JavaScript 01 B – Sintaxis (II)

JavaScript 01 B – Sintaxis (II) 1
Funciones 1
Parámetros 1
Funciones anónimas 3
Arrow functions (funciones flecha) 3
Estructuras y bucles 4
Estructura condicional: if 4
Operador condicional (ternario) 4
Estructura condicional: switch 4
Bucle while 5
Podemos usar el bucle while…do 5
O el bucle do…while: 5
Bucle: for 6
Bucle: for con contador 6
Bucle: for…in 6
Bucle: for…of 7

Funciones

Se declaran con la palabra reservada **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*).

Puede usarse una función antes de haberla declarado por el comportamiento de Javascript llamado *hoisting*: el navegador primero carga todas las funciones y mueve las declaraciones de las variables al principio y luego ejecuta el código.

> EJERCICIO **mensaje**: Realiza una función que te pida que escribas algo y muestre un alert diciendo 'Has escrito…' y el valor introducido. Pruébala en la consola (pegas allí la función y luego la llamas desde la consola)

Parámetros

Si se llama a 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;
}

let resultado = potencia(4); // devolverá 1 ya que no se ejecuta el for
console.log(resultado);
//console.log(potencia(4));    // devolverá 1 ya que no se ejecuta el for
```

Podemos dar un **valor por defecto** a los parámetros por si no los pasan asignándoles el valor al definirlos:

```
function potencia(base, exponente=2) {
    console.log(base);            // muestra 4
    console.log(exponente);       // muestra 2 la primera vez y 5 la segunda
    let valor=1;
    for (let i=1; i<=exponente; i++) {
      valor=valor*base;
    }
    return valor;
}

console.log(potencia(4));         // mostrará 16 (4^2)
console.log(potencia(4,5));       // mostrará 1024 (4^5)
```

NOTA: En ES5 para dar un valor por defecto a una variable se hacía:

```
function potencia(base, exponente) {
    exponente = exponente || 2;   // si exponente vale undefined se la asigna el valor 2
...
```

También es posible acceder a los parámetros desde el array **arguments[]** si no sabemos cuántos parámetros recibiremos:

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

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

En Javascript las funciones son un tipo de datos más, por lo que podemos hacer cosas como pasarlas como argumento o asignarlas a una variable:

```
const cuadrado=function(value){
```

```
    return value * value;
}
function aplicarFuncion(dato, funcion_a_aplicar){
    return funcion_a_aplicar(dato);
}
aplicarFuncion(3,cuadrado); // devolverá 9 (3^2)
```

A este tipo de funciones, que son tratadas como cualquier otra variable, se llaman *funciones de primera clase* y son típicas de lenguajes funcionales.

Funciones anónimas

Podemos definir una función sin darle un nombre. Dicha función puede asignarse a una variable, autoejecutarse o asignarse a un manejador de eventos. Ejemplo:

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

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

Como vemos, asignamos una función a una variable de forma que podamos "ejecutar" dicha variable.

```
let nuevaVariable = holaMundo; // asigno el valor de holaMundo a otra variable
nuevaVariable();  // Ejecutamos la nueva variable => la función holaMundo.
                  //Para la ejecución se añaden los paréntesis
```

Arrow functions (funciones flecha)

ES2015 permite declarar una función anónima de forma más corta. Ejemplo sin *arrow function*:

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

Al escribirla con la sintaxis de una *arrow function* lo que hacemos es:

- 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 eliminamr las { } y la palabra *return*

El ejemplo con *arrow function*:

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

```
}
```

Otro ejemplo, sin *arrow function*:

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

con *arrow function*:

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

> EJERCICIO: Haz una *arrow function* que devuelva el cubo del número pasado como parámetro y pruébala desde la consola. Escríbela primero en la forma habitual y luego la "traduces" a *arrow function*.

Estructuras y bucles

Estructura condicional: if

El **if** es como en la mayoría de los lenguajes. Puede tener asociado un **else** y pueden anidarse varios con **else if**.

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

Ejemplo:

```
if (edad < 18) {
    console.log('Es menor de edad');
} else if (edad > 65) {
    console.log('Está jubilado');
} else {
    console.log('Edad correcta');
}
```

Operador condicional (ternario)

Se puede usar el operador **? :** que es como un *if* que devuelve un valor:

```
let esMayorDeEdad = edad > 18 ? true : false;
```

Estructura condicional: switch

El **switch** también es como en la mayoría de lenguajes. 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':
        colorFondo='amarillo';
        break;
    default:            // Para cualquier otro valor
        colorFondo='negro';
}
```

Javascript permite que el *switch* en vez de evaluar valores pueda evaluar expresiones. En este caso se pone como condición *true*:

```
switch(true) {
    case age < 18:
        console.log('Eres muy joven para entrar');
        break;
    case age < 65:
        console.log('Puedes entrar');
        break;
    default:
        console.log('Eres muy mayor para entrar');
}
```

Bucle while

Podemos usar el bucle while...do

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

que se ejecutará 0 o más veces. Ejemplo:

```
let nota = prompt("Introduce una nota (o cancela para finalizar)");
while (nota) {
    console.log("La nota introducida es: " + nota);
    nota = prompt("Introduce una nota (o cancela para finalizar)");
}
```

O el bucle do...while:

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

que al menos se ejecutará 1 vez. Ejemplo:

```
let nota;
do {
    nota=prompt('Introduce una nota (o cancela para finalizar)');
    console.log('La nota introducida es: '+nota);
```

```
} while (nota)
```

EJERCICIO: Haz un programa para que el usuario juegue a adivinar un número. Obtén un número al azar (busca por internet cómo se hace o simplemente guarda el número que quieras en una variable) y ve pidiendo al usuario que introduzca un número. Si es el que busca le dices que lo ha encontrado y si no le mostrarás si el número que busca el mayor o menor que el introducido. El juego acaba cuando el usuario encuentra el número o cuando pulsa en 'Cancelar' (en ese caso le mostraremos un mensaje de que ha cancelado el juego).

Bucle: for

Tenemos muchos *for* que podemos usar.

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
```

EJERCICIO: El factorial de un número entero n es una operación matemática que consiste en multiplicar ese número por todos los enteros menores que él: **n x (n-1) x (n-2) x ... x 1**. Así, el factorial de 5 (se escribe 5!) vale **5! = 5 x 4 x 3 x 2 x 1 = 120**. Haz un script que calcule el factorial de un número entero positivo.

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
```

También sirve para recorrer las propiedades de un objeto:

```
let profe={
    nom:'Santi',
    ape1:'Blanco',
    ape2:'Arenal'
}

let nombre='';

for (var campo in profe) {          // no declarar campo con let
   nombre += profe.campo + ' ';     // o profe[campo];
```

```
}
// El valor de nombre será 'Santiago Blanco Arenal'
```

Bucle: for…of

Es similar al *for…in* pero la variable contador en vez de tomar como valor cada índice toma cada elemento. Es nuevo en ES2015:

```
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
```

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'
}
```

EJERCICIO: Haz 3 funciones a las que se le pasa como parámetro un array de notas y devuelve la nota media. Cada una usará un for de una de las 3 formas vistas. Pruébalas en la consola.