

2.4. Estructuras de control


Estructuras selectivas o condicionales

```
if (condición) {  
    instrucciones_si_true;  
}  
else {  
    instrucciones_si_false;  
}
```

```
console.log("Inicio");  
let local = 2;  
let visitante = 1;  
if (local == visitante) {  
    console.log("¡Hay empate!");  
}  
else {  
    console.log("¡NO hay empate!");  
}  
console.log("Fin");
```

```
if (condición1) {  
    instrucciones_si_condición1_true;  
}  
else if (condición2) {  
    instrucciones_si_condición2_true;  
}  
...  
else if (condiciónN) {  
    instrucciones_si_condiciónN_true;  
}  
...  
else {  
    instrucciones_si_todas_condiciones_false;  
}
```

```
console.log("Inicio");  
let local = 2;  
let visitante = 1;  
if (local > visitante) {  
    console.log("Local gana.");  
}  
else if (local < visitante) {  
    console.log("Visitante gana");  
}  
else {  
    console.log("| Hay empate!");  
}  
console.log("Fin");
```



```
console.log("Inicio");
let local = 2;
let visitante = 1;
if (local > visitante) {
    console.log("Local gana.");
    if ((local-visitante) > 1) {
        console.log("Y además por goleada.");
    }
    else {
        console.log("Pero por la mínima.");
    }
}
else if (local < visitante) {
    console.log("Visitante gana");
    if ((visitante-local) > 1) {
        console.log("Y además por goleada.");
    }
    else {
        console.log("Pero por la mínima.");
    }
}
else {
    console.log("¡Hay empate!");
}
console.log("Fin");
```


```
switch (expresión) {  
  case valor_1:  
    instrucciones_1  
    [break;]  
  case valor_2:  
    instrucciones_2  
    [break;]  
  ...  
  default:  
    instrucciones_predeterminadas  
    [break;]  
}
```

```
console.log("Menú abierto");  
let letra_pulsada = 'c';  
switch (letra_pulsada) {  
  case 'a':  
    console.log("Abrir archivo");  
    break;  
  case 'c':  
    console.log("Copiar");  
    break;  
  case 'p':  
    console.log("Pegar");  
    break;  
  default:  
    console.log("Opción incorrecta");  
}  
console.log("Menú cerrado");
```


Estructuras repetitivas

```
while (condición) {  
    instrucciones;  
}
```

```
let pases = 0;  
while (pases < 10) {  
    console.log(`Pase número ${pases+1}`);  
    pases++;  
}
```



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



```
do {  
    instrucciones;  
} while (condición);
```

```
console.log("--- Primeros 10 números pares ---");  
let contador = 0;  
let numero = 1;  
do {  
    if (numero%2 == 0) {  
        console.log(`PAR: ${numero}`);  
        contador++;  
    }  
    numero++;  
} while (contador<10);
```

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

Bucle **for** (con contador)

```
for ([expresiónInicial]; [expresiónCondicional]; [expresiónDeActualización]) {  
    instrucciones;  
}
```

```
const TABLA = 9;  
for (let contador=1; contador<=10; contador++) {  
    console.log(`${TABLA} x ${contador} = ${TABLA*contador}`);  
}
```

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]  Array...  
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
```


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 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 '
```



Bucle for...of

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

```
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
```

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'  
}
```

Estructuras de salto

La instrucción **break** se utiliza para terminar un bucle while, do while o for o una sentencia switch y transferir el control a la siguiente instrucción.

```
const TABLA = 9;
for (let contador=1; contador<=10; contador++) {
    console.log(`${TABLA} x ${contador} = ${TABLA*contador}`);
    if (contador == 5)
        break;
}
```


Al utilizar **continue** se finaliza la iteración actual de un bucle while, do while o for y continúa la ejecución del bucle con la siguiente iteración.

```
console.log("--- Primeros 10 impares NO x 3 ---");
let contador = 0;
let numero = 1;
while (contador < 10){
    if (numero%3 == 0) {
        numero++;
        continue;
    }
    if (numero%2 != 0) {
        console.log(`IMPAR: ${numero}`);
        contador++;
    }
    numero++;
}
```

Calcula los 10 primeros números impares que no son múltiplo de 3.

La declaración **labeled** viene a complementar la funcionalidad de las dos instrucciones anteriores.


Su utilidad reside en establecer puntos en el programa, a los que se asigna un nombre (una **etiqueta**) al que hacer referencia cuando se desea efectuar un salto.



```
let primero = segundo = 1;
buclePrincipal: while (true) {
  console.log(`(Bucle principal) Iteración ${primero}`);
  primero++;
  while (true) {
    console.log(`(Bucle secundario) Iteración ${segundo}`);
    segundo++;
    if (segundo == 5)
      break;
    else if (primero == 3)
      break buclePrincipal;
  }
}
```


Las instrucciones de salto break , continue y labeled son recursos que deben utilizarse en muy contadas ocasiones.

El motivo es que abusar de ellas produce algoritmos denominados “**spaghetti code**” debido a la cantidad de saltos adelante y atrás que es necesario realizar para seguir el flujo de ejecución del programa.



Esto es completamente **contrario a las normas de estilo más elementales de la programación, donde se busca simplicidad y legibilidad** en pro de un software de mantenimiento más sencillo. En muchas empresas está completamente prohibido su uso, salvo casos muy excepcionales.