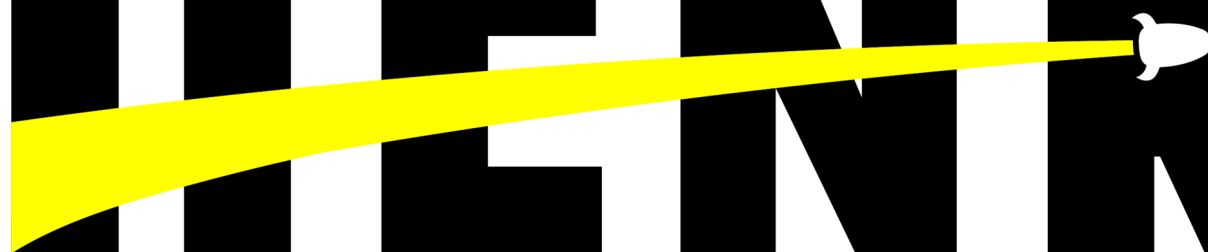


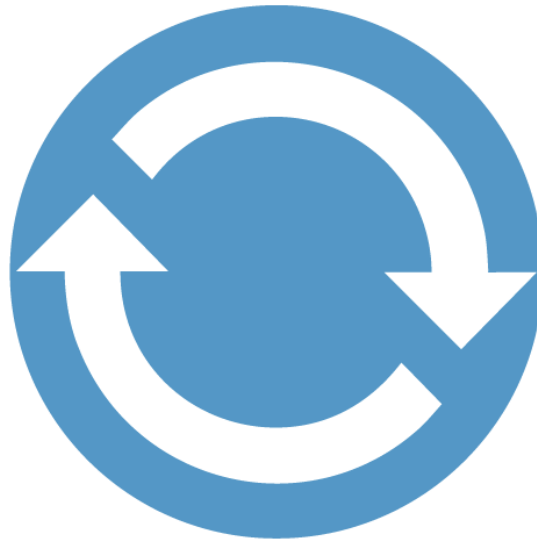
# HENRY



JS-III

# Bucles (Loops)

Un bucle es una secuencia de instrucciones de código que se ejecuta repetidas veces, hasta que la condición asignada a dicho bucle deja de cumplirse.



## Bucle *for*

```
1 for (var i = 0           ; i < 10           ; i++           ) {  
2 // | Declaramos una variable | Expresión condicional | Incrementamos la variable |  
3   console.log(i);  
4 }
```

```
1 for (var i = 0; i < 10 ; i++) {  
2   console.log(i);  
3 }
```

```
1 function encontrarVocalA(string){  
2   for(var i = 0; i < string.length; i++){  
3     if(string[i] === 'a'){  
4       return "Encontramos la vocal"  
5     }  
6   }  
7   return "El string no tiene ninguna letra a"  
8 }
```

```
1 // Bucle infinito  
2  
3 for (let i = 0; i >= 0; i++) {  
4   console.log(i);  
5 }
```

## Bucle *while*

```
1 while(unCondicion){  
2     //Código a ejecutarse mientras se cumpla la condición  
3 }
```

```
1 var count = 1;  
2 while(count < 6) {  
3     console.log("count es: "+count);  
4     count++;  
5 }
```

# Arreglos (Arrays)

Son usados para guardar información correspondiente a distintos tipos de datos como strings, números, booleanos, etc. Hacen parte del tipo de dato Object, el cual no es un primitivo.

```
1 var comidas = ['Pizza', 'Hamburguesa', 'Hot Dog', 'Lasagna'];
```

Así como con los strings, podemos acceder a los elementos que componen un arreglo a partir de sus índices e iniciando también desde el índice 0:

```
1 var comidas = ['Pizza', 'Hamburguesa', 'Hot Dog', 'Lasagna'];  
2               0           1           2           3
```

Así, si queremos acceder a alguno de ellos, basta usar el nombre del arreglo seguido por corchetes y dentro de estos la posición donde se encuentra el elemento:

```
1 console.log(comidas[1]) // Hamburguesa
```



También podemos guardar el elemento en una nueva variable:

```
1 var comidaFavorita = comidas[1];  
2 console.log(comidaFavorita) // Hamburguesa
```

Podemos asignar y reasignar cualquier índice en el arreglo usando corchetes, adentro de estos el índice y luego el signo '=', seguido del valor a asignar:

```
1 var comidas = ['Pizza', 'Hamburguesa', 'Hot Dog', 'Lasagna'];  
2  
3 comidas[2] = 'Pastas';  
4  
5 console.log(comidas) // ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna']
```

# Métodos de arreglos

## *.length*

Devuelve el número de elementos que hay en un arreglo:

```
1 var comidas = ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna'];  
2  
3 console.log(comidas.length) // 4
```

