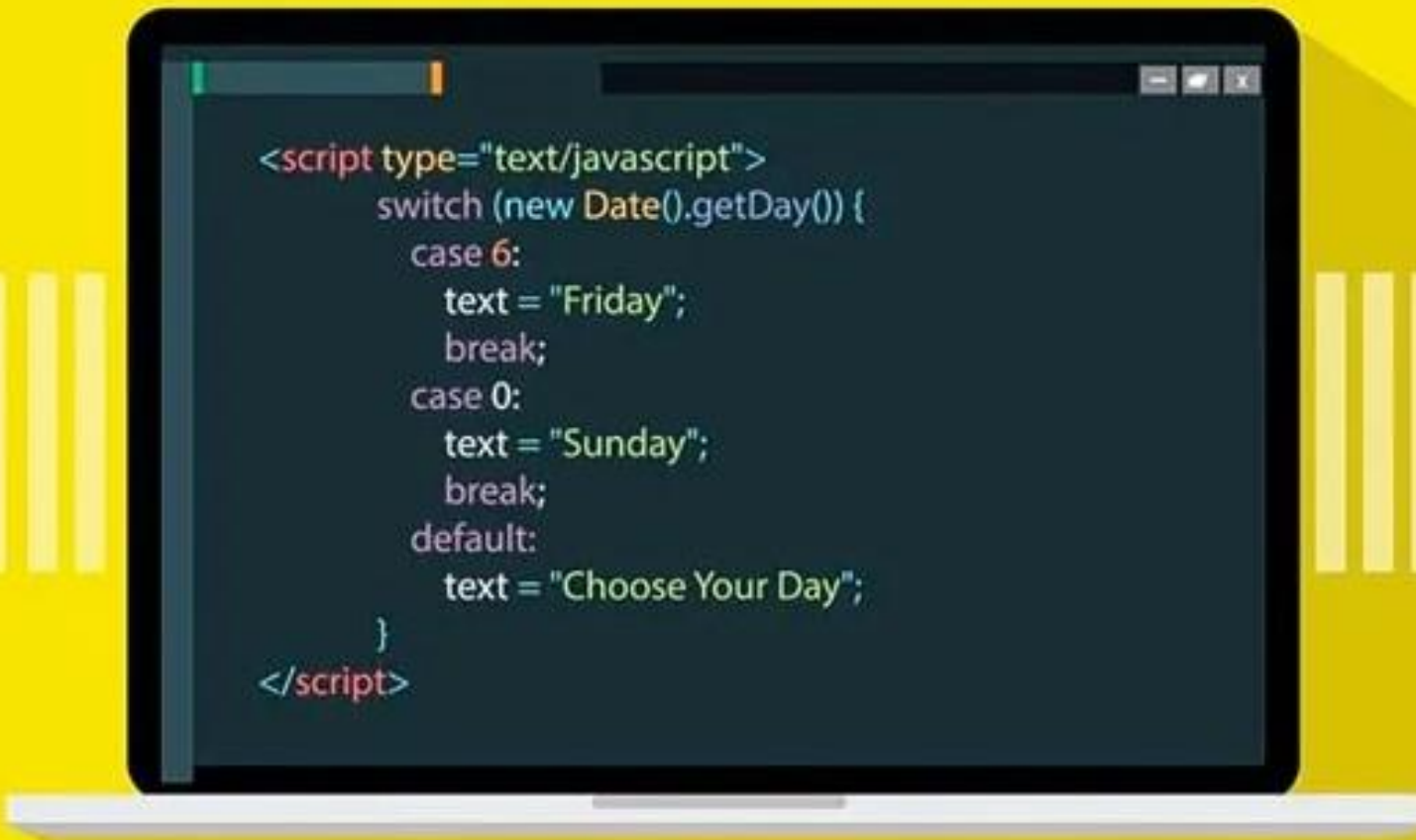
# UT 2_1 - Sintaxis

# Javascript

Marina Navarro Pina





# *Introducción*

Las aplicaciones tendrán estos 3 elementos separados en distintos ficheros:

- El HTML - fichero *index.html*
- El CSS - uno o más ficheros con extensión *.css* dentro de una carpeta llamada *styles*
- EL JS - uno o más ficheros con extensión *.js* en un directorio llamado *scripts*

# Características JS

1. lenguaje interpretado, no compilado
2. se ejecuta en el lado cliente (en un navegador web), aunque hay implementaciones como NodeJS para el lado servidor
3. lenguaje orientado a objetos
4. lenguaje débilmente tipado, con tipificación dinámica

