

## **Introducción al desarrollo de aplicaciones web**

### **Introducción**

Con este curso en línea descubrirás cómo se realizan estas apps web, comprenderás su arquitectura cliente-servidor y además aprenderás paso a paso a realizar una aplicación web desde cero. Concretamente, te vamos a guiar a lo largo del curso para que desarrolles tu propia red social.

Para este propósito comprenderás cómo utilizar de forma conjunta diferentes tecnologías web como son HTML, CSS, Python, JSON, JavaScript y Ajax.

### **Instalación de virtualenv en linux**

El siguiente es ejecutar aquí “virtualenv”, dice que no existe y entonces lo único que tengo que hacer es “sudo apt install” y en este caso queremos instalar VirtualEnv.

Se conecta a los repositorios, se da cuenta de que estamos trabajando con Python 3, se descarga la versión correcta, y cuando lo ejecutamos lo instala y, si ahora abro un nuevo terminal para ver las modificaciones, tengo instalado VirtualEnv en la máquina.

El siguiente paso es configurar y empezar a utilizar VirtualEnv.

En lo que se refiere a la configuración, lo único que vamos a hacer es crear un directorio en el que almacenar todos los entornos virtuales. Lo que yo os propongo es un directorio que se llame virtualenvs, que sea un directorio oculto, de ahí que esté el punto (.) delante del directorio y que se encuentren en el home del usuario. Si creamos ese directorio, inicialmente estará vacío.

Vamos a crear nuestro primer entorno virtual. Para ello, simplemente tenemos que llamar a virtualenvs, darle la ruta en la que queremos que almacene el entorno virtual y el nombre del entorno virtual, todo como un directorio.

Cuando hacemos esto, en lo que se traduce es que ese directorio que antes estaba vacío ahora tiene un nuevo directorio, que se llama web, y que dentro del directorio web existen otra serie de directorios y de ficheros donde están, por ejemplo, dentro del directorio bin, las herramientas para trabajar con VirtualEnv, el pip con el que vamos a trabajar, las versiones de Python, tenemos librerías en las que está, en este caso Python3 y esto se va a empezar a rellenar a medida que trabajemos con el entorno virtual.

Otra cosa que podemos hacer, relacionada con la configuración en este caso de pip, pero relacionado con los entornos virtuales, es obligar a que no podamos instalar ni desinstalar ningún módulo de Python si no es dentro de un entorno virtual. ¿Qué es lo que conseguimos con esto? Pues evitar errores, evitar que yo borre accidentalmente un módulo que utiliza una aplicación crítica o que actualice a una nueva versión un módulo que utiliza otra aplicación crítica.

Para ello, lo que vamos a utilizar es la variable entorno, PIP\_REQUIRE\_VIRTUALENV.

Si yo esta variable la fijo a true (“export PIP\_REQUIRE\_VIRTUALENV=true”),

lo que estoy haciendo es que pip no pueda instalar ningún módulo si no estás dentro de un entorno virtual. Si yo intento instalar, por ejemplo, el módulo pprint, como he fijado esa variable a true me dice que no puedo trabajar con pip, no se puede instalar, porque no estoy dentro de un entorno virtual de esta forma, como ya decía, evitamos errores.

Si quiero, ya porque estoy seguro de ello, instalar o borrar una librería general del sistema, lo único que tengo que hacer es quitar esa variable de entorno (“unset PIP\_REQUIRE\_VIRTUALENV”). Le quito el valor y ahora intento instalar el módulo pprint, en este caso.

Para no tener que hacerlo cada vez, lo que podemos hacer es fijar esa variable de entorno en nuestro bashrc. Aquí, en nuestro bashrc, lo que yo podría hacer es poner esta línea y, cada vez que abriera una sesión con bash, se fijaría y la variable PIP\_REQUIRE\_VIRTUALENV no tendría que ir fijándola cada vez.

Ahora que ya tenemos creado nuestro entorno virtual, el siguiente paso es empezar a trabajar con él. Para ello, lo que tenemos que hacer es llamar a través de source a uno de los ficheritos que ha dejado en el directorio bin, que veíamos antes.

```
source .virtualenvs/web/bin/activate
```

Este ficherito es “activate”, que lo que hace es activar ese entorno virtual; en este caso, el entorno virtual “web”. Vemos que ha cambiado la prompt, me ha puesto aquí entre paréntesis el nombre del entorno virtual activo, y eso significa que todo lo que llevaba ahora con Python, ya sea invocaciones al intérprete o instalaciones de módulos o bibliotecas a través de pip, se va a hacer en el directorio que representa ese entorno virtual.

Si yo quiero salir del entorno virtual, quiero dejar de trabajar con él, lo que hago es llamar a “deactivate”, y a partir de ese momento dejo de trabajar con ese entorno virtual.

Vemos la diferencia. Si yo digo que qué Python se está ejecutando aquí, que no tengo ningún entorno virtual activo, me dice que se está ejecutando el Python de usr/bin/python3, el Python 3 del sistema.

Si yo ahora activo el directorio del entorno virtual y hago lo mismo, me dice que está ejecutando el Python 3 del entorno virtual, y esto nos permite hacer cambios en las versiones de Python sin que haya modificaciones generales en el sistema. Esto se mantendrá siempre. Desactivo y vuelvo a trabajar con el Python 3 del sistema general.

Entonces, ahora, por último, vamos a ver cómo instalar un módulo en nuestro entorno virtual, y para ello vamos a utilizar uno de los módulos que seguro que vamos a utilizar después para el desarrollo web, porque es el que nos da toda la estructura necesaria para las aplicaciones que vamos a desarrollar, que es el módulo Flask.

Si intentamos instalarlo, sin tener ningún entorno virtual activo, ¿qué va a ocurrir? Que no me deja porque no tengo ningún directorio virtual activo. Si activo el entorno virtual web, y lo repito, ahora sí que se conecta y sí que instala todo lo que necesita. Pero veis dónde lo está instalando, lo está instalando en el entorno virtual.

Para volver a la máquina virtual, primero hay que instalar virtualenvwrapper con apt-get install y luego en .bashrc y .profile agregar las líneas:

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/web
```

Y luego

```
workon web
```

Si ahora hago "ls .virtualenvs/web/lib", sigo teniendo Python3 pero, si veo los binarios, me ha metido Flask, que es lo que yo necesitaba, que le he dicho que lo instale. No lo ha instalado en el sistema, sino que ha bajado el módulo aquí.

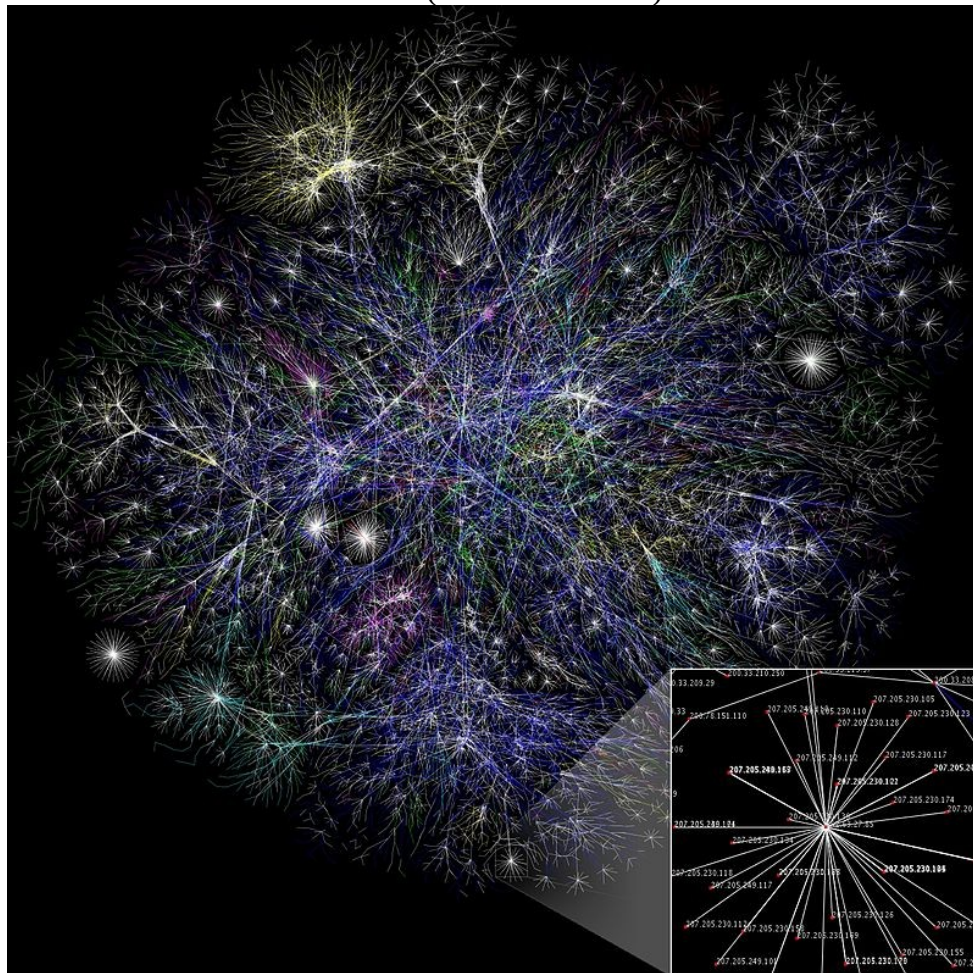
## **Unidad 1: WWW y sus aplicaciones**

### **Introducción**

"El internet (o, también, la internet) es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP/IP, lo cual garantiza que las redes físicas heterogéneas que la componen como una red lógica única de alcance mundial" (Wikipedia).

Internet ofrece una serie de servicios, siendo la World Wide Web (o Web o WWW) el de más éxito y conocido por todos. Pero Internet ofrece otros servicios o protocolos, como son: el correo electrónico (SMTP), la transmisión de archivos (FTP y P2P), las conversaciones en línea (IRC), el acceso remoto a otros dispositivos (SSH y Telnet), etc.

La siguiente imagen representa un mapa parcial de internet, donde cada línea entre dos nodos representa el enlace entre dos direcciones IP (Internet Protocol).



Para todos es habitual abrir un navegador y acceder a contenidos en la World Wide Web, también llamada web.

La web nos facilita poder buscar encontrar lo que buscamos, comunicarnos con amigos, realizar pagos...

Pero ¿cómo surgió la web? Su fundador es Sir Tim Berners-Lee.

En 1989 se encontraba realizando sus investigaciones en el CERN cuando propuso un lenguaje de marcado llamado HTML, un protocolo de comunicación llamado HTTP y una forma de acceder a cualquier elemento en internet llamado URL.

El primer sitio web que se creó con esta convención fue el del mismo CERN en 1991.

Veamos más en detalles estos tres elementos nombrados.

¿Qué es un URL?

URL son las siglas de Uniform Resource Locator, es decir, es la forma de poder acceder a cualquier fichero de forma internacional en cualquier sitio en internet.

Veamos el ejemplo del url de acceso al servicio web de la Universidad Autónoma de Madrid, con [www.uam.es](http://www.uam.es).

¿Y qué significa HTTP?

HTTP son las siglas de HyperText Transfer Protocol, es decir, protocolo de transferencia de datos entre el cliente web y el servidor web. Actualmente, dicho protocolo se está utilizando de forma segura, a lo que se denomina HTTPS.

¿Y qué es HTML?

HTML son las siglas de HyperText Markup Language, lenguaje de marcado hipertextual. Con este lenguaje vamos a poder crear documentos multimedia enlazados entre sí, donde vamos a poder tener texto, vídeos, imágenes...

Actualmente utilizamos la versión HTML5, previamente se estaban utilizando los estándares HTML 4.0 y HTML 4.01.

¿Cómo se gestiona la evolución de la Web?

Existe un consorcio internacional llamado Consorcio Web o World Wide Web Consortium que se dedica a hacer todos esos estándares y recomendaciones necesarios para poder gestionar una evolución a largo plazo de la web.

A modo de resumen, la web es un conjunto de servidores distribuidos por todo el mundo que dan respuesta a multitud de clientes. Por lo tanto, sigue una estructura cliente-servidor, en la cual el cliente introduce URL o link del servidor al que se quiere conectar, se genera una petición HTTP, la cual es contestada por el servidor, mandando al cliente una página codificada en HTML.



### **Historia de internet**

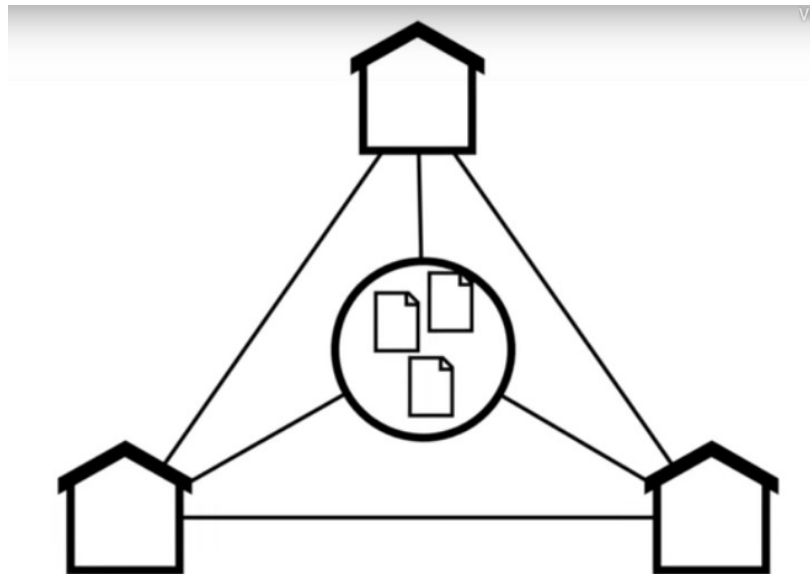
Todo empezó en 1957. Antes de ese año, las computadoras hacían una sola tarea a la vez (procesamiento por lotes, *batch processing*), y como cada vez se hacían más grandes para aumentar su velocidad, se hizo necesario almacenarlas en salas refrigeradas.

Esto hizo que los desarrolladores no pudiesen trabajar directamente en las computadoras, sino que se requerían especialistas para conectarlas. Programar era una tarea manual y el hecho de requerir un intermediario entre computadora y programador hacía que hubiesen cantidad de “bugs” y pérdida de tiempo.

Esto cambió en 1957 cuando se creó la posibilidad de conectarse de forma remota a éstas supercomputadoras, y al mismo tiempo, surgió la idea de tiempo compartido. Éste es el primer concepto en informática que comparte la capacidad de procesamiento de una computadora con varios usuarios.

El 4 de octubre de 1957 durante la Guerra Fría la Unión Soviética lanzó el Sputnik 1, lo cual lanzó un miedo de tener un “missile gap”. Con el fin de que Estados Unidos mantuviese el liderazgo tecnológico, fundaron DARPA (Agencia de Investigación de Proyectos Avanzados en Defensa, *Defense Advanced Technology Research Project Agency*) en febrero de 1958.

En ese momento, los conocimientos solo se daban de persona a persona (o de ordenador a ordenador), entonces DARPA planeó una red de computadoras de gran envergadura con el fin de acelerar la transferencia de conocimiento y evitar la duplicación de investigaciones existentes, la cual se llamó ARPANET.



Por otra parte, se desarrollaron otros 3 conceptos, el concepto RAND que era una red militar de dicha compañía para defensa en Estados Unidos, la red comercial del Laboratorio Nacional de Física (NPL) de Inglaterra y la red científica Cyclades de Francia.

Los enfoques científicos, de comercio y militares del concepto de red de computadoras, son la base del concepto actual de Internet.

### Arpanet

Su desarrollo comenzó en 1966. Las universidades eran reacias a compartir sus computadoras, por lo cual ponían pequeñas computadoras en frente de la unidad central. Esta computadora denominada Procesador de Mensajes (**IMP**, *Interface Message Processor*) tomaba el control de las actividades en la red mientras que la unidad central solo se encargaba de la inicialización de programas y archivos. Al mismo tiempo, la IMP servía únicamente como interfaz de la unidad central.

Como sólo los IMPs estaban interconectados en una red, a esto se lo llamó sub-red IMP. Para las primeras conexiones entre computadoras el *Network Working Group* desarrolló el **Protocolo de Control de Redes (NCP)**.

Más tarde NCP fue reemplazado por el **Protocolo de Control de Transmisión (TCP)**, el cual es más eficiente y se caracteriza por la verificación de transferencia de archivos.

Como la NPL se diseñó con fines comerciales, se esperaba una gran cantidad de usuarios y de transferencia de datos. A fin de evitar colisiones entre las líneas, los archivos se dividían en paquetes pequeños los cuales unía el receptor. Ésto es el nacimiento del concepto de **Conmutación de Paquetes** (*Packet-switching*).

En 1962, Estados Unidos detectó misiles de mediano y largo alcance en Cuba, capaces de alcanzar territorio americano, lo cual avivó el miedo a un conflicto atómico. En esa época, los sistemas de información se basaban en una arquitectura de red centralizada.

Para evitar una avería general durante un ataque, había que diseñar una arquitectura de red descentralizada, la cual pudiese seguir funcionando aunque fallasen algunos nodos. La comunicación se hacía por ondas de radio, lo cual hubiera causado problemas con un ataque

atómico pues las ondas de radio, no llegarían al receptor dado que la ionósfera se vería afectada y las ondas largas no llegarían. Por lo tanto, se debían usar ondas directas de radio pero éstas son de corto alcance.

Una mejor solución era el modelo de red distribuida, así largas distancias podían cubrirse con un mínimo de intrferencia.

### **Cyclades**

Como Cyclades tenía menos presupuesto de ARPANET, y por ende, con menos nodos, se enfocó en el desarrollo de las comunicaciones con otras redes y de allí surgió el término inter-net.

Además el concepto de Cyclades iba más allá que el de ARPANET y NPL. Durante la comunicación entre emisor y receptor las computadoras no debían intervenir, solo servir como nodo de transmisión.

El protocolo Cyclades pasaba por cada máquina usando una capa física implementada directamente en el hardware, proporcionando una conexión directa con el receptor en una estructura de punta a punta.

### **Internet**

Inspirados por la red Cyclades e impulsados por la incompatibilidad entre redes, su interconexión ganó popularidad en todos lados.

Las compañías telefónicas desarrollaron el protocolo X.25, que permitía la comunicación a través de sus servidores, por un determinado precio.

El protocolo TCP de DARPA debía conectar las computadoras a través de pasarelas (Gateways), y la Organización Internacional de Estandarización (ISO) diseñó el modelo de referencia OSI (*Open System Interconnection*).

La innovación de OSI era el intento de estandarizar la red desde sus extremos y la separación del canal en capas separadas.

Finalmente TCP asimiló las recomendaciones del modelo de referencia OSI y dio paso al protocolo TCP/IP, el cual es un estándar que garantizaba la compatibilidad entre redes y al final las unió, creando así internet.

El 28 de febrero de 1990, el hardware de ARPANET se retiró pero internet estaba en marcha

### **Las características de una aplicación Web**

Llamamos aplicación web o, de forma abreviada, web app, a la aplicación informática que un usuario puede utilizar accediendo a través de internet a un servidor web, y para ello hace uso de un cliente web o navegador.

Estas aplicaciones han cobrado gran popularidad ya que los usuarios pueden acceder con cualquier dispositivo. Simplemente con tener en el dispositivo un navegador o browser tipo Google Chrome, vamos a poder acceder al servidor web que nos la facilita.

¿Cómo es el funcionamiento de comunicación entre el cliente web y el servidor web?

El usuario introduce en su navegador, que es un cliente web, el URL o link para acceder al servidor web.

En ese momento se va a generar una petición HTTP desde el lado cliente al servidor, y el servidor le va a contestar, le va a responder con un documento web.

Dicho documento será visualizado en ese momento por el cliente, es decir, por el navegador web.

Por tanto, si estamos interesados en hacer una aplicación web, tenemos que tener en cuenta que vamos a tener lógica de programación tanto en el lado cliente como en el lado servidor.

Por un lado tenemos la interfaz de usuario, que está en el lado cliente, llamada en inglés Graphical User Interface, que la va a mostrar el navegador o browser al usuario de la aplicación. Y por otro lado tenemos los datos de la aplicación, que residen en el lado servidor.

Y vamos a distribuir la lógica de la aplicación entre lado cliente y lado servidor.

Para ello podemos hacer dos opciones.

- Tener la mayoría de la lógica de programación en el lado servidor, y por lo tanto tendremos un servidor pesado, es decir, también un cliente ligero.
- Tener parte de nuestra lógica de programación en el lado cliente, tendremos lo que llamamos un cliente pesado, que proviene del inglés flat client.

¿Qué lenguaje o lenguajes podemos utilizar para hacer nuestra aplicación web?

En primer lugar, todo documento web está creado con HTML; por lo tanto, será el primer lenguaje con el que vamos a trabajar.

Y seguidamente vamos a dar formato a cualquier documento con CSS, siglas en inglés que significan Cascading Style Sheets.

Pues bien, HTML junto con CSS van a ser la base de todo documento o página web de nuestra aplicación web.

Seguidamente tenemos que elegir cuál es el lenguaje de alto nivel, lenguaje que vamos a utilizar para la lógica de programación en el lado servidor. Podemos elegir, por ejemplo, PHP o Perl o Python. En nuestro caso, para este curso hemos elegido Python.

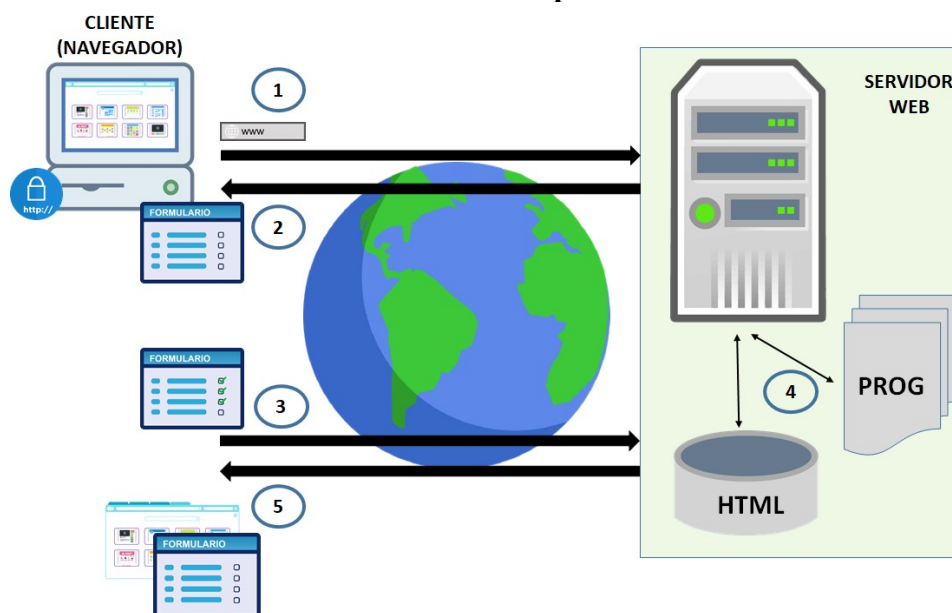
Python es un lenguaje de programación de alto nivel, de propósito general, que se puede utilizar o bien para temas de ciberseguridad, análisis de datos...

Y por este motivo, porque es un lenguaje muy utilizado, hemos pensado que es el ideal para este curso.

Como he comentado, la aplicación web puede necesitar manejar datos. Para este curso hemos elegido JSON, que es una notación de texto ligero para el intercambio de datos. Con el fin de dar cierta interactividad a nuestra aplicación y dar cierta lógica de programación en el lado cliente, vamos a utilizar Javascript, cuyo acrónimo es JS. Es un lenguaje interpretado por el navegador web.



## **Funcionamiento del modelo cliente-servidor de las aplicaciones Web**



1. El usuario, a través de su cliente Web (navegador), escribe la dirección (URL) de acceso a la aplicación web, entonces se lanza una petición HTTP del cliente al servidor.
2. El servidor envía un documento HTML que contiene normalmente un formulario que da acceso a interactuar con la aplicación Web.
3. El usuario podrá entonces rellenar el formulario recibido y dar a enviar. Se mandan los datos introducidos por el usuario desde el cliente al servidor Web.
4. El servidor web ejecuta los programas oportunos en función de los datos recibidos y genera otro documento HTML resultante de la programación de la aplicación web. En nuestro caso, tendremos en el servidor web la lógica de programación realizada con Python.
5. El nuevo documento generado por la aplicación web es mandado al cliente. Este documento puede a su vez volver a contener otro formulario, lo que facilitaría que cliente y servidor web sigan interactuando.

## **Unidad 2: La web interactiva**

### **Introducción**

Ahora vamos a empezar a crear nuestra aplicación web desde cero.

Para ello, vamos a empezar con el lenguaje de marcado HTML.

Vamos a crear formularios web, mediante los cuales vamos a facilitar que el usuario de nuestra aplicación interactúe con ella y seguidamente le daremos formato con CSS.

### **Introducción a HTML**

HTML son las siglas en inglés de HyperText Markup Language, lenguaje de marcado de hipertexto.

Se llama así porque se escribe utilizando etiquetas, etiquetas en las cuales utilizaremos los símbolos de menor (<), mayor (>) y en algunos casos la barra inclinada (/).

Es un estándar a cargo del W3C, una organización a cargo de la estandarización y las recomendaciones de toda tecnología relacionada con la web.

Actualmente utilizamos la versión quinta de HTML, es decir, HTML5.

HTML consta de varios componentes, de los cuales vamos a destacar los elementos y sus atributos.

Veamos un ejemplo de documento HTML, o documento web, muy sencillo, el cual lo visualiza el navegador de la siguiente manera.



Los elementos son la estructura básica de HTML. Dichos elementos tienen atributos y contenido, y además tienen una etiqueta de inicio y una etiqueta de fin.

Veamos, en el ejemplo, h1. Tenemos la etiqueta h1 inicio (<h1> ), h1 final (</h1>) y en medio tenemos el contenido. Dicho contenido es un título que aparecerá como cabecera de nivel 1.

En general, los atributos están formados por nombre-valor, separados por el signo de igual (=), y puede tener el valor entre comillas sencillas, o comillas dobles, o no necesitarlas. En el ejemplo, un atributo es <html lang = > y su valor es "es" con colillas.

Todo documento web empieza con la indicación del tipo de documento que es, es decir, con <!DOCTYPE html>

Seguidamente, una parte para la cabecera, entre las etiquetas y , donde vamos a tener metadatos como, por ejemplo, el título del documento.

Y seguidamente tendremos el contenido del documento propiamente dicho, entre las etiquetas de body. Algunos de los elementos más utilizados en los documentos web son, por ejemplo: p para párrafo, img para imagen, y a para link.

### El lenguaje HTML

os elementos del lenguaje HTML se especifican entre etiquetas (tags) de inicio (< >) y fin (</ >).

Un fichero HTML (con extensión .html o .htm), estará formado por texto plano, pero podrá mostrar todo tipo de elementos, incluidos multimedia:


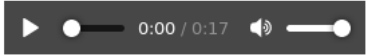
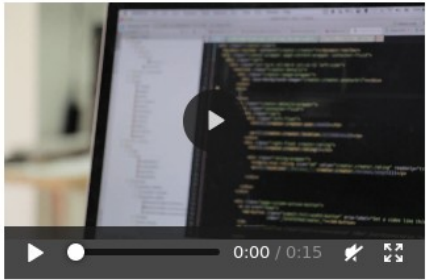
- Párrafos: <p> </p>

- Encabezados: `<h1>` `</h1>`
- Imágenes: `<img/>`
- Vídeos: `<video>` `</video>`
- Vínculos o enlaces: `<a>` `</a>`

### **Elementos HTML**

La estructura básica de HTML son los elementos, los cuales tienen dos propiedades básicas: atributos y contenido. Un elemento generalmente tiene una etiqueta de inicio (por ejemplo, `<h1>`) y una etiqueta de cierre (por ejemplo, `</h1>`).

El último estándar aceptado por la W3C es HTML5. La W3C nos ofrece un listado de los elementos que podemos utilizar en nuestros documentos HTML. Los elementos más utilizados a modo de resumen son:

Etiqueta	Descripción	Código	Visualización
<HTML> </HTML>	Documento HTML.	Mira el código de la página anterior.	
<HEAD> </HEAD>	Cabecera del documento. Habitualmente tendrá título en <TITLE></TITLE>	Mira el código de la página anterior.	
<BODY> </BODY>	Cuerpo del documento.	Mira el código de la página anterior.	
<!--...-->	Comentario.	<!-- esto es un comentario-->	No se ve en la página que visualiza el navegador, ¡es un comentario!
<P> </P>	Párrafo.	<P>Párrafo 1</P> <P>Párrafo 2</P>	Párrafo 1 Párrafo 2
<H1> </H1> ... <H6> </H6>	Encabezados de diferentes niveles.	<h1>Encabezado nivel 1</h1>	Encabezado nivel 1
<A HREF="LINK" ...> </A>	Hiperenlace (se redirige a link indicado en el atributo HREF).	<a href="https://www.uam.es">Universidad Autónoma de Madrid</a>	<a href="https://www.uam.es">Universidad Autónoma de Madrid</a>
<DIV> </DIV>	División en el documento; lo habitual es darle un nombre.	<DIV="cabecera"> <p> Párrafo en la cabecera de la página.</p></div>	Párrafo en la cabecera de la página.
<IMG> </IMG>	Imagen: se indica la imagen en el atributo SRC.	<IMG SRC="http://www.uam.es/UAM/imagen/1233310437165/LOGO_UAM.png"></IMG>	
<AUDIO> </AUDIO>	Audio: se indica el audio en el atributo SRC de elemento SOURCE.	<audio controls><source src="http://soundbible.com/grab.php?id=2213&type=mp3" ></audio>	
<VIDEO> </VIDEO>	Vídeo: se indica el video en el atributo SRC de elemento SOURCE.	<video width="320" height="240" controls><source SRC="//prod-edxapp.edx-cdn.org/assets/courseware/v1/13e4c857c2ac6c21c7a0cd06fb9424cb/asset-v1:UAMx+WebApp+1T2019a+type@asset+block/Pexels_Videos_2833.mp4" type="video/mp4"></video>	
<FORM> </FORM>	Formulario: veremos sus detalles a lo largo de esta unidad.	Veremos el código más adelante.	

Otros elementos son:

### Listas:

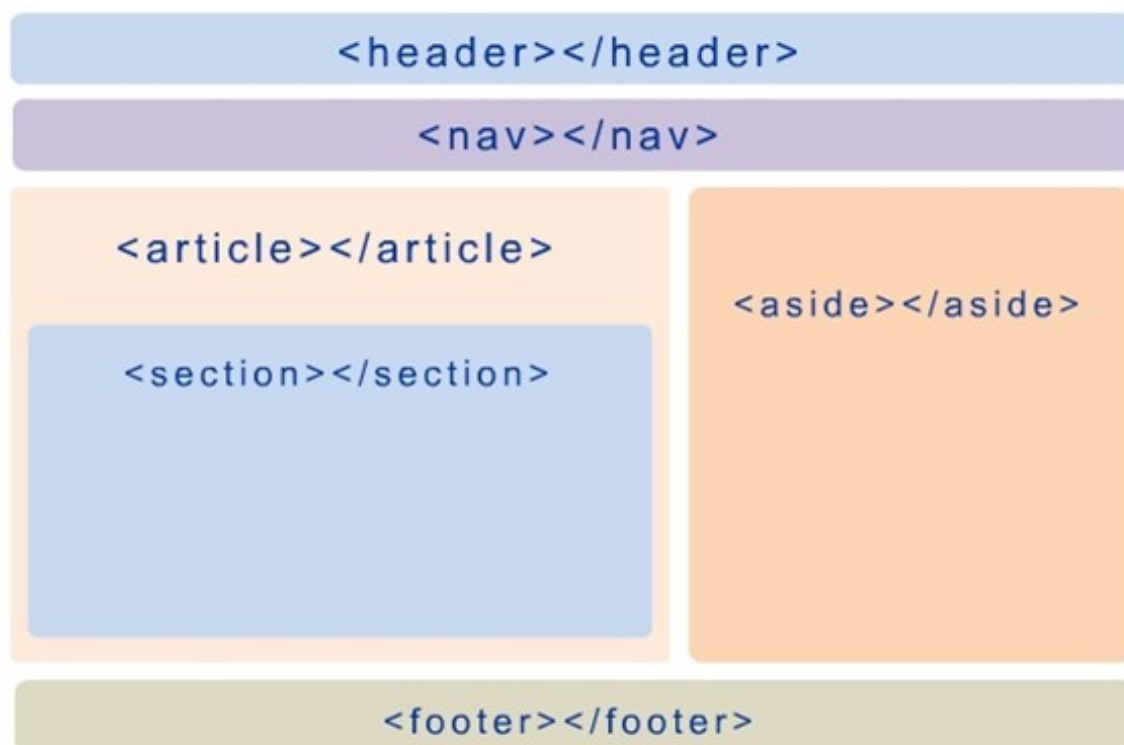
- Numeradas <OL> </OL>

- No numeradas <UL> </UL>
- Cada elemento en lista (numerada o no numerada) se define con: <LI> </LI>.

**Tablas.** Se define con <TABLE> </TABLE>.

- Fila se define con: <TR> </TR>
- Celda de datos: <TD> </TD>
- Celda de cabecera: <TH> </TH>

A partir de la versión HTML5 aparecen estas etiquetas para añadir diferentes secciones a los documentos HTML



### **Red Social: página de inicio**

Como ya te hemos adelantado, vamos a crear paso a paso una aplicación web que simula una red social muy sencilla a la que llamaremos de forma abreviada SocNet.

Dicha aplicación tendrá un documento HTML de inicio o entrada llamado index.html. Más adelante te damos las instrucciones de su contenido, pero es muy importante que dicho fichero lo pongas en el directorio correspondiente:

En linux: moocwebapp/mooc/U2/app/static

El código:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8"/><script
type="text/javascript">(window.NREUM||
(NREUM={})).loader_config={xpid:"XA4GVl5ACwAEV1JQAA=="};window.NRE

