

```
const personas = [
  { nombre: 'Juan', edad: 25 },
  { nombre: 'Maria', edad: 30 },
  { nombre: 'Pedro', edad: 22 }
];

// Utiliza forEach para incrementar la edad de cada persona en 1 año
personas.forEach(persona => persona.edad++);

console.log(personas);
```

Ej. 79: Tenemos un array de objetos, en donde cada objeto tiene 2 propiedades, `nombre` y `apellidos`. Utiliza un foreach para que cada objeto tenga un tercer campo, llamado `nombreCompleto`, con su apropiado valor.

Ej. 80: Suma las edades de todas las personas del ejemplo anterior con un foreach.

Foreach no devuelve nada

En todo caso, `foreach` no puede, como podrán otros métodos, encadenarse, puesto que foreah no devuelve nada:

```
let n = numeros.forEach(num => console.log(num + ','));  //Undefined
let p = personas.forEach(persona => persona.edad++) );  //Undefined
```

9. Transformar flujos

Reverse

Reverse es un método sin argumentos que invierte el orden de un array. No es un método de programación funcional, ya que transforma el array original.

```
const nums = [6, 4, 1, -4, -9];

const alReves = nums.reverse();  // Devuelve la misma referencia a nums
nums[0] = 0;
console.log(nums);     //[0, -4, 1, 4, 6]
console.log(alReves);  //[0, -4, 1, 4, 6]
```

Como vemos, reverse devuelve la propia referencia al array, no una copia, por lo que un cambio en el array nums se translada a alReves, incluso si la modificación se realizó tras el reverse.

Para simular una metodología de programación funcional, podemos crear una copia del array antes de llamar a reverse. Es posible, y muy común, hacer una copia de un array empleando el operador spread:

```
const nums = [6, 4, 1, -4, -9];

const alReves = [...nums].reverse();  //copia nums y luego invierte
nums[0] = 0;
console.log(nums);     //[ 0, -4, 1, 4, 6]
console.log(alReves);  //[-9, -4, 1, 4, 6]
```

Parte IV:
Servidor Node

Unidad Didáctica 1:

Gestión del proyecto

1. Iniciar proyecto

Abre un terminal en VSCode (`Terminal→New Terminal` o `Ctrl+May+`` abrirá una terminar el en panel inferior), o en el sistema operativo (sitúate, en ese caso, en el directorio del proyecto antes creado)

Lo primero será crear un directorio para el proyecto, en el lugar que desees. El nombre de directorio suele coincidir (aunque no tiene por qué), con el nombre de proyecto, el cual tiene ciertas restricciones de estilo: todo en minúsculas, sin espacios (usa guiones tales como `-` en su lugar) y sin caracteres especiales ni símbolos de puntuación. Luego, entra dentro del directorio con el comando `cd`.

Tras esto, podemos utiliar un gestor de paquetes interno para adquirir las dependencias de nuestros proyectos web. Podemos usar npm o yarn pero, para nuestros proyectos, debemos de tener también el paquete de node.

```
sudo apt install nodejs   //instala nodejs, que incluye npm
sudo apt install yarnpkg  //solo si deseamos usar yarn en de npm
```

Importante: Debemos restringirnos a usar, o bien siempre npm o bien usar siempre yarn en todo nuestro desarrollo, puesto que mezclar su uso puede dar efectos indeseados.

Respecto a las ventajas de uno y otro[3], npm es más conocido y extendido y, aunque ha mejorado últimamente, tanto en seguridad como en otros aspectos, yarn permite cara de módulos Plug'n'Play, así como una mayor velocidad y seguridad.

Npm

Para crear un nuevo proyecto web con npm, escribe el comando siguiente:

```
npm init  #creación con npm (dentro del paquete nodejs)
```

Completa, al menos, el nombre del proyecto, la descripción y el autor (este último debería coincidir con tu usuario en el repositorio remoto GitLab, GitHub o similar). El resto puedes dejar lo que te propone el sistema o dejarlo en blanco.

[3] https://www.upgrad.com/blog/yarn-vs-npm-which-package-monitor-to-choose/

Asegúrate, eso si, que el punto de entrada sea `index.js`. Cuando confirmes los datos verás que aparece un fichero llamado `package.json`. Si abres este fichero verás algo como

```
{
  "name": "npmbasic",
  "version": "1.0.0",
  "description": "introducción a npm y node",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "J.A. Sampedro",
  "license": "ISC"
}
```

Nota: Puedes ver este proceso en: https://www.youtube.com/watch?v=TF-TJJIjPsk, aunque cambian los atajos de teclado y algún que otro detalle.

Ej. 1: ¿Qué licencia es la que establece por defecto (di sus iniciales y su nombre completo) ¿Qué tipo de licencia es?

Ej. 2: Crea un proyecto con npm

Yarn

Podemos usar yarn, en vez de npm para crear el proyecto:

```
yarnpkg init  #creación con yarn (paquete yarnpkg)
```

En este caso, se nos crearán inicializará en el directorio una serie de fichero y directorios:

- Archivos de Git, aunque el repositorio remoto y demás configuración solo estarán establecidos si han sido especificados en el yarn init (no en nuestro caso).
- package.json: similar a npm, es el archivo principal de configuración del proyecto en Node.js. Inicialmente, a diferencia que npm, contiene el nombre del proyecto y el nombre y versión del gestor de paquetes yarn empleado.

- .pnp.cjs (Plug'n'Play – CommonJS): sado por Yarn cuando Plug’n’Play (PnP) está habilitado. Reemplaza la clásica carpeta node_modules optimizando la gestión de dependencias. Sirve para resolver las rutas de los paquetes sin necesidad de instalarlos físicamente en node_modules.
- .editorconfig: archivo de configuración para editors de código como VSCode, JetBrains, Sublime, etcétera Contiene reglas de formato (espaciado, codificación de caracteres, etc.).
- README.md: Archivo de documentación del proyecto en formato Markdown.
- yarn.lock: Archivo de bloqueo de versiones, realizado aquí en vez de en package.json.
- El directorio .yarn: Contiene archivos internos que usa Yarn para su funcionamiento interno, estos fichero son incluidos en el .gitignore.

Ej. 3: Crea un proyecto con yarn

2. Scripts

Un script es un comando, o un grupo de comandos, definidos por el desarrollador (o generados por las diversas herramientas) para realizar una serie de funciones, con un nombre. Están almacenados en el apartado `scripts` de `package.json`. Si hemos creado el proyecto con npm, podremos ver algunos:

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

En el ejemplo anterior, podemos ver los símbolos `&&`, que separan un comando y otro, de forma que se ejecuta el de la izquierda y luego el de la derecha. Podríamos encadenar más comandos con otros símbolos `&&`.

Si, por el contrario, hemos creado el proyecto con yarn, podemos añadir los scripts con ese mismo código (u otro que deseemos) a `package.json`.

Ejecutar scripts

Para ejecutar un script, nos dirigimos al terminal y usamos uno de los siguientes:

```
npm run «nombre_script»   #ejecución con npm
yarn «nombre_script»      #ejecución con yarn
yarn run «nombre_script»  #ejecución con yarn cuando el nombre del script
                           se solapa con un comando yarn (ej: si el script
                           se llamarara init).
```

Así por ejemplo para ejecutar el script `test`, podemos usar `npm run test` o las otras opciones. Recuerda eso si, restringirte a usar siempre yarn o siempre npm en la gestión de tus proyectos.

Ej. 4: Crea un script llamado `format` que ejecute el siguiente comando: `prettier --write`. Al ejecutarlo, dará un error porque prettier no está instalado.

Ej. 5: Crea un script llamado `clean` que borre la carpeta `dist` del proyecto junto a todos sus directorios y luego vuelva a crear el directorio `dist`.

3. Instalación de paquetes

Para instalar un paquete en nuestro proyecto, emplea:

```
npm install «paquete»    #ejecución con npm
yarn install «paquete»   #ejecución con yarn
```

Con npm, yarn v1 y yarn v2, los paquetes se instalarán en el directorio `node_modules`. Podemos borrar ese directorio y el proyecto no será dañado, aunque no funcionará hasta que se ejecuta el comando de instalación, que bajará de nuevo los paquetes y los grabará en `node_modules`:

```
npm install    #ejecución con npm
yarn install   #ejecución con yarn
```

Lo mismo sucede con yarn v2+(PnP), pero el contenido de los paquetes se crea en un único fichero llamado `.pnp.cjs`. Si se borra éste, debemos ejecutar, de igual forma, el `install`, para poder seguir desarrollando el proyecto. Este método, además, emplea una caché para que los módulos no tengan que ser descargados si es posible.

El borrado de estos paquetes es muy usual cuando queremos compartir el proyecto, ya que el destinatario podrá descomprimir un fichero que contenga todo nuestro proyecto excepto el `node_modules` (o el `.pnp.cjs`), y luego hacer él mismo el comando `install`.

Ej. 6: Crea un archivo `index.html` con una plantilla html básica (pon html en VSCode y luego selecciona `html:5`. Enlazalo con un archivo `main.js` con el siguiente contenido:

```
import dayjs from 'dayjs';
console.log("Fecha actual:", dayjs().format('YYYY-MM-DD HH:mm:ss'));
```

Prueba a ejecutarlo y, luego, a instalar el paquete `dayjs` antes de volver a ejecutarlo de nuevo.

Dependencias de desarrollo.

Existen dos tipos de dependencias, de forma que un paquete puede instalarse de forma normal, como hasta ahora, y éste será necesario para la versión de

producción, o bien como desarrollo, de forma que el paquete solo sea necesario durante el desarrollo. Las órdenes para instalar un paquete de esta forma son:

```
npm install «paquete» --save-dev
yarn add «paquete» --dev
```

Lo siguiente es un ejemplo de lo que se añadiría a `package.json` tras añadir como dependencia el paquete `eslint`:

```
  "devDependencies": {
    "eslint": "^8.59.0"
  }
```

Ej. 7: Cambia el archivo `index.html` para eliminar algunas de las tabulaciones del archivo (quedarán casi todas las etiquetas a la izquierda), guardando el resultado. Luego, instala el paquete como dependencia de desarrollo `prettier` y ejecuta el comando `format`.

Ej. 8: Borra el directorio node_modules e intenta ejecutar de nuevo el script `format`, que deberá dar error. Ejecuta `npm install` y verás como ahora si funciona.

4. Servir proyecto

Vamos a crear un pequeño proyecto web. Los ficheros de desarrollo, (tanto html, js, jsx, css y otros) se sitúan normalmente en el directorio `src` de nuestro proyecto. En ese directorio situamos el archivo src/index.html, que empleará un JavaScript y un archivo de estilo, por ejemplo:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Mi Proyecto Web</title>
    <link rel="stylesheet" href="style.css" />
  </head>
<html>
  <body>
    <h1>Hola</h1>
    <script src="main.js"></script>
  </body>
</html>
```

El archivo src/main.js va a ser algo muy simple, como por ejemplo:

```
document.write('Si, este es uno de mis primeros proyectos!');
```

Por su parte, el archivo CSS conta con algún formato relativo a la página web, como establecer el color de fondo, la letra, etcétera.

Finalmente, para servir es proyecto vamos a usar un servidor sencillo que podemos usar con `npm` y `yarn`, llamado `serve`. Debemos ejecutarlo de forma global con `npx`, en vez de instalarlo y usar `npm`.

```
npx serve src/  #no es la mejor idea el servir el directorio src.
```

Ej. 9: Crea un proyecto nuevo con `npm init`, y crea el directorio src junto a los archivos HTML, CSS y JS arriba mencionados. Luego sirve el directorio `src` con serve, y emplea el navegador para ir a la dirección indicada y ver el proyecto en él. Observa que, en la terminal donde servimos la web, se produce un error 404. Arregla dicho error.

Sin embargo, servir el proyecto src no es una buena idea. Lo normal es crear tener un directorio build (o similar) y copiar los ficheros en él. Para ello, podemos usar cpy para crear un script en `package.json` con el comando `cpy 'src/**/*' build` (si el directorio `build` no existe, entonces lo creará). Para mejorar aún más el script, podemos usar el paquete `rimraf` para poder borrar el directorio `build` y su contenido con el comando `rimraf build`. Observa que dichos paquetes deben instalarse como dependencias de desarrollo.

Ej. 10: Crea el mencionado script `build`, el cual llamará rimraf y luego (en el mismo script `build`) llamará a cpy para crear build y copiar el contenido de `src` en build.

5. Empaquetadores

Parcel

Borra `"main": "index.js"` de package.json, instala parcel y ejecuta `npx parcel build src/index.html` para empaquetar el proyecto hacia dist (el directorio por defecto de parcel). Usa `npx parcel src/index.html` para ejecutarlo en modo desarrollo.

Ej. 11: Crea un proyecto nuevo que implemente Parcel para empaquetar el proyecto. Instala Parcel como dependencia de desarrollo en el proyecto. El proyecto tendrá scripts para ejecutarlo y para construirlo. Observa que, como hemos instalado parcel de forma local, en los scripts no debe aparecer `npx`.

Webpack

Debemos instalar webpack y ciertos paquetes asociados, que serán dependencias de desarrollo

```
npm init -y
npm install --save-dev webpack webpack-cli html-webpack-plugin css-loader
webpack-dev-server style-loader
```

El HTML difiere de lo anterior, de forma que no hay ninguna llamada directa.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Mi Proyecto Web</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>Mi Proyecto Web</h1>
    <p>Bienvenido a mi proyecto. La fecha actual es:</p>
    <div id="fecha"></div>  <!-- Script lo inyecta HtmlWebpackPlugin -->
```

```
  </body>
</html>
```

En el index.js hacemos uso de datejs, de forma que deberemos instalarlo e importarlo (esta vez como dependencia normal). También importamos el estilo CSS, para que sea aplicado a los componentes aquí contenidos.

```
// Importamos el CSS para que webpack lo procese
import './style.css';
// Importamos la librería datejs instalada vía npm
import 'datejs';

// Seleccionamos el elemento donde mostraremos la fecha
const fechaElemento = document.getElementById('fecha');

// Usamos datejs para obtener la fecha actual y formatearla
fechaElemento.textContent = Date.today().toString('dd/MM/yyyy');
```

El archivo CSS, en nuestro caso llamado style.css, podría ser, por ejemplo:

```
body {
    font-family: 'Roboto', sans-serif;
    background-color: #f2f2f2;
    padding: 20px;
}
```

Webpack necesita una configuración que puede ser compleja. En nuestro caso, que es un proyecto sencillos, básicamente lo que indicamos en este archivo es la salida, los plugins que se utilizan para inyectar el estilo, los HTMLs y los scripts, y ciertas opciones de desarrollo. El archivo `webpack.config.js` sería:

```
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  // Archivo de entrada
  entry: './src/index.js',

  // Configuración de salida
  output: {
    filename: 'bundle.js', // Nombre del bundle generado
    path: path.resolve(__dirname, 'dist'), // Carpeta de salida
    clean: true, // Limpia la carpeta dist antes de cada build
  },
```

```
  // Reglas para procesar distintos tipos de archivos
  module: {
    rules: [
      {
        test: /\.css$/, // Para todos los archivos CSS
        use: ['style-loader', 'css-loader'], // Cargadores que permiten
importar CSS en JS
      },
    ],
  },

  // Plugins
  plugins: [
    new HtmlWebpackPlugin({
      template: './src/index.html', // Toma el index.html de src y le
inyecta el bundle
    }),
  ],

  // Modo de compilación: 'development' o 'production'
  mode: 'development',

  // Configuración del servidor de desarrollo
  devServer: {
    static: path.resolve(__dirname, 'dist'),
    port: 8080,
    open: true, // Abre el navegador al iniciar
  },
};
```

Unidad Didáctica 2:

Servidor web propio

Conectar scripts con node

Vamos ahora a definir otro script, uno llamado `start`. Este script lo que hará, será iniciar el servidor, cuyo punto de entrada, recordemos, es `index.js`. Para lograrlo, lo primero que haremos será crear un fichero con ese nombre (`index.js`) y, por el momento, ponemos tan solo un mensaje de que se ha iniciado el servidor:

```
console.log("servidor iniciado");
```

Ahora, si nos dirigimos al terminal, podemos ejecutar el siguiente comando para ejecutar el siguiente comando (no se apreciará gran cosa, tan solo se iniciará el servidor y luego terminará):

```
node index.js
```

Para conectar el script con node, vamos a establecer un script, llamado `start`, que ejecutará el anterior comando (no olvides poner comas para separar los distintos campos de un JSON:

```
"scripts": {
```

```
  "start": "node index.js,
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

A partir de ahora, si escribimos en la terminal el siguiente comando, nos ejecutará el script start, el cual lo que hace es iniciar el servidor:

```
npm run start
```

Vamos a crear otro script, llamado dev, que tenga la instrucción `npm -watch index.js`:

```
"scripts": {
  "start": "node index.js,
  "dev": "node --watch index.js",
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

Ej. 12: Cambia el script de `test` para que ponga el mensaje en español (o, si deseas conservarlo en inglés, pon un mensaje algo más completo).

Ej. 13: Añade un script llamado `help`, que muestre un mensaje de ayuda (pon cualquier texto referente al proyecto). Observa que el resultado de esta acción será éxito, es decir, deberá devolver `0`.

Nota: Puedes ver algo similar a lo anterior en https://www.youtube.com/watch?v=Ok3b8bbztzU&t=306s.

Ejecutar proyecto

Pulsa el icono de la barra lateral de ejecución (el tercer icono, que tiene forma de play), y verás que la barra lateral ofrece varias opciones. Dentro de `Run and Debug`, pulsa `create a launch.json file` y, cuando el sistema te ofrezca una serie de opciones, elige la de `Node.js`. Todo esto y modificará la barra lateral de ejecución, y también creará un archivo `launch.json` dentro del directorio `.vscode`:

```
{
// Use IntelliSense to learn about possible attributes.
// Hover to view descriptions of existing attributes.
// For more information, visit: https://go.microsoft.com/fwlink/?
linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Launch Program",
            "skipFiles": [
                "<node_internals>/**"
            ],
            "program": "${workspaceFolder}/index.js"
        }
    ]
}
```

A partir de este momento, en la barra lateral de ejecución se verán los paneles de ejecución y depuración. Podemos emplear el botón de “play” de arriba para ejecutar nuestro proyecto. Por ahora, al pulsarlo, la barra de estado inferior se tornará brevemente un color naranja, puesto que se lanzará el fichero `index.js` y luego finalizará (recordemos que, por ahora, solo posee una línea de código).

2. Crear el servidor

Muchas veces haremos uso de módulos existentes que ya implementan una funcionalidad que deseamos incorporar a nuestro proyecto. En nuestro caso, emplearemos el módulo `http` para gestionar las peticiones a nuestro servidor.

Crear el servidor con el módulo http

Vamos ahora a modificar el código del fichero `index.js` con lo siguiente:

```
const http = require('http');

// Página a servir
let page = '<!doctype html><title>Hola></title><body>Hola mundo ';

//Creación del servidor
const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-type': 'text/html' });
    res.end(page);
});

// Inicia el servidor para escuchar peticiones
server.listen(5000, 'localhost', () => {
    console.log('El servidor está escuchando en el puerto 5000')
});
```

En el código anterior se realizar 4 cosas. En primer lugar, se importa un módulo llamado `http`, que es un módulo integrado en el sistema y que no necesita ser instalado (llamados core modules).

En segundo lugar, definimos una página web, en formato html, que será servida por el servidor node cuando un cliente (un navegador) lo pida.

En tercer lugar, creamos un servidor, al cual debemos pasar una función. Dicha función tiene como parámetros una petición (request: req) y una respuesta (response: res). Por ahora, la función primero escribe en la respuesta una cabecera con un código 200 OK, y luego escribe la página definida arriba, terminando la conexión con `res.end`.

Finalmente, iniciamos el servidor en el puerto 5000 de localhost. El mensaje que escribimos con `console.log` será un mensaje escrito en el servidor al iniciar el servidor, y NO en la consola del navegador.

Ej. 14: Haz que la página a servir sea más completa, con un código html más completo: que tenga título (title), establezca el idioma (lang), con un contenido que posea un título (por ejemplo, hola mundo con <h1>) y un texto en un párrafo (<p>).

Ej. 15: Modifica el puerto de escucha a, por ejemplo, 9000. Emplea una constante, llamada LISTEN_PORT, y úsala en vez de escribir el número de puerto en mitad del código.

Ej. 16: Modifica el código para que se envíe un texto de `¡Hola Mundo!`. Observa que, ahora, el MIME devuelto debe ser de tipo `text/plain`.

Ej. 17: Busca cuales son los core modules y pon una muy breve descripción de cada uno.

Ej. 18: Establece el `content-type` a `text/html; charset=utf-8`. Prueba la página servida en el ejercicio 4, usando caracteres en la web tales como la admiración de inicio (`¡`).

