

JAVASCRIPT

**JS**

# Contenido

- ★ Iteradores
- ★ Funciones
- ★ Métodos de Array
- ★ Parseo de Datos

# Iteradores



¡For, Switch y mucho más!

# FOR

```
for (inicio; condición; incremento) {  
    // hacer esto mientras la condición sea verdadera  
}  
// =====  
for (var i = 0; i < 4; i++) {  
    console.log("Hola " + i);  
}
```

Esto imprimirá en la consola:

"Hola 0"

"Hola 1"

"Hola 2"

"Hola 3"

# SWITCH

```
var fruta = "pera";
```

```
switch (fruta) {  
  case "frutilla": console.log("La frutilla del postre");  
  break;  
  
  case "manzana": console.log("Me prestas tu reloj? Manzana");  
  break;  
  
  case "pera": console.log("2 pesitos la pera!"); 😊  
  break;  
  
  default: console.log("Es otra fruta");  
}
```

# WHILE & DO WHILE

```
while (condicion) {  
    // ejecutar mientras la  
    condición sea verdadera  
}
```

```
var a = 0;  
while (a < 3) {  
    console.log("Hola");  
    a++; // Siempre llegar a la condición  
    de corte  
}
```

Esto imprimirá en la consola:

"Hola"

"Hola"

"Hola"

```
do {  
    // lo que quiero hacer  
} while (condición)
```

```
var a = 0;  
do {  
    console.log("Hola"); // Siempre se  
    ejecuta al menos 1 vez  
} while(a > 100);
```

Esto imprimirá en la consola:

"Hola"

# BREAK & CONTINUE

```
for (var i = 0; i < 5; i++) {  
  console.log("Hola " + i);  
  if(i === 1){  
    break; // corta el b ucle FOR  
  }  
  console.log('Chau');  
}
```

El Break corta el bucle y sale de   l.

# BREAK & CONTINUE

```
for (var i = 0; i < 5; i++) {  
  console.log("Hola " + i);  
  if(i === 3){  
    continue; // Salta la iteración cuando i es 3 y continúa sin cortar el  
    bucle  
  }  
  console.log("Hola " + i);  
  console.log('Chau');  
}
```



**¡A practicar!**

Ejercicios Iteradores



```
// To Do  
console.log("Practice Time");
```

# Funciones



# ¿Qué es una Función?

Una función es un bloque de código asignado para hacer una tarea.

```
function elProducto(n1, n2) {  
    return n1 * n2; // Devuelve la multiplicación de n1 por n2  
}
```

La función es ejecutada cuando se la invoca:

```
elProducto(2, 5); // Esto me devolverá 10
```

# Declaración vs. Expresión

## Declaración

```
function hola () {  
    return console.log("hola");  
}
```

// invocación  
hola();

## Expresión

```
var hola = function () {  
    return console.log("hola");  
}
```

// invocación  
hola();

# PARÁMETROS

A las funciones les podemos pasar parámetros:

```
var hola = function(nombre, apellido){  
    return "Hola " + nombre + " " + apellido;  
}
```

¿Cómo ejecutamos la función hola?

# Scope

## Scope => Alcance

El Scope es el alcance de una variable y determina su accesibilidad. Javascript tiene 2 tipos de Scope: GLOBAL y LOCAL.

```
var saludo = "hola"; // variable GLOBAL
```

```
function saludar(nombre){  
    return saludo + " " + nombre;  
}
```

```
saludar("Ale"); // "hola Ale"
```

```
console.log(saludo); // "hola"
```

# Scope

## Scope => Alcance

El Scope es el alcance de una variable y determina su accesibilidad. Javascript tiene 2 tipos de Scope: GLOBAL y LOCAL.

```
function saludar(nombre){  
    var saludo = "hola"; // variable LOCAL  
    return saludo + " " + nombre;  
}
```

```
saludar("Ale"); // "hola Ale"
```

```
console.log(saludo); // Undefined variable
```

# CLOSURES

## ¿Qué son?

Son funciones que viven dentro de otras funciones, las cuales tienen acceso a todos los parámetros y variables de la función que las contiene.

```
function test(string){  
    var variableLocal = "Digital";  
    function funcionInterna(){  
        return variableLocal + " " + string;  
    }  
    return funcionInterna();  
}  
test("House"); // "Digital House"
```