```

```

UM||(NREUM={}),__nr_require=function(t,n,e){function r(e){if(!
n[e]){var o=n[e]={exports:{}};t[e][0].call(o.exports,function(n)
{var o=t[e][1][n];return r(o)||n}),o,o.exports}}return
n[e].exports}if("function"==typeof __nr_require)return
__nr_require;for(var o=0;o<e.length;o++)r(e[o]);return r}({1:
[function(t,n,e){function r(t)
{try{s.console&&console.log(t)}catch(n){}}var
o,i=t("ee"),a=t(18),s={};try{o=localStorage.getItem("__nr_flags").
split(", "),console&&"function"==typeof console.log&&(s.console=!
0,o.indexOf("dev")!==-1&&(s.dev=!0),o.indexOf("nr_dev")!==-
1&&(s.nrDev=!0))}catch(c){}s.nrDev&&i.on("internal-
error",function(t){r(t.stack)}),s.dev&&i.on("fn-
err",function(t,n,e){r(e.stack)}),s.dev&&(r("NR AGENT IN
DEVELOPMENT MODE"),r("flags: "+a(s,function(t,n){return
t}).join(", "))))},{}],2:[function(t,n,e){function r(t,n,e,r,s)
{try{p?p-=1:o(s||new UncaughtException(t,n,e),!0)}catch(f)
{try{i("ierr",[f,c.now(),!0])}catch(d){}}return"function"==typeof
u&&u.apply(this,a(arguments))}function UncaughtException(t,n,e)
{this.message=t||"Uncaught error with no additional
information",this.sourceURL=n,this.line=e}function o(t,n){var e=n?
null:c.now();i("err",[t,e])}var
i=t("handle"),a=t(19),s=t("ee"),c=t("loader"),f=t("gos"),u=window.
onerror,d=!1,l="nr@seenError",p=0;c.features.err=!
0,t(1),window.onerror=r;try{throw new Error}catch(h){"stack"in
h&&(t(8),t(7),"addEventListener"in
window&&t(5),c.xhrWrappable&&t(9),d=!0)}s.on("fn-
start",function(t,n,e){d&&(p+=1)}),s.on("fn-err",function(t,n,e)
{d&&!e[l]&&(f(e,l,function(){return!0}),this.thrown=!
0,o(e))}),s.on("fn-end",function(){d&&!this.thrown&&p>0&&(p-
=1)}),s.on("internal-error",function(t){i("ierr",[t,c.now(),!
0]))}),{}],3:[function(t,n,e){t("loader").features.ins=!0},{}],4:
[function(t,n,e){function r(t)
{}if(window.performance&&window.performance.timing&&window.perform
ance.getEntriesByType){var
o=t("ee"),i=t("handle"),a=t(8),s=t(7),c="learResourceTimings",f="a
ddEventListener",u="resourcetimingbufferfull",d="bstResource",l="r
esource",p="-start",h="-
end",m="fn"+p,w="fn"+h,v="bstTimer",y="pushState",g=t("loader");g.
features.stn=!0,t(6);var x=NREUM.o.EV;o.on(m,function(t,n){var
e=t[0];e instanceof
x&&(this.bstStart=g.now())}),o.on(w,function(t,n){var e=t[0];e
instanceof x&&i("bst",
[e,n,this.bstStart,g.now()]))},a.on(m,function(t,n,e)
{this.bstStart=g.now(),this.bstType=e}),a.on(w,function(t,n){i(v,
[n,this.bstStart,g.now(),this.bstType]}),s.on(m,function()
{this.bstStart=g.now()}),s.on(w,function(t,n){i(v,
[n,this.bstStart,g.now(),"requestAnimationFrame"])}),o.on(y+p,func
tion(t)
{this.time=g.now(),this.startPath=location.pathname+location.hash}
),o.on(y+h,function(t){i("bstHist",
[location.pathname+location.hash,this.startPath,this.time]))},f in

```

```

window.performance&&(window.performance["c"+c]?
window.performance[f](u,function(t){i(d,
[window.performance.getEntriesByType(l)]),window.performance["c"+c
]()),!1):window.performance[f]("webkit"+u,function(t){i(d,
[window.performance.getEntriesByType(l)]),window.performance["webk
itC"+c]()),!1)),document[f]("scroll",r,{passive:!0}),document[f]
("keypress",r,!1),document[f]("click",r,!1)}},{},5:
[function(t,n,e){function r(t){for(var n=t;n&&!
n.hasOwnProperty(u);)n=Object.getPrototypeOf(n);n&&o(n)}function
o(t){s.inPlace(t,[u,d],"-",i)}function i(t,n){return t[1]}var
a=t("ee").get("events"),s=t(21)(a,!
0),c=t("gos"),f=XMLHttpRequest,u="addEventListener",d="removeEvent
Listener";n.exports=a,"getPrototypeOf"in Object?
(r(document),r(window),r(f.prototype)):f.prototype.hasOwnProperty(
u)&&(o(window),o(f.prototype)),a.on(u+"-start",function(t,n){var
e=t[1],r=c(e,"nr@wrapped",function(){function t()
{if("function"==typeof e.handleEvent)return
e.handleEvent.apply(e,arguments)}var n={object:t,"function":e}
[typeof e];return n?
s(n,"fn-",null,n.name||"anonymous"):e});this.wrapped=t[1]=r}),a.on
(d+"-start",function(t){t[1]=this.wrapped||t[1]}),{}},6:
[function(t,n,e){var r=t("ee").get("history"),o=t(21)
(r);n.exports=r;var
i=window.history&&window.history.constructor&&window.history.const
ructor.prototype,a=window.history;i&&i.pushState&&i.replaceState&&
(a=i),o.inPlace(a,["pushState","replaceState"],"-" ),{}},7:
[function(t,n,e){var r=t("ee").get("raf"),o=t(21)
(r),i="requestAnimationFrame";n.exports=r,o.inPlace(window,
["r"+i,"mozR"+i,"webkitR"+i,"msR"+i],"raf-"),r.on("raf-
start",function(t){t[0]=o(t[0],"fn-")}),{}},8:[function(t,n,e)
{function r(t,n,e){t[0]=a(t[0],"fn-",null,e)}function o(t,n,e)
{this.method=e,this.timerDuration=isNaN(t[1])?
0:+t[1],t[0]=a(t[0],"fn-",this,e)}var
i=t("ee").get("timer"),a=t(21)
(i),s="setTimeout",c="setInterval",f="clearTimeout",u="-
start",d="-";n.exports=i,a.inPlace(window,
[s,"setImmediate"],s+d),a.inPlace(window,
[c],c+d),a.inPlace(window,
[f,"clearImmediate"],f+d),i.on(c+u,r),i.on(s+u,o)},{}},9:
[function(t,n,e){function r(t,n){d.inPlace(n,
["onreadystatechange"],"fn-",s)}function o(){var
t=this,n=u.context(t);t.readyState>3&&!n.resolved&&(n.resolved=!
0,u.emit("xhr-resolved",[],t)),d.inPlace(t,y,"fn-",s)}function
i(t){g.push(t),h&&(b?b.then(a):w?w(a):(E=-E,R.data=E))}function
a(){for(var t=0;t<g.length;t+
+)r([],g[t]);g.length&&(g=[])}function s(t,n){return n}function
c(t,n){for(var e in t)n[e]=t[e];return n}t(5);var
f=t("ee"),u=f.get("xhr"),d=t(21)
(u),l=NREUM.o,p=l.XHR,h=l.MO,m=l.PR,w=l.SI,v="readystatechange",y=
["onload","onerror","onabort","onloadstart","onloadend","onprogres
s","ontimeout"],g=[];n.exports=u;var