Añadir paquetes al proyecto

Conforme vayamos desarrollando el proyecto, iremos necesitando módulos que no están integrados en `node`. En el siguiente ejemplo vamos a usar el módulo stringify, que lo tendremos que instalar. Stringify nos permite convertir objetos de JS a texto.

Para realizar la instalación, nos dirigimos al terminal dentro de VSCode y ejecutamos lo siguiente:

```
npm install stringify
```

Lo que hará que se instale el módulo stringify en el directorio `node_modules`, y podamos usarlo ahora en el código. Para ello, tenemos que usar un `require`, como el en anterior módulo `http`:

```
const http = require('http');
const stringify = require('json-stringify-safe');

let page = `<h1>Hello <h2><p>Your age are <p>`;

const server = http.createServer((req, res) => {
              res.writeHead(200,   {   'Content-type':   'text/html;
charset=utf-8' });
    res.end(stringify(req));
});

// Inicia el servidor para escuchar peticiones
server.listen(5000, 'localhost', () => {
    console.log('Server is listening at localhost on port 5000')
});
```

Ej. 19: Busca un módulo (también llamado paquete) que ponga en mayúsculas las primeras letras de cada palabra. Puedes buscarlo en https://www.npmjs.com.

Ej. 20: Busca qué hace el módulo llamado `express`.

3. Rutas

Hasta ahora, el servidor mostraba siempre lo mismo, independientemente de la ruta que tenga el servidor. Vamos a cambiar la función de respuesta del servidor, de forma que calculamos la ruta de la petición (pathname) y devolvemos justo eso.

```
const http = require('http');
const stringify = require('json-stringify-safe');

// Página a servir
let page = '<!doctype html><title>Hola></title><body>Hola mundo ';

//Creación del servidor
const server = http.createServer((req, res) => {
             res.writeHead(200,   {   'Content-type':   'text/plain;
charset=utf-8' });
    const url = new URL(req.url, `http://${req.headers.host}`);
    res.end(stringify(url.pathname));
});

// Inicia el servidor para escuchar peticiones
server.listen(5000, 'localhost', () => {
    console.log('El servidor está escuchando en el puerto 5000');
});
```

Ej. 21: Haz que, en la página inicio, se muestre una página web con una bienvenida, mostrando un texto con la ruta en el resto de rutas. El resto de páginas mostrará la página web de `Hola mundo`.

Nota: La página de inicio deberá activarse cuando el cliente pida `/index.html` o `/`.

Ej. 22: Haz que, si la ruta solicitada es un archivo distinto a html (que su extensión sea distinta a `.html`, se devuelva un error 404 sin ningún `content-type` (usa la función `writehead` con un solo argumento, y luego la función `end` sin ningún argumento).

Nota: en Node, para saber si una cadena termina en .html puede usarse `«cadena».endsWith('.html')`.

Enviar ficheros

Crea, con el explorador de VSCode, un directorio llamado `files`, que será nuestro directorio base de ficheros a servir. Dentro de dicho directorio, crea un fichero llamado `base.html`.

Vamos ahora a usar otro módulo integrado en node, que es el módulo `fs`, para cargar las páginas desde archivo. Lo que hacemos es cargar el fichero en un stream y enviarlo a la respuesta. Si se produce algún error (básicamente, que no encuentre el archivo), se envía un código de error 404.

```
const BASE_PATH = "files";
const http = require('http');
const fs = require('fs');

//Creación del servidor
const server = http.createServer((req, res) => {

    const url = new URL(req.url, `http://${req.headers.host}`);

    var stream = fs.createReadStream(BASE_PATH + url.pathname);
    console.log(BASE_PATH + url.pathname);
    stream.on('error', function() {
        res.writeHead(404);
        res.end();
    });

                res.writeHead(200,    {    'Content-type':    'text/html;
charset=utf-8' });
    stream.pipe(res);
});

// Inicia el servidor para escuchar peticiones
server.listen(5000, 'localhost', () => {
    console.log('El servidor está escuchando en el puerto 5000')
});
```

Ej. 23: Haz que el servidor comprueba si la ruta de la página termina en .html, en cuyo caso devolverá la página web correspondiente (si existe). En otro caso, mostrará un mensaje de error.

Nota: esto evita al servidor tener que buscar ficheros que no sean html (por ejemplo, en el caso de que nuestro servidor solo contenga ficheros html, evitando accesos a disco.

Ej. 24: Usa `const homepage = fs.readFileSync(BASE_PATH + "/index.html", "utf8");` al inicio del código (justo debajo de los `require`) para cargar la página web de inicio hacia una cadena.

Ahora, comprueba si la ruta solicitada es `/` o `/ndex.html`, en cuyo caso deberás servir la página cargada `homepage`. En otro caso, haz lo que se indica en el ejercicio anterior.

Nota: esto nos permite tener en memoria una página que, supuestamente, se emplea con asiduidad, de forma que evitamos accesos a disco.

Ej. 25: Usa telnet para conectar con el servidor y ver, a bajo nivel el funcionamiento del servidor. Por ejemplo, usa `telnet localhost:5000` y pon lo siguiente para acceder a la página `base.html`:

```
GET /base2.html
host: localhost
```

(luego tendrás que pulsar `enter` dos veces). Explica brevemente (no pegues el resultado, en su lugar explícalo) el resultado de una petición encontrada una que no (distingue entre lo que es la cabecera y el contenido).

Unidad Didáctica 3:

Crear proyecto React

1. Create React App

Create React App es una aplicación conocida y cómoda para proyectos existentes, y tiene una comunidad más establecida. Hoy por hoy funciona solo con la versión anterior de React, la versión 18.

Crear aplicación

Para crear la aplicación, deberás ejecutar el siguiente comando, que bajará los paquetes necesarios y ejecutará la aplicación

```
npx create-react-app «directorio»
```

Posiblemente dará un error, porque Create React App emplea la versión 18 de React, pero tendrás instalada la versión 19. Vamos a hacer que nuestro proyecto emplee React 18, para lo cual tendremos que abrir el `package.json` y cambiarlo a dicha versión.

```
"react": "^18.X.X",
"react-dom": "^18.X.X"
```

Ahora tendremos que reinstalar, para lo cual deberemos primero borrar el directorio `node_modules` y hacer una instalación:

```
rm node_modules
npm install
```

Ej. 26: Ejecuta los comando anteriores para crear una aplicación de React usando Create React App. Busca en Internet para saber cuál es la última subversión de React 18.

```
npm run start
```

Se abrirá una pestaña en el navegador. Cada cambio guardado en el código automáticamente se mostrará en el Pulsa Ctrl-C para parar el servidor.

```
npm install --save-dev web-vitals
npx run build
```

```
npx serve -s build
```

Tras ello, puedes abrir la dirección http://localhost:3000 para ejecutar la aplicación de React. Pulsa Ctrl-C para parar el servidor.

Estructura

Create React App crea dos directorios fuente, src y public, que serán los que serán utilizados para crear la versión final del proyecto, la cual se sitúa en build.

2. Vite

Vite supera a Create React App en velocidad gracias a su compilación en tiempo real, y ofrece una configuración más sencilla basada en ESM, por lo que es preferible para proyectos nuevos.

Crear aplicación

Para crear un nuevo proyecto, podemos emplear:

```
npm create vite
```

Escribe el nombre del proyecto (recuerda usar solo minúsculas y guiones), selecciona React y, finalmente, selecciona JavaScript. Con ello te habrá creado un directorio con el proyecto. Para ejecutarlo, deberás introducirte dentro del directorio, instalar los paquetes y ejecutar el script `dev`.

```
cd «proyecto»
npm install
npm run dev
```

Ej. 27: Crea un proyecto de React (usando JavaScript) con Vite.

Estructura

Con Vite, no aparece la carpeta public. En package.json habrá menos dependencias que Create React App, porque éste último instala de base muchas más cosas de base. Si en Vite quisieramos usar algunas de ellas tendríamos que instalarlas y configurarlas manualmente. Tampoco hay un README.md o un manifiest que tendríamos que crear nosotros en caso de necesitarlo.

En Vite aparece un archive vite.config.js que permitirá configurar ciertos aspectos de Vite en nuestro proyecto.

3. Main y componente principal

En primer lugar, creamos una aplicación con Vite (React + JavaScript), y borramos todo el contenido del directorio src, dejando el resto como están. Para el nombre del proyecto puedes emplear `mi-primera-App` o el nombre que desees.

Vamos ahora a implementar el funcionamiento de una aplicación muy sencilla en un archivo jsx que podemos llamar de forma igual o parecida al nombre de la aplicación u otro que deseemos, por ejemplo `MiPrimeraApp.jsx`. Esta aplicación va, tan solo, a mostrar un mensaje:

```
export function MiPrimeraApp() {
    return <h1>¡¡Mi primera aplicación!!</h1>
}
```

Tenemos ahora que crear el archivo `main.jsx`, que será el que llame a la aplicación:

```
import React from "react";
import ReactDOM from 'react-dom/client';
import { MiPrimeraApp }  from "./MiPrimeraApp";

ReactDOM.createRoot(document.getElementById('root')).render(
    <React.StrictMode>
        <MiPrimeraApp />
    </React.StrictMode>
);
```

Observa que tenemos que importar ciertos elementos de React, y también el fichero que contiene la aplicación. La importación elegida no es con `export default`, por lo que tendremos que poner llaves en el import de `main.jsx`. Otra opción sería usar un `default export` en el archivo de la función, aunque esto ocasionaría que, en `main.jsx`, tendríamos que decidir un nombre para usarlo en ese fichero `main.jsx`.

Ejecutar la aplicación

Para ejecutar la aplicación para el desarrollo, podemos ejecutar el script `dev`, que realmente llama a vite:

```
npm run dev
```

Con esto se abrirá un navegador que cambiará cada vez que grabemos cambios en los archivos fuente.

Si quisiéramos crear una página web final, para publicarla en un servidor, podemos ejecutar el script build del proyecto, que lo que realmente hace es invocar internamente a Vite con el comando `vite build`:

```
npm run build
```

Para servir esa web generada en dist, podemos usar un servidor web incluido en npm llamado Serve. El siguiente comando bajará, si es necesario, el mencionado paquete y servirá la web como si de un Apache muy básico se tratara:

```
npx serve dist
```

Si examinamos el fichero index.html generado, vemos que es una página web que puede ser servida por cualquier servidor Apache, Nginx o similar.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
    <script type="module" crossorigin src="/assets/index-ulgK4Yt5.js">
    </script>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

4. Modificando app

Añadiendo CSS

Se crea un archivo css e luego se importa en `main.jsx`.

Ej. 28: Dentro del directorio `src`, Crea un archivo el nombre de `style.css`, que establezca el color de letra rojo oscuro (u otro color que desees) a todos los hijos `body` que sean de tipo h1 con un color de letra (u otro formato que desees). Para usarlo, añade `import "./style.css";` al fichero `main.jsx`.

Sustituye ahora el código devuelto para que sea un `h1` y un párrafo, ambos encapsulados con un `div`. Comprueba que el `h1` ha perdido el formato. Obsérvalo con el inspector del navegador.

Fragmentos

El archivo de la aplicación creado arriba devuelve tan solo un elemento HTML. Si quisiéramos devolver varios, uno tras otro, nos daría error. Una posibilidad es encapsular ambos con un `<div>`, pero eso nos conllevaría añadir ese `div` al código, que incluso podría afectar al código CSS involucrado.

React tiene una forma de solucionarlo, que es con fragmentos, los cuales actúan como un nodo HTML pero luego no se materializan en el código final. El problema es que hay que hacer una importación cada vez que son usados. React permite evitar el import si, en lugar de `<Fragment>` y `</Fragment>`, se usan `<>` y `</>`.

```
Import { Fragment } from 'react';  //Puede quitarse si se usan <> y </>.

export function MiPrimeraApp() {
    return (
       <Fragment>  //Puede reemplarse por: <> junto al cierre
                   //No aparece en la aplicación final.
           <h1>¡¡Mi primera aplicación!!</h1>
           <p>Una aplicación de React</p>
       </Fragment>  //Reemplarse por: </> junto a <>.
                    //No aparece en la aplicación final.
```

```
    );
}
```

Ej. 29: Usa Fragments en `App.jsx` para que vuelva a funcionar el formato en `h1`.

Código JS

Para insertar código JS en la función que devuelve la aplicación, podemos usar llaves.

```
const mensaje = {
    titulo: "¡¡Mi primera aplicación!!",
    texto: "Una aplicación de React",
};

const getTexto = (admiracion) => {
    return admitacion ? mensaje.texto : `¡${nombre}!`;
};

export function MiPrimeraApp() {
    return (
       <>
           <h1> {mensaje.titulo} </h1>
           <p> {getTexto(true)} </p>
       </>
    );
};
```

Se puede insertar booleanos, null, undefined (aunque todos ellos no muestran nada), números, texto, o incluso arrays, pero no puede insertarse un objeto directamente (podríamos usar JSON.stringify para convertir el objeto a cadena y entonces mostrarlo). Siempre que sea posible, debemos poner estos valores y funciones de forma externa a la función componente que exportamos para ser renderizada por React.

Ej. 30: Tras el nodo de párrafo (`<p>`) del ejemplo, añade una tabla de una sola fila que se construya llamando a una función. Dicha función devolverá

un array declarado fuera de la función con el valor `[null, undefined, “”, 0, “hola!”, true]`.

Ej. 31: Cambia el array por `[“El”, “fuego”, “no”, “tiene”, “sombra”]` (u otro que quieras). Haz que la función devuelva una tabla de una sola línea con tantas celdas como valores tenga el array, y cada palabra en una celda.

React Developer Tools

Instala el plugin de React Developer Tool. Habilitará, en la zona de Herramientas del desarrolador (normalmente mostradas con Ctrl-May+I), un par de pestañas que son Componenets y Profiler.

Propiedades

Las propiedades es un canal de comunicación del padre al hijo. Lo normal es realizar desestructuración en las propiedades.

```
export const MiPrimeraApp = (props) {  //(props) -> ({titulo})
    return (
       <>
           <h1> {props.titulo} </h1>  //{props.titlo} -> {titulo}
           <p> {getTexto(true)} </p>
       </>
    );
};
```

Puedes usar la pestaña de Componenets en React Developer Tools para cambiar las props de forma dinámica.

```
export const MiPrimeraApp = ({titulo = "¡Mio primera App!}) {
```

Para pasar valores del componente padre podemos hacer algo como lo siguiente. Para pasar cadenas se haría con comillas, para números y otros tipos con llaves, y para boleanos se puede hacer con llaves o simplemente poniendo el argumento para true o no ponerlo para false:

```
<MiPrimeraApp title="Mi 1ª App" subtitulo={123} subrayado />
```

Existe una librería para forzar que los las propiedades sean de un tipo concreto, de forma que si no lo son se mostraría un warning. Se debe instalar la librería, que se llama Proptypes con lo siguiente.

```
npm install prop-types
```

Para usarla en un componente, hay que importar la librería en él:

```
import PropTypes from’prop-types’
```

Ahora se pueden establecer tipos por defecto. También se pueden definir los valores por defecto con defaultProps.

```
MiPrimeraApp.propTypes = {
    titulo: PropTypes.string.isRequired,
    subtitulo: PropTypes.number.isRequired,
}

MiPrimeraApp.defaultProps = {
    titulo: "titulo por defecto"  //No tiene sentido si titulo es requerido
    codigo: 123  //No definido en los args de MiprimeraApp, pero es válido.
}
```

Cambiar estados con Hooks

```
import { useState } from 'react';
import PropTypes from 'prop-types';

export const CounterApp = ( {valorInicial} ) {

    const [counter, setCounter] = useState( valorInicial );

    const sumaUno = () {
        setCounter( counter + 1 ); //No vale counter++;
        //setCounter( () => counter + 1 ); //Otra forma de hacerlo
    }

    return (
       <>
           <h1> {props.titulo} </h1>
           <p> {getTexto(true)} </p>
           button onClick={ sumaUno }> +1 </button>
       </>
    );
};
```

Ej. 32: Realiza un componente que represente un producto. En los parámetros del padre se pasará un nombre de producto, una descripción corta, un precio, y un descuento (entre 0% y 100%), y el stock existente. La cantidad de productos inicial será cero.

Con dos botones, el usuario podrá modificar la cantidad, que será controlada con un Hook (useState). Esto hará que cada uno de los botones deba llamar a una función que hará uso del Hook. Controla para que el número de productos esté entre 0 y el stock existente.

El precio será calculado cada vez que cambie la cantidad, teniendo en cuenta el descuento.

Aplica CSS al componente. Para ello crea un archivo CSS con el mismo nombre que el componente hijo y, en dicho componente hijo, importa el archivo CSS.

Parte V:
Instalación web

Unidad Didáctica 1:

Instalación de Apache2

1. Instalación

Para instalar el servidor apache, debemos utilizar el gestor de paquetes del sistema. En sistemas Debian, bastaría con el comando:

```
apt install apache2
```

Tras la instalación, el servidor Apache se iniciará automáticamente. Para comprobar que está instalado y ejecutándose adecuadamente, se puede abrir un navegador y buscar la dirección `http://localhost/`. Ello debería mostrar una página web por defecto con el mensaje de que Apache2 está instalado correctamente. Observa que Apache2 es un programa demonio (daemon), es decir, un programa que se ejecuta en segundo plano, de forma que al iniciarse no se lanza ninguna interfaz que interactúe con el usuario.

Ej. 1: Instala el servidor Apache2 en linux. Durante la instalación, debería iniciarse el servidor automáticamente. Comprueba que funciona usando el método descrito arriba.

Documentación

La documentación de Apache2 puede encontrarse en `https://httpd.apache.org/docs/2.4/`

Ej. 2: Mirando la documentación: ¿el módulo mod_cache es nuevo en Apache 2.4 (es decir, no estaba en anteriores versiones)?

Directorios

El servidor web apache utiliza, principalmente, dos directorios:

- Directorio `/etc/apache2`. Almacena la configuración del servidor apache2.
- Directorio `/var/www/html`. Almacena los ficheros web que serán servidos o interpretados por apache2 cuando un navegador u otro cliente conecte con ellos.

Analizar la red

En el navegador Firefox, pulsa `Ctrl+Mayús+I` para abrir la consola de desarrollo web. Allí, se puede ver la red de una conexión con todos sus detalles. Pulsa "Desactivar Caché" y limita la conexión a "DSL" o bien a "Regular 4G/LTE".

Ej. 3: Dirígete a Google y haz una búsqueda cualquiera. Usando "Archivo→Guardar Como", guarda la página con un nombre sin espacios ni símbolos (se guardará un fichero .html y un directorio con el mismo nombre). Mueve dicha página al directorio de páginas web. Usa un navegador para que te muestre la página web que acabas de guardar.

Analiza la red del sistema web y contesta a las siguientes preguntas:

¿Cuantas conexiones tienes en total y qué cantidad de megabytes se han transferido?

Existen algunos archivos tienen un tamaño, pero la cantidad transferida es distinta ¿qué tamaño tienen en el servidor? ¿cómo es posible que la cantidad transferida es menor si el archivo se ha recibido correctamente?

¿Cual es el tipo de archivo que suele poseer mayor tamaño?

Existe un tipo de archivo que en nuestro servidor posee igual valor en tamaño real que en tamaño transferido, pero que en el servidor de Google cambia ¿qué tipo de fichero es? ¿Por qué sucede, a grandes rasgos, que en Google el tamaño transferido sea menor y en nuestro servidor no?

¿Qué servidor está, por tanto, más optimizado?

Algunos ficheros son recibidos cuando se accede a Google, pero luego en nuestro servidor fallan (no son servidos por nuestro servidor). ¿Por qué puede suceder esto?

Mira en las cabeceras de los archivos ¿que versión(es) de HTTP se usa(n)?

2. Estado del servidor

El servidor web (o, dicho de otro modo, el demonio apache2) puede iniciarse o pararse a través de su "programa interfaz", llamado apache2ctl. Los siguientes comandos paran, inician o reinician el servidor web:

```
apache2ctl stop     #Detiene apache2
apache2ctl start    #Inicia apache2
apache2ctl restart  #Reinicia apache2
```

Sin embargo, como veremos luego, se recomienda usar systemctl, en lugar los 3 comandos anteriores.

En todo caso, apache2ctl proporciona un par de funcionalidades adicionales, que no tienen contrapartida en systemctl. La primera de ellas es realizar una comprobación de la configuración sin reiniciar el servidor. Esto es especialmente útil, en ahorro de tiempo, en sistemas que poseen cientos o miles de servidores:

```
apache2ctl configtest  #Comprueba configuración
```

También se puede obtener el estado del servidor. Para ello, éste debe estar ejecutándose (no estar parado), y nuestro sistema debe tener instalado el navegador de texto Lynx, puesto que lo que realmente realiza este comando es usar el navegador Lynx para leer la página `http://localhost/server-status`.

```
apt install lynx       #Instala Lynx
apache2ctl status      #Muestra estado del servidor
apache2ctl fullstatus  #Muestra estado completo
```

Nota: la página `http://localhost/server-status` es creada por el módulo (mod) llamado `status`, el cual se define activo por defecto cuando realizamos la instalación inicial de Apache. Si desactiváramos dicho módulo, estos dos comandos darían error.