# Incluir en una página web

entre etiquetas *<script></script>*

en el *<head>* o en el *<body>*

en cuanto a rendimiento, lo mejor es ponerla al final del *<body>* para que no se detenga el renderizado de la página mientras se descarga y se ejecuta el código

*Ejemplo1*

también podemos ponerlo en el *<head>* pero usando los atributos *async* y/o *defer*

# Mostrar información

JS permite mostrar ventanas modales para pedirle o mostrarle información al usuario, utilizando el objeto *window:*

- **window.alert(mensaje):** Muestra en una ventana modal *mensaje* con un botón de *Aceptar* para cerrar la ventana.
- **window.confirm(mensaje):** Muestra en una ventana modal *mensaje* con botones de *Aceptar* y *Cancelar*. La función devuelve **true** o **false** en función del botón pulsado por el usuario.
- **window.prompt(mensaje [, valor predeterminado]):** Muestra en una ventana modal *mensaje* y debajo tiene un campo donde el usuario puede escribir, junto con botones de *Aceptar* y *Cancelar*. La función devuelve el valor introducido por el usuario como texto (es decir que si introduce 54 lo que se obtiene es "54") o **false** si el usuario pulsa *Cancelar*.

También funciona sin especificar el objeto *window,* ya que es el objeto por defecto de métodos y propiedades.

# *Variables*

JS es un *lenguaje dinámicamente tipado*: no se indica de qué tipo es una variable al declararla e incluso puede cambiar su tipo a lo largo de la ejecución del programa.

```
let miVariable;         // declaro miVariable y como no se asigno un valor valdrá undefined
miVariable='Hola';      // ahora su valor es 'Hola', por tanto contiene una cadena de texto
miVariable=34;          // pero ahora contiene un número
miVariable=[3, 45, 2];  // y ahora un array
miVariable=undefined;   // para volver a valer el valor especial undefined
```

JS es *débilmente tipado:* permite operaciones entre tipos de datos diferentes:

```
miVariable='23';
console.log(miVariable * 2);  // mostrará 46 ya que convierte la cadena '23' a número
```

# Declaración variables

Las variables de declaran con ***let*** o con ***var***. Con *let* el ámbito de la variable es sólo el bloque en que se declara mientras que con *var* su ámbito es global (o global a la función en que se declara).

RECOMENDACIÓN: utilizar let

Cualquier variable que no se declara *dentro de una función* (o si se usa sin declarar) es global. Debemos siempre intentar NO usar variables globales.

No estamos obligados a declarar una variable antes de usarla, aunque es recomendable para evitar errores que nos costará depurar:

use strict → error si no se declaran

# Declaración variables

Nombres variables - sintaxis *camelCase*: **miPrimeraVariable**

También podemos declarar constantes con ***const***. Se les debe dar un valor al declararlas y si intentamos modificarlo posteriormente se produce un error:

```
const PI=3.1416;
PI=3.14;    // dará un error

const miArray=[3, 4, 5];
miArray[0]=6;   // esto sí se puede hacer
miArray=[6, 4, 5];  // esto dará un error
```

# *Estructuras y bucles*

## Estructura condicional: if

Puede tener asociado un *else* y pueden anidarse varios con *else if:*

```
if (condicion) {
    ...
} else if (condicion2) {
    ...
} else if (condicion3) {
    ...
} else {
    ...
}
```

# *Estructuras y bucles*

## Estructura condicional: switch

Hay que poner *break* al final de cada bloque para que no continúe evaluando:

```
switch(color) {
    case 'blanco':
    case 'amarillo':    // Ambos colores entran aquí
        colorFondo='azul';
        break;
    case 'azul':
        color_lambda_Fondo='amarillo';
        break;
    default:            // Para cualquier otro valor
        colorFondo='negro';
}
```

# *Estructuras y bucles*

## Bucle *while* o *do … while*

Se ejecuta **0 o más veces** en función de la evaluación de la *condición*:

```
while (condicion) {
    // sentencias
}
```

Se ejecuta **1 o más veces** en función de la evaluación de la *condición*:

```
do {
    // sentencias
} while (condicion)
```

# *Estructuras y bucles*

## Bucle *for* con contador

Creamos una **variable contador** que controla las veces que se ejecuta el *for*:

```
let datos=[5, 23, 12, 85]
let sumaDatos=0;

for (let i=0; i<datos.length; i++) {
    sumaDatos += datos[i];
}
// El valor de sumaDatos será 125
```

Array...