```

```

x=window.XMLHttpRequest=function(t){var n=new
p(t);try{u.emit("new-xhr",[n],n),n.addEventListener(v,o,!
1)}catch(e){try{u.emit("internal-error",[e])}catch(r){}}return
n};if(c(p,x),x.prototype=p.prototype,d.inPlace(x.prototype,
["open","send"],"-xhr-",s),u.on("send-xhr-start",function(t,n)
{r(t,n),i(n)}),u.on("open-xhr-start",r),h){var
b=m&&m.resolve();if(!w&&!m){var
E=1,R=document.createTextNode(E);new h(a).observe(R,
{characterData:!0})}}else f.on("fn-end",function(t)
{t[0]&&t[0].type===v||a()}),{}}],10:[function(t,n,e){function r()
{var t=window.NREUM,n=t.info.accountID||null,e=t.info.agentID||
null,r=t.info.trustKey||null,i="btoa"in window&&"function"===typeof
window.btoa;if(!n||!e||!i)return null;var a={v:[0,1],d:
{ty:"Browser",ac:n,ap:e,id:o.generateCatId(),tr:o.generateCatId(),
ti:Date.now()}};return r&&n!
==r&&(a.d.tk=r),btoa(JSON.stringify(a))}var
o=t(16);n.exports={generateTraceHeader:r}},{}}],11:[function(t,n,e)
{function r(t){var n=this.params,e=this.metrics;if(!this.ended)
{this.ended=!0;for(var r=0;r<p;r+
+)t.removeEventListener(l[r],this.listener,!1);n.aborted||
(e.duration=s.now()-this.startTime,this.loadCaptureCalled||4!
==t.readyState?
null==n.status&&(n.status=0):a(this,t),e.cbTime=this.cbTime,d.emit
("xhr-done",[t],t),c("xhr",[n,e,this.startTime]))}}function o(t,n)
{var e=t.responseType;if("json"===e&&null!==n)return n;var
r="arraybuffer"===e||"blob"===e||"json"===e?
t.response:t.responseText;return w(r)}function i(t,n){var
e=f(n),r=t.params;r.host=e.hostname+": "+e.port,r.pathname=e.pathna
me,t.sameOrigin=e.sameOrigin}function a(t,n)
{t.params.status=n.status;var
e=o(n,t.lastSize);if(e&&(t.metrics.rxSize=e),t.sameOrigin){var
r=n.getResponseHeader("X-NewRelic-App-
Data");r&&(t.params.cat=r.split(", ").pop())}t.loadCaptureCalled=!
0}var s=t("loader");if(s.xhrWrappable){var
c=t("handle"),f=t(12),u=t(10).generateTraceHeader,d=t("ee"),l=["lo
ad","error","abort","timeout"],p=l.length,h=t("id"),m=t(15),w=t(14
),v=window.XMLHttpRequest;s.features.xhr=!0,t(9),d.on("new-
xhr",function(t){var
n=this;n.totalCbs=0,n.called=0,n.cbTime=0,n.end=r,n.ended=!
1,n.xhrGuids={},n.lastSize=null,n.loadCaptureCalled=!
1,t.addEventListener("load",function(e){a(n,t)},!1),m&&(m>34||
m<10)||window.opera||t.addEventListener("progress",function(t)
{n.lastSize=t.loaded,!1})),d.on("open-xhr-start",function(t)
{this.params={method:t[0]},i(this,t[1]),this.metrics={}}),d.on("op
en-xhr-end",function(t,n){{"loader_config"in NREUM&&"xpid"in
NREUM.loader_config&&this.sameOrigin&&n.setRequestHeader("X-
NewRelic-ID",NREUM.loader_config.xpid);var e=!1;if("init"in
NREUM&&"distributed_tracing"in NREUM.init&&(e=!
NREUM.init.distributed_tracing.enabled),e&&this.sameOrigin){var
r=u();r&&n.setRequestHeader("newrelic",r)}}),d.on("send-xhr-
start",function(t,n){var e=this.metrics,r=t[0],o=this;if(e&&r){var

```



```

i=w(r);i&&(e.txSize=i)}this.startTime=s.now(),this.listener=function(t){try{"abort"!==t.type||o.loadCaptureCalled||
(o.params.aborted=!0),("load"!==t.type||
o.called===o.totalCbs&&(o.onloadCalled||"function"!==typeof
n.onload))&&o.end(n)}catch(e){try{d.emit("internal-error",
[e])}catch(r){}}};for(var a=0;a<p;a+
+)n.addEventListener(l[a],this.listener,!1)),d.on("xhr-cb-
time",function(t,n,e){this.cbTime+=t,n?this.onloadCalled=!
0:this.called+=1,this.called!==this.totalCbs||!
this.onloadCalled&&"function"===typeof e.onload||
this.end(e)}),d.on("xhr-load-added",function(t,n){var e=""+"h(t)+!!
n;this.xhrGuids&&!this.xhrGuids[e]&&(this.xhrGuids[e]=!
0,this.totalCbs+=1)}),d.on("xhr-load-removed",function(t,n){var
e=""+"h(t)+!!n;this.xhrGuids&&this.xhrGuids[e]&&(delete
this.xhrGuids[e],this.totalCbs-=1)}),d.on("addEventListener-
end",function(t,n){n instanceof v&&"load"===t[0]&&d.emit("xhr-
load-added",[t[1],t[2]],n)}),d.on("removeEventListener-
end",function(t,n){n instanceof v&&"load"===t[0]&&d.emit("xhr-
load-removed",[t[1],t[2]],n)}),d.on("fn-start",function(t,n,e){n
instanceof v&&("onload"===e&&(this.onload=!0),
("load"===t[0]&&t[0].type)||
this.onload)&&(this.xhrCbStart=s.now()))},d.on("fn-
end",function(t,n){this.xhrCbStart&&d.emit("xhr-cb-time",[s.now()-
this.xhrCbStart,this.onload,n],n)}),{}],12:[function(t,n,e)
{n.exports=function(t){var
n=document.createElement("a"),e=window.location,r={};n.href=t,r.po
rt=n.port;var o=n.href.split("://");!
r.port&&o[1]&&(r.port=o[1].split("/")[0].split("@").pop().split(":")[1]),r.port&&"0"!==r.port||
(r.port="https"===o[0]?"443":"80"),r.hostname=n.hostname||
e.hostname,r.pathname=n.pathname,r.protocol=o[0],"/"!
==r.pathname.charAt(0)&&(r.pathn
ame="/"+r.pathname);var i=!n.protocol||":"===n.protocol||
n.protocol===e.protocol,a=n.hostname===document.domain&&n.port===e
.port;return r.sameOrigin=i&&(!n.hostname||a),r}},{}],13:
[function(t,n,e){function r(){}function o(t,n,e){return function()
{return i(t,[f.now()].concat(s(arguments)),n?null:this,e),n?void
0:this}}var
i=t("handle"),a=t(18),s=t(19),c=t("ee").get("tracer"),f=t("loader"
),u=NREUM;"undefined"===typeof window.newrelic&&(newrelic=u);var
d=["setPageViewName","setCustomAttribute","setErrorHandler","finis
hed","addToTrace","inlineHit","addRelease"],l="api-",p=l+"ixn-";a(
d,function(t,n){u[n]=o(l+n,!
0,"api")}),u.addPageAction=o(l+"addPageAction",!
0),u.setCurrentRouteName=o(l+"routeName",!
0),n.exports=newrelic,u.interaction=function(){return(new
r).get()};var h=r.prototype={createTracer:function(t,n){var
e={},r=this,o="function"===typeof n;return i(p+"tracer",
[f.now(),t,e],r),function(){if(c.emit((o?"":"no-")+"fn-start",
[f.now(),r,o],e),o)try{return n.apply(this,arguments)}catch(t)
{throw c.emit("fn-err",[arguments,this,t],e),t}finally{c.emit("fn-

```

```

end",
[f.now()],e)}]}]};a("actionText,setName,setAttribute,save,ignore,on
End,getContext,end,get".split(","),function(t,n)
{h[n]=o(p+n)}),newrelic.noticeError=function(t,n){"string"==typeof
t&&(t=new Error(t)),i("err",[t,f.now(),!1,n]}),{}},14:
[function(t,n,e){n.exports=function(t){if("string"==typeof
t&&t.length)return t.length;if("object"==typeof t){if("undefined"!
=typeof ArrayBuffer&&t instanceof ArrayBuffer&&t.byteLength)return
t.byteLength;if("undefined"!=typeof Blob&&t instanceof
Blob&&t.size)return t.size;if(!("undefined"!=typeof FormData&&t
instanceof FormData))try{return JSON.stringify(t).length}catch(n)
{return}}}},{}},15:[function(t,n,e){var
r=0,o=navigator.userAgent.match(/Firefox[\\\/\s](\d+\.\
d+)/);o&&(r+=o[1]),n.exports=r},{}},16:[function(t,n,e){function
r(){function t(){return n?15&n[e++]:16*Math.random()|0}var
n=null,e=0,r=window.crypto||
window.msCrypto;r&&r.getRandomValues&&(n=r.getRandomValues(new
Uint8Array(31)));for(var o,i="xxxxxxxx-xxxx-4xxx-yxxx-
xxxxxxxxxxxx",a="",s=0;s<i.length;s++)o=i[s],"x"===o?
a+=t().toString(16):"y"===o?(o=3&t()|
8,a+=o.toString(16)):a+=o;return a}function o(){function t()
{return n?15&n[e++]:16*Math.random()|0}var
n=null,e=0,r=window.crypto||
window.msCrypto;r&&r.getRandomValues&&Uint8Array&&(n=r.getRandomVa
lues(new Uint8Array(31)));for(var o=[],i=0;i<16;i+
+)o.push(t().toString(16));return
o.join("")}n.exports={generateUuid:r,generateCatId:o}},{}},17:
[function(t,n,e){function r(t,n){if(!o)return!1;if(t!==o)return!
1;if(!n)return!0;if(!i)return!1;for(var
e=i.split("."),r=n.split("."),a=0;a<r.length;a++)if(r[a]!
==e[a])return!1;return!0}var o=null,i=null,a=/Version\/(\S+)\
sSafari/;if(navigator.userAgent){var
s=navigator.userAgent,c=s.match(a);c&&s.indexOf("Chrome")===-
1&&s.indexOf("Chromium")===-
1&&(o="Safari",i=c[1])}n.exports={agent:o,version:i,match:r}},
{}},18:[function(t,n,e){function r(t,n){var e=[],r="",i=0;for(r in
t)o.call(t,r)&&(e[i]=n(r,t[r]),i+=1);return e}var
o=Object.prototype.hasOwnProperty;n.exports=r},{}},19:
[function(t,n,e){function r(t,n,e){n||(n=0),"undefined"==typeof
e&&(e=t?t.length:0);for(var r=-1,o=e-n||0,i=Array(o<0?0:o);+
+r<o;)i[r]=t[n+r];return i}n.exports=r},{}},20:[function(t,n,e)
{n.exports={exists:"undefined"!=typeof
window.performance&&window.performance.timing&&"undefined"!=typeof
window.performance.timing.navigationStart}},{}},21:
[function(t,n,e){function r(t){return!(t&&t instanceof
Function&&t.apply&&!t[a])}var
o=t("ee"),i=t(19),a="nr@original",s=Object.prototype.hasOwnPropert
y,c=!1;n.exports=function(t,n){function e(t,n,e,o){function
nrWrapper(){var
r,a,s,c;try{a=this,r=i(arguments),s="function"==typeof e?
e(r,a):e||{}}catch(f){l([f,"",[r,a,o],s])}u(n+"start",

```

```

[r,a,o],s);try{return c=t.apply(a,r)}catch(d){throw u(n+"err",
[r,a,d],s),d}finally{u(n+"end",[r,a,c],s)}}return r(t)?t:(n||
(n=""),nrWrapper[a]=t,d(t,nrWrapper),nrWrapper)}function
f(t,n,o,i){o||(o="");var
a,s,c,f="-"===o.charAt(0);for(c=0;c<n.length;c+
+)s=n[c],a=t[s],r(a)||(t[s]=e(a,f?s+o:o,i,s))}function u(e,r,o)
{if(!c||n){var i=c;c=!0;try{t.emit(e,r,o,n)}catch(a)
{l([a,e,r,o]))c=i}}function d(t,n)
{if(Object.defineProperty&&Object.keys)try{var
e=Object.keys(t);return e.forEach(function(e)
{Object.defineProperty(n,e,{get:function(){return
t[e]},set:function(n){return t[e]=n,n}})}),n}catch(r)
{l([r]))}for(var o in t)s.call(t,o)&&(n[o]=t[o]);return n}function
l(n){try{t.emit("internal-error",n)}catch(e){}}return t||
(t=o),e.inPlace=f,e.flag=a,e}},{}],ee:[function(t,n,e){function
r(){}function o(t){function n(t){return t&&t instanceof r?t:t?
c(t,s,i):i()}function e(e,r,o,i){if(!l.aborted||i)
{t&&t(e,r,o);for(var a=n(o),s=m(e),c=s.length,f=0;f<c;f+
+)s[f].apply(a,r);var d=u[g[e]];return
d&&d.push([x,e,r,a]),a}}function p(t,n)
{y[t]=m(t).concat(n)}function h(t,n){var e=y[t];if(e)for(var
r=0;r<e.length;r++)e[r]===n&&e.splice(r,1)}function m(t){return
y[t]||[]}function w(t){return d[t]=d[t]||o(e)}function v(t,n)
{f(t,function(t,e){n=n||"feature",g[e]=n,n in u||(u[n]=[])})}var
y={},g={},x={on:p,addEventListener:p,removeEventListener:h,emit:e,
get:w,listeners:m,context:n,buffer:v,abort:a,aborted:!1};return
x}function i(){return new r}function a(){(u.api||
u.feature)&&(l.aborted=!0,u=l.backlog={})}var
s="nr@context",c=t("gos"),f=t(18),u={},d={},l=n.exports=o();l.back
log=u,{}],gos:[function(t,n,e){function r(t,n,e)
{if(o.call(t,n))return t[n];var
r=e();if(Object.defineProperty&&Object.keys)try{return
Object.defineProperty(t,n,{value:r,writable:!0,enumerable:!
1}),r}catch(i){}return t[n]=r,r}var
o=Object.prototype.hasOwnProperty;n.exports=r,{}],handle:
[function(t,n,e){function r(t,n,e,r)
{o.buffer([t],r),o.emit(t,n,e)}var
o=t("ee").get("handle");n.exports=r,r.ee=o},{}],id:
[function(t,n,e){function r(t){var n=typeof t;return!t||"object"!
==n&&"function"!==n?-1:t===window?0:a(t,i,function(){return o+
+})}var o=1,i="nr@id",a=t("gos");n.exports=r,{}],loader:
[function(t,n,e){function r(){if(!E++){var
t=b.info=NREUM.info,n=p.getElementsByTagName("script")
[0];if(setTimeout(u.abort,3e4),!
(t&&t.licenseKey&&t.applicationID&&n))return
u.abort();f(g,function(n,e){t[n]||(t[n]=e)),c("mark",
["onload",a()+b.offset,null,"api"]);var
e=p.createElement("script");e.src="https://"+t.agent,n.parentNode.
insertBefore(e,n)}}function o()
{"complete"===p.readyState&&i()}function i(){c("mark",
["domContentLoaded",a()+b.offset,null,"api"]}function a(){return

```

```

R.exists&&performance.now?Math.round(performance.now()):
(s=Math.max((new Date).getTime(),s))-b.offset}var s=(new
Date).getTime(),c=t("handle"),f=t(18),u=t("ee"),d=t(17),l=window,p
=l.document,h="addEventListener",m="attachEvent",w=l.XMLHttpRequest,
v=w&&w.prototype;NREUM.o={ST:setTimeout,SI:l.setImmediate,CT:clearTimeout,XHR:w,REQ:l.Request,EV:l.Event,PR:l.Promise,MO:l.MutationObserver};var y="" + location,g={beacon:"bam.nr-data.net",errorBeacon:"bam.nr-data.net",agent:"js-agent.newrelic.com/nr-1130.min.js"},x=w&&v&&v[h]&&!/CriOS/.test(navigator.userAgent),b=n.exports={offset:s,now:a,origin:y,features:{},xhrWrappable:x,userAgent:d};t(13),p[h]?(p[h]("DOMContentLoaded",i,!1),l[h]("load",r,!1)):(p[m]("onreadystatechange",o),l[m]("onload",r)),c("mark",["firstbyte",s],null,"api");var E=0,R=t(20)},{},{}},{"loader",2,11,4,3)};</script><script type="text/javascript">window.NREUM||(NREUM={});NREUM.info={"beacon":"bam.nr-data.net","queueTime":2,"licenseKey":"1beac94c95","agent":"","transactionName":"NQQGYUZWDUNSVUMPWQx0IkBaVBdZXFGYCUYHDwFRTBkAX0FTGQJcAw8DWlVHE0MdVVGiQgcPEEZRRRVVQRhaD1IGDQFCVUUGCmBCVhJfASILW0BSDURGu0UQUxBPFEdbVAZDQGLFA0cXBBdB","atts":"GUMRFw5MQUJWR0IDRRY+F1BSUhFVQRQNRf4WFRRGDhhMU1xDRRVTEU8BUUwZDEJUGVQJQxASAUybVAXFQUVSS0BTWzF0eU9IZ1ZUdhZGSVAwBwQGWlEcVVGTRBEE1RGUkwIUlAGB1cDWAEGBgJXAVcGD1MBBgJQDQIAVwYDBlUHD00EAQAFULICALNVVFBWUgUFDVQCCFAEAQQCAwBdBQYFux8ECVYFQgsXBUFRaAFcXFVcOV8GXAZZW1QIHUUHElV3NyApTREFIWdWVHYWRkdTJgRgBVMBClcSVHQWGBRQEQTWEdbw0MEIAMiWldcRgQDBlZWVABVB1dRALpTBwUFXg8AWABXA1YCVAIBUQRUW1NGGRZFBkFGU0QSaRcSAUdrVgRVXUIVXBQvDh5cWFsCHwYYB0Ye0lBVDhR7Cl5GThceDlQ+UgEPFxFGCQAHSZLQSNQV1wMHwEGBLYGU1FVFXJeEVVWU9JAFJPVBcYFRZDVkRoD1JAW1YGDAFTAwAEG0REBxARUEdDPFFGQl85QhsRARc0FRBVQVEcVhPDhYYQVklXlxBWUQaQAILW0BSDURAU0UQUxBPB1RXXwZRUVpSRAwWExFQGBUAX11CUghCEQQWQ1FFTVNcWEMDWBY+CFBaFvKFCgEbRFUNDxBQWkMQVUFAUhQYEGaQXRYNQUUBaV4IUgcZ01xaXk1YR1tbRBpAAgtbQFINREBTRRBTEE8HWlpDBl5HaUMfRgdDXhdAUhtEHF5DC1pATUZHUUyWVUBCaBNFBxM7XFAVWQIADgFWBVFTSBdXWBZCQFNOD1JAW0ZWW0IRQ1YbQVcMnyApTR9gBlJyRkdNBzZTVAQNvKEcEVlFARRYQzF0eU9BHBFWWAhCBw8QRLFFVVBGFsJVQkEABCOUQJcQFMbRFUNDxBQWkMQVUFAUhQYBBMLWGtUB14RDFEHWhEEGUg=","applicationID":"3343327","errorBeacon":"bam.nr-data.net","applicationTime":19}</script>
<title>SocNet</title>
</head>
<body>
<div id="container">
<a url="/">SocNet | <a url="home">Home | <a url="login">Log In | <a url="signup">Sign Up
<h1>Hi, Welcome to the SocNet</h1>
<div>
<p>Choose <b>Sign Up</b> if you are a new user</p>
<p>Choose <b>Log In</b> if you are user of
SocNet</p>
<p>Choose <b>Home</b> to start to write Posts</p>
</div>
</div>

```

```
</body>  
</html>
```

Los problemas que tiene son 2:

La etiqueta `<a>` no se cierra

El nombre del atributo `<a>` no es correcto, por lo que hay que sustituir la línea:

```
<a url="/">SocNet | <a url="home">Home | <a url="login">Log In |  
<a url="signup">Sign Up
```

Por:

```
<a href="/">SocNet</a> | <a href="home">Home</a> | <a  
href="login">Log In</a> | <a href="signup">Sign Up</a>
```

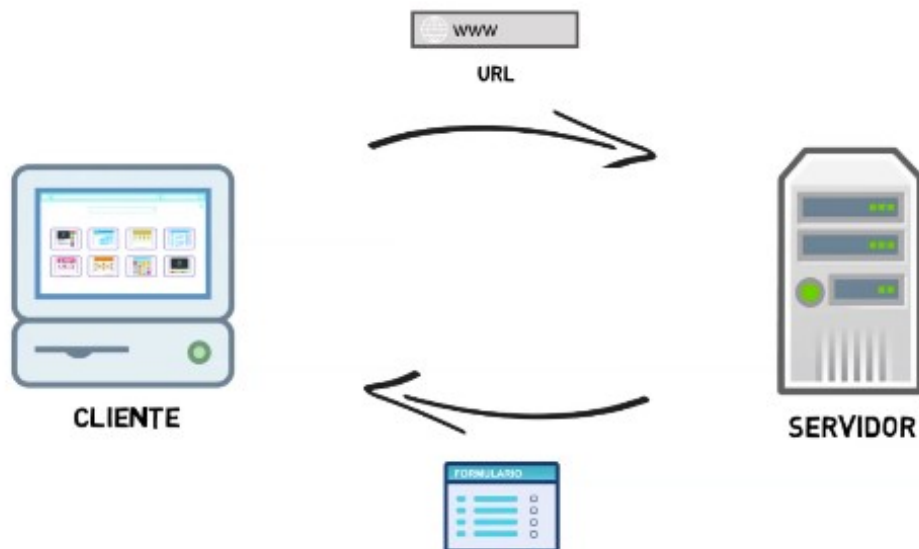
### **Introducción a los formularios web**

El primer paso para hacer una aplicación Web es dotarla de mecanismos de comunicación entre el lado cliente y el lado servidor. ¿Por qué? Porque tenemos el código de nuestra aplicación en el lado servidor y los usuarios interaccionan desde el lado cliente introduciendo sus datos.