# CALLBACKS

## ¿WTF? - Los callbacks

Son funciones que se pasan como parámetro de otra función para ser ejecutadas dentro de la función que las recibe.

```
function sumar(n1, n2){ return n1 + n2; }  
function restar(n1, n2){ return n1 - n2; }  
function operacionMatematica(n1, n2, operacion){  
    return operacion(n1, n2);  
}  
operacionMatematica(4, 17, sumar); // 21  
operacionMatematica(26, 12, restar); // 14
```

**¡A practicar!**

Ejercicios Funciones



```
// To Do  
console.log("Practice Time");
```

# Array y sus Métodos



¡forEach, for of, map, filter y  
mucho más!

# for of (ES6)

Este **for of** puede operar sobre Array o Strings.

```
var miArray = [1, 2, 3];
```

```
for(var numero of miArray){  
    console.log(numero);  
}
```

```
// 1  
// 2  
// 3
```

```
var miString = "hola";
```

```
for(var letra of miString){  
    console.log(letra);  
}
```

```
// h  
// o  
// l  
// a
```

# .forEach()

El iterador **forEach()** sólo puede iterar sobre arrays y recibe como parámetro una función.

```
var miArray = [1,2,3];  
  
miArray.forEach(function(numero){  
    console.log(numero);  
});
```

# map()

El `map()` recibe *un callback* como parámetro y devuelve un **array nuevo**.

```
var miArray = [1, 2, 3];
```

```
var arrayAlDoble = miArray.map(function(elemento){  
    return elemento * 2;  
});
```

```
// [2, 4, 6]
```

# filter()

El método **filter()** crea un **nuevo array** con todos los elementos que cumplan la condición implementada por el *callback*.

```
var miArray = [1, 2, 3];
```

```
var impares = miArray.filter(function(numero){  
    return numero % 2 !== 0;  
});
```

```
// [1, 3]
```

# reduce()

El método **reduce()** recibe un **callback** cuyo objetivo será retornar un único valor. Reduce el array dado a un solo valor.

El callback recibe 2 parámetros: **acumulador**, **elementoActual**.

**reduce()** puede recibir (optativamente) un valor inicial para el acumulador, va después del callback.

```
var miArray = [1, 2, 3];  
var sumado = miArray.reduce(function(acumulador, elementoActual){  
    return acumulador + elementoActual;  
}, 0);
```



# join()

El método **join()** une todos los elementos de un array en un string y devuelve ese string.

Le podemos indicar cómo deseamos que una los elementos. Si no le indicamos un separador los une sin espacios.

```
var frutas = ["Manzana", "Naranja", "Ananá"];
```

```
var texto = frutas.join(", ");
```

# find()

El método **find()** devuelve el **valor** del **primer elemento** del array que cumple la función de prueba proporcionada.

```
var frutas = ["Manzana", "Naranja", "Ananá"];
```

```
var match = frutas.find(function(fruta){  
    return fruta === "Manzana";  
});
```

```
console.log(match); // "Manzana"
```

**¡A practicar!**

Ejercicios Arrays



```
// To Do  
console.log("Practice Time");
```

# Parseo de Datos



# parseInt() vs. Number()

Convierte (parsea) un argumento de tipo string y devuelve un entero con su valor absoluto. Si no lo puede parsear a número devuelve un NaN.

```
var num = parseInt("01020");  
console.log(num); // 1020
```

Convierte (parsea) un argumento de tipo string y devuelve ese valor en número. Si no lo puede parsear a número devuelve un NaN.

```
var num = Number("22")  
console.log(num); // 22
```

# Array.from()

El método **Array.from()** crea un nuevo **Array** a partir de un dato.

```
console.log(Array.from('foo')); // ["f", "o", "o"]
```

También le podemos pasar un callback como segundo parámetro:

```
var multiplo = Array.from([1, 2, 3], function(elem){  
    return elem * 2;  
});
```

```
// [2, 4, 6]
```

**¡Hasta la  
próxima!**



```
// End The :P  
console.log("See you soon");
```