Ej. 4: Leyendo el estado del servidor, ¿qué versiones, como mínimo, de HTTP soporta? ¿cuál es el tamaño por petición?

FQDN

Al realizar el comando de configtest, Apache2 nos informará que no puede determinar el Full Qualified Domain Name (FQDN). El FQDN está formado por dos elementos:

- Nombre de dominio (domain name). El nombre de Internet asociado a nuestra organización, como example.com. Es asignado por la ICANN.

- Nombre de host (domain name). Es el nombre de la máquina dentro del dominio, como por ejemplo mail. En principio, es la organización a la que se le asigna el nombre del dominio la que establece los nombres de host.

Combinando ambos, conseguimos el FQDN, como por ejemplo mail.example.com, que identifica a una máquina concreta. Una máquina puede alojar varios FQDN, pero un FQDN no debe ser compartido por varias máquinas.

Ej. 5: Somos una organización a la que han asignado el dominio "iesvirgendelcarmen.com", y queremos asignarle a una máquina el nombre de "practicas" ¿Cuál sería el FQDN de la máquina?

¿Podríamos asignarle también dicho nombre a otra máquina de nuestra red?

¿Podríamos asignarle a esa misma máquina otro nombre, además del que ya tiene?

Fichero envvars

El fichero `/etc/apache2/envvars` define usa serie de aspectos usados por el comando apache2ctl. Básicamente, define el valor que tendrán una serie de variables de entorno. Por ejemplo, en éste fichero se define el valor de la variable de entorno `APACHE_RUN_USER` como `www-data`. Así, de esta forma, cuando un componente de apache haga uso de dicha variable, utilizará el valor definido aquí. En concreto, esta variable define que los ficheros creados por Apache2 dentro del directorio de páginas pertenecerán al usuario llamado `www-data`.

Las variables de entorno rara vez se modifican, salvo que se deseen varias instancias de apache simultáneas, o se necesite una cantidad excepcionalmente grande de descriptores de ficheros.

Ej. 6: Escoge una variable de entorno (solo una de ellas) que esté definida en el fichero `envvars` que no la comentada anteriormente. Describe brevemente qué hace y cómo afecta la variable al comportamiento de Apache2. Busca en Internet si es necesario.

3. Uso de systemctl

Es preferible usar systemctl, en vez de apache2ctl, para iniciar, parar o reiniciar el servidor Apache. La razón de ello es que, de usarse apache2ctl, el sistema base de linux (systemd) puede no quedar informado de los últimos cambios en el estado del servidor, cosa que no ocurres si se usa systemctl.

```
systemctl stop apache2.service     #Detiene apache2
systemctl start apache2.service    #Inicia apache2
systemctl restart apache2.service  #Reinicia apache2
```

Observa que, cada vez que un programa o demonio es ejecutado, el sistema le proporciona un número de proceso disponible (uno que no esté asignado a ningún proceso). Cuando un proceso termina, dicho número vuelve a estar disponible, pero no inmediatamente, para evitar errores.

Los comandos anteriores serán de gran utilidad puesto que, cada vez que cambiemos la configuración de Apache, deberemos reiniciar el servidor para que esa configuración surta efecto.

Además, systemctl también permite mostrar el estado del servidor. Sin embargo, esta información difiere de la proporcionada por apache2ctl. Aquí, se muestran los procesos que usa el demonio en cuestión (apache2), y los comandos que se llaman para parar, iniciar o reiniciar el servidor (realmente, se llama a apache2ctl):

```
systemctl status apache2.service  #Muestra info de sistema
```

Una vez se muestre el estado, puedes pulsar `:q` para volver a la línea de comandos.

Ej. 7: Usa systemctl para ver los números de proceso (PID: Process Identification Number) que Apache2 está usando. Usa el comando `ps -e` para ver todos los procesos que en ese momento se están ejecutando en el sistema ¿Coinciden los números de proceso? ¿Cuańtos procesos usa Apache en tu sistema?

Nota: el comando `grep «texto»` analiza un texto de entrada para luego mostrar solo las líneas que contienen «texto». Se usa mucho, combinado con pipes, para filtrar la salida de otros programas. Por ejemplo, el comando `ps -e | grep firefox` ejecuta en primer lugar `ps -e`, originando un listado de

todos los procesos. Luego, gracias al pipe `|`, ese listado se usa de entrada al comando `grep firefox`, que muestra solo las líneas que contengan el texto `firefox`. Con ello logramos mostrar únicamente los procesos de firefox.

Ej. 8: Reinicia el servidor apache, y luego usa el método que desees para ver los PIDs de Apache ¿Son los mismos que antes? ¿Por qué?

Ej. 9: ¿Qué diferencia hay entre ver el estado (status) con systemctl y con apache2ctl? ¿qué tipos de datos (del sistema, etc.) nos proporciona cada uno?

4. Desinstalación

Para una completa desinstalación, primero debemos parar el servidor, preferiblemente con systemctl:

```
systemctl stop apache2.service     #Detiene apache2
```

Tras ello, debemos desinstalar todos los archivos de apache2, a través de gestor de paquetes del sistema. En sistemas Debian seería:

```
apt purge apache2      #Elimina apache2
apt-get autoremove     #Elimina paquetes que no sean necesarios
```

Si, tras la instalación, los ficheros de configuración permanecieran aún en el sistema (para una posible reinstalación, etc.), podríamos reinstalarlos con:

```
rm -rf /etc/apache2
```

Ej. 10: Crea o copia (si es que no tienes ya) un par (o mas) de archivos cualquiera (algún fichero de texto, una página html sencilla, una copia de la web por defecto, etc.) dentro del directorio `/var/www/html`. Realiza una desinstalación del servidor ¿Qué sucede con los ficheros almacenados en `/var/www/html`?.

Sabiendo que otros paquetes y servidores también podrían usar dicho directorio ¿Por qué crees que se da dicho comportamiento?

Tras ello, vuelve a instalar el servidor.

5. Mime

MIME sirve para indicar de qué tipo es un fichero, de forma que cada fichero tiene un único tipo. Por ejemplo, un fichero de texto será de tipo `text/plain`, y un fichero de música codificado en formato MP3 será de tipo `audio/mpeg` [4].

Como norma general, el MIME de un fichero se calcula mirando extensión. Por ejemplo, al fichero `documento.odt` se le asignará un MIME de `application/vnd.oasis.opendocument.text`. Las extensiones reconocidas por el sistema se definen en `/etc/mime.types`.

Ej. 11: Busca un formato de imagen y apunta su MIME y su extensión. ¿Qué tipo MIME se le asignará a un fichero llamado cuya extensión sea `.ogg`?

Fichero magic

El fichero `/etc/apache2/magic` es usado por el módulo de Apache mime_magic, el cual por defecto está inactivo.) para averiguar el tipo MIME de un fichero. Si se activa este módulo, Apache2 mira primero la extensión para determinar el tipo MIME del archivo. Si no está ahí definido, entonces examina el contenido del fichero usando el fichero magic.

Cuando es llamado el módulo mime_magic, se examinan las líneas del fichero magic, intentando encontrar un parámetro que corresponda. Lo más sencillo es que una línea determine el forma con un valor. Por ejemplo, si a partir del 4 byte, el fichero contiene una cadena igual a moov, entonces el fichero es de tipo `video/quicktime`. (byte significa un byte concreto, belong significa big endian long de 4 bytes, etc.).

```
4      string          moov          video/quicktime
```

También existen la líneas condicionadas, de forma que la primera línea no posee un formato, sino que es seguida por otra o varias líneas con un mayor al inicio. Los siguiente determina que una aplicación es de tipo ichitaro (versión 4) si el fichero empieza por DOC y en el byte posición 43 vale 0x14 o bien en la posición 144 está la cadena JDASH.

[4] RFC 3003: https://tools.ietf.org/html/rfc3003.

```
0       string          DOC
>43     byte            0x14            application/ichitaro4
>144    string          JDASH           application/ichitaro4
```

Si aún así no puede determinarse el MIME del fichero, se le asigna `application/octet-stream`. También es posible cambiar la prioridad o el tipo devuelto si no se puede determinar el tipo.

Ej. 12: Mirando el archivo magic, averigua que MIME calcularía Apache para un fichero que empiece por `#!/bin/perl`.

Ej. 13: ¿Se tiene en cuenta el primer byte para saber si un archivo es de tipo `application/pgp-keys`?

Ej. 14: Un fichero que empiece por DOC podría ser candidato al mime `application/ichitaro4` ¿Qué otra/s condición/es debe cumplir para serlo?

Ej. 15: ¿Qué condiciones tiene que tener un archivo para que sea considerado como `image/x-portable-pixmap`.

Nota: En magic, cuando aparece el término 7bit al final de la línea, indica que el fichero a examinar será leído como si estuviera codificado en formato 7 bit (se ignora el bit más significativo al hacer la comparación). Para este ejercicio ignora este hecho.

Ej. 16: Busca un sistema multimedia en el que, cuando el bit 7º de su cuarto byte sea “1”, entonces se trate de un formato, pero si es “0”, se trata de otro formato.

Nota: El fichero tendrá otros requisitos, ya que lo anterior lo cumplen todos los ficheros.

Ej. 17: Busca otro formato en el que, a partir de la posición 4 tenga la cadena `ftypavc1` o `ftypavc1`.

Unidad Didáctica 2:

Directivas de configuración

1. Ficheros de configuración

El fichero `apache2.conf`[5] es el fichero de configuración maestro de Apache2, y está situado en el directorio `/etc/apache2/`. Es leído por Apache2 al iniciarse (usando `apache2ctl start` o, preferiblemente, a través de systemctl), y regula todo el comportamiento del servidor.

Incluir ficheros

Para evitar que `apache2.conf` sea demasiado grande para ser manejable, y para agrupar las directivas de configuración de forma ordenada, el fichero `apache2.conf` hace uso de dos directivas:

- `Include`. El fichero incluido es obligatorio de forma que si no está disponible o posee algún error de sintaxis, el servidor no podrá iniciarse (se originará un error).
- `IncludeOptional`. Si el fichero incluido no está disponible, simplemente ese componente no se carga, pero no se origina error.

Ej. 18: ¿Cuál es el único fichero que, inicialmente, es incluido de forma que si no está presente da error? ¿cuáles son los ficheros opcionales incluidos?

Inicialmente, los archivos importados en el fichero de configuración son los siguientes:

- Archivo `ports.conf`. Define los puertos en los que escucha el servidor web.
- Archivos .load y .conf del subdirectorio `mods-enabled`. Son los archivos de configuración de los módulos activos. Un módulo de Apache es un componente que añade funcionalidades al servidor web. Algunos de ellos están activos en una configuración por defecto.
- Archivos .conf del subdirectorio `conf-enabled`. Estos archivos definen comportamientos típicos del servidor, como establecer páginas de error personalizadas, establecer una seguridad básica, etc.

[5] En otros sistemas, el fichero `apache2.conf` tiene de nombre `httpd.conf`.

- Archivos .conf del subdirectorio `sites-enabled`. Define los sitios virtuales (Virtual Hosts) del servidor. Permite definir varias webs independientes, todas ellas usando el mismo demonio. Por defecto, solo está configurado el servidor virtual por defecto.

Importante: Cualquier cambio en el fichero `apache2.conf`, o de cualquiera de los ficheros importados, solo es efectivo cuando se reinicia el servidor.

`Ej. 19:` Cambia el fichero de configuración donde se definen los puertos de escucha para que se llame `listen_ports.conf`. Configura el servidor para que escuche en los puertos 80 y 8080 (tendrás que consultar en la documentación sobre el funcionamiento del fichero que controla los puertos de escucha) . Describe las líneas que hay que cambiar y/o añadir, y en qué archivos. Tras ello, revierte los cambios.

Nota: para comprobar que el servidor escucha en un puerto, puedes usar el navegador y acceder a una página existente, añadiendo dos puntos y el puerto a comprobar. Si no se especifica número, el puerto por defecto es 80. De esta forma, las direcciones `http://localhost` y `http://localhost:80` son equivalentes.

Directivas core y no core

Para la configuración de Apache, podemos incluir, en los archivos de configuración, una serie de directivas definidas por Apache, pudiendo diferenciar dos tipos.

Las ***directivas core***, que pueden ser usadas en todos los archivos de configuración, sin necesidad de tener instalado ningún módulo. Cada una de estas directivas tienen una sintaxis concreta, y afecta a un aspecto concreto del servidor.

Inicialmente, en el fichero maestro `apache2.conf` se definen las directivas mostradas más abajo. Establecen el directorio base de ejecución del servidor, configuración del log, así como varios comportamientos al servir los ficheros a un cliente (permite hasta 100 conexiones persistentes con máximo de espera de 5ms, etc.).

```
DefaultRuntimeDir ${APACHE_RUN_DIR}   #Directorio de ejecución
Timeout 300     #Segundos antes de cerrar conexión sin transferencia
KeepAlive On    #Múltiples solicitudes en la conexión
MaxKeepAliveRequests 100              #Solicitudes máx. Por conexión
KeepAliveTimeout 5                    #Tiempo máx. Entre solicitudes.
HostnameLookups Off                   #Convierte IPs en nombres.
ErrorLog ${APACHE_LOG_DIR}/error.log  #Directorio del log
LogLevel warn                         #Nivel de log
PidFile ${APACHE_PID_FILE}            #PID del proceso principal
User ${APACHE_RUN_USER}               #Usuario de los ficheros a servir
Group ${APACHE_RUN_GROUP}             #Grupo de los ficheros a servir
LogFormat "…….                        #Formato del archivo de log
```

La documentación de todas las directivas core puede encontrarse en https://httpd.apache.org/docs/2.4/es/mod/core.html

Ej. 20: ¿Dónde se definen los valores de ${APACHE_RUN_DIR} y ${APACHE_LOG_DIR} y similares?

Ej. 21: Escribe la ruta completa del fichero de log de Apache con la configuración por defecto.

Ej. 22: Busca en la documentación y escribe qué hacen dos de las directivas anteriores que no sean la de `Errorlog`.

Ej. 23: ¿Qué realiza la directiva `DocumentRoot`?

También están las ***directivas definidas por módulos***, los cuales son componentes adicionales que complementan la funcionalidad del servidor con nuevos comportamientos. Las directivas de estos módulos vienen en sus correspondientes ficheros de configuración, dentro de mods-enabled, como por ejemplo, en `dir.conf`:

```
DirectoryIndex index.html index.cgi index.pl index.php index.xhtml index.htm
#Establece los archivos de directorio
```

2. Directivas de contexto

Son directivas core, con una etiqueta de apertura y otra de cierre, al estilo de XML o HTML. Definen un objetivo, que puede ser un directorio, un conjunto de ficheros o una condición, de forma que las directivas en su interior solo son válidas para ese objetivo o si esa dirección se cumple.

- ***Directory***: Define configuraciones específicas para un directorio y todos sus subdirectorios. No permite el uso de caracteres especiales como si sucede en `Files`. Todas las rutas de directorio pasados por Apache terminarán en `/`.

  Ejemplo: `<Directory "/var/www/html/home/">` (la barra final es necesaria).

- ***DirectoryMatch***, afecta a los directorios cuya ruta case con una expresión regular, no incluyendo a los subdirectorios, a menos que éstos también casen con la expresión regular. Todas las rutas de directorio pasados por Apache terminarán en `/`.

  Ejemplo: `<DirectoryMatch "\/tmp\/$">` afecta a todos los directorios llamados tmp, y `<DirectoryMatch "\/tmp\/">` incluye también a los subdirectorios, mientras que `<DirectoryMatch "\/tmp\/.+">` afecta solo a los subdirectorios.

  Observa que `.` incluye el carácter `/`, por lo que deberás usar `[^/]` para indicar cualquier carácter que no tenga barra. Así, por ejemplo, para indicar los directorios con un nombre que empiece por tmp, pero no a sus subdirectorios, hará que poner `<DirectoryMatch "\/tmp[^/]*\/$">`.

- ***Files***: Aplica configuraciones a archivos específicos basándose en nombres exactos o comodines simples: `?` equivale a un carácter cualquiera, * equivale a cualquier número de caracteres. También puedes poner `!` al inicio para negar la condición. Files nunca tendrá que lidiar con el carácter `/`.

  Ejemplo: `<Files "!a*.jpg">`, equivale a todos los archivos que no sean una "a" seguida por cualquier cadena, y con extensión jpg.

- ***FilesMatch***: Afecta a los archivos que coincidan con una expresión regular. FilesMatch nunca tendrá que lidiar con el carácter `/`.

  Ejemplo: `<FilesMatch "\.jpe?g$">`, equivale a todos los archivos que tengan extensión jpg o jpeg.

Apache 2 no gestiona archivos que posean una barra (`/`) en el nombre, a pesar de que hay sistemas de archivos que permiten ficheros con dicha barra en su nombre.

- ***IfModule***: Evalúa si un módulo específico de Apache está habilitado antes de aplicar configuraciones dentro de su bloque. También es posible indicar el código fuente del módulo, que terminará en `.c`. Si se quiere una regla para cuando el módulo NO esté activado, se debe incluir una admiración al inicio del nombre del módulo.

  Ejemplos: `<IfModule mod_dir.c>`, `<IfModule mod_php>` y `<IfModule ! mod_rewrite>`.

- ***Location***: Configura reglas basadas en la URL solicitada, en lugar de la ubicación en el sistema de archivos. `Ejemplo, <Location "/miubicacion">`. Rara vez se anidan las directivas Location/LocationMatch con las directivas Directory/DirectoryMatch, aunque sería posible hacerlo. Esta directiva NO permite el uso de la admiración para negar la condición, ni el uso de comodines (`*` o `?`).

- ***LocationMatch***: Permite afectar a las rutas que correspondan con una archivos que coincidan con una expresión regular. Por ejemplo, para indicar a todas las URLs que empiecen por `/private`. `<LocationMatch "^/private/.*">`.

  Recuerda que todas las rutas empiezan por `/`, y observa que se pueden hacer consideraciones similares a las de DirectoryMatch, de forma que tendrás que elegir bien cuando usas `.`, `[^/]` y la terminación `^`.

- ***Limit*** y ***LimitExcept***: Restringen configuraciones según el método HTTP (GET, POST, PUT, etc.). Ejemplo `<LimitExcept GET POST>`, que se aplica a todas las peticiones que no se hayan realizado con los métodos GET o POST.

Todas estas directivas se declaran, como se ha visto, encerradas por `<` y `>`. Tras las directivas que sea, aparecerá una etiqueta de finalización. Por ejemplo, lo siguiente afecta a todos los ficheros con nombre de 6 caracteres y que terminen en `a.jpg`.

```
<Files "?a.jpg">
    # Las directivas aquí especificadas solo afectan a los archivos
```

```
    # de 6 caracteres que terminen en a.jpg, en cualquier ruta.
</Files>
```

Varias de estas directivas pueden anidarse entre sí:

```
<IfModule mod_dir.c>
  <Directory "/var/www/html/img/">
    <FilesMatch "^[a-zA-Z][0-9]{1,2}\.jpg$">
      # Las directivas aquí especificadas solo afectan a los archivos jpg
      # en "/var/www/html/img/" y sus subdirectorios que empiecen por una
      # letra y sigan por uno o dos números letra, y solo solo si el
      # módulo mod_dir está activo.
    </FilesMatch>
  </Directory>
</IfModule>
```

Ej. 24: Escribe las directivas necesarias para poder afectar a todos los ficheros que terminen en `.es` que estén alojados en `/var/www/html/lang` o en sus subdirectorios.

Ej. 25: Crea una directiva que solo se aplique si el módulo `mod_rewrite` está habilitado y afecte a los métodos DELETE y POST.

Ej. 26: Crea una regla que afecte a todos los ficheros php en los que no se emplee el método GET.

Ej. 27: Haz una regla que afecte a todo los ficheros de las URLs que empiecen por `/src/` seguidas un un número entre 100 y 5000 (sin extensión).

Ejemplo: afectaría a URLS como `/src/120` y `/src/1000`, pero no a `/src/1`, `/src/a100`, `/src/1000/` ni `/src/1000/100`.

Nota: Para el patrón que conforme el número, hay 3 posibilidades: (1) que esté entre 0 y 999, (2) que esté entre 1000 y 4999, o (3) que sea 5000. Puedes contemplar las tres posibilidades usando un or y paréntesis: `( | | )`.