## *.push*

Agrega un elemento al final del arreglo, incrementando así la longitud de este, y devuelve la nueva longitud:

```
1 var comidas = ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna'];  
2  
3 comida.push('Sandwich');  
4  
5 console.log(comidas) // ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna', 'Sandwich']
```

## *.pop*

Elimina el último elemento del arreglo disminuyendo en 1 su longitud.  
Devuelve el elemento eliminado:

```
1 var comidas = ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna', 'Sandwich'];  
2  
3 comidas.pop();  
4  
5 console.log(comidas) // ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna']
```

## *.unshift*

Agregará un elemento al inicio del arreglo:

```
1 var comidas = ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna'];  
2  
3 comidas.unshift('Hot Dog');  
4  
5 console.log(comidas) // ['Hot Dog', 'Pizza', 'Hamburguesa', 'Pastas', 'Lasagna']
```

## *.shift*

Eliminará el primer elemento del arreglo:

```
1 var comidas = ['Hot Dog', 'Pizza', 'Hamburguesa', 'Pastas', 'Lasagna'];  
2  
3 comidas.shift();  
4  
5 console.log(comidas) // ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna']
```

# Utilizando bucles en arreglos

## *for*

Usando la técnica de acceso al índice ("index access technique") podemos acceder a cada elemento del arreglo. Para hacer esto, usamos el método `.length` como punto de parada para el ciclo.

```
1 var comidas = ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna'];
2
3 for (let i = 0; i < comidas.length; i++) {
4     console.log(comidas[i]);
5 }
6
7 // 'Pizza'
8 // 'Hamburguesa'
9 // 'Pastas'
10 // 'Lasagna'
```

## *for ... of*

Itera sobre los valores del arreglo. Su estructura es un poco diferente, dado que no tendrá expresión condicional, sino que, para cada iteración, se asigna el valor de cada elemento a una variable y finaliza cuando ya no tenga más de estas asignaciones por hacer:

```
1 var comidas = ['Pizza', 'Hamburguesa', 'Pastas', 'Lasagna'];
2
3 for(comida of comidas){
4   console.log(comida);
5 }
6 // 'Pizza'
7 // 'Hamburguesa'
8 // 'Pastas'
9 // 'Lasagna'
```

## *.forEach*

Toma un callback como su único argumento, e itera sobre cada elemento del arreglo y llama a la función en él. El callback puede tomar dos argumentos, el primero es el elemento en sí, el segundo es el índice del elemento (este argumento es opcional).

```
1 var autos = ["Ford", "Chevrolet", "Toyota", "Tesla"];
2
3 // Podemos escribir el callback en los paréntesis como una función anónima
4 autos.forEach(function (elemento, indice) {
5     console.log(elemento);
6 });
7
8 // O podemos crear una instancia de una función para usarla como callback.
9 // Además, no necesitamos usar el argumento de índice,
10 // si no lo necesitas, no dudes en omitirlo.
11 function mostrarNombres(elemento) {
12     console.log(elemento);
13 }
14
15 // Y llamar a esta función dentro de los paréntesis del método .forEach
16 autos.forEach(mostrarNombres);
```

## *.map*

Se usa cuando queremos cambiar cada elemento de un arreglo bajo las mismas condiciones, de la misma manera.

```
1 var numeros = [2, 3, 4, 5];
2
3 //Definimos la función que será pasada por callback en el método .map
4 function multiplicarPorTres(elemento) {
5     return elemento * 3;
6 }
7
8 var triple = numeros.map(multiplicarPorTres);
9
10 console.log(triple); // [ 6, 9, 12, 15 ]
11
12 // O simplemente pasamos como callback una función anónima
13 var doble = numeros.map(function (elemento) {
14     return elemento * 2;
15 });
16
17 console.log(doble); // [ 4, 6, 8, 10 ]
```



# *.reduce*

Ejecutará un bucle en nuestro arreglo con la intención de reducir cada elemento en un elemento que se devuelve.

```
1 var numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9];
2 var palabras = ["Hola,", "mi", "nombre", "es", "Martin"];
3
4 // Podemos escribir la función anónima directamente en los paréntesis de .reduce
5 // Si omitimos el elemento inicial, siempre comenzará en el primer elemento.
6 var suma = numeros.reduce(function (acc, elemento) {
7     return acc + elemento;
8 });
9
10 // Podemos escribir una función fuera de los parents de .reduce
11 // (para usar varias veces después)
12 function multiplicarDosNumeros(a, b) {
13     return a * b;
14 }
15
16 var productos = numeros.reduce(multiplicarDosNumeros);
17
18 // .reduce funciona en cualquier tipo de datos.
19 // En este ejemplo configuramos un acumulador de arranque
20 var frases = palabras.reduce(function (acc, elemento) {
21     return acc + " " + elemento;
22 }, "Frase completa:");
23
24 console.log(suma); // 45
25 console.log(productos); // 362880
26 console.log(frases); // "Frase completa: Hola, mi nombre es Martin"
```