¿Y dónde los van a introducir? A través de formularios web.

Entonces, ¿cómo es el mecanismo de funcionamiento de una aplicación web?

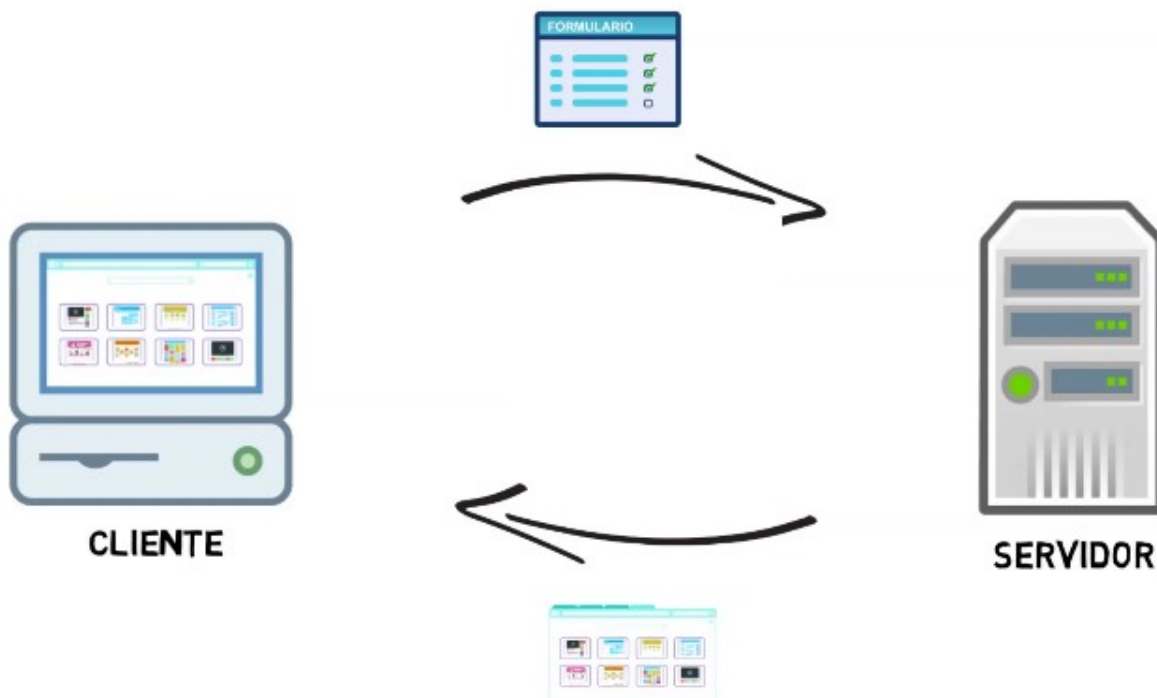
El cliente, desde su navegador (que es su cliente web) introduce el URL o link de acceso al servidor web, donde está la aplicación web. Seguidamente, el servidor le manda un formulario.



Ese formulario es visualizado por el navegador, el cliente web.

El usuario podrá rellenarlo y dar al botón de envío. En ese momento, recibe el servidor la petición del cliente, donde recibe los datos a través del formulario.

El servidor, con su código, realiza la lógica de programación que tenga programada y seguidamente va a devolver al cliente un documento web en el cual puede ir codificado otro formulario y, de esa manera, seguiría interactuando el usuario de la aplicación con el servidor de la misma.



Por lo tanto, el formulario es una pieza clave a la hora de hacer una aplicación web, porque es la interfaz del usuario con su aplicación.

Saber cómo se crean formularios es necesario para hacer cualquier aplicación web, sea cual sea el lenguaje que hayamos elegido en el lado servidor.

En nuestro caso hemos elegido Python, y vamos a mostrar unos ejemplos sencillos para procesar los formularios que vamos a hacer a lo largo de esta unidad.

Veamos este ejemplo sencillo de formulario, en el cual un usuario puede hacer login en nuestra aplicación.

Así se vería en el navegador, pero vamos a ver el código.

El formulario tiene dos atributos vitales en este proceso.

Uno llamado ACTION, donde ponemos el nombre del código que va a procesar lo que se ponga en el formulario.

```
<form action="processLogin" method="post" name="login">
```

Y otro llamado METHOD, donde indicamos cómo se van a mandar los datos: si por método POST o método GET.

```
<form action="processLogin" method="post" name="login">
```

En este caso, **el cliente efectúa la llamada al servicio login** (porque action="processLogin"), implementado en el servidor, servicio que procesará los datos ingresados por el usuario en este formulario, **y el envío de datos se realiza mediante el método POST.**

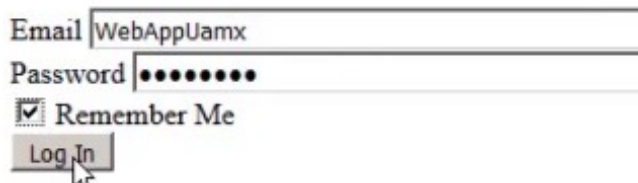
Como se puede observar hay más elementos dentro del mismo formulario. En este ejemplo tenemos dos campos donde el usuario podrá poner su dirección de email y su contraseña, los datos básicos para hacer el login en la aplicación Web. Y además se ofrece la oportunidad de recordar la contraseña.

Y finalmente, el formulario facilita un botón, que es el de login submit para que al pincharlo se manden los datos al servidor.

Entonces el html se vería de la siguiente manera:

SocNet | [Home](#) | [Log In](#) | [Sign Up](#)

## Log In



### Los elementos de un formulario

El elemento HTML para definir un formulario Web se especifica con la etiqueta <FORM> y tiene este aspecto:

```
<FORM ACTION="url" METHOD="método de envío">  
  <INPUT> | <SELECT> | <TEXTAREA> | <BUTTON> | <DATALIST> | <OUTPUT>  
</FORM>
```

Estos son los atributos que encontramos en la etiqueta FORM:

**ACTION:** la URL de un programa que procesa la petición en el lado servidor.

**METHOD:** el método HTTP que el navegador utiliza para mandar los datos del formulario del cliente al servidor. Hay dos opciones:

- post: corresponde al método POST HTTP, por el cual los datos del formulario son incluidos en el cuerpo del formulario y son enviados al servidor. Los más habitual es elegir este método.
- get: corresponde al método GET HTTP, por el cual los datos del formulario son adjuntados a la URI del atributo action con un '?' como separador, y la URI resultante es enviada al servidor.

### Los elementos del formulario

Los elementos que se pueden tener en un formulario son:

- <INPUT>: entrada o campo del formulario. Los tipos de input más utilizados son:
  - text: cuadro de texto.
  - password: cuadro de texto para introducir contraseña (no se ve contenido).
  - hidden: texto oculto, el navegador no lo visualiza en el documento Web, sin embargo si vemos su código podemos ver su contenido.
  - checkbox: casilla de verificación.
  - radio: casilla de selección.
  - submit: botón de envío del contenido del formulario al servidor.
  - reset: botón que facilita el borrado de los datos completados en el formulario.
  - Otros tipos que aparecieron con la version de HTML5 son: color, date, datetime-local, email, month, number, range, search, tel, time, url y week.
- <SELECT>: lista de selección con múltiples valores.
- <TEXTAREA>: campo de texto multilínea.
- <BUTTON>: botón de acción.
- <DATALIST>: facilita un listado de opciones predeterminadas para un cuadro de texto.
- <OUTPUT>: facilita mostrar la salida a un calculo realizado en el formulario.

Al igual que la etiqueta FORM, cada uno de los elementos del formulario tiene sus propios atributos. A modo de ejemplo, estos son los más habituales del elemento <INPUT> cuadro de texto (text):

- id: identificador del elemento.
- name: nombre del elemento. Lo habitual es que tenga el mismo valor que el atributo "id".
- type: para indicar el tipo (text, password, hidden, etc).
- required: indica si es imprescindible poner información en el campo, se pone en el campo sólo la palabra "required" (tambien es posible escribiendo required="true").
- placeholder: da información del dato que se espera introduzca el usuario (aparece en el cuadro de texto hasta que el usuario hace click en él).
- size: tamaño del cuadro de texto.
- maxlength: longitud máxima del texto.
- value: para inicializar con dicho valor el cuadro de texto.

### **Red Social: Analizando el formulario login.html**

Las partes más importantes que tiene este código en relación con el formulario:



```
<form action="processLogin" method="post" name="login">
  <label for="email">Email</label>
  <div class="inputs">
    <input id="email" name="email" required="true" placeholder="Email" size="80" type="text"
value=""/>
  </div>
  <label for="passwd">Password</label>
  <div class="inputs">
    <input id="passwd" name="passwd" required="true" placeholder="Password" size="80"
type="password" value=""/>
  </div>
  <label for="remember_me">
    <input id="remember_me" name="remember_me" type="checkbox" value="y"/>
    Remember Me
  </label>
  <div class="inputs">
    <input id="login_submit" name="login_submit" type="submit" value="Log In"/>
  </div>
</form>
```

1. Se indica en el elemento FORM que será procesado el formulario por "processLogin". Coincide con el nombre de una función en el fichero server.py que precisamente hace un procesamiento sencillo del formulario que recibe (por ahora imprime los datos recibidos en un documento web). Más adelante te explicaremos los detalles de este fichero server.py; por ahora, utilízalo como se te indica en el vídeo.
2. Se indica que el método de envío es POST HTTP.
3. El formulario tiene dos campos para entrada de texto (cuadro de texto): "email" (email del usuario en la red social), "passwd" (password o contraseña). También tiene una casilla de verificación o checkbox "remember\_me" para indicar si se desea recordar la contraseña.
4. Para cada campo del formulario se pone el nombre en un elemento del tipo label. Esto será útil cuando demos enseguida formato con CSS.
5. Los campos usuario y contraseña no pueden estar vacíos, por ello se especifica el atributo "required=true" en ambos.
6. Para ayudar al usuario de la Red Social se describe para cada campo la información a escribir con el atributo "placeholder" del formulario.
7. Cada campo del formulario tiene atributo id con el mismo valor que el atributo name.
8. Cada campo del formulario está dentro de una división de clase llamada inputs. Veremos la utilidad de esto cuando demos formato con CSS.

### **Red Social: Nuestro primer formulario**

Según la estructura de directorios que te comentamos durante la instalación, coloca los siguientes ficheros en los directorios que te indicamos:

- Pon login.html en moocwebapp\mooc\U2\app\static o moocwebapp/mooc/U2/app/static
- Pon server.py en moocwebapp\mooc\U2\app o moocwebapp/mooc/U2/app/
- En el caso de MAC y Linux, modificar server.py para que el puerto sea 8080 en vez de 80, si no, da error

Tenemos un directorio llamado moocwebapp. A partir de él tenemos una serie de subdirectorios, como ya hemos explicado, y en cualquiera de ellos de la unidad 2, 3 o 4 o 5 vamos a tener una serie de ficheros.

Acabamos de ver server.py y también tenemos login.html e index.html; uno lo habéis creado y otro lo hemos proporcionado.

Vamos a abrir PyCharm o Atom y vamos a decir que queremos crear un proyecto moocwebapp.

Lo abriremos y podremos ver efectivamente que tenemos dentro de dicho directorio una serie de directorios; lo mismo que acabamos de ver en el explorador. Podemos pinchar en cualquiera de ellos.

Por ejemplo, este login.html que acabamos de proporcionar, y vemos el código. Podríamos ver su contenido en una página web o bien pulsando el botón de la derecha y dar a navegador, elegir uno, y de esta manera vamos a ver su contenido, cómo lo visualiza un navegador o cliente web.

Y esto es lo que vemos: un formulario donde tenemos la posibilidad de introducir un email, que va a ser el nickname o forma de entrada a la aplicación, que si lo dejamos en blanco nos da un error porque tenemos siempre que poner un contenido.

Podemos poner, por lo tanto, un email, temporal, de prueba, y lo siguiente sería también escribir una password.

Pasaría lo mismo: si no escribimos la password, si lo dejamos en blanco, daría un error. Escribimos la password y le daríamos al login.

¿Qué ocurre? Da un error.

¿Por qué? Pues muy sencillo, porque realmente el servidor no está activado, no está levantado.

Por lo tanto, estamos viendo una página que simplemente muestra un contenido, pero no funciona la aplicación web, webapp.

Para que funcione tenemos simplemente que activar, lanzar el server.

Entonces en la consola, hacemos workon web, nos paramos sobre el directorio que contiene a server.py y lo ejecutamos. Y luego abrimos login.html en el navegador.

Entonces, si lo ponemos tal cual, aparece nuestro index que habíamos hecho, y si damos al login, aparecerá login.

Entonces, ahí simplemente lo que tendríamos que hacer es escribir un email, y después haremos lo mismo con el campo de password.

Podemos dar a remember me o no, y dar.

¿Qué ocurre?

Nos sale un mensaje de lo que hemos escrito porque nuestro server.py que tenemos ahora mismo es muy sencillo, es un server que simplemente visualiza lo que acabamos de escribir.

Nota Personal: Me parece que hay que reescribir a login poniendo .html a cada una de las páginas, porque no las encuentra

### **Red Social: Formulario signup.html**

Teniendo la página de signup.html, la idea es que en el navegador despliegue ésto:

## Sign Up

Nickname	
Nickname	I
Email	
Email	
Password	
Password	
Repeat Password	
Repeat Password	
Sign Up	

En el sign up de una pagina, tiene sentido que, si no ponemos datos a algunos de los campos y damos al botón de sign up, dé un error.

Bien, vamos entonces por tanto a rellenar el primer campo con un nickname.

Ponemos seguidamente su email.

A continuación, vamos a poner password y repetimos la password, lo cual tiene sentido que éstos 2 campos sean exactamente iguales.

Si se baja el archivo y se pincha el botón de submitir, por ahora lo que hace es consultar a server.py, que por ahora simplemente muestra los datos que hemos puesto en cada uno de los campos.

[SocNet](#) | [Home](#) | [Log In](#) | [Sign Up](#)

## Data from Form: Sign Up

Nickname: Jose  
 email: jose@socnet.com  
 passwd: jose  
 confirm: jose

### **Red social: creación de la página de sign up**

El código de ésta página es:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8"/><script
type="text/javascript">(window.NREUM||
(NREUM={})).loader_config={xpid:"XA4GVl5ACwAEV1JQAA=="};window.NRE
UM||(NREUM={}).__nr_require=function(t,n,e){function r(e){if(!
n[e]){var o=n[e]={exports:{}};t[e][0].call(o.exports,function(n)
{var o=t[e][1][n];return r(o||n),o,o.exports)}}return
n[e].exports}if("function"===typeof __nr_require)return
__nr_require;for(var o=0;o<e.length;o++)r(e[o]);return r}({1:
[function(t,n,e){function r(t)
{try{s.console&&console.log(t)}catch(n){}}var
o,i=t("ee"),a=t(18),s={};try{o=localStorage.getItem("__nr_flags").
split(", "),console&&"function"===typeof console.log&&(s.console=!
0,o.indexOf("dev")!==-1&&(s.dev=!0),o.indexOf("nr_dev")!==-
1&&(s.nrDev=!0))}catch(c){s.nrDev&&i.on("internal-
error",function(t){r(t.stack)}),s.dev&&i.on("fn-
err",function(t,n,e){r(e.stack)}),s.dev&&(r("NR AGENT IN
DEVELOPMENT MODE"),r("flags: "+a(s,function(t,n){return
t}).join(", ")))},{}],2:[function(t,n,e){function r(t,n,e,r,s)
{try{p?p-=1:o(s||new UncaughtException(t,n,e),!0)}catch(f)
```

```

{try{i("ierr",[f,c.now(),!0])}catch(d){}}return"function"==typeof
u&&u.apply(this,a(arguments))}function UncaughtException(t,n,e)
{this.message=t||"Uncaught error with no additional
information",this.sourceURL=n,this.line=e}function o(t,n){var e=n?
null:c.now();i("err",[t,e])}var
i=t("handle"),a=t(19),s=t("ee"),c=t("loader"),f=t("gos"),u=window.
onerror,d=!1,l="nr@seenError",p=0;c.features.err=!
0,t(1),window.onerror=r;try{throw new Error}catch(h){"stack"in
h&&(t(8),t(7),"addEventListener"in
window&&t(5),c.xrWrappable&&t(9),d=!0)}s.on("fn-
start",function(t,n,e){d&&(p+=1)}),s.on("fn-err",function(t,n,e)
{d&&!e[l]&&(f(e,l,function(){return!0}),this.thrown=!
0,o(e))}),s.on("fn-end",function(){d&&!this.thrown&&p>0&&(p-
=1)}),s.on("internal-error",function(t){i("ierr",[t,c.now(),!
0]))}),{3:[function(t,n,e){t("loader").features.ins=!0},{}],4:
[function(t,n,e){function r(t)
{}if(window.performance&&window.performance.timing&&window.perform
ance.getEntriesByType){var
o=t("ee"),i=t("handle"),a=t(8),s=t(7),c="learResourceTimings",f="a
ddEventListener",u="resourcetimingbufferfull",d="bstResource",l="r
esource",p="-start",h="-
end",m="fn"+p,w="fn"+h,v="bstTimer",y="pushState",g=t("loader");g.
features.stn=!0,t(6);var x=NREUM.o.EV;o.on(m,function(t,n){var
e=t[0];e instanceof
x&&(this.bstStart=g.now()))},o.on(w,function(t,n){var e=t[0];e
instanceof x&&i("bst",
[e,n,this.bstStart,g.now()])),a.on(m,function(t,n,e)
{this.bstStart=g.now(),this.bstType=e}),a.on(w,function(t,n){i(v,
[n,this.bstStart,g.now(),this.bstType]}),s.on(m,function()
{this.bstStart=g.now()}),s.on(w,function(t,n){i(v,
[n,this.bstStart,g.now(),"requestAnimationFrame"])}),o.on(y+p,func
tion(t)
{this.time=g.now(),this.startPath=location.pathname+location.hash}
),o.on(y+h,function(t){i("bstHist",
[location.pathname+location.hash,this.startPath,this.time]))},f in
window.performance&&(window.performance["c"+c]?
window.performance[f](u,function(t){i(d,
[window.performance.getEntriesByType(l)],window.performance["c"+c
]()),!1):window.performance[f]("webkit"+u,function(t){i(d,
[window.performance.getEntriesByType(l)],window.performance["webk
itC"+c]()),!1)),document[f]("scroll",r,{passive:!0}),document[f]
("keypress",r,!1),document[f]("click",r,!1)}),{5:
[function(t,n,e){function r(t){for(var n=t;n&&!
n.hasOwnProperty(u);)n=Object.getPrototypeOf(n);n&&o(n)}function
o(t){s.inPlace(t,[u,d],"-",i)}function i(t,n){return t[1]}var
a=t("ee").get("events"),s=t(21)(a,!
0),c=t("gos"),f=XMLHttpRequest,u="addEventListener",d="removeEvent
Listener";n.exports=a,"getPrototypeOf"in Object?
(r(document),r(window),r(f.prototype)):f.prototype.hasOwnProperty(
u)&&(o(window),o(f.prototype)),a.on(u+"-start",function(t,n){var
e=t[1],r=c(e,"nr@wrapped",function(){function t()

```

```

{if("function"==typeof e.handleEvent)return
e.handleEvent.apply(e,arguments)}var n={object:t,"function":e}
[typeof e];return n?
s(n,"fn-",null,n.name||"anonymous"):e});this.wrapped=t[1]=r}},a.on
(d+"-start",function(t){t[1]=this.wrapped||t[1]}),{}},6:
[function(t,n,e){var r=t("ee").get("history"),o=t(21)
(r);n.exports=r;var
i=window.history&&window.history.constructor&&window.history.const
ructor.prototype,a=window.history;i&&i.pushState&&i.replaceState&&
(a=i),o.inPlace(a,["pushState","replaceState"],"-")},{}},7:
[function(t,n,e){var r=t("ee").get("raf"),o=t(21)
(r),i="requestAnimationFrame";n.exports=r,o.inPlace(window,
["r"+i,"mozR"+i,"webkitR"+i,"msR"+i],"raf-"),r.on("raf-
start",function(t){t[0]=o(t[0],"fn-")}),{}},8:[function(t,n,e)
{function r(t,n,e){t[0]=a(t[0],"fn-",null,e)}function o(t,n,e)
{this.method=e,this.timerDuration=isNaN(t[1])?
0:+t[1],t[0]=a(t[0],"fn-",this,e)}var
i=t("ee").get("timer"),a=t(21)
(i),s="setTimeout",c="setInterval",f="clearTimeout",u="-
start",d="-";n.exports=i,a.inPlace(window,
[s,"setImmediate"],s+d),a.inPlace(window,
[c],c+d),a.inPlace(window,
[f,"clearImmediate"],f+d),i.on(c+u,r),i.on(s+u,o)},{}},9:
[function(t,n,e){function r(t,n){d.inPlace(n,
["onreadystatechange"],"fn-",s)}function o(){var
t=this,n=u.context(t);t.readyState>3&&!n.resolved&&(n.resolved=!
0,u.emit("xhr-resolved",[],t)),d.inPlace(t,y,"fn-",s)}function
i(t){g.push(t),h&&(b?b.then(a):w?w(a):(E=-E,R.data=E))}function
a(){for(var t=0;t<g.length;t+
+)r([],g[t]);g.length&&(g=[])}function s(t,n){return n}function
c(t,n){for(var e in t)n[e]=t[e];return n}t(5);var
f=t("ee"),u=f.get("xhr"),d=t(21)
(u),l=NREUM.o,p=l.XHR,h=l.MO,m=l.PR,w=l.SI,v="readystatechange",y=
["onload","onerror","onabort","onloadstart","onloadend","onprogres
s","ontimeout"],g=[];n.exports=u;var
x=window.XMLHttpRequest=function(t){var n=new
p(t);try{u.emit("new-xhr",[n],n),n.addEventListener(v,o,!
1)}catch(e){try{u.emit("internal-error",[e])}catch(r){}}return
n};if(c(p,x),x.prototype=p.prototype,d.inPlace(x.prototype,
["open","send"],"xhr-",s),u.on("send-xhr-start",function(t,n)
{r(t,n),i(n)}),u.on("open-xhr-start",r),h){var
b=m&&m.resolve();if(!w&&!m){var
E=1,R=document.createTextNode(E);new h(a).observe(R,
{characterData:!0})}}else f.on("fn-end",function(t)
{t[0]&&t[0].type===v||a()}),{}},10:[function(t,n,e){function r()
{var t=window.NREUM,n=t.info.accountID||null,e=t.info.agentID||
null,r=t.info.trustKey||null,i="btoa"in window&&"function"==typeof
window.btoa;if(!n||!e||!i)return null;var a={v:[0,1],d:
{ty:"Browser",ac:n,ap:e,id:o.generateCatId(),tr:o.generateCatId(),
ti:Date.now()}};return r&&n!
==r&&(a.d.tk=r),btoa(JSON.stringify(a))}var

```