Ej. 28: Haz una regla que afecte a las operaciones POST con una URLs que apunte a los ficheros que estén contenidos en todos los directorios que se

llamen `downloads` (los directorios llamados `downloads` podrán estar en cualquier sitio.

Ej. 29: Haz una regla solo efectiva con el módulo modrewrite no esté activo, que afecte a todos los ficheros en el directorio `/var/www/tmp` (no a ficheros en subdirectorios), y cuyo nombre esté compuesto solo por números (cualquier cantidad de éstos, pero al menos uno) y tenga una extensión de `tmp` o `swap`.

~~**Ejemplo:** Afectaría a operaciones POST en `/user1/download/fich1` o `/download/fich2.js`, pero no a `/download/` o a `/general/download/tmp/` o `/general/download/tmp/fich`.~~

Ej. 30: Escribe una regla, usando Files (no FilesMatch) que corresponda con todos los ficheros de exactamente 5 caracteres, más una extensión cualquiera de exactamente 3 caracteres. Solo será aplicable para el método POST.

Ej. 31: Escribe una directiva que afecte a todos los ficheros de máximo 3 caracteres que estén en el directorio `/var/www/resources`, pero que no afecte a los subdirectorios de éste.

Ej. 32: Escribe una regla que, en caso de no estar activo mod_alias, afecte a todos los subdirectorios directos de `/backups`.

Ejemplo: afectará a `/backups/01/`, pero no a `/backups/01/tmp/`.

Ej. 33: La ruta `/link/` en las URLs apunta a un directorio concreto de nuestro sistema. Escribe una regla que afecte a todos los ficheros, subdirectorios y ficheros en subdirectorios de ese directorio, pero no al propio directorio `/link/`.

Nota: en este caso, asume que, para acceder a esos ficheros y directorios, se hace a través de la URL que empiece por `/link/`).

Ej. 34: Tenemos una configuración especial en la que todos los archivos .es de nuestro sistema con extensión `.es` son servidos con la URL `/lang/«archivo».es` (por ejemplo, `/var/www/html/esp/hi.es` será servido

con la URL `/lang/hi.es`). Crea una regla que afecte a todos esos archivos usando `.es`.

Ej. 35: Crea una única directiva que afecte a cada uno de los siguientes:

a) A todos los ficheros que terminen en `.log`. No uses FilesMatch.

b) A cualquier URL que contenga `/admin/` en cualquier parte del path.

c) Al directorio `/var/www/html/privado/` y todos sus subdirectorios.

d) A todos los ficheros llamados `config.php` o `config.perl`.

e) A URLs que terminen en `/api`.

f) A todos los directorios llamados `backup` y a todos sus respectivos subdirectorios.

g) A todos los directorios dentro de `/opt/logs/` y todos sus respectivos subdirectorios, pero no al propio directorio `/opt/logs/`.

h) A las URLs que empiecen con `/intranet/seguro`.

i) A todos los ficheros con 3 o menos caracteres. No uses FilesMatch.

j) A todas las URLs que apunten a un directorio con nombre `home`.

k) A todas las URLs que apunten a un fichero o directorio con un nombre o alias que tenga extensión `.html`.

l) A cualquier fichero cuyo nombre esté formado por una o más letras inglesas, números y/o guiones bajos, seguido de la extención `.backup`.

m) A URLs que contengan `/debug/` en cualquier parte del path.

n) Al directorio `/etc/apache2/`, pero no a subdirectorios.

ñ) A los subdirectorios directos de `/home/user/`.

o) A las URLs que apunten a un directorio `config` que esté dentro de otro directorio llamado `home` (entre ambos, pueden haber uno o varios directorios).

p) A todos los ficheros formados por uno o más dígitos y extensión `.log`.

q) A los subdirectorios directos de los directorios llamados `secure`, pero no a los propios directorios llamados `secure`.

30: <Limit POST> <Files "?????.???">
31: <DirectoryMatch "^\/var\/www\/resources\/$"> <FilesMatch ".{,3}">
32: <Ifmodule "!mod_alias"> <DirectoryMatch "^\/backups/[^\/]+^\/">
33: <DirectoryMatch "^\/link/.+"> <Files "*.es">
34: <Directory "/lang/"> <Files "*.es">
35:

a) <Files "*.log">
b) <LocationMatch "\/admin\/">
c) <Directory "/var/www/html/privado/">
d) <FilesMatch "config\.(php|perl)">
e) <LocationMatch "\/api$">
f) <DirectoryMatch "\/backup\/"> #Afecta a todas las rutas con direct. backup
g) <DirectoryMatch "^\/opt\/logs\/.+"> # ".*" fuerza que no sea ese direct.
h) <LocationMatch "^\/intranet\/seguro"> #Sin barra final (URL)
i) <Files "!????*"> #Lo que no sea algo con 4 o mas (????*) caracteres.
j) <LocationMatch "/home/$">
k) <LocationMatch "\.html\/?$"> #La "?" es para apuntar a fichero o direct.
l) <FilesMatch "^\w+\.backup$"> #La barra es para que . Sea el caracter "."
m) <LocationMatch "\/debug\/">
n) <DirectoryMatch "^\/etc\/apache2\/$" /> #Directorymatch obligatorio
ñ) <DirectoryMatch "^\/home\/user\/[^\/]+\/" /> #solo direct. directos
o) <LocationMatch "^\/home(\/.*\/|\/)config\/$">
p) <FilesMatch "^\d+\.log$"> #Barra para escapar el punto.
q) <DirectoryMatch "/secure/[^/]+\/$">

3. Directivas de acceso

Las directivas `Require` permiten o deniegan el acceso a recursos, ficheros y directorios. Las directivas `Require` deben estar bajo una de las siguientes:

- Dentro de alguna de las siguientes directivas de contexto Files, FilesMatch, Directory, DirectoryMatch, Location, LocationMatch.
- Dentro de una directiva de `VirtualHost`.
- En un archivo `.htaccess`.
- A nivel global.

Un ejemplo de esta directiva sería:

```
<Directory "/public">
    Require all granted  #Acceso incondicional al directorio /public
</Directory>
```

Existen diversas condiciones para permitir o denegar acceso:

```
Require all granted        #Permite acceso incondicionalmente
Require all denied         #Restringe acceso incondicionalmente
Require ip 192.168.1.100   #Permite acceso a esa IP
Require not ip 10.0.0.0/8  #Permite acceso a esa red
Require user admin         #Permite acceso identificado a ese usuario
```

También pueden combinarse varias de estas directivas entre sí con `RequireAll` o con `RequireAny`:

```
<RequireAll> #Permite acceso al usuario admin que esté en la red indicada
    Require ip 192.168.1.0/24
    Require user admin
</RequireAll>

<RequireAny> #Permite acceso si se accede en la red y/o se está
             #identificado como user1
    Require ip 192.168.1.0/24
    Require user user1
</RequireAll>
```

Ej. 36: Crea las directivas necesarias para que los archivos en `/debug` (pero no sus sbdirectorios) sea `accedidos` solo por el usuario `admin` y el usuario `reviewer`.

Ej. 37: Crea una regla para que `/debug/private` y todos sus subdirectorios solo sean accesibles por `admin`, y solo desde la red con IPs desde 192.168.0.1 hasta 192.168.0.254 (la 0 y la 255 se reservan para la dirección de red y la dirección broadcast).

Ej. 38: Crea una regla para que los ficheros que empiecen por .log (en cualquier directorio), sea solo accesible solo por la ip 10.8.0.1, y solo cuando esté autentificado como admin.

4. Directiva Options

La directiva `Options` establece qué funcionalidades más alla del acceso a archivos y directorios. Al igual que `Require`, puede emplearse en los siguientes contextos:

- Dentro de alguna de las siguientes directivas de contexto Files, FilesMatch, Directory, DirectoryMatch, Location, LocationMatch. La opción Indexes solo es posible en Directory o DirectoryMatch.
- Dentro de una directiva de `VirtualHost`.
- En un archivo `.htaccess`.
- A nivel global.

En los apartados siguientes se detallan los argumentos posibles de `Options`, pudiendo combinar varios de ellos. `Options` habilita la(s) opción(es) indicada(s) y niega el resto, a menos que se use `+` / `-`. Ejemplos de directivas serían:

```
<Directory "/public/">
    Options None  #No permite ninguna funcionalidad adicional
</Directory>

<Directory "/dmz/">
    Options All  #Activa todas las opciones posibles excepto MultiViews.
                 #Desaconsejada por cuestiones de seguridad.
</Directory>

<Directory "/dmz/">
    Options Indexes FollowSymLinks  #Activa dos opciones, negando el resto.
</Directory>

<Directory "/dmz/">
    Options +Indexes -FollowSymLinks  #Deja la configuración existente, pero
                                      #añade indexes y elmina FollowSymLinks
</Directory>
```

Índice de directorios

Con `Options Indexes` se permite mostrar un listado de los archivos de un directorio si no hay un archivo de índice, como index.html o index.php (el

mostrar el archivo de índice tiene preferencia sobre el listado de directorios). El mostrar estos listados puede exponer archivos sensibles.

Enlaces simbólicos

Apache es capaz de usar los enlaces simbólicos del sistema, pero esta opción está desactivada por defecto, a menos que se establezca en apache2.conf o similar (como así sucede para algunos directorios). Para habilitar los enlaces simbólicos en un directorio utilizaremos Directory o DirectoryMatch o en el correspondiente .htaccess, aunque también lo podemos situar a nivel global o a nivel de VirtualHost para que afecte a todo el servidor o a todo el host virtual:

```
<Directory "/hub/dmz">
    Options FollowSymLinks
</Directory>

<DirectoryMatch "/auth/.*/linkToApp/">
    Options SymLinksIfOwnerMatch
</DirectoryMatch>
```

La opción de SymLinksIfOwnerMatch funciona igual que FollowSymLinks, pero solo habilita los enlaces en los que el propietario del enlace es el mismo que el destino del enlace.

Ej. 39: En mi web, todos los archivos han sido copiados o generados con el usuario y el grupo de Apache (en una instalación por defecto de Ubuntu, `www-data:www-data`) ¿tendré problemas si uso `SymLinksIfOwnerMatch`?

Repaso: ¿cómo e definían ese usuario y ese grupo?

Ej. 40: Crea dos directorios dentro de `/var/www/html/`, llamados `original` y `link`. Dentro del directorio original, crea una página web mínima llamada `original.html`. , que muestre, tan solo, el mensaje `Página original`. Crea en directorio link un enlace a esa página, con nombre `link.html`.

Escribe el comando necesario para crear el enlace.

Nota: para comprobar que el enlace está creado correctamente, puedes usar un navegador y acceder a la dirección `http://localhost/link/link.html`. El servidor debería servir la página web mostrando el mensaje `Página original`.

Nota 2: los enlaces simbólicos se crean con el comando `ln -s`.

Ej. 41: Teniendo el enlace creado del ejercicio anterior, establece en apache2.conf (justo tras el cierre de la directiva `<Directory /srv/>...<Directory>`), las directivas necesarias para que en el directorio `/var/www/html/link` no se permitan enlaces. Escribe las directivas creadas.

Nota: para comprobar que no se siguen los enlaces en ese directorio, usa un navegador para acceder a la dirección `http://localhost/link/`. Debería mostrarse el directorio, sin ningún archivo.

Nota 2: ¡Recuerda! Tras cualquier cambio en `apache2.conf` o en los ficheros incluidos por éste, debes reiniciar el servidor para que los cambios surtan efecto.

Multiviews

La opción `Options Multiview` habilita la negociación de contenido basado en el nombre del archivo. Permite que el servidor web sirva automáticamente la variante más adecuada de un recurso solicitado, sin que el cliente especifique la extensión completa del archivo. La selección puede basarse en parámetros como idioma, tipo MIME, codificación o contenido comprimido.

Ej. 42: Habilita la opción Multiviews en un directorio y crea 4 archivos:

example.es.html
example.en.html
example.html.gz

Por un mensaje distinto en cada archivo. Usa about:config, y luego cambia los valores de `network.http.accept-encoding` y `intl.accept_languages`.

Ejcución de archivos

Los archivos HTML pueden incluir etiquetas que indiquen al servidor que deben ejecutar Server Side Includes (SSI), tales como.

```
<!--#include file="filename.html" -->
```

Para permitirlo, las directivas `Options Includes` y `Options IncludesNOEXEC` (esta última es más segura y restringe la ejecución de comandos en el servidor). Actualmente se usan muy poco, en favor de otras tecnologías como React, JS y demás.

Por su parte, `Options ExecCGI` permite ejecutar scripts CGI, aunque estos son proclives a vulnerabilidades y son poco escalables, por lo que actualmente se usan otras tecnologías como PHP y JS.

5. Configuración de directorio

Una función muy conocida de Apache es la utilización de archivos de configuración de directorio, comúnmente llamados archivos `.htaccess`, pues es el nombre por defecto de estos archivos.

La configuración de éstos viene también dada en `apache2.conf`. En concreto, se establece el nombre que toman estos archivos, y se imposibilita que sean vistos directamente:

```
AccessFileName .htaccess   #Nombre de los ficheros de configuración
    #Ejemplo para varios fich. de conf:  AccessFileName .htaccess .ht_

<FilesMatch "^\.ht">
    Require all denied   #Restringe acceso a ficheros que empicen por .ht
</FilesMatch>
```

AllowOverride

Esta directiva es la permite o deniega el uso de archivos de configuración de directorio. Hay que recordar que, en el fichero apache2.conf, se eliminaba la posibilidad del uso de estos archivos en todo el sistema:

```
#Esto es parte del contenido del fichero por defecto de apache2.conf
<Directory "/">
    AllowOverride None
    ...
</Directory>
```

Por tanto, para permitir el uso de archivos `.htaccess`, en un directorio concreto, será necesaria la inclusión, en `apache2.conf` (o en algún fichero incluido por éste), de una directiva `<Directory>` que contenga un `AllowOverride All` o similar.

El funcionamiento de estos ficheros es sencillo. Cuando un directorio contiene un fichero de este tipo (si se permiten este tipo de ficheros en tal directorio), las directivas contenidas en él se aplican a ese mismo directorio, como si estuvieran escritas en `apache2.conf`.

Por ejemplo, si quisiramos habilitar índices en el directorio `/var/www/html/dir`, y sus subdirectorios, tendríamos la opción de permitirlos usando una directiva

<Directory> en `apache2.conf`, o bien podríamos permitir el uso de `.htaccess` en dicho directorio, y allí incluir dicha directiva.:

(1) Permitir índices directamente

apache2.conf

```
<Directory /var/www/html/dir>
       Options Indexes
</Directory>
```

(2) Permitir el uso .htaccess y, allí, permitir los índices.

apache2.conf

```
<Directory /var/www/html/dir>
       AllowOverride All
</Directory>
```

/var/www/html/dir/.htaccess

```
Options Indexes
```

Las directivas de un directorio sobreescriben, en caso de solaparse, las directivas de directorios padres o anteriores. Todas estas directivas sobreescriben a las del fichero de configuración de Apache. En todo caso, la directiva AllowOverride es ignorada si es encontrada en un archivo de configuración de acceso.

La documentación para estos archivos puede encontrarse en https://httpd.apache.org/docs/2.4/howto/htaccess.html.

Ej. 43: Establece que los archivos de configuración se van a llamar `.htac`, en vez de `.htaccess`.

Crea un directorio llamado `sobreescrtura` que cuelgue de la raíz de archivos que sirve el servidor web, en el archivo de configuración de Apache, establece que, en ese directorio y sus subdirectorios, se permita los archivos de configuración de directorio.

Emplea el navegador para acceder a ese directorio (la URL a poner terminará en /), y verás que se mostrará el listado del directorio, aunque este listado mostrará que existen archivos. Dirígete al “Parent Directory” y explica qué sucede y por qué.

Crea un archivo `.htac` con el contenido `Options FollowSymLinks`. Vuelve a cargar el directorio en en navegador y explica por qué no puede verse el directorio.

Ej. 44: Busca en la documentación de Apache el cómo se sobreescriben las directivas de directorios entre si y con el archivo de configuración de Apache2.

Ej. 45: Crea el subdirectorio `lista` dentro del directorio `sobreescrtura`, y añade un archivo `.htac`, en el directorio `lista`, en donde haya una directiva `Options +Indexes`. Prueba a acceder al directorio `/sobreescritura/lista` en el navegador.

Ej. 46: Modifica la configuración para que se permitan archivos `.htaccess` (además de los `.htac`). Crea un enlace simbólico hacia `lista`, llamado `list`. Comprueba que funciona en enlace.

Crea otro enlace dentro de `lista`, llamado `sobre-esc`, hacia el directorio padre. Comprueba si funciona o no.

6. Configuración inicial

Inicialmente, en el fichero de configuración maestro se establecen, inicialmente, una serie de restricciones a los directorios del sistema. Ello se hace a través de varias directivas <Directory>. Por ejemplo, la primera que nos encontramos es:

```
<Directory "/">               #Hdirectorio del sistema, no web
       Options FollowSymLinks  #Hab. enlaces simb.
       AllowOverride None      #No permite .httpaccess
       Require all denied      #Restringe acceso a todos
</Directory>
```

En ella, se establece que, en el directorio `/` y todos sus subdirectorios (es decir, en todo el sistema), se establece que (1) se permiten los enlaces simbólicos (2) no se permite el uso de .httpaccess, y (3) nadie tiene acceso a esos archivos. Esta última directiva no es del core, sino del módulo mod_authz_core, habilitado por defecto.

Más abajo, se muestran directivas que permiten el acceso (`Require all granted`) a los siguientes directorios:

- `/usr/share`. Esto es necesario para que los módulos accedan a diversos programas del sistema, como programas para ejecutar scripts.
- `/var/www/`. Hemos de dar acceso a este directorio para que puedan servirse los ficheros que hay en él. Observa que el directorio base del servidor es `/var/www/html`. Esto es así para permitir acceso a ficheros de forma indirecta. Por ejemplo, era muy típico situar ficheros de script en `/var/www/cgi`, de forma que no se pudieran cargar directamente por un navegador, pero si pudieran ser llamados por páginas situadas bajo `/var/www/html`, que si podían ser solicitadas por el navegador.
- `/srv/`. Aquí se sitúan los diversos servidores virtuales.

Ej. 47: Con la configuración inicial de `apache2.conf`, ¿está permitido el acceso al directorio `/srv/`) ¿Por qué?

Ej. 48: Haz que `/var/www/host/` y sus subdirectorios sigan los enlaces simbólicos (FollowSymLynks). Dentro de `/var/www/host/`, todos los directorios list, pero no en sus subdirectorios, solo se permitirá hacer listado de directorio (Indexes), no enlaces u otra cosa, y los ficheros que empiecen por punto serán inaccesibles (require all denied) ¿Qué mods deberán estar activos?

Ej. 49: Haz que el directorio /var/www/html tenga las opciones activas de seguir enlaces y listado de directorio, exclusivamente. Haz que todos los directorios que cuelguen de dicho directorio no puedan seguir enlaces.

Ej. 50: En `/var/www/host/home`, pero no en sus subdirectorios, permitiremos .htaccess (AllowOverride All). En esos .htaccess, creamos una configuración para que ese directorio y sus subdirectorios solo sean accesibles por el usuario correspondiente (`/var/www/host/home/sam` requerirá el usuario sam). Estos .htaccess no serán accesibles (require all denied).

Ej. 51: Haz que las URLs que empiecen por `/user/` sean accesibles. Sin embargo, si, dentro de `/user/`, se accede a ficheros que empiecen por punto, solo serán accesibles desde localhost o desde `192.168.10.0/24`.

Ej. 52: En los directorios home y todos sus subdirectorios, debemos establecer que solo pueden establecerse las opciones de Multiviews y de seguir los enlaces que sean de la misma propiedad. Luego, en los subdirectorios directos de todos los directorios llamados `home`, deseamos asegurarnos que no se pueden listar los directorios.

Ej. 53: Haz que los subdirectorios directos de todos los directorios llamados `home`, deseamos asegurarnos que no se pueden listar los directorios.

Unidad Didáctica 3:

Gestión de módulos y configuraciones

1. Configuración y activación

Un servidor web es, básicamente, un programa que escucha en una serie de puertos y espera que un navegador, u otro tipo de cliente, establezca una conexión en alguno de esos puertos. Entonces, usando el protocolo http o https, el cliente solicita una ruta web, y el servidor le transmite el fichero que corresponda con esa ruta. Si ese fichero no está, o la ruta no corresponde a un fichero (corresponde, por ejemplo, a un directorio), un servidor web básico enviará un error[6].

Un módulo es un componente que añade funcionalidades al servidor web. Apache viene con varios de ellos activados por defecto.

Por su parte, una configuración es, simplemente, un conjunto de directivas, normalmente destinadas a un mismo fin (mejorar la seguridad, añadir una funcionalidad o un conjunto de funcionalidades asociadas entre si, etc.).