# *Estructuras y bucles*

## Bucle *for … in*

El bucle se ejecuta una vez para **cada elemento del array** (o **propiedad del objeto**) y se crea una variable *contador* que toma como valores la *posición* del elemento en el array:

```
let datos=[5, 23, 12, 85]
let sumaDatos=0;

for (let indice in datos) {
    sumaDatos += datos[indice];     // los valores que toma indice son 0, 1, 2, 3
}
// El valor de sumaDatos será 125
```

Array…

# *Estructuras y bucles*

Bucle *for … in*

O recorrer propiedades de un objeto:

```
let profe={
    nom:'Juan',
    ape1='Pla',
    ape2='Pla'
}
let nombre='';

for (var campo in profe) {
   nombre += profe.campo + ' '; // o profe[campo];
}
// El valor de nombre será 'Juan Pla Pla '
```

Objeto…

# *Estructuras y bucles*

## Bucle *for … of*

Similar a *for…in* pero la variable contador en vez de tomar como valor cada índice **toma cada elemento**. Nuevo en ES2015:

```
let datos = [5, 23, 12, 85]
let sumaDatos = 0;

for (let valor of datos) {
    sumaDatos += valor;       // los valores que toma valor son 5, 23, 12, 85
}
// El valor de sumaDatos será 125
```

# *Estructuras y bucles*

## Bucle *for … of*

También sirve para recorrer los caracteres de una cadena de texto:

```
let cadena = 'Hola';

for (let letra of cadena) {
    console.log(letra);     // los valores de letra son 'H', 'o', 'l', 'a'
}
```

# *Funciones*

Se declaran con **function** y se les pasan los *parámetros* entre paréntesis.

La función puede devolver un valor usando **return** (si no tiene return es como si devolviera *undefined*):

```
function functionName(Parameter1, Parameter2, ..)
{
   // Function body
}
```

¿Hoisting?

# *Parámetros*

Si se llama una función con **menos parámetros** de los declarados el valor de los parámetros no pasados será *undefined*:

```
function potencia(base, exponente) {
    console.log(base);              // muestra 4
    console.log(exponente);         // muestra undefined
    let valor=1;
    for (let i=1; i<=exponente; i++) {
      valor=valor*base;
    }
    return valor;
}

potencia(4);    // devolverá 1 ya que no se ejecuta el for
```

Podemos dar un **valor por defecto** a los parámetros.

# *Parámetros - arguments[]*

Es posible acceder a los parámetros desde el array **arguments[]**:

```
function suma () {
    var result = 0;
    for (var i=0; i<arguments.length; i++)
        result += arguments[i];
    return result;
}

console.log(suma(4, 2));                    // mostrará 6
console.log(suma(4, 2, 5, 3, 2, 1, 3));     // mostrará 20
```

# *Un tipo de datos más…*

Podemos *pasarlas por argumento* o *asignarlas a una variable*:

```
const cuadrado = function(value) {   // funciones como objetos
  return value * value
}
function aplica_fn(dato, funcion_a_aplicar) {
    return funcion_a_aplicar(dato);
}

aplica_fn(3, cuadrado);     // devolverá 9 (3^2)
```

# *Funciones anónimas*

Definir una función **sin darle un nombre**. Dicha función puede asignarse a una variable, autoejecutarse o asignarse a un manejador de eventos.

```
let holaMundo = function() {
    alert('Hola mundo!');
}

holaMundo();        // se ejecuta la función
```

Podemos “ejecutar” la variable

# *Arrow functions (funciones flecha)*

- Eliminamos la palabra *function*
- Si sólo tiene 1 parámetro podemos eliminar los paréntesis de los parámetros
- Ponemos el símbolo =>
- Si la función sólo tiene 1 línea podemos eliminar las { } y la palabra *return*

```
let potencia = function(base, exponente) {
    let valor=1;
    for (let i=1; i<=exponente; i++) {
      valor=valor*base;
    }
    return valor;
}
```

```
let cuadrado = function(base) {
    return base * base;
}
```

```
let potencia = (base, exponente) => {
    let valor=1;
    for (let i=1; i<=exponente; i++) {
      valor=valor*base;
    }
    return valor;
}
```

```
let cuadrado = base => base * base;
```