```

o=t(16);n.exports={generateTraceHeader:r},{},11:[function(t,n,e)
{function r(t){var n=this.params,e=this.metrics;if(!this.ended)
{this.ended=!0;for(var r=0;r<p;r+
+)t.removeEventListener(l[r],this.listener,!1);n.aborted||
(e.duration=s.now()-this.startTime,this.loadCaptureCalled||4!
==t.readyState?
null==n.status&&(n.status=0):a(this,t),e.cbTime=this.cbTime,d.emit
("xhr-done",[t],t),c("xhr",[n,e,this.startTime]))}}function o(t,n)
{var e=t.responseType;if("json"===e&&null!==n)return n;var
r="arraybuffer"===e||"blob"===e||"json"===e?
t.response:t.responseText;return w(r)}function i(t,n){var
e=f(n),r=t.params;r.host=e.hostname+": "+e.port,r.pathname=e.pathna
me,t.sameOrigin=e.sameOrigin}function a(t,n)
{t.params.status=n.status;var
e=o(n,t.lastSize);if(e&&(t.metrics.rxSize=e),t.sameOrigin){var
r=n.getResponseHeader("X-NewRelic-App-
Data");r&&(t.params.cat=r.split(", ").pop())}t.loadCaptureCalled=!
0}var s=t("loader");if(s.xhrWrappable){var
c=t("handle"),f=t(12),u=t(10).generateTraceHeader,d=t("ee"),l=["lo
ad","error","abort","timeout"],p=l.length,h=t("id"),m=t(15),w=t(14
),v=window.XMLHttpRequest;s.features.xhr=!0,t(9),d.on("new-
xhr",function(t){var
n=this;n.totalCbs=0,n.called=0,n.cbTime=0,n.end=r,n.ended=!
1,n.xhrGuids={},n.lastSize=null,n.loadCaptureCalled=!
1,t.addEventListener("load",function(e){a(n,t)},!1),m&&(m>34||
m<10)||window.opera||t.addEventListener("progress",function(t)
{n.lastSize=t.loaded},!1)),d.on("open-xhr-start",function(t)
{this.params={method:t[0]},i(this,t[1]),this.metrics={}),d.on("op
en-xhr-end",function(t,n){n.loader_config"in NREUM&&"xpid"in
NREUM.loader_config&&this.sameOrigin&&n.setRequestHeader("X-
NewRelic-ID",NREUM.loader_config.xpid);var e=!1;if("init"in
NREUM&&"distributed_tracing"in NREUM.init&&(e=!
NREUM.init.distributed_tracing.enabled),e&&this.sameOrigin){var
r=u();r&&n.setRequestHeader("newrelic",r)}},d.on("send-xhr-
start",function(t,n){var e=this.metrics,r=t[0],o=this;if(e&&r){var
i=w(r);i&&(e.txSize=i)}this.startTime=s.now(),this.listener=functi
on(t){try{"abort"!==t.type||o.loadCaptureCalled||
(o.params.aborted=!0),("load"!==t.type||
o.called===o.totalCbs&&(o.onloadCalled||"function"!==typeof
n.onload))&&o.end(n)}catch(e){try{d.emit("internal-error",
[e])}catch(r){}}};for(var a=0;a<p;a+
+)n.addEventListener(l[a],this.listener,!1)),d.on("xhr-cb-
time",function(t,n,e){this.cbTime+=t,n?this.onloadCalled=!
0:this.called+=1,this.called!==this.totalCbs||
this.onloadCalled&&"function"===typeof e.onload||
this.end(e)}),d.on("xhr-load-added",function(t,n){var e="" +h(t)+!!
n;this.xhrGuids&&!this.xhrGuids[e]&&(this.xhrGuids[e]=!
0,this.totalCbs+=1)},d.on("xhr-load-removed",function(t,n){var
e="" +h(t)+!!n;this.xhrGuids&&this.xhrGuids[e]&&(delete
this.xhrGuids[e],this.totalCbs-=1)},d.on("addEventListener-
end",function(t,n){n instanceof v&&"load"===t[0]&&d.emit("xhr-

```

```

load-added",[t[1],t[2]],n)),d.on("removeEventListener-
end",function(t,n){n instanceof v&&"load"===t[0]&&d.emit("xhr-
load-removed",[t[1],t[2]],n)),d.on("fn-start",function(t,n,e){n
instanceof v&&("onload"===e&&(this.onload=!0),
("load"===(t[0]&&t[0].type)||
this.onload)&&(this.xhrCbStart=s.now()))}),d.on("fn-
end",function(t,n){this.xhrCbStart&&d.emit("xhr-cb-time",[s.now()-
this.xhrCbStart,this.onload,n],n)}},{}],12:[function(t,n,e)
{n.exports=function(t){var
n=document.createElement("a"),e=window.location,r={};n.href=t,r.po
rt=n.port;var o=n.href.split("://");!
r.port&&o[1]&&(r.port=o[1].split("/")[0].split("@").pop().split(":")[1]),r.port&&"0"!==r.port||
(r.port="https"===o[0]?"443":"80"),r.hostname=n.hostname||
e.hostname,r.pathname=n.pathname,r.protocol=o[0],"/"!
==r.pathname.charAt(0)&&(r.pathn
ame="/" +r.pathname);var i=!n.protocol||":"===n.protocol||
n.protocol===e.protocol,a=n.hostname===document.domain&&n.port===e
.port;return r.sameOrigin=i&&(!n.hostname||a),r}},{}],13:
[function(t,n,e){function r(){function o(t,n,e){return function()
{return i(t,[f.now()].concat(s(arguments)),n?null:this,e),n?void
0:this}}var
i=t("handle"),a=t(18),s=t(19),c=t("ee").get("tracer"),f=t("loader"
),u=NREUM;"undefined"===typeof window.newrelic&&(newrelic=u);var
d=["setPageViewName","setCustomAttribute","setErrorHandler","finis
hed","addToTrace","inlineHit","addRelease"],l="api-",p=l+"ixn-";a(
d,function(t,n){u[n]=o(l+n,!
0,"api")}),u.addAction=o(l+"addAction",!
0),u.setCurrentRouteName=o(l+"routeName",!
0),n.exports=newrelic,u.interaction=function(){return(new
r).get()};var h=r.prototype={createTracer:function(t,n){var
e={},r=this,o="function"===typeof n;return i(p+"tracer",
[f.now(),t,e],r),function(){if(c.emit((o?"":"no-")+"fn-start",
[f.now(),r,o],e),o)try{return n.apply(this,arguments)}catch(t)
{throw c.emit("fn-err",[arguments,this,t],e),t}finally{c.emit("fn-
end",
[f.now()],e)}}}},a("actionText,setName,setAttribute,save,ignore,on
End,getContext,end,get".split(","),function(t,n)
{h[n]=o(p+n)}),newrelic.noticeError=function(t,n){"string"===typeof
t&&(t=new Error(t)),i("err",[t,f.now(),!1,n]}},{}],14:
[function(t,n,e){n.exports=function(t){if("string"===typeof
t&&t.length)return t.length;if("object"===typeof t){if("undefined"!
=typeof ArrayBuffer&&t instanceof ArrayBuffer&&t.byteLength)return
t.byteLength;if("undefined"!==typeof Blob&&t instanceof
Blob&&t.size)return t.size;if(!("undefined"!==typeof FormData&&t
instanceof FormData))try{return JSON.stringify(t).length}catch(n)
{return}}}},{}],15:[function(t,n,e){var
r=0,o=navigator.userAgent.match(/Firefox[\s\S](\d+\.\
d+)/);o&&(r+=o[1]),n.exports=r},{}],16:[function(t,n,e){function
r(){function t(){return n?15&n[e++]:16*Math.random()|0}var
n=null,e=0,r=window.crypto||

```

```

window.msCrypto;r&&r.getRandomValues&&(n=r.getRandomValues(new
Uint8Array(31)));for(var o,i="xxxxxxxx-xxxx-4xxx-yxxx-
xxxxxxxxxxxx",a="",s=0;s<i.length;s++)o=i[s],"x"===o?
a+=t().toString(16):"y"===o?(o=3&t())|
8,a+=o.toString(16)):a+=o;return a}function o(){function t()
{return n?15&n[e++]:16*Math.random()|0}var
n=null,e=0,r=window.crypto||
window.msCrypto;r&&r.getRandomValues&&Uint8Array&&(n=r.getRandomVa
lues(new Uint8Array(31)));for(var o=[],i=0;i<16;i+
+)o.push(t().toString(16));return
o.join("")}n.exports={generateUuid:r,generateCatId:o}},{}],17:
[function(t,n,e){function r(t,n){if(!o)return!1;if(t!==o)return!
1;if(!n)return!0;if(!i)return!1;for(var
e=i.split("."),r=n.split("."),a=0;a<r.length;a++)if(r[a]!
==e[a])return!1;return!0}var o=null,i=null,a=/Version\/(\S+)\
s+Safari/;if(navigator.userAgent){var
s=navigator.userAgent,c=s.match(a);c&&s.indexOf("Chrome")===-
1&&s.indexOf("Chromium")===-
1&&(o="Safari",i=c[1])}n.exports={agent:o,version:i,match:r}},
{}],18:[function(t,n,e){function r(t,n){var e=[],r="",i=0;for(r in
t)o.call(t,r)&&(e[i]=n(r,t[r]),i+=1);return e}var
o=Object.prototype.hasOwnProperty;n.exports=r},{}],19:
[function(t,n,e){function r(t,n,e){n||(n=0),"undefined"===typeof
e&&(e=t?t.length:0);for(var r=-1,o=e-n||0,i=Array(o<0?0:o);+
+r<o;)i[r]=t[n+r];return i}n.exports=r},{}],20:[function(t,n,e)
{n.exports={exists:"undefined"!==typeof
window.performance&&window.performance.timing&&"undefined"!==typeof
window.performance.timing.navigationStart}},{}],21:
[function(t,n,e){function r(t){return!(t&&t instanceof
Function&&t.apply&&!t[a])}var
o=t("ee"),i=t(19),a="nr@original",s=Object.prototype.hasOwnPropert
y,c=!1;n.exports=function(t,n){function e(t,n,e,o){function
nrWrapper(){var
r,a,s,c;try{a=this,r=i(arguments),s="function"===typeof e?
e(r,a):e||{}}catch(f){l([f,"",[r,a,o],s])}u(n+"start",
[r,a,o],s);try{return c=t.apply(a,r)}catch(d){throw u(n+"err",
[r,a,d],s),d}finally{u(n+"end",[r,a,c],s)}}return r(t)?t:(n||
(n=""),nrWrapper[a]=t,d(t,nrWrapper),nrWrapper)}function
f(t,n,o,i){o||(o="");var
a,s,c,f="-"===o.charAt(0);for(c=0;c<n.length;c+
+)s=n[c],a=t[s],r(a)||t[s]=e(a,f?s+o:o,i,s))}function u(e,r,o)
{if(!c||n){var i=c;c=!0;try{t.emit(e,r,o,n)}catch(a)
{l([a,e,r,o])}c=i}}function d(t,n)
{if(Object.defineProperty&&Object.keys)try{var
e=Object.keys(t);return e.forEach(function(e)
{Object.defineProperty(n,e,{get:function(){return
t[e]},set:function(n){return t[e]=n,n}})}),n}catch(r)
{l([r])}}for(var o in t)s.call(t,o)&&(n[o]=t[o]);return n}function
l(n){try{t.emit("internal-error",n)}catch(e){}}return t||
(t=o),e.inPlace=f,e.flag=a,e}},{}],ee:[function(t,n,e){function
r(){}function o(t){function n(t){return t&&t instanceof r?t:t?

```



```

c(t,s,i):i()}function e(e,r,o,i){if(!l.aborted||i)
{t&&t(e,r,o);for(var a=n(o),s=m(e),c=s.length,f=0;f<c;f+
+)s[f].apply(a,r);var d=u[g[e]];return
d&&d.push([x,e,r,a]),a}}function p(t,n)
{y[t]=m(t).concat(n)}function h(t,n){var e=y[t];if(e)for(var
r=0;r<e.length;r++)e[r]==n&&e.splice(r,1)}function m(t){return
y[t]||[]}function w(t){return d[t]=d[t]||o(e)}function v(t,n)
{f(t,function(t,e){n=n||"feature",g[e]=n,n in u||(u[n]=[])})}var
y={},g={},x={on:p,addEventListener:p,removeEventListener:h,emit:e,
get:w,listeners:m,context:n,buffer:v,abort:a,aborted:!1};return
x}function i(){return new r}function a(){(u.api||
u.feature)&&(l.aborted=!0,u=l.backlog={})}var
s="nr@context",c=t("gos"),f=t(18),u={},d={},l=n.exports=o();l.back
log=u},{},gos:[function(t,n,e){function r(t,n,e)
{if(o.call(t,n))return t[n];var
r=e();if(Object.defineProperty&&Object.keys)try{return
Object.defineProperty(t,n,{value:r,writable:!0,enumerable:!
1}),r}catch(i){}return t[n]=r,r}var
o=Object.prototype.hasOwnProperty;n.exports=r},{},handle:
[function(t,n,e){function r(t,n,e,r)
{o.buffer([t],r),o.emit(t,n,e)}var
o=t("ee").get("handle");n.exports=r,r.ee=o},{},id:
[function(t,n,e){function r(t){var n=typeof t;return!t||"object"!
==n&&"function"!=n?-1:t===window?0:a(t,i,function(){return o+
+})}var o=1,i="nr@id",a=t("gos");n.exports=r},{},loader:
[function(t,n,e){function r(){if(!E++){var
t=b.info=NREUM.info,n=p.getElementsByTagName("script")
[0];if(setTimeout(u.abort,3e4),!
(t&&t.licenseKey&&t.applicationID&&n))return
u.abort();f(g,function(n,e){t[n]||(t[n]=e)),c("mark",
["onload",a()+b.offset,null,"api"]);var
e=p.createElement("script");e.src="https://"+t.agent,n.parentNode.
insertBefore(e,n)}}function o()
{"complete"===p.readyState&&i()}function i(){c("mark",
["domContentLoaded",a()+b.offset,null,"api"]}function a(){return
R.exists&&performance.now?Math.round(performance.now()):
(s=Math.max((new Date).getTime(),s))-b.offset}var s=(new
Date).getTime(),c=t("handle"),f=t(18),u=t("ee"),d=t(17),l=window,p
=l.document,h="addEventListener",m="attachEvent",w=l.XMLHttpRequest,
v=w&&w.prototype;NREUM.o={ST:setTimeout,SI:l.setImmediate,CT:cle
arTimeout,XHR:w,REQ:l.Request,EV:l.Event,PR:l.Promise,MO:l.Mutatio
nObserver};var y=""+"location,g={beacon:"bam.nr-
data.net",errorBeacon:"bam.nr-data.net",agent:"js-
agent.newrelic.com/nr-1130.min.js"},x=w&&v&&v[h]&&!/
CriOS/.test(navigator.userAgent),b=n.exports={offset:s,now:a,origi
n:y,features:{},xhrWrappable:x,userAgent:d};t(13),p[h]?(p[h]
("DOMContentLoaded",i,!1),l[h]("load",r,!1)):(p[m]
("onreadystatechange",o),l[m]("onload",r)),c("mark",
["firstbyte",s],null,"api");var E=0,R=t(20)},{},{}],{}},
["loader",2,11,4,3]);</script><script
type="text/javascript">window.NREUM||

```

```
(NREUM={});NREUM.info={"beacon":"bam.nr-
data.net","queueTime":1,"licenseKey":"1beac94c95","agent":"","tran
sactionName":"NQQGYUZWDUNSVUMPWQx0IkBaVBdZXfYgYCUYHDwFRTBkAX0FTGQJc
Aw8DWlVHE0MdVVGiQgcPEEZRRRVVQRhaD1IGDQFCVUUGCmBCVhJfASILW0BSDURgU0
UQUxBPFEdbVAZDQGLFA0cXBBdB","atts":"GUMRFw5MQUJWR0IDRRY+F1BSUhFVQR
QNRF4WFRRGDhhMU1xDRRVTEU8BUUwZDEJUGVQJQxASAUybVAXFQUVSS0BTWzF0eU9I
Z1ZUdhZGSVAwBwQGWLEcVVgTRBEEE1RGUkwIUlAGB1cDWAEGBgJXAVcGD1MBBgJQDQ
IAVwYDBlUHD00CAQdVUVQFVgQBVg9WVgVXDAMBA1VXUwMCU1kAV1AHWx8GCVYFQgsX
BUFRaAFcXFVcOV8GXAZZW1QIHUUHElV3NyApTREFIWdWVHYWRkdTJgRgBVMBclcSVH
QWGBRQEQTWEdbw0MEIAMIWldcRgQDAAAEDlZUAQcBU1UHB1RVUA8GBVZQV1QGUQMC
BQAGVFJGGRZFBkFGU0QSaRcSAUdrVgRVXUIVXBQvDh5cWfsCHwYYB0Ye0lBVDhR7Cl
5GThceDlQ+UgEPFxFGCQAHSALZLSNQV1wMHwEGBLYGU1FVFXJeEVVWU9JAFJPVBcY
FRZDVkRoD1JAW1YGDAFTAwAEG0REBxARUEdDPFFGQl85QhsRARc0FRBVQEVeCVhPDh
YYQVklXlxBWUQaQAILW0BSDURAU0UQUxBPB1RXXwZRUVpSRAwWExFQGBUAX11CUghC
EQQWQ1FFTVNcWEMDWBY+CFBaFVkBwEHShQBDgpBUVkXQ1ZEQQNETBEFQVwVWRJAX1
AIQxI+DVtdGQtEXloVShQBDgpBUVkXQ1ZEQQNETAILW0BSDURsQk4WU0BbRkFRTxcf
W0JaChROQxZQRUIGQ0dpQhVTED4NURYNUQMLAAdVBVBNRLZbQhFDVmleAhRYQwdaQU
UQVR5ABlxjIywcHmNSAXFDRhxXYlBRVQxVFU8SXERQRAXANCV4TBVPElBZWRJTDBUX
UEZBBkIdWlGfXQcFRg9SVg9DVhVbVbKMFQfBQEQGQkVTRUhQEA4JaldTDRIJUFYKRQ
ccGQ==","applicationID":"3343327","errorBeacon":"bam.nr-
data.net","applicationTime":21}</script>
```

```
<title>
    Sign Up - SocNet
</title>
</head>
<body>
    <div id="container">
        <a href="/">
            SocNet
        </a>
        |
        <a href="home">
            Home
        </a>
        |
        <a href="login">
            Log In
        </a>
        |
        <a href="signup">
            Sign Up
        </a>
        <h1>
            Sign Up
        </h1>
        <form action="processSignup" method="post"
name="signup">
            <label for="nickname">
                Nickname
            </label>
            <div class="inputs">
```

```
        </div>
        <label for="email">
            Email
        </label>
        <div class="inputs">

        </div>
        <label for="passwd">
            Password
        </label>
        <div class="inputs">

        </div>
        <label for="confirm">
            Repeat Password
        </label>
        <div class="inputs">