Directorio

Por defecto, los módulos y las configuraciones activos en un momento dado se encuentran en el subdirectorio `mods-enabled`, y `conf-enabled`, dentro del directorio de configuración de Apache (en definitiva, en `/etc/apache2/mods-enabled` y `/etc/apache2/conf-enabled`).

Recuerda que, en el fichero de confguración maestro de Apache2 (llamado `apache2.conf` en sistemas Debian), se realiza un `IncludeOptional` de todos ficheros `.conf` y `.load` que se hallen en el directorio de mods, y todos los `.conf` del directorio de configuraciones:

```
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
IncludeOptional conf-enabled/*.conf
```

En el directorio `/etc/apache2/mods-enabled/` podemos ver los módulos activos en un momento dado, mientras que en el directorio `/etc/apache2/mods-available/`, están todos los módulos disponibles. Así, un módulo activo estará en ambos directorios, mientras que uno no activo pero disponible (instalado),

6 Realmente, Apache2 recién instalado, no devuelve error si se solicita un directorio, puesto que el módulo mod_autoindex viene activado por defecto.

estará solo en el `mods-available`. Realmente, el contenido de `mods-enabled` son enlaces al directorio `mods-available`.

De forma similar, `conf-enabled` tendrá las configuraciones habilitadas, mientras que `conf-available` tendrá las disponibles. El primero de éstos también se trataría de enlaces al segundo.

Activación

La activación de un módulo en apache2 se realiza a través del comando `a2enmod` seguido por el nombre del módulo (sin el prefijo `mod_`). De forma similar se realiza con las configuraciones con `a2enconf`:

```
a2enmod «nombre_del_módulo»
a2enconf «nombre_de_la_configuración»
```

Este comando crea los enlaces en `mods-enabled` que apuntarán a `mods-available`. Observa que, si el módulo no está instalado, no podrá ser activado. De igual forma funcionarán las configuraciones.

Desactivación

La desactivación de un módulo se realiza a través de `a2dismod` seguido por el nombre del módulo (sin el prefijo `mod_`). Para configuraciones se realiza con `a2disconf`:

```
a2dismod «nombre_del_módulo»
a2disconf «nombre_de_la_configuración»
```

Esto eliminará los enlaces del módulo en `mods-enabled`, dejando intactos los ficheros de `mods-available`, y de igual forma con las configuraciones.

Ej. 54: Desactiva el módulo `status`. Escribe el comandos necesario para ello. Indica qué sucede ahora cuando abres la localización `/server-status`. ¿Funciona el comando `apache2ctl status`?

Ej. 55: Activa el módulo `info` ¿Qué cambios se realizan en el directorio `mods-enabled`? ¿Qué localización web se activa y qué información proporciona?

Ej. 56: En una instalación por defecto de apache ¿Está el módulo `ldap` disponible para ser activado? ¿Y el módulo php? ¿Cómo lo sabemos?

2. Módulo mod_alias

Este módulo permite asociar una localización web concreta con un fichero o directorio que se encuentre en otro lugar. Ello se hace con la directiva Alias:

```
Alias «Localización Web» «Ruta de Destino»
```

Por ejemplo, supongamos un servidor web tan solo contiene, en el directorio `/var/www/html` el fichero por defecto `index.html`. Podríamos hacer que, cuando un navegador u otro cliente pidiera la localización web `home.html`, se le sirviera ese fichero `index.html`, en vez de dar error. Ello se realizaría de la siguiente forma:

```
Alias /home.html "/var/www/html/index.html"
```

La directiva `Alias` también funciona cuando «Ruta de Destino» está fuera del directorio raíz de documentos (`/var/www/html`), siempre que se tenga acceso a dicha ruta (las restricciones de acceso iniciales permiten acceso a `/var/www` y `/usr/share`).

También es posible que, «Ruta de Destino» sea un directorio. En este caso, todo bajo ese directorio sería afectado por un alias, como un directorio más.

Redirect

Esta directiva le informa al cliente que debe pedir la dirección web indicada. Por ejemplo:

```
Redirect /example “www.example.com/example”
```

Cuando un navegador solicita la ruta `/example` en nuestro servidor, éste le informará que lo siguiente que deberá hacer será abrir una conexión con `www.example.com/example`. Se trata de una redirección, por lo que la conexión se cierra sin más resultado y el navegador ya hará una nueva petición a esa nueva dirección.

Ej. 1: Crea, en el sistema de ficheros anfitrión, el directorio `/var/www/recursos`. Y dentro, otro directorio llamado `iconos`. Copia el logo

de ubuntu (`/usr/share/apache2/icons/ubuntu-logo.png`) a ese último directorio.

Modifica alias.conf para establecer la localización web `/iconos` hacia ese directorio. Comprueba, poniendo en un cliente (navegador, etc.) la localización `/iconos/ubuntu-logo.png`, y que aparezca la imagen.

Escribe las directivas incluidas en el archivo `alias.conf`.

Ej. 2: Crea el directorio `/var/www/recursos/error/`. Dentro, crea un fichero web html5 mínimo, de nombre `error404.html` que tan solo muestre el siguiente mensaje: `404. No encontrado`.

Configura `alias.conf` para que, cuando un cliente pida la localización web `/no_encontrado.html`, se le sirva el archivo antes especificado.

Escribe las directivas incluidas en el archivo `alias.conf`.

Ej. 3: Con la configuración actual, ¿es posible acceder a través del navegador al contenido del directorio `/var/www/recursos` ¿Por qué?

Ej. 4: Establece una directiva, en `alias.conf`, para que la localización web `/google` redirija a `www.google.com`. Estribe la directiva creada.

3. Módulo mod_autoindex

Estando éste módulo activo, cuando un navegador u otro cliente solicitan una localización que corresponde a un directorio, en vez de enviar un error, se sirve una web que muestra el contenido del directorio. Dicha página web servida no existe realmente, sino que se crea en ese momento, sin grabarse en ningún lado.

El fichero de configuración `autoindex.conf` define varios aspectos de cómo se construye esta web.

Iconos

La directiva `AddIcon` define la imagen a usar para fichero de una o más extensiones, mientras que `AddIconByType` define el icono para los archivos de un tipo MIME concreto. Además, existe la directiva `DefaultIcon`, que define el icono a usar cuando ninguna de estas directivas es aplicable al fichero en cuestión:

```
AddIconByType (TXT,/icons/text.gif) text/*
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
DefaultIcon /icons/unknown.gif
```

Cabecera y pie

Es posible definir la cabecera y el pie que se muestran en la menciona web. Con las siguientes directivas, el servidor busca el fichero los ficheros con ese nombre. En este caso, si encuentra el fichero `HEADER.html`, sustituye la cabecera por defecto (Index of «dir»), y si encentra el fichero `README.html`, sustituye la nota al pie por defecto.

```
        ReadmeName README.html
        HeaderName HEADER.html
```

Estos ficheros son relativos al directorio que se está mostrando. Si poseen una barra al principio, se mostrará un fichero relativo a la raíz de documentos.

Ej. 5: Bájate de Internet tres iconos relativos a carpeta (folder) y texto (text) y página web (layout), cuyo tamaño sea alrededor de 20x20. Copialos en el directorio `/var/www/recursos/iconos`, creado en anteriores ejercicios.

Modificando `autoindex.conf`, cambia el aspecto de los textos, los ficheros web y las carpetas producidos por autoindex para que aparezcan los iconos de ese directorio.

Crea un directorio llamado `dir` en la raíz de documentos (en definitiva, `/var/www/html/dir`). Comprueba que se ven esos documentos usando ese directorio: ¿qué dirección web tendrás que usar?

Escribe también las directivas modificadas en `autoindex.conf`.

Ej. 6: Modificando `autoindex.conf`, establece que el nombre del fichero de cabecera sea `_heading.html`. Luego, crea un fichero para que el directorio dir antes creado tenga una cabecera de la forma: `Contenido del directorio:`. Escribe el cambio hecho en `autoindex.conf`.

Ej. 7: Modifica `autoindex.conf` para que todas las páginas de directorio muestren, bajo el contenido del directorio (es decir, al pie de página) un enlace (<a href …) con el texto <home> que dirija a la página de inicio.

Escribe el cambio hecho en `autoindex.conf`.

4. Módulo mod_deflate

Éste módulo establece los tipos de fichero que serán susceptibles de ser comprimidos antes de enviarse al cliente. El protocolo HTTP permite este proceso como medida de aumentar la eficiencia de la red.

Normalmente, los ficheros que mejor se comprimen son los ficheros que están codificados con texto como, por ejemplo, los archivos web (html, css, etc.), los archivos de texto (txt), los archivos de código fuente (javascript, de otros lenguajes, etc.) o los archivos XML.

Sin embargo, otros ficheros que, de forma natural, vienen ya codificados, poseen malos ratios de compresión, y son especialmente poco indicados de incluirse aquí. Estos archivos son, por ejemplo, los archivos multimedia o los archivos comprimidos.

Filtro con DEFLATE

Por ejemplo, la directiva siguiente indica que los archivos de tipo application/xml serán manejados con DEFLATE.

```
AddOutputFilterByType DEFLATE application/xml
```

Ej. 8: Abre el archivo deflate.conf y observa los tipos MIME que el servidor comprime. Menciona otro tipo de ficheros que sería bueno de incluirse aquí, y crea una directiva para usarla

Ej. 9: Comprueba, mirando la configuración, si los archivos HTML son comprimidos. Comprueba en el navegador que es así. Comprueba si google realiza la compresión de los HTMLs.

Nota: para comprobar que se ha usado compresión, puedes ver si la cantidad de bytes trasferidos es menor que el tamaño del archivo.

5. Módulo mod_dir

Éste módulo hace que, cuando un navegador u otro cliente solicitan un recurso que resulta ser un directorio, se sirva en su lugar un fichero contenido en el directorio solicitado.

El nombre del fichero a servir se define en `dir.conf`. Inicialmente, éste fichero solo contiene una directiva `DirectoryIndex` (definida por el propio módulo mod_dir), y describe el orden de los ficheros que se busca cuando el cliente pide una localización que es un directorio:

```
DirectoryIndex index.html index.cgi index.pl index.php
               index.xhtml index.htm
```

En este caso, el servidor Apache, si se le pide una localización web como `/dir`, y resulta que esa localización es un directorio, entonces busca en `/var/www/html/dir/index.html`. Si no encuentra ese fichero, lo intenta con el `cgi`, el `pl`, el `php`, el `xhtml` y, por último, el `html`. Si ninguno de ellos está disponible, es entonces cuando se envía un error o, si el módulo mod_autoindex está activo, se pasa el control a dicho módulo.º

Ej. 10: Usa la directiva `Directory` para que cuando un cliente (un navegador web, etc.) solicite el directorio `/dir` o cualquiera de sus subdirectorios, se muestre por defecto la página `dir.html` contenida en el directorio solicitado (en vez de `index.html`). Si `dir.html` no estuviera disponible, el servidor deberá devolver `dir.php` y, si ésta tampoco estuviera disponible, devolverá `dir.htm`.

Escribe las directivas creadas para ello.

Ej. 11: En `/var/www/html/dir`, crea un directorio llamado `subdir` que contenga el fichero `dir.html`. Dicho fichero contendrá un fichero html5 con el mensaje `Esto es dir.html`.

Teniendo en cuenta la directiva del ejercicio anterior, ¿qué resulta si un navegador solicita la ruta `/dir/subdir/` (la cual es un directorio)? ¿Por qué?

Ej. 12: Crea otro fichero, también en el directorio `subdir`, llamado `dir.htm`, y que contenga el mensaje `Y esto es dir.htm`.

Teniendo en cuenta la configuración de los ejercicios anteriores, ¿que se mostrará si un navegador solicita ahora la ruta `/dir/subdir/`? ¿Por qué no se muestra el contenido del otro fichero?

Ej. 13: Con la configuración inicial, en la que tanto mod_autoindex como mod_dir están activos, y la configuración de mod_dir es la inicial.

¿Es posible que el servidor nos sirva una web que sea creada usando mod_autoindex en la que dentro exista un archivo con nombre index.html?

6. Módulo mod_mime

Determina varios aspectos de un fichero, que son:

- Tipo de contenido (Content-Type): Define el tipo MIME del archivo (por ejemplo, text/html, image/png, application/json).
- Codificación (Content-Encoding): Especifica si el archivo está comprimido o usa una codificación especial (por ejemplo, gzip).
- Idioma (Content-Language): Indica el idioma del archivo (por ejemplo, en, es).
- Charset (Charset): Define la codificación de caracteres, como UTF-8 o ISO-8859-1.

Este módulo también gestiona ciertos handlers para archivos ejecutables, es decir para CGI y SSI.

MIME

Respecto a la gestión de mimes, la directiva `TypesConfig` apuntará al fichero tipos mimes del sistema. También podemos usar `AddType` para añadir un mime, o sobreescribirlo si éste ya se encontraba en el fichero de mimes del sistema:

```
TypesConfig /etc/mime.types
#AddType application/x-gzip .tgz
```

Ej. 14: Ve al fichero de tipos mime del sistema y mira a qué mime están asignados los ficheros .tgz. Crea un archivo tgz cualquiera y descargalo con un navegador. Observa la cabecera y el mime declarado en la cabecera de respuesta ¿cuál sería en caso de que `#AddType application/x-gzip .tgz` no estuviera comentado?

Ej. 15: Indica qué módulo básico (de los vistos anteriormente) que hace uso de los tipos mime para realizar alguna función específica.

Codificación

Mime también permite identificar codificaciones. Con las siguientes directivas, cuando un cliente solicite un archivo .Z, Apache añadirá `Content-Encoding: x-compress` en la cabecera de la respuesta. Este permite, a navegadores modernos, descomprimir el archivo al vuelo y usarlo/mostrarlo directamente.

```
AddEncoding x-compress .Z
AddType application/x-compress .Z
```

Esta información puede usarse por otras directivas o por otros módulos para realizar ciertas operaciones según el tipo de fichero (con AddOutputFilter, AddHandler, etc.).

Idioma

En el fichero de configuración de mod_mime también se definen una serie de idioma.

pagina.es.html para español.

```
AddLanguage es .es
# DefaultLanguage nl
```

Con el anterior `AddLanguaje`, el sistema podrá identificar, por ejemplo, una página como `pagina.es.html` con el idioma español, de forma que añadirá, a la cabecera de respuesta `Content-Language: es`. Este comportamiento NO está limitado a páginas web, también es posible aplicarlo a imágenes, archivos comprimidos, scripts o todo tipo de ficheros.

Observa que este comportamiento está relacionado con la opción `Multiviews`, No es estrictamente necesario que esté activa `Multiviews` para que esta funcionalidad funcione.

También es posible establecer un lenguaje por defecto (en caso de que no se detecte ningún lenguaje conocido) pero, tal y como afirma la propia configuración, es mejor no especificar lenguaje que hacerlo de forma incorrecta.

Ej. 16: ¿Qué sucede si no está activa `Multiviews`, pero si lo está esta funcionalidad de `AddLanguage`?

Ej. 17: ¿Tiene sentido crear imágenes distintas según el idioma en una web?

Nota: puedes mirar webs como `https://www.peppercarrot.com/` para meditar tu respuesta.

Charset

A veces, un fichero está codificado de cierta manera, aunque hoy día se usa casi siempre UTF-8. Con la directiva `AddCharset`, hacemos que los archivos con una extensión final sean devueltos con una cabecera que incluirá un `Content-Type`. Por ejemplo:

```
AddCharset ISO-8859-1   .iso8859-1  .latin1
```

También es posible combinarlo con el idioma, aunque la extensión de charset debe quedar al final. Así, por ejemplo, un archivo `pagina.html.es.iso8859-1` tendrá, en su cabecera lo siguiente: `Content-Type: text/plain; charset=iso-8859-1`.

Ej. 18: Define el Conten-Type resultante para un archivo llamado hola.html.es.utf-7?

Ej. 19: ¿Qué codificación tendrá un archivo que sea enviado con la cabecera `Content-Type: text/html; charset=cp-1251`?

7. Módulo mod_reqtimeout

Este módulo establece límites de tiempo para la conexión del cliente. El fichero de configuración reqtimeout.conf contiene:

```
RequestReadTimeout header=20-40,minrate=500
RequestReadTimeout body=10,minrate=500
```

Con la primera directiva, el cliente tiene 20 segundos para transmitir las cabeceras y, por cada 500 bytes enviados, dispone de un segundo adicional. En todo caso, si tarda más de 40 segundos, el servidor termina la conexión.

La segunda directiva tiene el mismo funcionamiento, solo que respecto a la recepción del cuerpo de la transferencia (el fichero que se transmite en sí). En este caso, solo se establece un límite de 10 segundos, con un incremento de 1 segundo por cada 500 bytes transmitidos.

Ej. 20: Escribe una directiva para establecer que el cuerpo de la conexión tiene 8 segundos, como máximo 30, y se dispone de 1 segundo más por cada 750 bytes recibidos.

8. Módulo mod_status

Este módulo hace que el servidor, al ser preguntado por una localización web previamente definida (inicialmente, `/server-status`), sirva una web que muestra el estado del servidor.

Entre otras directivas que añaden más información (proxy, extended status), el corazón del fichero de configuración `status.conf` es:

```
<Location /server-status>
       SetHandler server-status
       Require local
       #Require ip 192.0.2.0/24
</Location>
```

Básicamente, define que la localización web `/server-status` es gestionada por el Handler server-status (que es el que crea la página de estado). Dicha localización solo es accesible por un usuario local, gracias a la directiva `Require local`.

Ej. 21: Con ifconfig, averigua la IP del interfaz de red del ordenador/máquina virual (por ejemplo, 192.168.0.1). Escribe qué haría falta añadir, dentro de Location, para que la página de estado sea accesible por esa red. Escribe qué haría falta para permitir el acceso a la IP 93.184.216.34 (y solo a esa IP).

Ej. 22: Escribe qué haría falta cambiar para que la página web de estado sea servida en la localización web `/status`, en vez de `/server-status`.

Ej. 23: Escribe las directivas necesarias para que la página web de estado sea servida en /status y también en /server-status.

9. Configuración Security

Gestiona opciones de seguridad en el servidor. El contenido por defecto es el siguiente:

```
ServerTokens OS  #En los encabezados de respuesta HTTP, muestra solo el
                 # nombre del servidor, su versión y el sistema operativo.
ServerSignature On  #En las páginas de error o respuestas generadas
                    # automáticamente, incluye la firma del servidor.  Ej.:
                    # Apache/2.4.41 (Ubuntu) Server at example.com Port 80
TraceEnable Off  #Desactiva el registro de las llamadas al sistema, usado
                 # para analizar errores o falta de rendimiento.
#RedirectMatch 404 /\.git  #redirige los .git a error.
#Header set X-Content-Type-Options: "nosniff"
    #Indica los navegadores para que no procesen un archivo que tenga un
    # tipo de contenido distinto al especificado en el content-type.
#Header set Content-Security-Policy "frame-ancestors 'self';"
    #Solo permite que el contenido de la web sea incrustado en un iframe o
    # frame si la página que lo incluye es del mismo dominio.
```

10. Módulos y configs por defecto

Por defecto, Apache2 viene con varios módulos y configuraciones activados por defecto. Inicialmente, los módulos y configuraciones activos, además de los ya comentados son:

- Módulo access_compat. Aumenta el rendimiento del servidor, combinando y compactando las directivas de acceso (elimina redundantes, o combina varias reglas en una, etc).
- Módulos authX. Se usa para autentificar los usuarios debidamente.
- Módulo env. Permite controlar variables de entorno internas para pasarlas a scripts.
- Módulo Filter. Permite que ciertos tipos de ficheros sean procesados de otra forma. Introduce, entre otras, la directiva AddOutputFilterByType. Es usado por otros módulos.
- Módulo mpm_event. Permite crear hilos para gestionar las peticiones, liberando los procesos principales.
- Módulo negotiation. El servidor escoge uno entre varios elementos de recursos, basándose en las preferencias del cliente. Por ejemplo, escoger un recurso u otro según el idioma.
- Módulo setenvif: Habilita variables de entorno según la petición del cliente.

- En cuanto a configuraciones, están charset.conf, other-vhosts-access-log.conf. serve-cgi-bin.conf y localized-error-pages.conf.

11. Instalación de módulos

Se realiza a través de los comandos típicos de instalación del sistema operativo. En Ubuntu, por ejemplo, sería el comando apt. Tras instalar el paquete concreto, debe de aparecer en mods-available, pudiéndose activar como cualquier otro módulo.

Algunos módulos necesitarán programas adicionales, normalmente también instalables según el mismo método.

Ej. 24: Usando apt-cache, busca un módulo para apache2 y dí cual el.

Ej. 25: Por defecto, el módulo php no está disponible para apache2. Busca la forma de instalar dicho módulo y cómo activarlo. Documenta brevemente los pasos realizados.

