DESARROLLO WEB FULL STACK

Control de flujo

¡Nuestro programa puede decidir!

Comisión/Clase/Versión/Autor

Control de flujo

Hasta ahora trabajamos en programas que ejecutan pasos lineales. La solución a los problemas de la vida real muchas veces requieren tomar decisiones en base a condiciones que surgen en el momento.

En lenguajes de programación, las estructuras de control permiten modificar el flujo de ejecución de las instrucciones de un programa.

Ejemplo clásico

Hagamos la lógica sobre prepararme algo rápido para comer

Ej: Prepararme una cena rápida

¿Hay atún?
Si:
- ¿Hay un abrelatas?
  Si:
  1. Abrir la lata
  2. Como arroz con atún

  No:
  1. Como arroz solo

No:
1. Como arroz solo

Bloque de control “if”

IF ELSE

Nuestra primera estructura de control, nos permite redirigir el flujo del programa en base a una condición simple.

```
if (condicion) {
    // Bloque de código
} else {
    // Bloque de código
}
```

IF ELSE IF

else if nos permite preguntar nuevamente inmediatamente después de que la condición principal fué falsa.

```
if (condicion) {
    // Bloque de código
} else if (condicion2) {
    // Bloque de código
} else {
    // Bloque de código
}
```

IF ELSE anidados

Se pueden anidar tantas veces como queramos, pero no hay que abusar, muchos **if else** anidados empobrecen la legibilidad y aumentan la **complejidad ciclomática** haciendo nuestro código es más propenso a errores.

```
if (condicion) {
    // Bloque de código
} else {
    if (condicion2) {
        // Bloque de código
    } else {
        // Bloque de código
    }
}
```

IF ELSE anidados

Como quedaría nuestro problema de la vida resuelto en un código con una estructura de control.

```
if (hayAtun) {
    if (hayAbrelatas) {
        abrirLaLata();
        comerArrozConAtun();
    } else {
        comerArrozSolo();
    }
} else {
    comerArrozSolo();
}
```

Verdaderos y Falsos

Las condiciones en base a las que tomamos decisiones siempre tienen dos posibles valores “verdadero” y “falso”.

Estos valores en javascript son: `true` `false`

Tengo café? `true` → me hago café

Tengo hambre? `false` → paso de largo

Operadores de comparación

Un operador de la comparación compara sus operando y devuelve un valor lógico basado en si la comparación es verdad o no.

```
operando1 comparador operando2
```

Comparadores

== → Comparación
=== → Comparación estricta (checkea tipo)
!= → Comparación negada
!== → Comparación estricta negada
> → Mayor que
< → Menor que
>= → Mayor o igual que
<= → Menor o igual que

Ejemplos

1 == 1 true

'Pepe' == 'Pepe' true

'Pepe' == 'Juan' false

'Pepe' == 'pepe' false

2 > 1 true

2 >= 2 true

3 < 4 true

'1' == 1 true

Ejemplos

'1' === 1 false

'Pepe' != 'Pepe' false

'Pepe' === 'Juan' false

'Pepe' === 'pepe' false

true true

!true false

'1' !== 1 true

Operadores lógicos

Los operadores lógicos se usan típicamente con valores Booleanos. En tal caso, regresan un valor Booleano. Sin embargo, los operadores && y || regresan en realidad el valor de uno de los operandos especificados, por lo que si estos operadores se usan con valores no Booleanos, posiblemente regresen un valor no Booleano.

```
expresion1 operador expresion2
```

Operadores lógicos

&& → Operador lógico “Y”
|| → Operador lógico “O”
! → Operador lógico “NOT”

Tabla de la verdad https://es.wikipedia.org/wiki/Tabla_de_verdad

p	q	$\neg p$	$\neg q$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
V	V	F	F	V	V	V	V
V	F	F	V	F	V	F	F
F	V	V	F	F	V	V	F
F	F	V	V	F	F	V	V

Ejemplos

true && true `true`

false || true `true`

true || true `true`

false && true `false`

(3 > 2) && (2 < 3) `true`

('pepe' == 'pepe') || (2 < 1) `true`

!(4 < 5) `false`

!(4 < 5) && (2 == 2) `false`

Bloque de control “switch”

SWITCH

La sentencia **switch** evalúa una expresión, comparando el valor de esa expresión con una instancia **case**, y ejecuta sentencias asociadas a ese **case**, así como las sentencias en los **case** que siguen.

```
switch(entrada) {
    case 'valor':
        //Bloque de código
        break;
    case 'otrovalor':
        //Bloque de código
        break;
    default:
        //Bloque de código
}
```

Práctica