

# PRACTICAS JAVASCRIPT

- 1) **prc\_drag\_drop**: Arrastrar y soltar una imagen de una capa a otra.



- 2) **js\_location**: Obtener la ubicación y mostrar un mapa. Para obtener el mapa. Insertar un enlace: `<a id="map-link" target="_blank"></a>` y mediante js se le modifica la propiedad href y textContent. Donde lat / lon serán variables donde se almacena la latitud, longitud.

- ✓ `enlace.href = `https://openstreetmap.org/#map=10/${lat}/${lon}`;`
- ✓ `enlace.textContent = `Latitud: ${lat}, Longitud: ${lon}`;`
- ✓ Para añadir la ubicación con un marcador:

`enlace.href = `https://openstreetmap.org/?mlat=${lat}&mlon=${lon}#map=20/${lat}/${lon}`;`

- 3) **reloj**: mostrar un reloj con las imágenes proporcionadas. Configurar dos botones para parar y arrancar el reloj.



Utilizar el objeto **Image** para cargar las imágenes. Ejecutar una función con **setInterval** cada 1000 ms. Esta función se puede detener con **clearInterval**. Con el objeto Date podemos obtener la fecha y la hora (cada campo por separado).

- 4) **Test**: desarrollar un test con limitación de tiempo en javascript. Material en practicas/ La práctica consiste en hacer un test con limitación de tiempo. Inicialmente se mostrará al usuario una página con esta información:

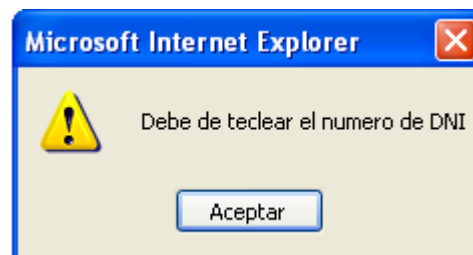
DNI:  
8787

NOMBRE COMPLETO:  
8787

Comenzar Test

Su tiempo restante es: 00:00:30

Al pulsar el botón de Comenzar Test se chequearán si están rellenos los datos. Es decir, si el usuario ha rellenado los datos (dni y nombre) el test puede comenzar. Si los datos no están rellenos se mostrará mensajes como este para indicar al usuario que los datos son obligatorios:



Cuando el usuario haya rellenado los dos campos, se le mostrarán las preguntas del test (se puede hacer visible la capa que contiene las preguntas) y el cronómetro empezará a descontar. El usuario verá algo así:

DNI:  
51.000.00F

NOMBRE COMPLETO:  
Antonio

Comenzar Test

Su tiempo restante es: 00:00:26

1. Indica cual de esta palabras no es de java

- ☐ A. break
- ☐ B. continue
- ☒ C. dim
- ☐ D. for
- ☐ E. while

2. Cual es el orden correcto de menor a mayor restricción en los modificadores

- ☐ A. private, default, protected, public.

Una vez que el test ha comenzado se pueden dar dos situaciones, el tiempo termina, en este caso se avisa y se muestra el número de aciertos. O el usuario rellena todas las preguntas, en este caso se corta el crono se y presenta el número de aciertos.



## 5) objetos:

Definir una clase Empleado con las propiedades: numero, nombre y departamento y un método mostrar que junta todas las propiedades y las muestra con un alert / console.log. Crear un par de objetos con datos fijos. Y mostrar sus datos usando el método mostrar().

Idem del anterior, pero añadimos una propiedad que a su vez es un objeto. Es la clase nómina que se compone de sueldo, paga\_extra, dieta, km. Además, tendrá un método que nos devuelva el sueldo final que será la suma de todas las propiedades.

Nos apoyamos en el ejercicio anterior, pero los empleados los metemos en un array y tenemos que decir el número de empleados que tenemos y calcular el sueldo medio.

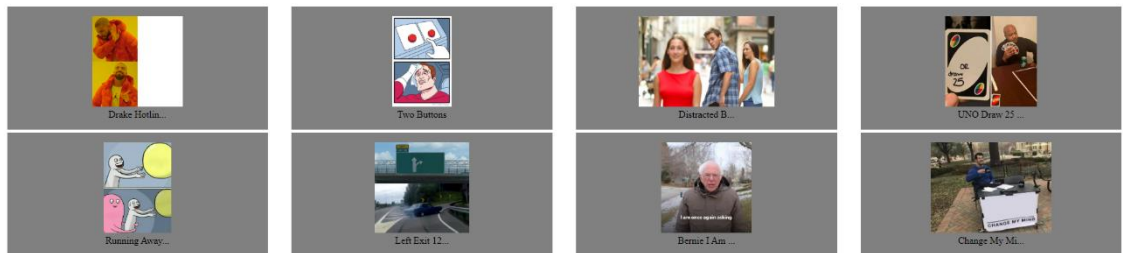
## 6) En la url: [https://api.imgflip.com/get\\_memes](https://api.imgflip.com/get_memes)

Podemos obtener un json con los datos sobre memes:: titulo, foto ..