Nota: el módulo php necesita conectar con el programa php, que es posible que deba ser instalado también. Se recomienda el uso de, como mínimo, la versión php 8.1 o, a ser posible, la 8.3 (que posee grandes mejoras) o posterior.

Unidad Didáctica 4:

Host virtuales

1. Directiva VirtualHost

Hasta ahora, hemos visto que Apache2 es un proceso que escucha por una serie de puertos establecidos en el fichero `ports.conf` (por defecto el 80). Cuando cualquier máquina o proceso abre una conexión a cualquiera de esos puertos, y pide (usando el protocolo http o https) un recurso, como por ejemplo, `www.example.com/dir/web.html`, el servidor apache se queda con la segunda parte, en este caso `/dir/web.html`. Apache busca y sirve ese documento o, si no lo encuentra, envía un código de error. En un principio, parece ser que el servidor apache ignora la primera parte (`www.example.com`).

Recuerda que el fichero de configuración apahe2.conf realizaba un IncludeOptional a todos los ficheros .conf de sites-enabled:

```
IncludeOptional sites-enabled/*.conf
```

El contenido del fichero es el siguiente (se omiten, por brevedad, algunos comentarios del fichero):

```
<VirtualHost *:80>
	#ServerName www.example.com

	ServerAdmin webmaster@localhost
	DocumentRoot /var/www/html

	ErrorLog ${APACHE_LOG_DIR}/error.log
	CustomLog ${APACHE_LOG_DIR}/access.log combined

	#Include conf-available/serve-cgi-bin.conf
</VirtualHost>
```

Este fichero define, con la directiva `DocumentRoot`, que se usará como raíz el directorio de sistema `/var/www/html`.

`Ej. 1:` Modifica el fichero 000-default.conf para que sirva los ficheros contenidos en /var/www/apache. Abre un navegador y por la dirección `http://localhost/` ¿Qué sucede? ¿Por qué?

Nota: recuerda que, todo cambio en apache2.conf y/o sus ficheros incluidos, necesita reiniciar el servidor para que dicho cambio se active.

Ej. 2: Teniendo en cuenta la configuración anterior, crea los directorios y ficheros necesarios para que `http://localhost/` sirva una página web básica donde, simplemente, se muestre el mensaje `Funciona` (u otro mensaje que prefieras) ¿Qué ficheros y directorios son necesarios?

Habilitar y desabilitar hosts virtuales

De forma similar a como sucedía con los módulos, también existe un directorio sites-available, que define los "sites" disponibles en el servidor apache. Podemos escribir nuevos ficheros en tal directorio. Para habilitar un fichero de configuración de "sites", se usa el comando:

```
a2ensite «nombre_del_site»
```

Con ello, veremos que aparece, automáticamente, un nuevo enlace simbólico en `sites-enabled` (dirigido al correspondiente fichero de `sites-available`). Esto en similar a los que sucede con los módulos y las configuraciones.

Para desactivar un site, usamos:

```
a2dissite «nombre_del_site»
```

Tras lo cual, desaparecerá el enlace en `sites-enabled`, permaneciendo en `sites-available`.

Ej. 3: Copia el fichero 000-default.conf, situado en sites-available, a otro que se llame 001-init.conf. Habilita 001-init.conf y desabilita 000-default.conf. Escribe los comandos de terminal usados para ello.

2. Parámetros de Virtualhost

La directiva VirtualHost tiene un parámetro que indica las ips y los puertos a los que se aplica. En caso de que varias directivas de virtualhost sean factibles, se mira primero la directiva Servername y, si no coincide ninguna, la directiva ServerAlias. Si ninguna coincide, se selecciona la directiva VirtualHost más específica o, si no es posible seleccionar ninguna por.

Hosts por ServerName

El servidor Apache2 es capaz de servir múltiples dominios. Ello se realiza con la directiva `ServerName`. Por ejemplo, en la configuración siguiente, se sirven dos dominios: (1) www.example.com, cuya raiz de ficheros está en `/var/www/ex`, y (2) www.example2.com, cuya raiz de ficheros está en `/var/www/other`.

```
<VirtualHost *:80>
       ServerName www.example.com
       DocumentRoot /var/www/ex
</VirtualHost>
```

```
<VirtualHost *:80>
       ServerName www.example2.org
       DocumentRoot /var/www/other
</VirtualHost>
```

De esta forma, cuando una navegador, u otro cliente, abre una conexión al servidor en el puerto 80 y, pide www.example.com/index.html, se le proporcionará el contenido de `/var/www/ex/index.hml`. Por otro lado, si pidiera www.example2.org/index.html, se le proporcionará el contenido de `/var/www/other/index.html`.

Si se pide cualquier otro dominio, entonces el servidor apache utilizará la primera configuración que encuentro (en este caso, la de `www.example.com`.

Ej. 4: Establece las directivas para que www.asir00.net en `/var/www/apache` y www.example.org en `/var/www/html`. Escribe las directivas usadas.

Nota: para hacer que los dominios anteriores apunten al propio PC, será necesario editar `/etc/hosts`.

Ej. 5: Edita el fichero de hosts para que asir99.net apunte también al PC, y solicita, en un navegador, la página www.asir99.net/ ¿qué página te muestra? ¿Por qué?

Hosts por IP/puerto

También es posible definir diversas configuraciones, dependiendo de la IP o del puerto por el que procede la petición.

```
<VirtualHost 172.20.40.10:80>
    DocumentRoot "/www/example1"
    ServerName www.example.com
</VirtualHost>
```

```
<VirtualHost 172.20.30.*:80>
    DocumentRoot "/www/example2"
    ServerName www.example.org
</VirtualHost>
```

En este caso, todas las peticiones hacia el puerto 80 de la IP de nuestro servidor 172.20.40.10 serán tratadas por el primer virtualhost. Las peticiones hacia cualquier dirección de nuestro servidor que esté en el rango 172.20.30.* (también en el puerto 80), emplearán el segundo bloque.

Ej. 6: Configura el servidor apache para que escuche en los puertos 80 y 8080. Escribe las directivas necesarias para crear la siguiente configuración (eliminando todas las otras configuraciones):

- Dirección externa : 80 → /var/www/com
- Dirección externa : 8080 → /var/www/org
- Localhost:80 → /var/www/html

Virtualhost por defecto

Puede usarse un Virtualhost `_default_`, de forma que, para toda petición de dominios que no estén explicitamente definido en otras directivas Virtualhost, se use una configuración concreta. Por ejemplo:

```
<VirtualHost _default_:*>
    DocumentRoot "/var/www/default"
</VirtualHost>
```

Esta directiva es distinta a *:*, ya que solo será activa cuando ninguna otra sea aplicable.

Ej. 7: Crea los directorios, ficheros y directivas necesarios para que el, cuando el dominio solicitado no sea ni `www.asir00.net`, ni `www.example.org` , se use /var/www/error.

Parte VI:
Plataformas

Unidad Didáctica 1:

Virtualbox

1. Instalar VirtualBox

Instalación de prerequisitos

Para que VirtualBox funcione de forma completa, vamos a usar varios paquetes, que pueden ser utilizados con:

```
sudo apt install virtualbox
```

La instalación de VirtualBox conlleva la instalación de las claves de Oracle. Dichas claves se utilizan para firmar digitalmente los paquetes de software de VirtualBox. Las claves de virtualbox son necesarias por los siguientes motivos:

- **Verificación de Autenticidad:** se asegura que el software que estás descargando e instalando es el auténtico proporcionado por Oracle y no ha sido alterado o comprometido de ninguna manera.

- **Integridad del Software:** se asegura que los paquetes de software no hayan sido manipulados o dañados durante la transferencia. Cualquier cambio en el paquete después de su firma sería detectado durante la verificación de la firma, alertando al usuario sobre posibles problemas de seguridad.

- **Seguridad de la Fuente:** Al agregar las claves GPG de Oracle y su repositorio a tu sistema, estás indicando a tu gestor de paquetes (APT en el caso de Ubuntu) que confías en estos paquetes y que pueden ser instalados. Esto previene advertencias y errores relacionados con la seguridad al intentar instalar software de una fuente no verificada.

- **Actualizaciones Seguras:** Utilizar las claves y el repositorio oficial también asegura que las futuras actualizaciones de VirtualBox provendrán de una fuente confiable y segura, manteniendo tu sistema protegido contra software malicioso o no autorizado.

Instalación del Extension Pack

Para instalar el Extension Pack, tan solo es necesario:

```
sudo apt install virtualbox-ext-pack
```

La instalación del Extension Pack proporciona diversas posibilidades en la máquinas que se instalen:

- **Soporte para USB 2.0 y 3.0:** permite a las máquinas virtuales conectar dispositivos USB 2.0 y 3.0, ofreciendo una mejor compatibilidad y rendimiento con dispositivos USB.

- **VirtualBox Remote Desktop Protocol (VRDP):** proporciona soporte para el protocolo RDP (Remote Desktop Protocol). Esto permite a las máquinas virtuales funcionar como servidores RDP, a los que se puede acceder de forma remota.

- **Soporte para PXE Boot para Intel Cards:** permite a las máquinas virtuales arrancar desde una red usando el Entorno de Ejecución Pre-arranque (PXE) con tarjetas de red Intel.

- **Virtualización de E/S basada en hardware (VT-x/AMD-V):** agrega soporte para ciertas funciones de virtualización de hardware, mejorando el rendimiento de las máquinas virtuales.

- **Encriptación de discos virtuales:** esta característica permite la encriptación de discos virtuales utilizando el Algoritmo de Encriptación Estándar (AES).

- **Grabación de sesión de máquina virtual:** permite grabar y reproducir la actividad dentro de la máquina virtual, que puede ser útil para fines de demostración o para diagnóstico de problemas.

2. Descargar ISO

Descarga la máquina

Descarga la máquina virtual de Ubuntu Server en https://ubuntu.com/download/server.

Lo normal es descargar la versión del último Ubuntu LTS, en nuestro caso, Ubuntu 24.04.01 LTS.

Verificar el fichero ISO

Debemos conseguir el fichero de SHA256SUMS de Ubuntu correspondiente a nuestra versión. Esto puede conseguirse en http://releases.ubuntu.com/. Descarga ese fichero SHA256SUMS en el mismo directorio del fichero ".iso". Luego, ejecuta, en dicho directorio el siguiente comando:

```
sha256sum -c SHA256SUMS
```

Te dirá si la suma de verificación coincide. Si no es así, se ha producido un fallo en la descarga.

3. Crear máquina

Vamos ahora a crear una Nueva Máquina Virtual. Abre VirtualBox y pulsa, en el menú de arriba, “Máquina → Nueva" para crear una nueva máquina virtual.

En el cuadro de diálogo que aparece, escribe el nombre que le quieras asignar a la máquina virtual. Selecciona el ISO que descargarte y te debería detectar la el tipo de sistema operativo ("Ubuntu 64 bits). Dale a continuar.

Parámetros de la máquina

Corrige los fallos en el nombre de la máquina anfitriona. Por el momento, no instales las “Guest Additions”, por lo que no marques la opción. Finalmente, dale a continuar

Pasará a la ventaja para configurar el número de MBs y CPUs. Para un entorno de pruebas o ligero con Ubuntu Server basta con poner 1024MBs y 1CPU. Para sistemas como Ubuntu Desktop deberían doblarse ambos valores, quizás más memoria para un Windows. Selecciona los valores deseados y dale a continuar.

En la ventana de disco, deberás crear un disco para el sistema operativo a instalar. Lo normal es usar un disco de tamaño varible aunque, se se conoce el tamaño que se va a necesitar, un tamaño fijo pequeño puede tener ligeras ventajas al rendimiento. Para Ubuntu Server, un disco variable con 25GB debería ser suficiente. Finalmente dale a continuar.

En la siguiente pantalla, revisa los datos y dale a terminar. El sistema se instalará automáticamente.

Instalación de la máquina

Selecciona el idioma y selecciona descargar el entorno de instalación nuevo.

Selecciona el interface propuesto y configura el proxy (ninguno en nuestro caso). Se descargarán paquetes de instalación y pasará a la ventana de selección de disco.

Selecciona el disco entero, revisa los datos y dale a continuar, tras lo que te pedirá el nombre de usuario, contraseña y demás, y pulsa continuar.

Deja sin seleccionar la opción de Ubuntu Pro, de OpenSSH y demás paquetes. Finalmente, la instalación empezará.

4. Guest Additions

Son un conjunto de herramientas y controladores a instalar en la máquina invitada que proporcionan, entre otras mejoras, una mejor integración de ratón, teclado, portapapeles, y mejoran el rendimiento y la conectividad de red.

Insertar CD con los ficheros de la GuestAdditions

Primero, debemos hacer que la máquina anfitrión tenga disponible el ISO de las guest additions, para poder proporcionárselo a la máquina invitada. Para ello, instalamos, en la máquina anfitrion, el siguiente paquete.

```
sudo apt install virtualbox-guest-additions-iso
```

Ahora, en la máquina virtual, ve al menú de arriba, y selecciona `Dispositivos → Insertar imagen de CD de las «Guest Additions»`. En ese momento es como si hubiéramos insertado, en la máquina invitada un dispositivo un USB o CD que contiene los ficheros de las Guest Additions.

Sin embargo, en Ubuntu Server, no se "automontan" los dispositivos, es decir, que si insertamos un USB, un CD u otro dispositivo, éste pasa a estar disponible, pero no accesible de forma automática. Podemos montar el CD recién insertado con los siguientes comandos:

```
sudo mkdir /media/cdrom
sudo mount /dev/cdrom /media/cdrom
```

Esto lo que hace es crear un directorio donde se accederán los ficheros del CD (media/cdrom). En Linux, los dispositivos suelen montarse en directorios que están en `/media`. Luego, realiza el montaje del dispositivo `cdrom` que está dentro del directorio `dev`, que es donde están todos los dispositivos. (que realmente es un enlace a `/dev/sr0`).

Instalar las GuestAdditions

Primero, vamos a preparar el sistema para la instalación. Debemos instalar los paquetes de gcc, make, perl y bzip2,

```
sudo apt update
sudo apt upgrade
sudo apt install gcc make perl bzip2 build-essential module-assistant
sudo m-a prepare
```

Finalmente, accedemos al directorio que contiene el contenido de las Guest Additions, que en nuestro caso ha sido `/media/cdrom`, y ejecutamos el comando de dichas Guest Additions:

```
cd /dev/cdrom
sudo ./VboxLinuxAdditions.run
```

El proceso tarda un poco en terminar, puesto que tiene que compilar los kernels, aparte de la copia de archivos y configuración.

Reinicia la máquina invitada para que las Guest Additions empiecen a funcionar.

```
Sudo shutdown -r now
```

5. Configurar la red

Para poder ver los interfaces de red en consola debemos instalar un paquete, llamado `net-tools`:

```
sudo apt intall net-tools
```

Con esto, podremos, por ejemplo, ver los interfaces de red:

```
ifconfig
```

la cual nos mostrará todos los interfaces de red habilitados.

Vamos a poner otra configuración. En la máquina virtual, ve a `Dispositivos → Red → Preferencias de la red`, y aparecerá una ventana con la configuración de la red.

Establece `Conectado a` a `Adaptador puente`, y selecciona, justo más abajo, el interface de la máquina anfitriona usado para conectar a la red. Puedes verlo abriendo, en la máquina anfitriona, una terminal, y usando el mismo comando `ifconfig`.

Reinicia, usando el comando shutdown, y comprueba que la interfaz ha cambiado (usando, de nuevo, ifconfig).

Unidad Didáctica 2:

Instalar Wordpress

1. Apache2

En la máquina invitada, instala Apache2. El servidor debe tener varios módulos activos:

- **mod_rewrite:** necesario para implementar las "permalinks" de WordPress.
- **mod_alias:** habilita redireccionamientos y rutas de archivos.
- **mod_deflate** o **mod_gzip:** para el envío comprimido de archivos, mejorando la velocidad de carga. Esto requiere instalar también **mod_filter**.
- **mod_headers:** para controlar encabezados HTTP.
- **mod_expires:** Para la gestión de caché de navegador mediante encabezados de expiración.
- **mod_mime:** para asignar extensiones de archivo a tipos MIME.
- **mod_ssl:** para implementar HTTPS.

De los anteriores, el único imprescindible es mod_rewrite.

Tras instalar Apache2, comprueba que funciona: desde la máquina anfitrión, pon la dirección de red de la máquina invitada en un navegador, y debería mostrarse la web de bienvenida de Apache2.

2. Instalación de PHP

Instala también el módulo de PHP, empleando, en la máquina invitada, tanto el sistema php como el módulo que conecta apache con php y la base de datos.

```
sudo apt-get install php libapache2-mod-php php-mcrypt php-mysql
```

Comprueba que está instalado creando una página php:

```
<?php
  phpinfo();
?>
```

En php, deben estar instalados los siguientes módulos:

- **mysqli** o **mysql:** para conectar con mysql o mariadb..
- **gd:** para la manipulación de imágenes.
- **curl:** para varias funcionalidades de red.
- **xml** y **json:** permiten el procesamiento de datos XML y JSON.
- **Mbstring:** para soporte de caracteres multibyte.
- **Zip:** para descomprimir y comprimir archivos.

Deberás localizar qué paquetes del sistema (que se instalarán con `apt`) corresponden a los módulos de arriba.

3. Instalar MariaDB

Instala, en la máquina virtual, MariaDB:

```
sudo apt install mariadb-server
```

Inícialo y habilítalo:

```
sudo systemctl start mariadb
sudo systemctl enable mariadb
```

MariaDB incluye un script de seguridad para cambiar algunas de las opciones predeterminadas menos seguras: :

```
sudo mysql_secure_installation
```

Lo anterior debería permitirte establecer la contraseña de la base de datos. Finalmente, accede a la base de datos con:

```
sudo mysql -u root -p
```

Crea una base de datos para Wordpress

```
CREATE DATABASE wordpress_db;
CREATE USER 'wordpress_user'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON wordpress_db.* TO 'wordpress_user'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

sustituye wordpress_user y password por unos a tu elección (no deben quedarse, bajo ningún concepto, esos nombres).

4. Despliega Wordpress

Cambia, si no lo estabas ya, en la máquina invitada, al directorio de Apache2, y descarga la última versión de wordpress y sube los ficheros al directorio `/var/www/html` de la máquina invitada. Para descargar un archivo, puedes usar `wget`.

Si necesitas descomprimir un archivo, puedes utilizar el siguiente comando:

```
sudo tar xzf archivo_a_descomprimir.tar.gz
```

Establece que todos los archivos y directorios allí contenidos, incluidos los subdirectorios, tengan el usuario y grupo `www-data:www-data`.

```
sudo chown -R www-data:www-data /var/www/html/wordpress/
```

Finalmente, establece los permisos de todos esos ficheros a 755:

```
sudo chmod -R 755 /var/www/html/wordpress/
```

Fichero de configuración de Wordpress

Copia el fichero de ejemplo de Wordpress y copialo a `wp-config.php`:

```
cd /var/www/html/wordpress
sudo cp wp-config-sample.php wp-config.php
```

Cambia las líneas que definen `DB_NAME`, `DB_USER`, y `DB_PASSWORD` con los valores adecuados de la base de datos.

5. Instala Wordpress

Finalmente, con un navegador, pon la dirección de la máquina invitada, y sigue los pasos de instalación.

6. Configuración de red

Para configurar la red debemos instalar un paquete, llamado `net-tools`:

```
sudo apt intall net-tools
```

Con esto, podremos, por ejemplo, ver los interfaces de red:

```
ifconfig
```

la cual nos mostrará todos los interfaces de red habilitados.

Parte VII:
Herramientas

Unidad Didáctica 1:

Git

1. Configuración

Tras instalar git (con `sudo apt install git`), debemos realizar una serie de configuraciones iniciales para poder usarlo debidamente.

Niveles

En primer lugar, tenemos la configuración de sistema, que atecta a todos los repositorios de un sistema. Esta configuración se almacena en `/etc/gitconfig`, `C:\ProgramData\Git\config` o similares. Cuando queremos establecer una configuración como sistema deberemos usar la opción `--system`, lo que posiblemente necesitará permisos de administrador. Para ver la configuración de sistema podemos usar:

```
git config --system --list
```

Tenemos, por otro lado, la configuración global, que afecta a todos los repositorios de un usuario. Para establecer una configuración como blobnal debemos usar la opción `--global` Esta configuración se almacenará en `~/.gitconfig`, `C:\Users\«usuario»\.gitconfig` o similar.

```
git config --global --list
```

Finalmente, podemos establecer una configuración para el repositorio local en el que nos encontremos. Por defecto, cuando se establece una configuración (a través del comando `git config` o similar), se establecerá en el repositorio existente en el directorio donde nos encontremos.