        </div>
        <div class="inputs">
            <input id="signup_submit" name="signup_submit"
type="submit" value="Sign Up"/>
        </div>
    </form>
</div>
</body>
</html>
```

Las partes importantes es que:

- **nickname.** Tiene tanto por identificador (id) como por nombre (name) 'nickname', es un campo requerido, de tamaño 80 y de tipo texto (text).
- **email.** Tiene tanto por identificador (id) como por nombre (name) 'email', es un campo requerido, de tamaño 80 y de tipo texto (text).
- **passwd.** Tiene tanto por identificador (id) como por nombre (name) 'passwd', es un campo requerido, de tamaño 80 y de tipo contraseña (password).
- **confirm.** Tiene tanto por identificador (id) como por nombre (name) 'confirm', es un campo requerido, de tamaño 80 y de tipo contraseña (password). Éste campo es el que debería verificar que lo metido en el campo passwd, sea igual

Para que sea tomado en cuenta, debe inscribirse así modificando el código

- `<input id="nickname" name="nickname" required="true" size="80" type="text" />`
- `<input id="email" name="email" required="true" size="80" type="text" />`
- `<input id="passwd" name="passwd" required="true" size="80" type="password" />`

- `<input id="confirm" name="confirm" required="true" size="80" type="password" />`

### **Red social: Formulario Home**

Veamos Home. Tenemos un cuadro de texto.



Si lo dejamos vacío y damos a enviar (post) debe dar error. Pongamos, por tanto, en dicho cuadro de texto un primer mensaje o post.

Damos a enviar dando al botón de post, seguidamente veis que va a aparecer abajo. Podemos escribir otro mensaje o post. También daríamos al botón.

Y esos mensajes se van a ir concatenando abajo, en la misma página. Esto lo realiza server.py.

Por ahora, nuestro server.py hace esta tarea, la de ir concatenando nuestros mensajes, y podemos seguir así continuamente.

### **Creación de Home**

Completa el formulario home.html para que el usuario pueda escribir mensajes (posts). Para ello, arregla este fichero llamado home\_ini.html en el que encontrarás parte del código, pero le faltan partes que tendrás que codificar y por las cuales te preguntamos en el ejercicio siguiente.

El fichero que crees debe llamarse home.html y se debe guardar en el mismo directorio donde está el fichero login.html (C:\moocwebapp\mooc\U2\app\static o moocwebapp/mooc/U2/app/static, según el sistema operativo que estés usando). Los nombres de los campos de este formulario y sus características deben ser los siguientes:

- message: Tiene tanto por identificador (id) como por nombre (name) 'message', es un campo requerido, de tamaño 80, de longitud máxima 128 y de tipo texto (text).  
`<input id="message" name="message" required="true" size="80" maxlength="128" type="text" />`
- last: Tiene tanto por identificador (id) como por nombre (name) 'last' y es de tipo oculto (hidden)

```
<input id="last" name="last" required="true" type="hidden" />
```

## **Introducción a CSS**

Es importante personalizar nuestra aplicación web. Para ello, veamos cómo darle un estilo utilizando CSS, siglas en inglés de Cascading Style Sheets o, en castellano, hojas de estilo en cascada.

**CSS es un lenguaje de diseño gráfico que utilizamos para dar presentación a documentos estructurados realizados en lenguajes de marcado.**

En este curso vamos a utilizarlo para darle el formato visual a los documentos y páginas web de nuestra aplicación web.

Además, HTML fue inicialmente diseñado para ofrecer contenidos y no para dar formato. Si bien es cierto, que por ejemplo, a partir de la versión 3.2 de HTML, se ofrecía la etiqueta para dar formato al texto, pero ni lo era entonces una buena opción ni lo es ahora mezclar contenidos con formato.

Por otro lado, escribir un documento separado de su formato facilita que dicho formato pueda ser compartido entre varios documentos y, de esta manera, podamos crear una aplicación web con un formato y un estilo homogéneo.

Vamos a crear reglas CSS.

Una regla CSS está formada por dos partes:

- un selector: por ejemplo h1
- una o varias declaraciones: que van apareadas con clave:valor, ejemplo color: black;

```
h1{  
    color:black;  
}
```

Y en el código ya conocido, login.html, se pone al inicio la referencia a un código donde vamos a tener las reglas css:

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="utf-8"/>  
    <link href="socnet-style.css" rel="stylesheet" type="text/css"/>  
    <title>Log In - SocNet</title>  
  </head>
```

Dentro de, en nuestro caso “socnet-style.css” va a ir las reglas de CSS:

```
/*socnet-style.css */  
/* Own stylesheet for SocNet */  
  
h1 {color:blue; font-size:20pt;}  
a:link {color:black;}  
a:hover {color:green;}
```

Este código significa que todos los headers 1 (en el caso de login.htm la palabra Login) tengan letra de color azul y un fontsize de 20, y por otro lado, los enlaces que aparecen al inicio de la página serán ahora de color negro en lugar de azul, como estaban antes. Y pasarán a tener color verde cuando se pase el ratón por encima de ellos.

### **Red Social: CSS a login.html**

Lo primero que hay que hacer es bajar de

<https://prod-edxapp.edx-cdn.org/assets/courseware/v1/fd02496f3c29fc16e5b521b91f5f2107/asset-v1:UAMx+WebApp+1T2019a+type@asset+block/socnet-style.css> el archivo de estilo CSS y **luego poner en login.html la referencia dentro de la etiqueta head**

```
<link href="static/css/socnet-style.css" rel="stylesheet" type="text/css"/>
```

Con éstos archivos se le va a dar formato a LogIn

Tenemos un fichero que os hemos proporcionado, con extensión CSS, donde tendremos una serie de reglas CSS, sencillas, que vamos a utilizar para dar formato, como decimos, al fichero login.html.

Pero por ahora vamos a lanzar primero la aplicación web como la tenemos, sin hacer ningún cambio.

Y lo que vamos a hacer es irnos a PyCharm o Atom, y lo que vamos a hacer es añadir en el fichero login.html una fila. una línea, con el contenido que dice que vamos a usar como fichero de reglas, para dar formato a este formulario.

Damos todo el camino para llegar a él, desde el fichero html hasta el fichero css, salvamos el fichero y volveremos a ejecutar la aplicación:

En la terminal sobre el directorio /moocwebapp/mooc/U2/app

workon web

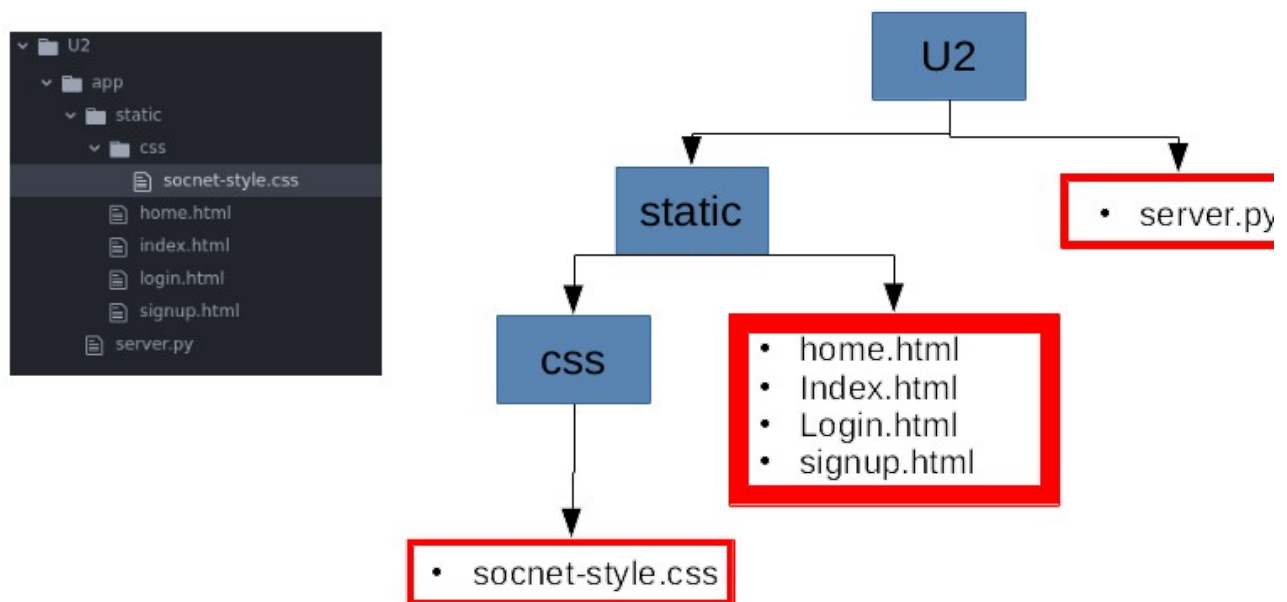
```
python3 server.py
```

Al irnos al navegador, refrescamos para ver los cambios y vamos a ver que efectivamente h1 ha cambiado a color azul, y luego los links, al pasar el ratón por encima de ellos, se ven de color verde.

Es decir, estamos teniendo en cuenta las reglas CSS del fichero que hemos puesto enlazado. Como vemos, el formulario ha cambiado, pero en el resto de páginas no.

Lo que deberíamos hacer es copiar esta fila en el resto de ficheros HTML para unificar dicho formato.

Estructura de ficheros:



### Reglas CSS

En este ejemplo se está modificando el formato de los párrafos p: ahora será su texto de color verde y de tamaño 10 px:

```
p {color:green; font-size:10px;}
```

p es el selector de párrafo y el resto son las declaraciones y ésto va en el archivo css

Ahora, **en el html**:

```
<html>
```

```
<head>
```

```
<style type="text/css">
```

```
    #para1 {text-align:center; color:red;}
```

```
</style>
```

```
</head>
<body>
  <p id="para1">Hello World!</p>
  <p>This paragraph is not affected by the style.</p>
</body>
</head>
```

O sea css usa el valor del atributo "id" del elemento HTML a modificar y en la regla CSS se utiliza una almohadilla (#) para referenciarlo.

Por otra parte, el selector **class** se utiliza para especificar el formato de varios elementos HTML que tienen en común el valor del atributo "class". En la regla CSS se utiliza un punto (.) para referenciarlo. Fíjate en este ejemplo, donde se cambia el formato tanto del título como del párrafo:

```
<html>
  <head>
    <style type="text/css">
      .center {text-align:center;}
    </style>
  </head>
  <body>
    <h1 class="center">Center-aligned heading</h1>
    <p class="center">Center-aligned paragraph.</p>
  </body>
</html>
```

### Entonces, cómo especificar el estilo:

- El **estilo en línea** (inline style) se da cuando especificamos el formato del elemento HTML a partir del atributo style. En este caso, el cambio de formato solo afecta a dicho elemento HTML pero, como ya te hemos mencionado anteriormente, no es la opción más recomendable.

Ejemplo: <p style="color:sienna; margin-left:20px">This is a paragraph.</p>

- **hoja de estilo interna** (*internal style sheet*) es la que has visto en los ejemplos de más arriba. Se utiliza la etiqueta <style> en la cabecera del documento para especificar el formato CSS, y este cambio afectará únicamente a los elementos HTML del documento en que está contenido.

Ejemplo:



```

<head>
  <style type="text/css">
    hr {color:sienna;}
    p {margin-left:20px;}
    body {background-image:url("images/back40.gif");}
  </style>
</head>

```

- La **hoja de estilo externa** (external style sheet) es con la que ya hemos trabajado en nuestra aplicación web. En ella se describe el formato en un fichero externo CSS (normalmente con extensión .css) y todos los documentos del proyecto que incluyan la etiqueta <link> y especifiquen dicha hoja de estilos compartirán el mismo formato. Es la forma recomendada de utilizar CSS, separando estilo de contenido en nuestro código.

### Combinación de selectores

En CSS hay varias formas de seleccionar unos elementos respecto a su relación con otros, lo cual se expresa mediante combinaciones. Sean A y B dos selectores cualquiera (p, h1, div, etc.):

Combinación	Se aplica el estilo a:	Ejemplo
A, B	Cualquier elemento A y/o B. Es decir, se aplica la misma regla a varios elementos.	A, B {color:gold;}
A B	Cualquier elemento B descendiente de un elemento A; es decir, un hijo, un hijo de otro hijo, etc.	A B {color:gold;}
A > B	Cualquier elemento B que sea hijo directo de un elemento A.	A > B {color:gold;}
A + B	Cualquier elemento B que sea hermano siguiente de un elemento A; es decir, los siguientes hijos del mismo padre.	A + B {color:gold;}
A ~ B	Cualquier elemento B que sea hermano siguiente y contiguo de un elemento A; es decir, los hijos B que siguen a A sin otros hijos diferentes en medio (C).	A ~ B {color:gold;}

El selector CSS `h1 ~ p {}` afecta por igual a los dos párrafos ya que ambos son hermanos posteriores del encabezado (h1): están contenidos directamente en el mismo elemento html, ambos (el título y los párrafos) tienen al div como padre. Si sustituimos `h1 ~ p {}` por `h1 + p {}`, el estilo solo afectará al primer párrafo.

`h1 ~ p {}`

```
<div>
```

```
  <h1>Título</h1>
```

```
  <p>Érase una vez...</p>
```

```
  
```

```
  <p>La historia sigue...</p>
```

```
</div>
```

## **Precedencia de reglas**

En un mismo documento HTML podemos utilizar diferentes formas de dar formato:

- Poner estilo directamente (inline) en los elementos HTML.
- Poner estilo utilizando "id": #elemento
- Poner estilo utilizando "class": .elemento
- Utilizando etiquetas o elementos que tienen estilo implícito como son: span, div, h1, etc

Entonces, si se usan varias hojas de estilo, se puede producir un conflicto sobre qué estilo se aplica a un selector en particular. En este caso se tendrán en cuenta las siguientes reglas:

- **! important**

Puede establecerse una regla como importante al especificar ! important. Un estilo designado como importante prevalecerá sobre estilos contradictorios de similar nivel.

Ejemplo:

```
BODY {  
    background: url(bar.gif) white;  
    background-repeat: repeat-x ! important  
}
```

- **Reglas de selector: calculando especificidad**

Las hojas de estilo también pueden primar sobre hojas de estilo en conflicto basándose en su nivel de especificidad, donde un estilo más específico siempre prevalecerá sobre uno menos específico.

La prioridad es en orden de importancia:

1. prevalece y se aplica estilo inline
2. sobre poner estilo con "id"
3. "class"
4. sobre el estilo del elemento HTML (p, h1, div, etc.).

## **Homework, darle los formatos de la red social**

Las características para dar el formato deseado son:

- Para la aplicación (*container*): la letra será Helvetica de 20px, el fondo de color blanco (#FFFFFF), la letra de color gris oscuro (#2A2929, dark grey), con un margen de 10px.

```
#container {  
    font-family: Helvetica, Geneva, Arial, sans-serif;
```

```
padding-left: 20px;
background-color: #FFFFFF; /* beige */
font-color: #2A2929; /* dark grey */
margin: 10px;
}
```

- Los enlaces o links (*a*) serán de tamaño algo mayor a la letra normal de la aplicación (120%) y sin subrayado.

```
a:link{
    text-decoration: none;
    font-size: 120%;
}
```

- Cuando se pase ratón por los enlaces (*a:hover*) se deberán ver de color bronce claro (#cd7f32, light bronze).

```
a:hover {
    color: #cd7f32;
}
```

- Los títulos o cabeceras de primer nivel (*h1*) serán de color bronce (#d69544, bonze) de tamaño doble a la letra normal de la aplicación (200%) y con un margen de 20px.

```
h1 {
    color: #d69544;
    font-size: 200%;
    margin: 20px;
}
```

- El formulario (*form*) estará rodeado por una línea de borde de 2px y color sólido bronce. Dicha línea de borde tendrá las esquinas redondeadas (10px). Si el contenido interior al borde sobrepasa los límites, que no aparezcan (hidden) barras de scroll. El texto tendrá un margen de 5px. El tamaño de letra al 120%.

```
form {
    border: 2px solid #d69544;
    border-radius: 10px;
    border-style: solid;
    overflow: hidden; /* ésto es para que no se vea la barra de scroll */
    margin: 5px;
}
```

- Tamaño de letra de las etiquetas (*label*) al 110%, con un espacio alrededor de su contenido (padding) de 5px. El texto aparecerá alineado a la izquierda y con un margen de 5px.

```
label {  
  font-size: 110%;  
  padding: 5px;  
  text-align: left;  
  margin: 5px;  
}
```

- Todos los elementos de la clase "inputs" tendrán un espacio alrededor de su contenido (padding) de 5px. El texto aparecerá alineado a la izquierda y con un margen de 5px.

```
.inputs {  
  border-radius: 5px;  
  padding: 5px;  
  text-align: left;  
  margin: 5px;  
}
```

Nota: el label selector de css es para que marque todo lo que tenga etiqueta en el html.

Por otra parte, el label tag de html define una etiqueta para los elementos <button>, <input>, <meter>, <output>, <progress>, <select> o <textarea>.

El atributo <label> debe ser igual al atributo id del elemento relacionado para unirlos.

Ejemplo de uso:

```
<form action="/action_page.php">  
  <label for="male">Male</label>  
  <input type="radio" name="gender" id="male" value="male"><br>  
  <label for="female">Female</label>  
  <input type="radio" name="gender" id="female"  
value="female"><br>  
  <label for="other">Other</label>  
  <input type="radio" name="gender" id="other"  
value="other"><br><br>  
  <input type="submit" value="Submit">  
</form>
```

## **Módulo 3: La aplicación web en el lado servidores**

En esta unidad vamos a empezar a estudiar cómo se debe programar la componente servidor de nuestra aplicación web.

Es decir, vamos, lo que vamos a hacer es ver cómo se puede responder a las peticiones que en el módulo anterior hicimos desde el HTML.

### **Introducción a python somera:**

- Python está orientado a objetos, es decir, todo valor es un objeto con una clase asociada.
- Las variables y expresiones en Python tienen un tipo, también conocido como clase.
- El tipo de una variable se determina dinámicamente, lo cual supone que no hace falta declarar variables.

En este ejemplo de código puedes ver la asignación de dos variables: en a (nombre) se asigna un entero (clase) y en b (nombre) un string (clase).

```
>>> a = 10
>>> type(a)
<class 'int'>
>>> b = 'cadena'
>>> type(b)
<class 'str'>
>>> a+b
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

Ésto da error porque a es de tipo int y b es de tipo cadena, sin embargo:

```
>>> str(a)+b
'10algo'
```

Si funciona porque la función str convierte a la variable “a” en una cadena y por lo tanto, el resultado es ‘10algo’. Pero ojo al gol, ese comando de por sí, no hace que a cambie de tipo int a str porque así como está el comando, no lo asigna a la variable “a”.

Por otra parte, Python es sensible a la diferencia entre mayúsculas y minúsculas (case sensitive). En este ejemplo, "VAR", "Var" y "var" son tres variables diferentes.

En Python la ayuda siempre está disponible a través del intérprete. Si quieres saber cómo funciona un objeto, todo lo que tienes que hacer es llamar “help (<objeto>)”.

### Sintaxis

- Python no tiene un carácter especial para indicar el fin de una sentencia como el lenguaje C, que utiliza un punto y coma (;). Sin embargo, para indicar que un conjunto de instrucciones forma un bloque, se deben poner con **indentación**.
- La forma más habitual de poner comentarios es con el símbolo hashtag (#) y van hasta el final de la línea. También se pueden poner comentarios de múltiples líneas encerrando el texto entre comillas triples.  
La diferencia entre comentar con # y comillas triples es que, cuando se si se escribe una librería, al hacer help(<objeto>) al importar la, va a aparecer lo que está entre comillas triples pero no lo que está con el hashtag
- Se asignan valores a variables con el símbolo igual (=) y comparamos igualdad con dos iguales (==), mientras que la exclamación de cierre y un igual (!=) consulta desigualdad.
- Para las cadenas se pueden usar comillas simples (') o dobles (")

### Tipos de Datos

Las estructuras de datos disponibles en Python son: listas, tuplas y diccionarios.

- Las listas son como arrays de una dimensión, pero se pueden construir listas de listas.
- Los diccionarios son arrays asociativos, también llamados tablas hash.
- Las tuplas son arrays de una dimensión inmutables.

En todos estos casos, los elementos contenidos pueden ser de cualquier tipo; por lo tanto, se pueden mezclar, por ejemplo, enteros con cadenas de caracteres.

Recuerda que en la mayoría de lenguajes de programación el índice del primer elemento es siempre 0. Por su parte, los índices negativos se usan para referenciar los elementos de una lista contando desde el final: -1 es el último elemento.

**Programación del servidor**

La acción en una aplicación web siempre comienza con una petición desde el cliente al servidor y éste cliente normalmente va a ser un navegador web o browser, por ejemplo, Internet Explorer (ahora es el Microsoft Edge), el Chrome, el Firefox...

Cualquier navegador que esté disponible en el dispositivo que estéis utilizando.

Esto va a ser cierto tanto si el dispositivo es un ordenador de sobremesa, como si es un ordenador portátil, aunque también puede ser un teléfono móvil o una tableta.

Esa petición puede ser para ver una página.

Otro ejemplo de petición del cliente al servidor es cuando, después de completar nuestros datos de registro, pinchamos en el botón “Log In”.

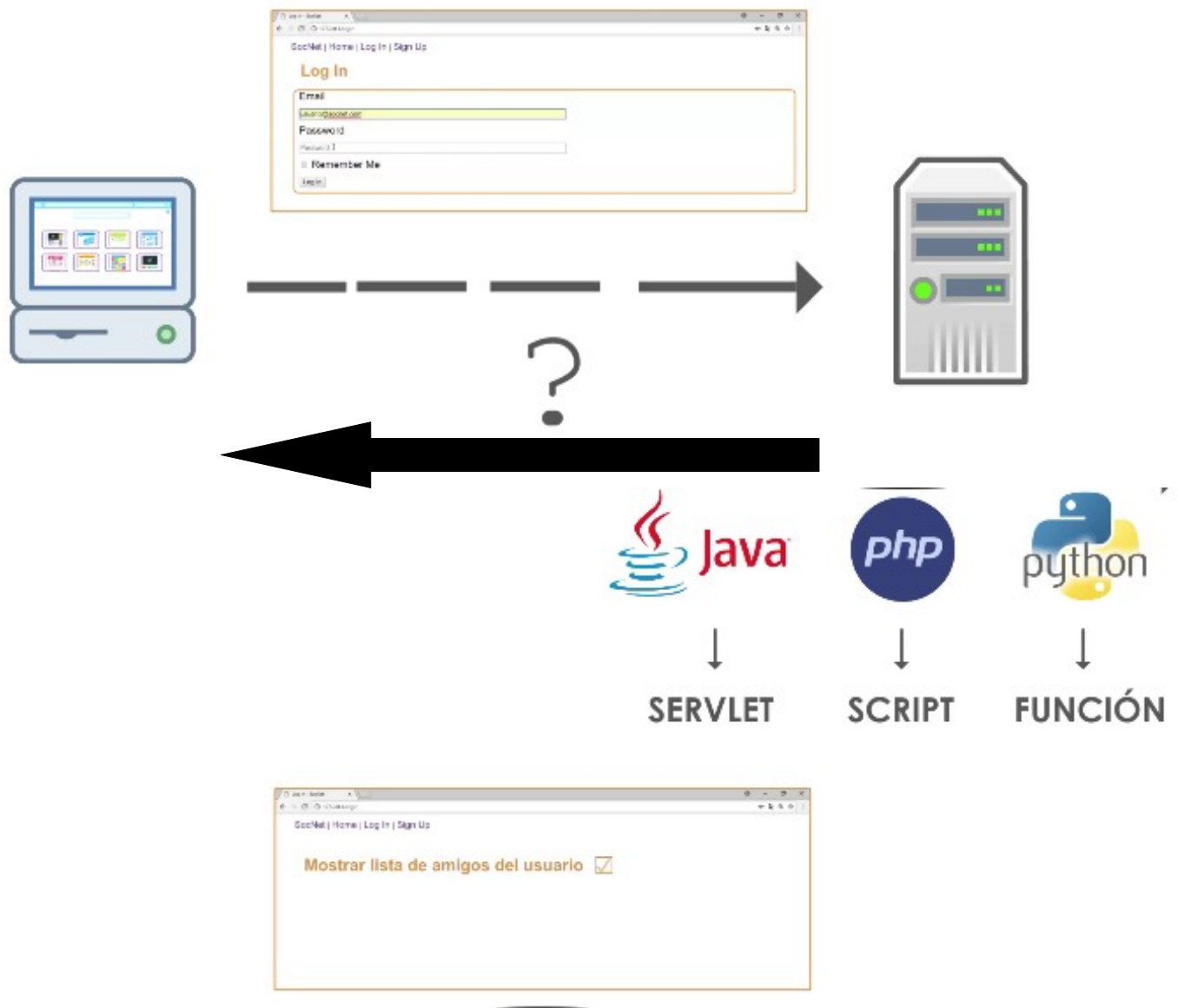
En todo caso, lo que hace el navegador es enviar una petición http al servidor.

Así que ahora lo que vamos a aprender es cómo debe hacer el servidor para responder a estas peticiones pues, de forma general, se puede decir que necesitamos programar cómo atender cada petición.

Si desde el cliente se envía una petición de “log in”, ¿qué debemos hacer?

¿Y si lo que se envía es un comando “mostrar la lista de amigos del usuario”?

Lo que deberíamos hacer entonces es asignar un “programa” (en realidad, un fragmento de código) por cada tipo de petición que puede hacer el cliente; y la forma que tiene este fragmento depende del lenguaje de programación.



Por ejemplo, en Java se podría programar un servlet para que responda a cada tipo petición, en PHP se escribiría un script (fichero), y en Python se programará una función por cada tipo de petición.

Precisamente, nosotros vamos a programar el servidor utilizando el lenguaje de programación denominado Python.

Una de las grandes ventajas este lenguaje de programación es que casi siempre que tengamos un problema, alguien ya lo habrá solucionado. Estas soluciones vienen en forma de **librerías** o en Python se denominan paquetes.



En particular, nosotros, para programar nuestro servidor web, vamos a utilizar un paquete existente que se denomina Flask.

Ejemplo: el código Python sacado del fichero server.py Este código Python está sacado del fichero server.py que ya te hemos facilitado en la unidad anterior. Como se puede ver al inicio del mismo, se hace la importación de la biblioteca Flask:

```
from flask import Flask, request  
app = Flask(__name__)
```

Hay dos formas de procesar cada una de las peticiones que desde el cliente hace al servidor:

Por ejemplo, en el caso de login, primero vemos cuando se carga por primera vez el formulario en el cual el usuario de la aplicación desea poner sus datos para entrar en la aplicación. En este caso, el código en Python:

```
@app.route('/login', methods=['GET'])  
def login():  
    return app.send_static_file('login.html')
```

muestra la página web que tenemos ya creada llamada login.html.

Sin embargo, una vez el usuario de la aplicación ha rellenado el formulario y pulsado el botón submit, el código Python primero revisa que estén todos los campos en el formulario y segundo crea la página web resultado del código HTML y de la evaluación del código Python que hay entre las etiquetas html, justo después de la palabra reservada return.

```
@app.route('/processLogin', methods=['GET', 'POST'])  
def processLogin():  
    missing = []  
    fields = ['email', 'passwd', 'login_submit']  
    for field in fields:  
        value = request.form.get(field, None)  
        if value is None:  
            missing.append(field)  
    if missing:  
        return "Warning: Some fields are missing"  
    return '<!DOCTYPE html> ' \\  
        '<html lang="es">' \\  
        '<head>' \
```

```
'<link href="static/css/my-socnet-style.css"
rel="stylesheet" type="text/css"/>' \
'<title> Home - SocNet </title>' \
'</head>' \
'<body> <div id ="container">' \
'<a href="/"> SocNet </a> | <a href="home"> Home </a>
| <a href="login"> Log In </a> | <a href="signup"> Sign Up </a>' \
'<h1>Data from Form: Login</h1>' \
'<form><label>email: ' + request.form['email'] + \
'</label><br><label>passwd: ' + request.form['passwd']
+ \
'</label></form></div></body>' \
'</html>'
```

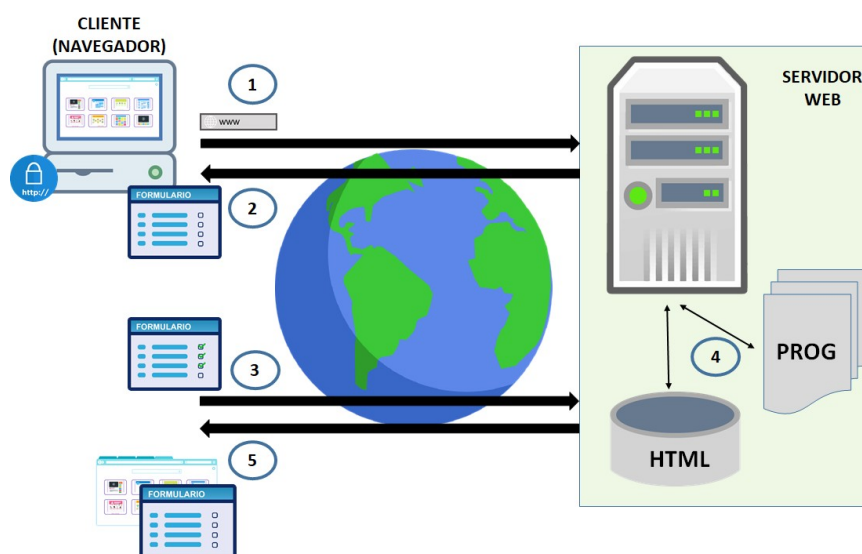
Explicación, si faltan rellenar campos, va a devolver que hay datos para llenar antes de hacer submit, porque entra primero a ese return, si está todo bien, regresa código html que va a ser la página resultado, y notar que el lenguaje usado es HTML

En resumen, para poder programar el servidor de nuestra aplicación web, vamos a necesitar aprender dos cosas. Primero, cómo programar en el lenguaje de programación Python.

Y segundo, cómo utilizar la biblioteca Flask para programar las respuestas a las peticiones del cliente.

### **Introducción a Python para implementar el Servidor Web**

Como ya hemos dicho varias veces, una aplicación web es un diálogo entre una componente llamada **cliente** y otra componente llamada **servidor**. Este diálogo se produce utilizando el protocolo HTTP: es el "lenguaje" en el que ambas componentes se comunican. Este lenguaje no es el lenguaje de programación en el que las componentes están implementadas. Cada una estará implementada, normalmente, en un lenguaje diferente. Es como dos personas que hablan idiomas diferentes, una castellano y otra alemán, que para comunicarse entre sí hablan en un tercer idioma: el inglés.



El cliente normalmente es un **navegador web**; ya hay muchos de ellos disponibles, casi todos gratuitos, así que no será necesario que nos preocupemos de su implementación. Nuestro problema será la implementación del servidor, que haremos en **Python**. Y nuestro servidor se comunicará con los clientes utilizando ese lenguaje común **HTTP**.

Si bien ya conocemos el lenguaje que utilizan para comunicarse, todavía no sabemos qué mensajes se envían. El cliente envía mensajes solicitando recursos, a los que hace referencia a través de una **URL**. Por ejemplo, la siguiente URL hace referencia a la página de horarios de un supuesto cine: [http://www.cine\\_de\\_mi\\_barrio.com/horarios.html](http://www.cine_de_mi_barrio.com/horarios.html)

Se puede observar que esta URL en particular apunta a un **fichero HTML**: este es el tipo de recurso que normalmente solicita el cliente, porque es la base de casi todos los documentos alojados en la web. Incluso cuando el recurso que solicita el cliente no existe previamente (o al menos no existe un documento HTML que lo represente), el servidor genera dinámicamente una respuesta en forma de documento HTML. Es prácticamente lo único que son capaces de mostrar los navegadores.

Si el cliente, además de describir el recurso al que quiere acceder, necesita enviar al servidor parámetros para que el servidor pueda adaptar su respuesta, estos parámetros pueden ser parte de la URL (si el cliente utiliza el método **GET** del protocolo HTTP) o estar incrustados dentro de la solicitud HTTP (si el cliente utiliza el método **PUT**).

En todo caso, para que el cliente no tenga que especificar esta petición HTTP "a mano", normalmente en el navegador se le ofrecen formas amigables de realizar las peticiones.

Así, cuando el cliente pincha un enlace (link), el navegador construye una petición HTTP que contenga una URL que apunte al recurso (documento) representado por ese enlace. De la misma forma, si tiene que enviar parámetros adicionales, se le ofrece un medio relativamente sencillo para especificarlos: los **formularios**, de los cuales ya te hablamos en el apartado de HTML.

## Nuestro actual servidor web

### Análisis previo a un servidor de flask

## Inicialización

Para que nuestro servidor Python pueda recibir las peticiones de los clientes, se debe realizar una instancia de aplicación. Para ello normalmente se crea una instancia de la clase Flask (previa importación para que el intérprete de Python la conozca).

Para importar:

```
from flask import Flask  
app = Flask(__name__)
```

Con esto, se inicializa la instancia llamada app. Esta instancia debe recibir un nombre de aplicación; lo más normal es pasar el parámetro `__name__` (cuidado, son 2 guiones bajos ( 2 guiones bajos antes y después de "name")), que contiene el nombre del programa que se está ejecutando.

## Ejecución del servidor

Para lanzar el programa, que finalmente será el que esté esperando las peticiones del cliente, se utiliza el método `run` de la instancia de Flask antes creada y que hemos almacenado en la variable `app`.

```
if __name__ == '__main__':  
    app.run(debug=True)
```

La condición `__name__ == '__main__'` nos garantiza que el servidor únicamente se iniciará cuando el script sea ejecutado directamente, no cuando sea importado por otro script.

Una vez que el servidor comience a ejecutarse, se meterá en un bucle esperando por peticiones y atendiéndolas a medida que llegan. Esto continuará hasta que la aplicación sea detenida.

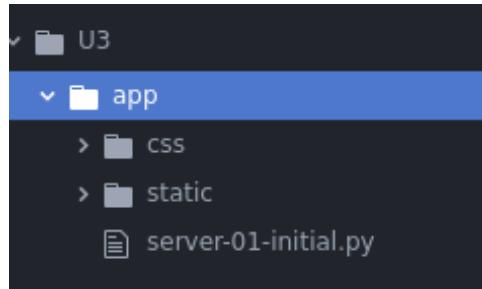
Por otra parte en éste código

```
# start the server with the 'run()' method  
if __name__ == '__main__':  
    if sys.platform == 'windows': # different port if running on Windows  
        app.run(debug=True, port=80)  
    else:  
        app.run(debug=True, port=8080)
```

Cuando se ejecute este código, primero se verificará si la plataforma sobre la que está ejecutando es el sistema operativo **del servidor** de Windows (en cuyo caso el valor de la variable de sistema `sys.platform` será 'windows'), pues de esto depende el puerto de salida dado que para mac y linux, el puerto es 8080.

Nota mental: La ruta donde tiene que estar el archivo de server es dentro de la carpeta llamada app, al mismo nivel que static y css

En nuestro ejemplo:



Por último, si ejecutamos y navegamos:

en el directorio raíz, va a imprimir el mensaje del siguiente código:

```
@app.route('/')
```

```
def index():
```

```
    """
```

```
    It process the '/' url.
```

```
    :return: basic HTML
```

```
    """
```

```
    return "Hello Word! This is the answer of the server"
```

como app en una instancia `@app.route('/')` vendría a se como pasarle al objeto y se define para el directorio raíz la función index, la cual lo que despliega la frase "Hello Word! This is the answer of the server" del navegador.

### Devolución de cosas más complejas

```
@app.route('/ex2')
```

```
def function_for_ex2():
```

```
    """
```

```
    It process the '/' url.
```

```
    :return: basic HTML for /ex2
```

```
    """
```

```
    return '<!DOCTYPE html> ' \
```

```
        '<html lang="es">' \
```

```
        '<head>' \
```

```
        '<title> This is the page title </title>' \
```

```
'</head>' \
'<body> <div id ="container">' \
'<h1>Example of HTML Content</h1>' \
'Return to the homepage by clicking <a href="/">
here</a> </html>' \
'</html>'
```

Eso es lo que era la solución a que el return en python de una html: se pone entre comillas simples por renglón cada tag y su contenido y para pasar al renglón siguiente solo pide haber 1 espacio entre la comillas t el “\”

### Otras opciones de rutas

El mecanismo de asociar URLs con funciones (las rutas) permite describir más situaciones. Por ejemplo, es posible especificar que una ruta solo se asocie con una URL cuando se utilice un método específico dentro del protocolo HTTP; es decir, según se la petición se haga con GET o con POST.

Recuerda que, en general, se utiliza el método POST para enviar datos de formularios, mientras que GET se utiliza en las peticiones que no envían datos.

## Sección 5: javascript

El código JavaScript se incluye en los documentos HTML mediante la etiqueta `<script>`. Puedes incluir tantos bloques de código dentro de etiquetas `<script>` como necesites en cualquier parte del documento dentro del head o el body. Existen dos formas de añadir el código dentro de una etiqueta `<script>`. En la más simple, el bloque de código se inserta "tal cual" entre la etiqueta script de apertura y la de cierre:

```
<script>

    // Tu código irá aquí.

</script>
```

El problema de hacerlo de esta forma es que el código está incrustado dentro del documento, por lo que no puede ser reutilizado en otros documentos. Además, tampoco podrá ser cacheado en el cliente para optimizar futuros accesos. Por eso, lo más habitual es que el código JavaScript de un documento HTML se descargue en archivos externos. Igual que antes, en un mismo documento podrás incluir tantos ficheros externos con código Javascript

como necesites (recuerda el ejemplo de la web de la Universidad Autónoma de Madrid que veíamos en la introducción).

Para indicar el fichero de código vinculado al documento se utiliza el atributo `src` de la etiqueta `<script>`, que le indica al navegador la URL (relativa o absoluta) de la que descargar el código:

```
<script src="codigo.js"></script>
```

NOTA IMPORTANTE 1. Por simplicidad, en los ejemplos que vamos a describir en el curso el código se incluye dentro del propio documento HTML, pero todo lo que se describe es aplicable al caso de ficheros de código externos.

NOTA IMPORTANTE 2. El código JavaScript se ejecuta en el cliente una vez se ha descargado el documento HTML al que está vinculado. En este escenario, los mecanismos por los cuales se ha generado el documento en el lado del servidor son una caja negra para el navegador. Por ello, también por simplicidad, en todos los ejemplos y ejercicios propuestos vamos a trabajar con contenido estático. Esto te va a permitir probar el código simplemente abriendo tus ficheros locales en un navegador sin necesidad de tener arrancado el servidor de la aplicación.

Una vez pasemos a implementar funcionalidad JS en el contexto de nuestra red social, ya sí llevaremos el código JavaScript a ficheros externos y comenzaremos a trabajar con contenido dinámico facilitado por un servidor.

## Sentencias, variables y tipos de datos

Al igual que en lenguajes como C o Java, las sentencias de código JavaScript se separan por punto y coma (;) aunque, a diferencia de estos lenguajes, solo es obligatorio cuando hay más de una sentencia en una misma línea. Por ejemplo, el siguiente fragmento de código es válido y no produciría errores aunque al final de la segunda y la cuarta línea se haya omitido el punto y coma final. En estos casos, el propio salto de línea actúa como separador de sentencias:

```
<script>
```

```
  a = 3;
```

```
  a = 12
```

```
  a = 7; a = 15;
```

```
a = 45; a = 14  
</script>
```

Sin embargo, la tercera línea del siguiente fragmento producirá un error de sintaxis al no haber separador entre las dos sentencias que contiene:

```
<script>  
a = 3;  
a = 12  
a = 7 a = 15;  
a = 45; a = 14  
</script>
```

La sintaxis para definir e inicializar una variable en JavaScript es:

```
var nombre = valor;
```

siendo nombre el nombre de la variable y valor el valor que se le quiere asignar inicialmente. Una variable se puede definir sin inicializar, en cuyo caso tomará el valor undefined.

JavaScript es un lenguaje orientado a objetos, aunque el concepto de objeto en JavaScript es ligeramente diferente al de otros lenguajes orientados a objetos. En JavaScript, un objeto en realidad es un array asociativo y, por ejemplo, el concepto de clase no se introduce hasta versiones muy recientes del lenguaje (ECMAScript 6 - ECMAScript 2015). Un objeto JavaScript es una estructura de datos que contiene propiedades que definen sus características y métodos (funciones) que implementan una determinada funcionalidad sobre el objeto. Por ejemplo, "Esto es un texto" es un objeto de tipo String (cadena de caracteres). Los strings tienen una propiedad de solo lectura length que nos permite saber su longitud (es decir, el número de caracteres que contiene) y un método toUpperCase que devuelve el valor del string convertido a mayúsculas. Más adelante profundizaremos en otras propiedades y métodos de los strings.

Para acceder a las propiedades y métodos de un objeto en JavaScript, se utiliza la misma sintaxis de C o Java para acceder a los elementos de una estructura u objeto:

"Esto es un texto".length devuelve el tamaño de la cadena de caracteres (16).



"Esto es un texto".toUpperCase() devuelve "ESTO ES UN TEXTO".

s.length devuelve la longitud de s si s es un String.

s.toUpperCase() devuelve el valor de s si s es un String.

JavaScript no es un lenguaje tipado en lo que se refiere a la definición de variables y parámetros. Esto permite que, a diferencia de lo que ocurre en lenguajes como C o Java, el siguiente fragmento de código sea válido:

```
<script>
```

```
var a = 3;
```

```
a = "String";
```

```
</script>
```

Sin embargo, en realidad, JavaScript sí que es un lenguaje tipado en el que se distinguen los siguientes tipos de datos:

- Number para valores numéricos, independientemente de si son enteros o en coma flotante.
- String para cadenas de caracteres.
- Boolean para valores lógicos: verdadero (true) y falso (false).
- Null para el valor nulo o vacío (null).
- Undefined para el valor desconocido (undefined). Por ejemplo, cuando se define una variable y no se le asigna un valor de inicialización, esta toma el valor undefined. Debes tener en cuenta que en JavaScript los valores null y undefined no son equivalentes.
- Symbol para valores inmutables, similares a las constantes de otros lenguajes.
- Object para cualquier tipo de objeto, ya sea del propio JavaScript o definido por el usuario.

Lo que ocurre es que JavaScript es lo que se conoce como lenguaje de tipado dinámico (a veces también se habla de tipado débil). Esto quiere decir que el tipo de una variable o parámetro se determina dinámicamente en tiempo de ejecución. Esto hace que en el siguiente fragmento de código, la llamada al método toUpperCase() produzca un error cuando la variable a vale 3, ya que es un método que no está definido para un Number, mientras que después de asignar a la variable a un String devolverá el valor asignado en letras mayúsculas:

```
<script>
  var a = 3;
  a = a.toUpperCase(); // ERROR!!!
  a = "String";
  a = a.toUpperCase(); // a pasa a valer "STRING"
</script>
```

También debes tener en cuenta que en tu código JavaScript podrías asignar valores y utilizar variables que previamente no han sido definidas. El intérprete JavaScript asume que la primera vez que se utiliza una variable no definida previamente es el momento de su definición. Esto hace que el primer fragmento de código que mostrábamos en esta sección sea válido aunque la variable `a` no se había definido previamente.

Aunque es posible utilizar variables no definidas, te recomendamos que, como buena práctica de programación para evitar errores, siempre defines e inicialices las variables que utilizas en tus programas al comienzo de tus ficheros y fragmentos de código JS.

## Estructuras de control

Código	Denominación
<pre>if(condición) {     código true } else {     código false }</pre>	Estructura de control condicional
<code>(condición) ? valor_true : valor_false</code>	Operador de control condicional
<pre>switch(expresión) {     case condición_1:         código condición_1         [break;]     ...     case condición_n:         código condición_n         [break;]     default:         código default }</pre>	Estructura de control condicional basada en la evaluación del valor de una expresión
<pre>for(ini; condición; actualización){     código bucle }</pre>	Bucle condicional
<pre>while(condición) {     código bucle }</pre>	Bucle incondicional
<pre>do {     código bucle } while(condicion)</pre>	Bucle incondicional
<code>break</code>	Salida de un bloque de código
<code>continue</code>	Parada de la ejecución de un bucle para volver a evaluar la condición de parada
<pre>try {     código js } catch(err) {     código que gestiona el error } finally {     código a ejecutar haya o no error }</pre>	Captura y gestión de errores en el código

## Ejecución del código JavaScript

El código *JavaScript* se interpreta conforme va siendo descargado, ya sea dentro del propio documento, ya sea como fichero externo. De esta forma, en el siguiente ejemplo, al “pasar” por el bloque *head*, el intérprete creará una variable (*a*) con valor 3 que se podrá utilizar en cualquier momento mientras se esté visualizando el documento HTML correspondiente en el navegador. En este caso, por ejemplo, para mostrar una ventana de aviso con su valor al invocar a la función *alert*. De nuevo, el intérprete invoca a esta función al “pasar” por ese fragmento de código:

```
<html>
<head>
  <script>
    var a = 3;
  </script>
</head>
<body>
  Ejemplo sencillo de uso de <i>JavaScript</i>
  <script>
    alert(a);
  </script>
</body>
</html>
```

El segundo mecanismo para ejecutar funcionalidad *JavaScript* en un documento HTML es crear una **función** que se ejecutará cuando se produzca un determinado "evento". La definición de una función tiene la siguiente sintaxis:

```
<html>
<head>
  <script>
    var a = 3;

    function mostrarVariable(valor) {
      alert(valor);    // Aquí mostramos el valor que recibe
                        // como parámetro
      alert(a);        // Aquí, el valor de la variable
                        // global
    }
  </script>
</head>
<body>
  Ejemplo sencillo de uso de <i>JavaScript</i>
  <input type="button" value="Pulsar..."
  onclick="mostrarVariable(a)" />
```

```
<script>
    alert(a);
    a = 5;
</script>
</body>
</html>
```

La secuencia de acciones realizadas por el intérprete *JavaScript* al interpretar este código es:

- Crear una variable *a* y asignarle el valor numérico 3.
- Crear una función *mostrarVariable* que queda a la espera de ser invocada.
- Invocar a la función *alert* para mostrar el valor actual de *a*.
- Cambiar el valor de *a* asignándole el valor numérico 5.

¿Cuándo se invoca la función que hemos creado? En este caso, al pulsar el botón que hemos añadido en el cuerpo del documento. Para ello utilizamos su atributo *onclick*, en el que se indica el código *JavaScript* a ejecutar al "hacer clic" sobre el elemento.

## Eventos HTML

Ya te hemos introducido el concepto de **evento** como "algo" que va a disparar la ejecución de un fragmento de código *JavaScript* en el lado del cliente. Ahora vamos a dar una definición más formal de qué es un evento HTML, estudiando además cómo vincular código *JavaScript* con un evento determinado, identificando los principales eventos HTML y viendo en detalle algunos ejemplos sencillos.

En la mayoría de los casos, la programación *JavaScript* en el lado del cliente es una programación basada en eventos. Lo que le vamos a decir al motor *JavaScript* del navegador es que cuando ocurra un determinado evento HTML, ejecute un fragmento de código. Un evento HTML puede ser el resultado de una acción que realiza el usuario de la aplicación (por ejemplo, que pulse un botón, que pulse una tecla estando dentro de un *input* o que pase el ratón por encima de un elemento HTML determinado), pero también puede estar asociado a acciones internas gestionadas por el propio navegador (por ejemplo, que se cumpla un *timer* o que una imagen haya terminado de cargar). Todos los elementos HTML tienen propiedades que permiten vincular código *JavaScript* a sus eventos asociados. Podrás identificar fácilmente estas propiedades en la documentación de referencia del lenguaje HTML porque todas ellas comienzan por el prefijo "**on**".

La sintaxis general en el documento HTML para asignar una funcionalidad determinada a un elemento HTML es:

```
<tag onXXX="tu código para XXX" onYYY='tu código para YYY'">
</tag>
```

Este código **tiene que ir entre comillas**

Con este código estaríamos indicando al navegador que cuando el elemento definido por la etiqueta (tag) HTML (una imagen —*<img>*—, el cuerpo del documento —*<body>*—, una tabla —*<table>*

—, etc.) "lance" el evento XXX o el evento YYY, se ejecute el código que se le proporciona en cada caso. Como puedes ver, esta asignación es equivalente a la asignación del valor de cualquier propiedad HTML, por lo que, al igual que ocurre con el resto propiedades, el valor que se da a la propiedad puede ir entre comillas simples (') o dobles (").

Estos son algunos de los eventos HTML que más habitualmente se procesan en las aplicaciones web. Aunque procesando estos eventos podrías desarrollar cualquier aplicación web típica, como ya sabes, puedes encontrar la lista completa de eventos HTML en la [documentación de referencia de la W3C](#). Recuerda que para procesar un evento tienes que dar valor a la propiedad correspondiente en el elemento HTML, por ejemplo, para procesar el evento click debes fijar la propiedad *onclick*:

- *click*: se produce cuando el usuario hace click sobre el elemento HTML.
- *mouseover*: se produce cuando el usuario entra con el ratón dentro del elemento HTML.
- *mouseout*: se produce cuando el usuario sale con el ratón del elemento HTML.
- *focus*: se produce cuando el elemento HTML recibe el foco.
- *blur*: se produce cuando el elemento HTML pierde el foco.
- *change*: se produce cuando el valor del elemento HTML cambia (para `<input>`, `<select>` y `<textarea>`).
- *keydown/keypress*: se producen cuando el usuario presiona una tecla estando dentro del elemento HTML. Aunque realmente son eventos distintos, en la mayoría de las ocasiones se pueden considerar equivalentes.
- *keyup*: se produce cuando el usuario suelta una tecla estando dentro del elemento HTML.
- *submit*: se produce cuando se hace el submit de un formulario (para `<form>`).
- *load*: se produce cuando finaliza la carga del elemento HTML. Típicamente se utiliza para validar la carga del documento.
- *unload*: se produce cuando se descarga la página (para `<body>`).
- *error*: se produce cuando se produce un error al cargar un fichero externo.

### **Atributos para leer la información proporcionada por el usuario**

El comportamiento dinámico del **atributo *value*** de los objetos que representan *inputs* es especialmente relevante para poder validar desde *JavaScript* los datos introducidos por el usuario.

Dado que es un atributo de tipo ***String***, en la mayoría de los casos la validación del valor de un *input* se realizará mediante alguna de las funciones ofrecida por el lenguaje que operan sobre strings:

- **Métodos de *String***. Ninguno de los métodos de *String* es destructivo, es decir, que todos devuelven copias del objeto sobre el que se invocan y no modifican el valor de este. Aquí damos una lista de los métodos más destacados, en la documentación de la W3C puedes encontrar un [listado completo con ejemplos de uso](#) de cada uno de ellos:
  - *charAt()* devuelve el carácter que ocupa una posición determinada del string.
  - *endsWith()* predicado que indica si el string acaba de una forma determinada.
  - *includes()* predicado que indica si el string contiene una subcadena.

- *indexOf()* busca la primera aparición de una subcadena.
- *lastIndexOf()* busca la última aparición de una subcadena.
- *match()* predicado que indica si el string se adapta a un patrón determinado.
- *repeat()* concatena n copias del string en un nuevo string.
- *replace()* sustituye una subcadena por otra.
- *slice()*, *substr()* o *substring()* devuelve una subcadena.
- *split()* divide el string en tokens en función de un separador.
- *startsWith()* predicado que indica si el string empieza de una forma determinada.
- *toLowerCase()* convierte el string a minúsculas.
- *toUpperCase()* convierte el string a mayúsculas.
- *trim()* elimina espacios en blanco de los extremos.
- Funciones de conversión y validación de tipos:
  - [\*isNaN\(string\)\*](#) predicado que valida si un string representa un valor numérico.
  - [\*parseFloat\(string\)\*](#) devuelve el string que se le pasa como parámetro en forma de número en coma flotante.
  - [\*parseInt\(string\)\*](#) devuelve el string que se le pasa como parámetro en forma de número entero.
  - [\*String\(objeto\)\*](#) convierte el objeto que se le pasa como parámetro a String.
- Interpretación de código JS:
  - [\*eval\(string\)\*](#) interpreta el código JS que se le pasa como parámetro.

De manera análoga al atributo *value*, en los inputs de tipo **check** y **radio** podemos saber si el usuario ha marcado o no el elemento en pantalla mediante el atributo *checked* del *HTMLInputElement* que representa al input en el modelo DOM del documento. En este caso, se trataría de un atributo booleano que se inicializaría con el valor asignado al atributo *checked* del elemento en el documento HTML y, posteriormente, pasaría a valer *true* (seleccionado) o *false* (no seleccionado) según el usuario interactúa con el elemento en pantalla.

Finalmente, recordemos que los `<select>` y `<textarea>` también son campos de formulario que nos permite recibir información del usuario de la aplicación.

El único ingrediente que nos falta para implementar la validación de formularios en *JavaScript* es saber **cómo invocar** el código que implementa las validaciones desde nuestra aplicación web. Aunque lo cierto es que de forma indirecta ya hemos visto un posible mecanismo para hacerlo.

Una primera opción es validar los datos suministrados por el usuario en el momento en el que los **introduce** en la aplicación. Para ello, podríamos utilizar como disparador el evento **change** del campo de formulario que recoge los datos. Esto ya lo hemos hecho en el último ejercicio que os proponía, con la salvedad que en el ejercicio no validábamos la lista de amigos, sino que mostrábamos todos los que estaban seleccionados.

### **Validación de datos en el momento del envío al servidor**

Realizar la validación de datos según van siendo introducidos en el formulario es un mecanismo de validación muy ágil para indicar al usuario lo que está mal y que realice las correcciones oportunas

en el momento. Sin embargo, este mecanismo no evita el envío del formulario al servidor cuando contiene datos no válidos.

Para esto disponemos del evento **submit** de los formularios, que se lanza cada vez que se va a proceder al envío del formulario. Este evento espera que el código *JavaScript* que lo procesa, ya sea una función o un conjunto de sentencias, devuelva un valor booleano: *true* significa que se puede proceder al envío del formulario al servidor y *false* que, por las circunstancias que sea, el formulario no se debe enviar.

Este mecanismo se puede utilizar junto al mecanismo anterior (es decir, los datos se validan según se introducen, pero hay una validación final de conjunto para decidir si el formulario se envía o no al servidor para su procesamiento), o se puede utilizar sin ningún tipo de validación previa para validar en conjunto todos los datos que contiene un formulario.

Ahora sí que dispones de todos los ingredientes necesarios para validar los formularios de tus aplicaciones web con *JavaScript*. Sin embargo, antes de pasar a ver en detalle algún ejemplo, vamos a ver el atributo **forms** del *document*. Este atributo contiene un objeto de la clase *HTMLCollection* en el que están almacenados todos los formularios del documento HTML.

Un formulario, es decir, un elemento HTML definido con la etiqueta **<form>**, en el modelo DOM se representa con un objeto de tipo *HTMLFormElement*. El atributo más destacado de estos objetos es el atributo *elements*, del que se puede obtener una colección con todos los campos de formulario (*inputs*, *selects*, *textareas*...) incluidos dentro del formulario.

*HTMLFormElement* ofrece también una sintaxis abreviada de acceso a sus campos de manera que, si "miFormulario" es el nombre del formulario, con *document["miFormulario"]* accederemos al objeto que representa el campo de formulario cuyo nombre/id es "campoFormulario". El atributo *forms* del *document*, por tanto, permite acceder a los formularios y sus campos sin necesidad de realizar búsquedas en el árbol DOM.

Además, de los elementos que contienen, los objetos *HTMLFormElement* tienen dos métodos que se suelen utilizar muy habitualmente. El método **submit()** permite enviar el formulario al servidor mediante código JS (de forma análoga a lo que haría un botón de tipo *submit* cuando lo pulsa el usuario de la aplicación), mientras que el método **reset()** devuelve todos los campos del formulario a su valor por defecto.

### **Validación de datos usando js en un fichero aparte**

Lo que tenemos en este formulario son cuatro campos de texto y las reglas de validación que vamos a aplicar son que todos ellos estén informados y que el password y su repetición tengan el mismo valor.

En realidad, esto ya son reglas que estabais aplicando, del lado del servidor, pero lo que vamos a hacer ahora es que cuando yo trate de enviar un formulario que no sea válido, si alguno de los campos no satisface sus reglas de validación, directamente no se manda al servidor, se muestra una alerta con un mensaje de error.

Cuando yo cierre la alerta, informando del error, el foco se va a posicionar en el campo que lo producía para que el usuario pueda corregirlo de forma sencilla y, entonces, cuando yo proporcione



datos válidos, el formulario se envía y, tras una validación, que luego veremos que no se puede realizaren el lado del cliente, dará de alta al nuevo usuario.

Pues para implementar esta funcionalidad en el lado del cliente, lo que hemos hecho ha sido vincular al evento submit del formulario que queremos validar, en este caso el formulario de registro, una funcionalidad que debe devolver true en caso de que queramos que se envíe el formulario al servidor, y devolver false en caso de que haya un error y no se vayan a enviar esos datos al servidor.

```
<!--...-->
<form action="signup" method="post" name="signup" onsubmit="return validateProfile(this)">
  <ul class="errorMessages" style="..." id="errores_profile"></ul>

  <label for="nickname">
    Nickname
  </label>
  <div class="inputs">
    <input id="nickname" maxlength="64" name="nickname" placeholder="Nickname" size="80" type="text" value=""/>
  </div>
  <label for="email">
    Email
  </label>
  <div class="inputs">
    <input id="email" name="email" placeholder="Email" size="80" type="text" value=""/>
  </div>
  <label for="passwd">
    Password
  </label>
  <div class="inputs">
    <input id="passwd" name="passwd" placeholder="Password" size="80" type="password" value=""/>
  </div>
  <label for="confirm">
    Repeat Password
  </label>
  <div class="inputs">
    <input id="confirm" name="confirm" placeholder="Repeat Password" size="80" type="password" value=""/>
  </div>
  <div class="inputs">
    <input id="signup_submit" name="signup_submit" type="submit" value="Sign Up"/>
  </div>
</form>
</div>
</body>
</html>
```

Entonces, en este caso, la función es la función `validateProfile()`, que está definida, en el fichero `profile.js`. Ya no va a estar incluida dentro del propio código, sino que en la estructura que tenemos definida para el proyecto, para los ficheros estáticos, dentro del directorio JS vamos a tener un fichero con esta funcionalidad.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8"/>
    <link href="static/css/my-socnet-style.css" rel="stylesheet" type="text/css"/>
    <title>
      Sign Up - SocNet
    </title>
    <script src="static/js/profile.js"></script>
  </head>
```

Si nos vamos a este fichero, `profile.js`, vemos esta función.

Entonces, la primera regla de validación que va a aplicar es validar que el nickname está informado. Para eso accede al `value`, le hace un `trim` para descartar posibles espacios que haya delante o haya detrás.

De esta forma, una cadena con solo caracteres en blanco (se considera la cadena vacía) dará error. Y en caso de que no satisfaga la regla, va a mostrar la alerta con el error, va a poner en el foco en el campo y va a indicar que el retorno de la función debe ser false.

```

function validateProfile(form) {
    var retorno = true;

    if(form["nickname"].value.trim() == "") {
        alert("Nickname is required");
        form["nickname"].focus();
        retorno = false;
    }
    else if(form["email"].value.trim() == "") {
        alert("Email is required");
        form["email"].focus();
        retorno = false;
    }
    else if(form["passwd"].value.trim() == "") {
        alert("Password is required");
        form["passwd"].focus();
        retorno = false;
    }
    else if(form["passwd"].value != form["confirm"].value) {
        alert("Password and Repeat Password are not equal");
        form["passwd"].focus();
        retorno = false;
    }

    return(retorno);
}

```

## DHTML

Practica de modificado dinámico de HTML.

```
<script type="text/javascript">
```

```
    var document= signup.html;
```

```
    document.title = .getElementById('nuevo_titulo');
```

```
    alert(document.title)
```

```
</script>
```

```
<link href="static/css/my-socnet-style.css" rel="stylesheet" type="text/css"/>
```

```
<title >
```

```
    Titulo original
```

```
    <input id="nuevo_titulo" name="next" type="submit" value="nuevo_titulo"
onclick="tituloNuevo()"/>
```

```
</title>
```

Después de modificar dinámicamente el valor de cualquier propiedad de un elemento del documento, el valor original que se descargó del servidor se pierde. Por ejemplo, en el caso del

ejercicio anterior, al modificar el título del documento ya no podremos recuperar el título original, a no ser que lo hayamos almacenado en una variable *JavaScript* antes de realizar el cambio.

Puedes realizar la prueba accediendo al atributo *title* del *document* después de pulsar el botón. Verás que ahora este atributo contiene el nuevo título y no el que se indicaba en el momento de la descarga la etiqueta `<title>`. Sin embargo, también es importante resaltar que los cambios DHTML **tienen lugar en el cliente**, la versión del documento HTML que hay en el servidor no cambia. Por ejemplo, si vuelves a cargar el documento después de haber cambiado el título, comprobarás que al realizar una nueva petición al servidor, el documento se descarga nuevamente y su título es el que habíamos especificado dentro del código HTML.

Si quieres que los cambios que realizas vía DHTML en el cliente sean **persistentes**, debes implementar una funcionalidad que explícitamente realice estos cambios en el servidor, ya sea en el propio documento HTML en el caso de documentos estáticos, o en los almacenes de datos en los que se guarde la información en el caso de la generación dinámica de documentos. Para ello, hay que utilizar lo que has aprendido en los módulos anteriores para enviar y procesar información desde el cliente al servidor.

Los ingredientes fundamentales que te van a permitir realizar modificaciones de DHTML en tus documentos son los mismos que permiten realizar la validación de formularios en el lado del cliente.

Por un lado, la capacidad del navegador de ejecutar código JavaScript. Por otro, la programación basada en eventos HTML, que dispara la ejecución de dicha funcionalidad. Y finalmente, los objetos DOM y sus atributos, que representan el estado de los elementos del documento en un momento dado.

Estos tres ingredientes nos van a permitir modificar dinámicamente el documento HTML que se muestra en el navegador.

Esta modificación dinámica hace referencia tanto al contenido, es decir, a la información que se muestra al usuario en pantalla, pero también hace referencia a la forma en que se presenta la información al usuario en pantalla.

Lo que vamos a implementar es una lista de elementos seleccionables partiendo del documento que utilizábamos para trabajar con las funciones que nos permitían navegar por el árbol DOM.

Y lo primero que hemos hecho en este documento es un cambio meramente estético para que los distintos elementos de la lista se iluminen según voy posicionándome sobre ellos con el ratón. Pero los cambios importantes son los que hacen referencia a la funcionalidad de selección. Y el primero de esos cambios son estos checkboxes que hemos puesto a la izquierda de cada uno de los elementos para que el usuario seleccione cuáles de esos elementos son los que quiere seleccionar para realizar una determinada acción sobre ellos.

Luego, además, hemos añadido dos componentes en la parte inferior. El primero, para indicar el tipo de selección, que va a poder ser única o múltiple.

En caso de selección única, solo voy a poder tener un elemento seleccionado cada vez, van alternando, cambia la selección, y en la selección múltiple podré tener múltiples elementos seleccionados.

Luego, además, tengo un componente para marcar todos los elementos de una vez y desmarcarlos todos. Obviamente, la opción marcar todos solo tiene sentido con el tipo de selección múltiple. Y de la misma forma, tenemos un control de errores para cuando tengo más de un elemento seleccionado.

En el caso de la selección múltiple, no poder transicionar al tipo de selección única, a no ser que solo tenga un elemento seleccionado en la lista.

Esta es una funcionalidad típica, que aparece en multitud de aplicaciones, tanto web como no web, para que el usuario seleccione un conjunto de elementos sobre los que realizar una acción en bloque, ya sea en el cliente o en el servidor.

(ver documento dhtml.html en la carpeta U5)