Construir una galería que se pueda cambiar de forma dinámica el número de memes a mostrar, ya sean:1, 2, 3, o 4 por fila. En total se descargan 100 memes.

Lista de Memes (100)

Memes / lista




- 7) **Módulos\_poo\_herencia:** Definir la clase Persona(nombre, edad, altura), probar los atributos privados. Después implementar la clase Guía (ámbito, idiomas). Importar y exportar. Crear los ficheros: package.json con {"type":"module"} El código se prueba desde un fichero: main.js. Ambas clases tienen el método hablar (en el caso de los guías) solo pueden hablar en caso de que coincidan en un idioma.
- 8) **Carrito:** implementar un carrito almacenando los datos en localStorage en formato json. Los productos están cargados en un array


```
const productos = [  
  { nombre: "Portatil 1", img: "./img/foto1.jpg", precio: 550 },  
  { nombre: "Portatil 2", img: "./img/foto2.jpg", precio: 650 },  
  { nombre: "Portatil 3", img: "./img/foto3.jpg", precio: 725 },  
];
```

Se compra arrastrando el producto.


### Productos disponibles (3)



Portatil 1 550€



Portatil 2 650€



Portatil 3 725€

### La compra

comprar

Cantidad

### Carrito

No hay productos en el carrito

El carrito se va actualizando en el momento que se compra un producto. Al entrar en la web se recupera del **LocalStorage** la información del carrito.

- 9) **Worker:** Crear dos worker que se encargan de obtener el contenido del fichero seleccionado. Además, el hilo principal. En la práctica el hilo principal recuperar el texto de un fichero y lo incrusta en una capa. Los otros dos hilos hacen lo mismo (insertan en la capa el fichero seleccionado, pero se les añade un retardo de 15 y 20 segundos respectivamente).
  - a. El hilo1: va ligado al fichero 1
  - b. El hilo 2: va ligado al fichero 2
  - c. Se trata de comprobar como funcionan por detrás varios hilos mientras realizamos otras tareas en el hilo principal. Seleccionar los ficheros 1 y 2 se pondrán en marcha los dos hilos, pero con un retardo de 15 y 20 segundos. Mientras en el hilo principal seleccionar el fichero 3.
- 10) **dom:**
  - a. **práctica1:** Obtener mediante el dom
    - i. Número de enlaces de la pagina
    - ii. Dirección del penúltimo enlace
    - iii. Número de enlaces que apuntan a http://prueba
    - iv. Número de enlaces del tercer párrafo
  - b. **práctica2:** Crear una lista de forma dinámica. Con el botón de insertar, añade un elemento a la lista con el texto tecleado. De la misma forma pulsando el botón de borrar se puede eliminar un texto.
- 11) **File:** Utilizando el API File probar a cargar dentro de la capa un fichero de texto, una imagen o un fichero binario (por ejemplo: un fichero de javascript o de CSS).

12) **Sockets:** Por un lado, necesitamos el servidor con Node y por otro utilizamos el api basado en sockets de javascript. Implementar un sistema de mensajería donde se pueden enviar distintos tipos de mensajes (abrir varios navegadores para que se puedan conectar):  
0 -> un mensaje de bienvenida al usuario que se conecta: type:0, contenido:

1 -> se ha conectado un nuevo cliente: type:1, nick:xxx

Se añade al mapa

Se comunica a todos

2 -> se ha desconectado un nuevo cliente: type:2, nick:xxx

Se borra del mapa

Se comunica a todos

3 -> mensaje de origen a destino: type:3, nick:origen, nick2:destino, contenido: mensaje

Se envía el mensaje solo al destinatario

4 -> mensaje de difusión a todos: type:4, nick:origen, contenido: mensaje

Se envía el mensaje a todos.

La url de conexión puede ser: ***const url = "ws://localhost:8081";***

***8081 sería el puerto que se pondría en el método listen del servidor.***

13) **Canvas:** Chequear el api de Canvas de javascript. Recuperar el contexto en 2d a partir del canvas recuperado del html. Iniciar con beginPath, establecer color de relleno y color de trazado. Probar a generar círculos concéntricos, insertar texto, etc.

#### 14) Fetch / JSON / DOM / usuarios

A partir de la URL: <https://randomuser.me/api/?results=10>

Podemos indicar el número de resultados aleatorios sobre usuarios que vamos a obtener en formato JSON. Se trata de implementar una petición Ajax a la URL para generar una lista de usuarios en una lista desordenada (ul) con el siguiente formato:

- a. Encadenar dos promesas then:
- b. Una para obtener los datos del servidor, utilizar función json()
- c. Y otra para procesar el json y generar la lista de usuarios.
- d. Datos: name.first / name.last / picture.medium
- e. Crear la lista ul de forma dinámica rellenando los datos de los usuarios



- Ashton Robinson



- Gordislav Burachinskiy



- Svetlana Muller



- Leona Harper



- Emil Christiansen



- Elizabeth Lévesque



- Elif Ilcali