```
git config --local --list
```

Para ver todas las configuraciones activas:

```
git config --list
```

Establecer, mostrar y borrar

Existen ciertos parámetros que podemos configurar, y lo haremos a nivel de sistema, global o local. Por ejemplo, para establecer el parámetro `color.ui` al valor `always` pondremos:

```
git config --system color.ui always //a nivel de sistema
git config --global color.ui always //a nivel global
git config color.ui always //a nivel local
```

para mostrar la configuración de un valor concreto usamos:

```
git config --system color.ui always //a nivel de sistema
git config --global color.ui always //a nivel global
git config --local color.ui always //a nivel localizar
git config color.ui always //el valor activo
```

Para borrar la configuración usamos:

```
git config --system --unset color.ui always //a nivel de sistema
git config --global --unset color.ui always //a nivel global
git config --unset color.ui always //a nivel local
```

`Ej. 1:` En linux, muestra la configuración de git a nivel de sistema. Luego, también a nivel de sistema, establece a `false` la opción `color.ui`. Finalmente, muestra, de nuevo, la configuración a nivel de sistema.

`Ej. 2:` Borra la configuración creada en el ejercicio anterior y vuelve a mostrar la configuración a nivel de sistema.

Configuración personal

Antes de nada, hay que proporcionar nuestro ***email, nombre y apellido(s)*** a Git, para que éste pueda identificar envíos de código hacia el repositorio remoto, de forma que, en dicho repositorio remoto, toda línea de código estará identificada por su autor. Para ello, empleamos los dos siguientes comandos:

```
git config --global user.name "«Nombre» «Apellido»"
```

```
git config --global user.email email@email.com
```

Para modificarlos se emplearían los siguientes:

```
git config --global --replace-all user.name "Nombre Apellido"
git config --global --replace-all user.email email@email.com
```

Ej. 3: Establece tu nombre, apellido y email a nivel global ¿por qué lo hacemos a nivel global y no a nivel local o de sistema?

Ej. 4: Establece tu email a nivel de sistema, pero uno distinto al ejercicio anterior. Emplea `git config --list` para ver cual prevalece. Finalmente, borra el email a nivel de sistema.

Configuración operativa

En entornos Windows los finales de línea son marcados con el carácter LF, pero, en los sistemas de repositorio, los finales de línea deben estar marcados por CRLF, tal y como sucede en Linux. Para que git te realice la conversión automática se puede usar:

```
git config core.autocrlf true
```

En sistemas Linux, como también usan CRLF para el fin de línea, este comando no suele ser necesario.

También podemos habilitar la autocorreción, de forma que nos sugiera el comando correcto cuando nos equivocamos al usar git:

```
git config --global help.autocorrect 1
```

Con lo siguiente, establecemos que, al iniciar un nuevo repositorio, la rama que se cree por defecto que se llame main en vez de master.

```
git config --global init.defaultBranch main
```

Credenciales

Cuando realicemos ciertos accesos al repositorio remoto (push, clone en repositorios privados, etc.) deberemos contar con permiso para ello. Git nos preguntará el usuario y la contraseña de acceso al repositorio. Para evitar insertarlos en cada operación, podemos guardar las credenciales en nuestro sistema con el siguiente comando, de forma permanente (store), durante 15 min (cache) o durante un tiempo concreto (cache –timeout).

```
git config credential.helper store
git config credential.helper cache
git config --global credential.helper 'cache --timeout=1800'
```

El gran problema de lo anterior es que las credenciales se almacenan sin cifrar en `~/.git-credentials`. Para evitarlo, podemos usar GCM. Para ello, hay que bajar la última versión dese `https://github.com/git-ecosystem/git-credential-manager/releases` (puede bajar el deb e instalarlo con dpkg -i). Tras ello, puedes usar:

```
git config  credential.helper manager
git config  credential.credentialStore cache
```

~~Primero, vamos a crear una clave privada y su correspondiente clave pública empleando ssh (se guardarán en `~/.ssh`). Utilizaremos el algoritmo Ed25519, que más rápido y seguro que RSA. También añadiremos un comentario a la clave pública para indicar el correo en cuestión:~~

```
ssh-keygen -t ed25519 -C "tuemail@example.com"
```

2. Iniciar un repositorio

Un repositorio tiene dos partes:

- Directorio local: contendrá los archivos del proyecto software que estemos desarrollando en un directorio concreto.

- Repositorio remoto, que estará situado en un servidor remoto. A menudo, estos repositorios tiene una interfaz web para poder configurarlos. Ejemplos de sitios web que ofrecen repositorios remotos son GitLab o GitHub.

Ej. 5: Ve a GitLab y crea una cuenta. Una vez hecho eso, crea un repositorio remoto público, en el que habrá, como mínimo, un fichero `README.md`. El nombre del repositorio puede ser `daw` o cualquiera que se te ocurra.

Podemos iniciar un repositorio partiendo de cualquiera de los extremos, pero los pasos se diferenciarán entre uno y otro. En caso de ***tener un repositorio remoto con código***, realizaremos uno de los siguientes (el directorio remoto será algo como `https://gitlab.com/techurbana/daw.git/` o similar:

```
git clone «directorio_remoto»
git clone «directorio_remoto» «ruta_local»
```

En el primer caso, se creará un subdirectorio en el directorio local, con el nombre del repositorio remoto (ej: con un repositorio remoto de `https://gitlab.com/techurbana/daw.git/` se creará un directorio `daw`. Con la segunda opción se descargarán los archivos en el directorio especificado, que debe estar vacío. Si nos situamos dentro del directorio y ejecutamos el siguiente comando:

```
git remote -v
```

nos mostrará la dirección del directorio remoto en cuestión.

En caso de que el ***directorio local que tenga un código*** que queramos publicar en un nuevo escritorio remoto, debemos realizar lo siguiente (será explicado más adelante):

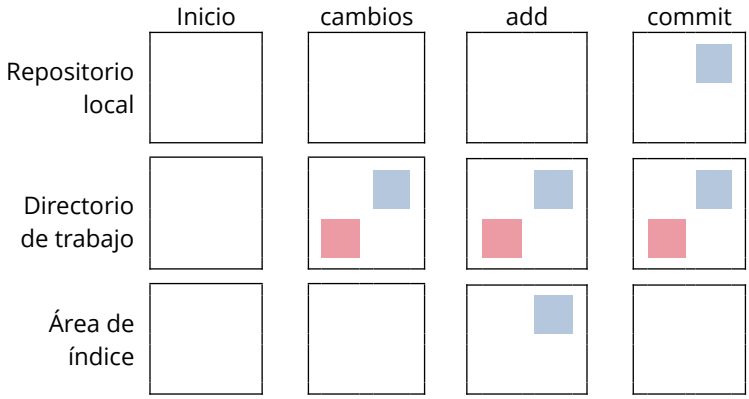
```
git init
```

```
git add .
git commit -m "First commit"
git remote add origin https://gitlab.com/techurbana/daw.git
git remote -v
git push -u origin master  #O bien: main en vez de master.
```

Ej. 6: Crea un repositorio local, clonando el directorio remoto que antes has creado.

3. Desarrollo en local

El repositorio local guarda siempre el estado del último commit, que son los últimos cambios aceptados. Ese estado se llama HEAD.



A partir de ahí, en nuestro proceso de desarrollo, realizamos diversos cambios en los ficheros del código, empleando el editor o entorno que deseemos, guardando los cambios en los ficheros. Al directorio que contiene el proyecto se le llama directorio de trabajo.

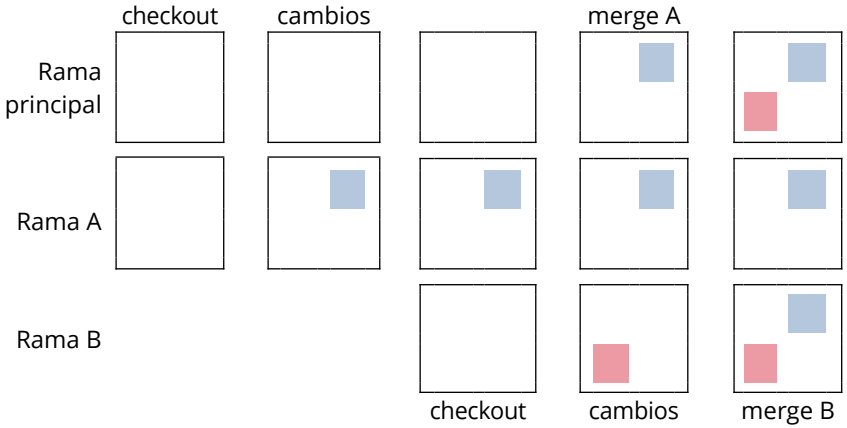
En cualquier momento podemos seleccionar cualesquiera archivos del directorio de trabajo y establecemos en el "stagin area" o área de índice, empleado `git add`, así como quitarlo empleando `git reset`. Esto no mueve realmente ficheros, tan solo le indica a git los ficheros que conforman ese área de índice. Observa que los ficheros seleccionados para conformar este área no tienen por qué ser todos los ficheros que hayamos modificado, tan solo los que conformarán el siguiente estado aceptado.

Finalmente, aceptamos cambios seleccionados en el área de índice realizando un `git commit`: esos cambios forman ahora parte del repositorio local, y el área de índice se vaciará. Observa que el directorio de trabajo conservará también los cambios de los archivos que no formaban parte del área de índice.

Trabajar con ramas

Como hemos visto antes, es posible estar realizando cambios en varias partes del código, de forma que solo parte de esos cambios forma parte del próximo estado.

Esto puede ser debido a que estemos realizando diversas pruebas, pero también puede ser debido a que estemos trabajando en varios cambios a la vez. Por esta razón, y para conseguir un seguimiento más limpio de los cambios, se suele trabajar con ramas.



La rama principal del repositorio local se llama main (en antiguos repositorios puede llamarse master). Partiendo de un estado en el que todos los cambios estén aceptados, podemos crear una nueva rama y cambia a ella con `git checkout`. Tras ello, realizaremos los cambios relativos a cierta nueva funcionalidad o arreglo de errores, y realizaremos los commits que estimemos. Cuando queremos fusionar los cambios con la rama principal, volveremos a la rama main con `git checkout` y emplearemos un `git merge` para fusionar los cambios de la rama creada con la rama principal. Podremos volver a cambiar a la rama creada y realizar más cambios (fusionándolos posteriormente), o podemos borrar la rama con `git checkout`, de forma que los cambios ya fusionados permanecerán en la rama principal.

Podemos tener varias ramas activas, de forma que, al crear una rama, el estado inicial de esa rama parte del estado de la rama principal, sin tener en cuenta los cambios hechos en ramas no fusionadas.

Trabajar con varias ramas

Podemos crear varias ramas, lo más típico es que cuelguen de la rama principal. Para crear una segunda rama nos dirigimos a la rama principal con checkout y volvemos a hacer un checkout -b.

Hay que tener en cuenta que, cuando fusionamos dos ramas, ambas quedan idénticas. Si, por ejemplo, fusionamos una rama con la principal, habiendo sido esta cambiada (con un commit anterior en la rama principal o con un merge anterior de otra rama), los cambios se fusionarán tanto desde la rama que estamos fusionando hacia la principal como viceversa.

~~*Actualización del remoto*~~

~~El pull y el clone fusionan los cambios en el repositorio local con nuestros cambios, mientras que el fetch actualiza el repositorio local.~~

~~El merge se encargará de fusionar los cambios realizados de nuestro directorio de trabajo con los cambios en el repositorio remoto. Un pull realmente es un fetch con un merge.~~

4. Área de índice

El área de índice consiste en seleccionar los ficheros que serán objeto del siguiente commit, para conformar el siguiente grupo de cambios aceptados. Para añadir ficheros al index, podemos realizar lo siguiente:

```
git add archivo
git add *.extensión
git add nombreCarpeta/
git add nombreCarpeta/*.extensión
git add nombreCarpeta/archivo
git add **/*.extensión
```

Los siguientes comandos afectan a todos los directorios. El primero, uno de los comandos git más usados, añade al índex todos los ficheros modificados, creados y eliminados. El segundo es igual, pero no añade los ficheros nuevos no rastreados:

```
git add .  #añade todos los archivos
git add -u  #añade todos excepto nuevos no rastreados
```

Para eliminar ficheros del área índex podemos emplear `reset`. Podemos emplear los mismos que con `add`, excepto el `git add -u`, que no tiene contrapartida en `reset`:

```
git reset archivo
git reset *.extensión
git reset nombreCarpeta/
git reset nombreCarpeta/*.extensión
git reset nombreCarpeta/archivo
git reset **/*.extensión
reset  #Elimina todos los archivos
```

Para saber qué archivos están en el área de Index, puedes emplear cualquiera de los siguientes comandos, mostrando el segundo una información más compacta que el primero:

```
git status
git status -s
```

Ej. 7: En el repositorio local, crea un directorio nuevo llamado source, y dento crea un fichero nuevo (por ejemplo `code1.js`) y añade dicho fichero al área de índice (hazlo desde el directorio base). Luego usa git status para ver los archivos que están en dicho área.

Ej. 8: Ejecuta el comando `git commit -m "new js files"`. Crea otro fichero dentro del directorio source (por ejemplo `code2.js`). Ejecuta `git add -u` ¿por qué solo se mueve el área de índice el fichero code1.js?

Es posible cancelar los cambios tanto del área de trabajo como del área index, con el siguiente comando:

```
git reset --hard
```

Para borrar los archivos y directorios nuevos no rastreados, de forma que podamos restaurar el estado anterior, podemos usar:

```
git clean -fd
```

Es posible usar `git clean -nd` para saber cuáles son estos archivos y directorios nuevos no rastreados.

Ej. 9: Modifica alguno de los archivos creados y crea un fichero y algún directorio nuevo. Luego cancela todos los cambios.

5. Commit

Cuando queremos aceptar los cambios hechos por los distintos archivos que están en el área de índice, deberemos hacer un commit. Cada commit debe llevar un mensaje que describa los cambios realizados (un fix sobre cierto bug, una optimización de una función, etc.). Para realizarlo, usaremos:

```
git commit -m "«Mensaje»"
```

También tenemos disponible un comando que añade todos los archivos excepto los nuevos no rastreados (como git add -u), y luego realiza el commit:

```
git commit -am "Mensaje"
```

Si queremos editar el mensaje del último commit, podemos escribir:

```
git commit --amend -m "Nuevo Mensaje"
```

Finalmente, es posible que no hayamos incluido algún archivo en el último commit, o que hayamos incluido uno que no debiéramos. O también es posible que queramos cambiar algo en algún fichero. Podemos editar los ficheros, y/o usar git add y/o git reset, y luego usar el siguiente comando para arreglar el error:

```
git commit --amend --no-edit
```

Deshacer cambios

Podemos deshacer todos los cambios del último commit con uno de los siguientes:

```
git reset --soft HEAD~1
git reset --hard HEAD~1
```

El primero deshace el último commit, pero deja los cambios hechos en el área de índice. El segundo descarta esos cambios.

Commits realizados

Para ver los commits realizados, puede emplearse:

```
git log
git log --oneline           #versión compacta
git log -n 5                #solo los últimos 5 commits
git log origin/main..HEAD   #solo desde el último push
```

En el siguiente resultado de git log --oneline, se ven dos commits, el primero es último commit realizado en nuestro repositorio local, que no ha sido aún pusheado, y el siguiente es un commit que indica el estado del repositorio remoto.

```
3e9e19b (HEAD -> main) nuevo js
f89b7d6 (origin/main, origin/HEAD) Initial commit
```

Ej. 10: Modifica alguno de los archivos creados y crea un fichero y algún directorio nuevo. Añade todos ellos al index y luego haz un commit. Realiza un git log para ver los commits realizados.

Ej. 11: Descarta los cambios del último commit y devuélvelos al área de index. Luego descarta esos cambios del área de index. Realiza un git log para ver los commits realizados.

Ej. 12: Realiza los siguientes pasos:

1) Crea una carpeta llamada `proyecto_web` o similar. Crea un nuevo repositorio en ella (no haremos uso del repositorio remoto).

2) Crea una carpeta src y, dentro de ella, crea un archivo `index.html` vacío y realiza un commit inicial.

3) Edita el fichero y ponle una estructura web básica, sin conexión a hora de estilo ni a archivos JS. Desde el directorio base, ~~añade únicamente el archivo `index.html` al área de índice~~ **añade el archivo al área de índice con `add .` y** realiza un commit.

4) Crea una nueva rama llamada agregar-estilo para trabajar en el diseño de la página web y cambia a esa rama.

5) Agrega un archivo CSS llamado style.css con un estilo simple para el cuerpo y el título de la página (u otro estilo que desees).

6) Vincula el archivo CSS en el archivo `index.html`.

7) Haz commit de estos cambios en la rama agregar-estilo.

8) Vuelve a la rama main, de forma que volveremos al estado antes de que se hicieran los cambios de estilo.

9) Crea una nueva rama llamada agregar-artitulo.

10) Añade un pie de página a index.html (un `<footer>`), y haz commit de los cambios en esta nueva rama.

11) Utiliza git log para visualizar el historial de commits y confirmar que todo está registrado correctamente.

12) En la rama actual, mueve el proyecto a un estado anterior en el historial de commits. Edita ahora `index.html` para añadir un `<article>` con solo un texto dentro. **Añade al área de índice el archivo y haz commit**.

13) Crea una nueva rama llamada agregar-contenido-articulo y agrega una imagen dentro del articulo y pon también un título con h2 y modifica el texto.

14) Revisar el estado actual con `git status`.

15) Haz un commit de los cambios en la rama actual, realiza un git log para ver todo el estado del repositorio.

```
git config --global user.name "«tu nombre»"
git config --global user.email «tur correo»
git config --global init.defaultBranch main
git config core.autocrlf true  #Solo necesario si estamos en Windows

# 1)
mkdir proyecto_web
cd proyecto_web
git init

# 2)
mkdir src
touch src/index.html
git add .
```

```
git commit -m "Commit inicial con index.html vacío"
# 30db3fa (HEAD -> main) Commit inicial con index.html vacío

# 3)
# (Edita el archivo index.html)
git add .
git commit -m "HTML básico en index.html"
# * 3aefc38 (HEAD -> main) HTML básico en index.html
# * 30db3fa Commit inicial con index.html vacío

# 4)
git checkout -b agregar-estilo
# * 3aefc38 (HEAD -> agregar-estilo, main) HTML básico en index.html
# * 30db3fa Commit inicial con index.html vacío

# 5)
touch src/style.css
# (Edita el archivo style.css)

# 6)
# (Edita index.html para vincular style.css con <link>)

# 7)
git add .
git commit -m "Agregado y vinculado style.css"
# * dc5810b (HEAD -> agregar-estilo) Agregado y vinculado style.css
# * 3aefc38 (main) HTML básico en index.html
# * 30db3fa Commit inicial con index.html vacío

# 8)
git checkout main

# 9)
git checkout -b agregar-artitulo
# * dc5810b (agregar-estilo) Agregado y vinculado style.css
# * 3aefc38 (HEAD -> agregar-artitulo, main) HTML básico en index.html
# * 30db3fa Commit inicial con index.html vacío

# 10)
# (Edita index.html para agregar <footer>)
git add .
git commit -m "Agregado footer a index.html"
# * 6c26167 (HEAD -> agregar-artitulo) Agregado footer a index.html
# | * dc5810b (agregar-estilo) Agregado y vinculado style.css
# |/
```

```
# * 3aefc38 (main) HTML básico en index.html
# * 30db3fa Commit inicial con index.html vacío

# 11)
git log  # git log --oneline --graph --decorate --all

# 12)
git reset --hard HEAD~1
# * dc5810b (agregar-estilo) Agregado y vinculado style.css
# * 3aefc38 (HEAD -> agregar-artitulo, main) HTML básico en index.html
# * 30db3fa Commit inicial con index.html vacío
# (Edita index.html para agregar <article>)
git add .
git commit -m "Agregado artículo a index.html"
# * 494a922 (HEAD -> agregar-artitulo) Agregado artículo a index.html
# | * dc5810b (agregar-estilo) Agregado y vinculado style.css
# |/
# * 3aefc38 (main) HTML básico en index.html
# * 30db3fa Commit inicial con index.html vacío

# 13)
git checkout -b agregar-contenido-articulo
# (Edita index.html: agrega imagen, título h2, modifica <article>)
# * 494a922 (HEAD -> agregar-contenido-articulo, agregar-artitulo) Agregado
# | * dc5810b (agregar-estilo) Agregado y vinculado style.css
# |/
# * 3aefc38 (main) HTML básico en index.html
# * 30db3fa Commit inicial con index.html vacío

