



Estructura de Repetición: Do - While

While

- `while (condición) { /* Bloque del bucle while */ }`
- La estructura while, primero evalúa la condición de su cabecal, y luego entra al bloque de código.

Do While

- `do {`
 `// Bloque de código`
} `while(condición);`
- El do-while primero ejecuta al menos una vez el bloque, y luego evalúa la condición.

Ejemplo

```
let valores = "";  
let i = 0;  
  
do {  
    i += 1;  
    valores += i.toString();  
} while (i < 5);
```

Cuáles son los valores finales de “i” y “valores” ?

Funciones

- En JavaScript las funciones son bloques compuestos por un conjunto de instrucciones encargadas de realizar tareas específicas.
- Sirven para tener un código mas organizado
- Sirven para ahorrar código. En caso que tengamos una o varias líneas que usemos varias veces, las podemos encapsular en una función, y llamar dicha función las veces que queramos
-

Funciones

- Sintaxis en JavaScript:

```
function nombre(parámetros) {  
    // Bloque de la función
```

```
}
```

- Los parámetros son opcionales, y pueden haber tantos parámetros como el programador quiera.

Ejemplo

```
// función declarada por el programador:
function controlarFaltas(faltas) {
    if (faltas > 24) { alert("Se excede de faltas"); }
}

while (true) { // while true: bucle infinito
    let opcion = prompt(
        "(1)- Consultar faltas, (2)- Salir"
    );

    if (opcion === "1") {
        let faltas = prompt("Ingresar faltas");
        controlarFaltas(faltas);
    } else if (opcion === "2") {
        break; // paramos el bucle
    }
}
```



Retorno de valores en funciones

- En JavaScript, el valor por defecto de todo es “Undefined”, incluso en las funciones.
- Si queremos hacer una funcion que retorne algo, usamos la sentencia **return**

Ejemplo de return

```
function suma(a, b) {  
    return a + b;  
}  
  
console.log(suma(1, -4));
```

Analizando la función anterior.

```
function suma(a, b) {  
  return a + b;  
}  
  
const resultado1 = suma(10, 10);  
const resultado2 = suma("Fuck", " you");  
  
console.log(resultado1);    // 20  
console.log(resultado2);    // Fuck you
```

El operador + se comporta distinto

Antes de mejorar la función anterior, veamos algunos operadores lógicos:

Negación de una expresión (Operador NOT):

- Para negar una expresión del tipo booleana, puedo usar el operador !

```
> !true
< false

> !false
< true

> !(1 == 1)
< false

> !(1 == 7)
< true

>
```

Operador lógico AND:

- En este curso de programación no pondremos énfasis en las tablas de verdad, o proposiciones lógicas (De donde salen los operadores and, or, xor, not, etc)
- El operador and (&) retorna true si **todas** las condiciones de una expresión son ciertas.
- (condicion1) && (condicion2) && (condicionN)

Operador lógico AND

```
> (1 === 1) && (4 === 4)
```

```
< true
```

```
> (1 === 2) && (4 === 4)
```

```
< false
```

```
> (4 === 4) && (2 === 4)
```

```
< false
```

```
> ("H" === "H") && (true === true)
```

```
< true
```

Operador lógico OR:

- Similar al *and*, pero retorna **true** con que una de las condiciones sea verdadera.
- Es decir, una expresión será **false** si todas sus condiciones son false.
- En javascript este operador es: `||`

```
> ("a" === "a") || (true === true)
```

```
< true
```

```
> (1 === "x") || (false === false)
```

```
< true
```

```
> (!(1 === "x")) || (false === false)
```

```
< true
```

```
>
```

Mejorando la función:

```
function sumaNumeros(a, b) {  
  let resultado;  
  
  if (typeof(a) == "number" && typeof(b) == "number") {  
    resultado = a + b;  
  }  
  return resultado;  
}
```



Arrow functions

- En JavaScript podemos definir una función sin usar la sentencia `function`.
- También podemos guardar una función en una variable.
- Este tipo de definiciones los hacemos con las funciones flecha, o “arrow functions”.

Arrow functions

```
const suma = (a, b) => {  
  if (typeof(a) == "number" && typeof(b) == "number") {  
    return a + b;  
  } else {  
    return "Deben ser ambos números."  
  }  
}  
  
let miSuma = suma(1, 2);
```

Se le llama función flecha, porque la definición se hace con los símbolos de igual y de mayor

Ejemplos de arrow functions:

```
const saludar = (mensaje, nombre) => alert(mensaje, nombre);  
const numeroPar = numero => console.log(numero % 2 === 0);
```



Práctica

- Hagamos algunos ejercicios del práctico...