# 14)
git status

# 15)
git add .
git commit -m "Completar articulo"
git log
# * 05db88e (HEAD -> agregar-contenido-articulo) Completar articulo
# * 494a922 (agregar-artitulo) Agregado artículo a index.html
# | * dc5810b (agregar-estilo) Agregado y vinculado style.css
# |/
# * 3aefc38 (main) HTML básico en index.html
# * 30db3fa Commit inicial con index.html vacío
```

6. Checkout y merge

Para crear una rama, con objeto de implementar nueva característica o corrección de errores, podemos usar:

```
git checkout -b «nombre_de_la_rama»
```

Esto creará una rama nueva, cuyo estado inicial será el que tenga la rama principal en ese momento, es decir, desde el último commit o merge realizado en la rama principal (también puede haberse visto afectado este estado por clone, fetch o pull o rebase). También nos moverá a esa rama.

Como nos ha movido a esa rama, a partir de ahora, los cambios y commits que realicemos, serán realizarán en esa rama.

Podemos cambiar entre ramas con el comando:

```
git checkout «nombre_de_la_rama»
```

Llegará un momento en que queramos fusionar los cambios realizados en la nueva rama con la rama principal, para ello debemos movernos a la rama principal y hacer un merge:

```
git checkout main
git merge  «nombre_de_la_rama_a_fusionar_con_la_principal»
```

Podemos borrar la rama con el siguiente comando:

```
git branch -d «nombre-de-la-rama»
```

Observa que es posible que hayamos fusionado los cambios con la rama principal, pero que queramos seguir trabajando con esa rama, por lo que nos podremos mover a esa rama, seguir haciendo commits y, tarde o temprano, hacer otro(s) merge.

7. Stash

Puede realizar un guardado de los cambios realizados desde el último commit junto con el estado de tu área de índice. El siguiente comando guarda esos cambios y vuelve el estado del área de trabajo y el del índice al estado del repositorio tras el último commit o similar. Puedes hacer varios stash, que se irán apilando como en una pila:

```
git stash  # Crea un stash con todos los cambios hechos
git stash push  «ruta_a_archivo1» «ruta_a_archivo2»
           #Solo con los cambios en esos archivos
git stash push --staged  #Solo de los ficheros en área de índice
git stash push -m "Descripción"  # Stash con mensaje
```

Para examinar los stash existentes:

```
git stash show  # Muestra ellos stash existentes
git stash show -p stash@{n}  #Muestra detalle de un stash
git stash show -p -1         #Igual que el anterior
```

Podemos restaurar el estado de un stash, descartando el estado actual. Para ello, podemos usar:

```
git stash pop   # Restaura el estado del último stash y lo borra
git stash apply # Igual, pero sin borrarlo de la pila
git checkout stash@{0} -- «ruta_a_archivo1» «ruta_a_archivo2»
                #restaura solo el archivo( especificado
```

El almacenado en el stash es útil para trabajar con ramas, puesto que el área de trabajo y de índice se pierden cuando se cambia de rama. El stash es independiente de las ramas, de forma que quedará inalterado al cambiar de una rama a otra.

Ej. 13: Reinicia tu repositorio para que apunte al estado inicial, o bien borra todos los archivos y directorios excepto README.md y realiza un commit. En este ejercicio vamos a crear una aplicación que codifique y decodifique una URL.

1) En primer lugar, crearemos una página web que tenga un campo de texto, un botón y una etiqueta. Cuando se pulse el botón, el programa ejecutará un

script, escrito en la misma página HTML, que establecerá la etiqueta con la URL codificada. Para codificar, usamos encodeURIComponent:

```
const original = "https://ex.com/?name=John Doe&age=30";
const encoded = encodeURIComponent(original);
```

Realizaremos los cambios en la rama principal y aplicaremos varios commits para ello.

2) Crea ahora otra rama, con el nombre `decode` u otro a tu elección. Implementa un nuevo componente, un radiobutton, y haz un commit.

3) Aplica un cambio menor en el HTML, de forma que el radiobutton se llame de otra forma, u otro cambio, como englobar todo con un div. Crea un stash con el estado actual

4) Dirígete ahora a la rama principal con checkout y crea otra rama, llamada `css` o similar. Implementa el css para la página tenga un estilo más elaborado. Observa que ambas son independientes.

5) Fusiona la primera rama con la rama principal, luego fusiona la segunda rama. Esto hará que los cambios de la primera rama se incorporen a la segunda.

6) Ve a la primera rama y comprueba que no estará el cambio hecho en 3. Restáuralo desde el stash ¿qué ha pasado con el link al css?

7) Sigue trabajando en ella, haciendo que, en caso de que el radiobutton esté en un estado se codifique, y que si esté en otro estado, decodifique, usando varios commits para ello.

8) Fusiona la rama con la principal.

8. Merge vs rebase

Como vimos, cuando creamos una rama nueva, solemos realizar una serie de commits y, antes o después, querremos incorporar esos cambios a la rama principal. En el sisguente ejemplo, creamos una nueva rama con un commit (se muestra el resultado de (`git log --oneline --graph --decorate --all`).

```
* cbe72f8 (HEAD -> feature, origin/main, origin/HEAD, main) Initial commit
```

Tras crear una rama nueva no habrá cambios en el log, salvo que se indicará el nuevo puntero `HEAD→feature`, ya que ahora estamos en esa rama:

```
* cbe72f8 (HEAD -> feature, origin/main, origin/HEAD, main) Initial commit.
```

Ahora bien, si creamos ciertos cambios, estando en la nueva rama, y realizamos un commit en ella, el puntero de `feature` se moverá hacia un nuevo estado:

```
* f84af05 (HEAD -> feature) minor changes
* cbe72f8 (origin/main, origin/HEAD, main) Initial commit
```

Cambiamos ahora a la rama principal, y realizamos un `merge`, pero lo haremos forzandolo para que sea sin fast-forward (`git merge «rama_a_fusionar» --no-ff`). Se avanzará el puntero de la rama principal hacia el estado de la rama fusionada, y se conservarán los commits en la rama en la que fueron hechos:

```
*   7131068 (HEAD -> main) Merge branch 'feature'
|\
| * f84af05 (feature) minor changes
|/
* cbe72f8 (origin/main, origin/HEAD) Initial commit
```

Observa que hemos tenido que forzar --no-ff porque un merge, por defecto, aplica un fast forward si es posible. En caso de que tanto la rama principal como la rama a fusionar tuvieran commits cada uno, no sería posible realizar fast forward, y el merge se realizaría siempre sin ese fast forward.

Fast forward

Cuando una de las ramas a fusionar no tiene commits (como el caso anterior), es posible realizar un merge con fast forward (`git merge «rama_a_fusionar»`).

Vamos a usar el apartado anterior. Primero vamos a deshacer el merge y, para hacerlo correctamente, primero asegúrate qué rama contiene el merge. Vete a esa rama y haz el reset.

```
git branch --contains 7131068
git checkout «rama»
git reset --hard HEAD~1
```

Ahora vamos hacer un merge con fast forward. En este caso, los commits de la rama fusionada pasan a formar parte de la rama principal (o de la rama hacia la que fusionemos). Prueba a hacer, desde main, haz el `git merge feature`., y muestra con `git log --oneline --decorate --all`:

```
* f84af05 (HEAD -> main, feature) minor changes
* cbe72f8 (origin/main, origin/HEAD) Initial commit
```

Rebase

En caso de que tengamos ramas a fusionar con cambios en ambas ramas, también podemos usar rebase (`git rebase «rama_a_fusionar»`). Este comando, ademas de fusionar los commits de la rama indicada con la rama actual, aplica todos los cambios de la rama indicada hacia la rama actual.

Los rebase se hace siempre desde la rama local, nunca desde ramas remotas (origin/*).

Ej. 14: Crea un nuevo repositorio con tan solo el commit inicial. Tras clonarlo, crea un archivo index.html y haz un commit en la rama main local. Luego vuelve a la rama main, al commit inicial (`git checkout HEAD~1` o similar) y crea una rama llamada, por ejemplo, `feature`, en dicha rama, un archivo main.js y realizando un commit en dicha rama. Realiza un `git log --oneline --graph --decorate --all` para ver el resultado. Deben de aparecer dos ramas, cada una con un commit.

Fuerza un merge con fast-forward (usa `git merge -ff-only`) y verás que te da error.

Realiza ahora un merge fast-forward (usa `git merge «rama» --ff «mensaje»` o similar) y observa el hecho de que no te fusiona los commits de las dos ramas (no te “aplana”). Es decir, te ha realizado un git SIN fast-forward, aún indicándole que lo hiciera CON fast-forward.

Retrocede (usa `git reset --hard HEAD~1` o similar) hasta antes del merge y realiza ahora un merge sin fast-forward (`git merge «rama»` o `git merge «rama» --no-ff` o similar). Verás que no te fusiona commits, tal y como el merge sin fast-forward hace normalmente.

Retrocede una vez más (`git reset --hard HEAD~1` o similar) y realiza ahora un rebase desde main (`git rebase «rama»`). Verás que ahora si que te fusiona los commits de la rama hacia main, aunque no te fusiona ramas.

Resumen:

	merge --ff-only	merge --no-ff	rebase
o feat *cambio* | o main *Inicio*	o feat, main *cambio* | o *inicio* *[[merge]]*	o main *merge* | \\ | o feat *cambio* | / o *Inicio*	o feat, main *cambio* | o *inicio*
o feat *cFeat* | o main *cMain* | / o *inicio*	error	o feat *merge* | \\ | o main *cMain* o | *cFeat* | / o *[[merge]]*	o main *cMain* o feat *cFeat* o *inicio*

9. Fetch, pull y push

El comando fetch descarga el repositorio remoto, en su versión más reciente, y la almacena en el repositorio local (nota, realmente, solo descarga las ramas bajo "origin" del remoto especificadas en refspec, aunque en repositorios simple esto suele ser todas las ramas remotas) . Este hecho no originará nunca un conflicto, porque lo único que avanza es la rama origin/head:

```
git fetch
```

Ej. 15: Crea un repositorio, de nuevo desde clone, con tan solo el commit inicial. Realiza un cambio en un apartado concreto de README.md, luego crea un archivo index.html y haz un commit en main.

Tras el clone, navega hasta el archivo README.md (hasta que veas el contenido), y edita el mismo apartado que editastes antes, ahora en la web (en la pestaña de "Edit", la opción "Edit single file"), proporcionando una descripción para el commit.

Dirígete a tu repositorio local y realiza un git fetch. Observa el resultado con git log.

En proyectos más complejos, con varias ramas en repositorios, git fetch actualizaría todas las ramas remotas en nuestro repositorio local (todas las origin/«rama»).

Seguramente, ahora querremos fusionar los cambios hechos en remoto con el estado de la rama actual, para ello nos situamos en la rama main y realizamos uno de los siguientes:

```
git merge origin/main  #válido sobre todo si en local no hay cambios.
git rebase origin/main  #siempre situado en la rama local, nunca la remota
```

Si tuvieramos más ramas en local y remoto, tendríamos que situarnos en cada una de ellas y fusionaralas una a una.

Si existen conflictos en el merge, deberás resolverlos manualmente, abriendo y modificando cada uno de los archivos que generen conflictos (puedes usar `log` `status` para saber cuáles son estos conflictos).

Ej. 16: A partir del ejercicio anterior, realiza un rebase y observa los cambios con git log.

Pull

Básicamente, equivale a un fech de tan solo la rama actual con la correspondiente rama del remoto (origin/«nombre_de_la_rama»), seguido por un merge.

```
git pull
```

Si has realizado cambios en la rama en cuestión de tu repositorio local (has hecho commits en ella) desde el último pull que hiciste, el comando puede que te diga que hay conflictos, que se resolverían de igual forma que al hacer un fetch y luego un merge.

Push

Finalmente, vemos que, tras la fusión, el puntero de origin/HEAD sigue apuntando a una posición más antigua que la de main. Esto es porque no hemos enviado los resultados al servidor. Para ello, usamos:

```
git push
```

Esto enviará el código fusionado al servidor remoto.

Observa que un push no tiene por qué realizarse después de una fusión. Si queremos enviar un código sobre algo que nadie más ha cambiado, como por ejemplo un componente nuevo, etcétera, podremos realizar un push directamente.

Ej. 17: Realiza un push al respositorio remoto y observa los cambios en git log.

Ej. 18: Con un compañero, uno de vosotros creará un repositorio remoto y le dará permiso al otro usuario para modificar dicho repositorio remoto (en la web de Gitlab, en el repositorio en cuestión, usa la ruta de Manage→Members (más información en `https://docs.gitlab.com/ee/user/project/members/`).

Luego, cread repositorios locales en cada uno de vuestros equipos, de forma que los dos repositorios de ambos sistemas locales se enlacen con él.

Uno de vosotros realizará cambios (crea algún archivo que tenga algún código y también crea alguna carpeta) y los publicará al escritorio remoto. Luego el otro compañero los bajará ¿son ahora iguales los archivos de ambos repositorios locales? Emplead `git log --oneline --graph --decorate --all` en ambos sistemas para ver los cambios.

Ej. 19: Probad ahora a editar cada uno de vosotros, cada uno en vuestro respectivo repositorio local, el mismo fichero (sustituid todo su contenido por una línea que no sea un comenario), y subidlos al repositorio remoto mediante. El último en hacer el push deberá resolver el conflicto.

Ej. 20: En la web de Gitlab, en el repositorio remoto, dirígite a Code > Branches y crea una nueva rama (más información en https://docs.gitlab.com/ee/user/project/repository/branches/). Haced un fetch y observad, con git log la nueva rama remota.

Cada uno de vosotros, haced vuestra propia rama en local y haced un cambio en esa rama (por ejemplo, un nuevo fichero que se llamará igual para ambos sistema locales con una línea de texto distinta en ambos sistemas) y luego haced commit.

Estando en esa rama (no en main), realizad un pull (que lo haga primero el que en el ejercicio anterior fue el último en hacer el push), y observad los cambios en local y en remoto. Posiblemente, el último en hacer pull deberá arreglar conflictos.

```
const a = [{a:3, b:1},{a:2, c:2},{d:1,a:1, c:3}];
```

```
const b = a.reduce( (acc,e) => {
  for (const [key, value] of Object.entries(e)) {
    acc[key] = value + (acc[key] || 0)
  }
  return acc;
},{});

console.log(b);
console.log(Object.entries({a:3, b:1}));
console.log(" ");

const c = a.reduce( (acc,e) => [...acc, ...Object.entries(e)], [])
//[ [a,3], [... ]
           .reduce( (acc,[key, value]) => ({...acc, [key]:value +
(acc[key]||0) }), {} );

console.log(c);
```

Ej. 21: Realiza los siguientes pasos. Antes de todo, asegúrate que tiene las configuraciones de email, nombre y, si estás en Windows, de `crlf`. También es recomendable hacer un caché de las credenciales. Ve haciendo un git log (con las opciones necesarias) para ir viendo como evolucionan los repositorios.

a) Crea un directorio, entra en él, e inicializa un repositorio git.

b) Crea un `README.md` con un texto inicial de una sola línea (con el contenido `TODO` o similar).

c) Realiza el clásico commit inicial (`Initial Commit`).

d) Realiza un `git remote add origin «URL_rep_remoto»`, y verifica que el nuevo repositorio ha sido añadido con `git remote -v`.

e) Realiza un push. Comprueba que el repositorio remoto coincide con el local. (Nota: también puedes comprobar que sucede con la rama `origin→main` con el `git log`).

f) Crea una nueva rama. En ella, Añade una carpeta `src` y, dentro, crea un nuevo archivo `main.js`, que contendrá solo una variable de la forma `let a=2;` , y luego un if - else if - else. El primer if comprobará si a es igual a 1 y, si es así, hará un `console.log('Es un uno');`. El elseif comparará si a es

igual a 2 (y pondrá un mensaje acorde), y el else final mostrará el mensaje de que es otro número.

g) Realiza commit de los cambios en esta nueva rama. Luego haz un push (o bien un fetch y luego un push, que es más típico, aunque en este caso el resultado es el mismo, ya que en remoto no se ha cambiado nada).

h) Dirígete al escritorio remoto, en la página web. Asegurándote de hacerlo en la rama `main`, cambia `README.md` para que ahora contenga tres cabeceras de primer nivel, la primera cabecera se llamará `install`, la segunda `config` y la tercdera `run`. Bajo cada cabecera habrá un texto que tu desee. Los cambios deberán ser hechos con un commit (y su respectivo mensaje de commit), que te deberá ser preguntado en la web al grabar los cambios.

Nota: una cabecera de primer nivel, en este tipo de archivos, se implementa con un carácter `#`, y los subapartados con `##`, `###`, etcétera.

i) Ve al repositorio local, a la rama main. Modifica el `README.md` para que tenga dos cabeceras, una llamada `config` y otra llamada `notes`. Bajo cada cabecera habrá un texto que elijas, que será distinto del que pusieras en el anterior apartado. Realiza un commit en local de este cambio.

j) Siguiendo en la rama principal, realiza un pull. Verás que se produce un error, ya que las ramas son divergentes. No continues con el pull, sino que realiza un fetch.

k) Fusiona la dos ramas main (la local y la remota), de forma que los commits se fusionen también.

l) Vuelve atrás con git reset --hard «ID», siendo «ID» la id anterior al rebase (usa git reflog para averiguarlo si no la sabes). Haz un merge de las ramas main.

a)
```
mkdir proyecto
cd proyecto
git init
```

b) Crea el archivo README.md con un texto cualquiera

c)
```
git add README.md  #también valdría: git add .
git commit -m "Initial Commit"
# * 0aa013e (HEAD -> main) Initial Commit
```

d) Crea el repositorio remoto en blanco, desmarcando el creear "README.md".
URL: https://gitlab.com/mydaw/daw2# → https://gitlab.com/mydaw/daw2.git

```
git remote add origin «URL_rep_remoto»
git remote -v  #Permite ver los remotes
#Borrar todos los remotes: git remote | xargs -n1 git remote remove
```

e)
```
git push -u origin main  #El -u asocia las ramas remota y local
# * 0aa013e (HEAD -> main, origin/main) Initial Commit
```

f)
```
git checkout -b crear_script  #Crea una rama y cambia a ella
mkdir src
```

Crea `src/main.js` con el contenido:

```
let a = 2;
if (a == 1) { console.log('Es un uno'); }
else if (a == 2) { console.log('Es un dos'); }
else { console.log('Es otro número'); }";
```

```
# * 0aa013e (HEAD -> crear_script, origin/main, main) Initial Commit
```

g)
```
git add .
git commit -m "Añadido src y main.js"
git push -u origin crear_script  #El -u asocia las ramas remota y local
# * 8b7465e (HEAD -> crear_script, origin/crear_script) Añadido...
# * 0aa013e (origin/main, main) Initial Commit
```

h) Modifica, en remoto, el `README.md`, usando el botón de edit (edit single file). Tras modificar el contenido, los cambios se aceptan con el botón "commit changes":

```
# install
Texto para instalación.
# config
Texto para configuración.
# run
Texto para ejecución.
```

i)
```
git checkout main
# * 8b7465e (origin/crear_script, crear_script) Añadido src y main.js
# * 0aa013e (HEAD -> main, origin/main) Initial Commit
```

Cambia el README.md local para que tenga "config" y "notes".

```
git add .
git commit -m "edit README.md with 2 headers"
```

```
   # * f66a1c1 (HEAD -> main) edit README.md with 2 headers
   # | * 8b7465e (origin/crear_script, crear_script) Añadido src...
   # |/
   # * 0aa013e (origin/main) Initial Commit
j) git pull
   git fetch
   # * f66a1c1 (HEAD -> main) edit README.md with 2 headers
   # | * 96cdcc7 (origin/main) Edit README.md
   # |/
   # | * 8b7465e (origin/crear_script, crear_script) Añadido src...
   # |/
   # * 0aa013e Initial Commit
k) # git checkout main #(solo si estábamos en otra rama
   git checkout #Solo para ver que divergen
   git rebase origin/main
```

Edita el contenido de `README.md`, o usa `git rebase –abort` para abortar.

```
   git add README.md
   git rebase --continue
   # * 3106887 (HEAD -> main) edit Merged 3 and 2 README.md headers
   # * 96cdcc7 (origin/main) Edit README.md
   # | * 8b7465e (origin/crear_script, crear_script) Añadido src ...
   # |/
   # * 0aa013e Initial Commit
l) git reglog #apunta la ID del estado anterior al rebase
   git reset --hard «ID» #Sustituye la ID
   git merge origin/main  #asumimos que seguimos en rama main
```

Edita el contenido de `README.md`.

```
   git add README.md
   git commit -m "Merge 2 y 3 headers in README.md"
   # *   84028ae (HEAD -> main) Merge 2 y 3 headers in README.md
   # |\
   # | * 96cdcc7 (origin/main) Edit README.md
   # * | f66a1c1 edit README.md with 2 headers
   # |/
   # | * 8b7465e (origin/crear_script, crear_script) Añadido src y main.js
   # |/
   # * 0aa013e Initial Commit
```