Introducción a la Programación con JavaScript

Nacho Viano

Índice

1. Sobre la programación

1.1 ¿Qué es un programa?

Un programa es una lista de instrucciones escritas en un lenguaje de programación que se usa para controlar las tareas de una maquina, normalmente una computadora (en cuyo caso se denomina programa informático).

La ejecución de un programa consiste en una serie de acciones que siguen las instrucciones que este contiene. Cada instrucción produce efectos que alteran el estado de la máquina conforme a su significado predefinido.

1.2 ¿Qué es un lenguaje de programación?

Un lenguaje de programación es un lenguaje formal (o artificial, es decir, un lenguaje con reglas gramaticales bien definidas) que le proporciona a una persona, en este caso el programador, la capacidad de escribir (o programar) una serie de instrucciones o secuencias de órdenes en forma de algoritmos con el fin de controlar el comportamiento físico o lógico de un sistema informático, de manera que se puedan obtener diversas clases de datos o ejecutar determinadas tareas. A todo este conjunto de órdenes escritas mediante un lenguaje de programación se le denomina programa informático

2. Organización de un programa informático

2.1 Estructura básica

Existen dos partes o bloques que componen un programa:

1. Bloque de declaraciones: en este se detallan todos los objetos que utiliza el programa (constantes, variables, archivos, etc).

2. Bloque de instrucciones: conjunto de acciones u operaciones que se han de llevar a cabo para conseguir los resultados esperados.

2. Organización de un programa informático

2.1 Estructura básica

El bloque de instrucciones está compuesto a su vez por tres partes, aunque en ocasiones no están perfectamente delimitadas, y aparecerán entremezcladas en la secuencia del programa, podemos localizarlas según su función. Estas son:

1. Entrada de datos: instrucciones que almacenan en la memoria interna datos procedentes de un dispositivo externo.
2. Proceso o algoritmo: instrucciones que modifican los objetos de entrada y, en ocasiones, creando otros nuevos.
3. Salida de resultados:conjunto de instrucciones que toman los datos finales de la memoria interna y los envían a los dispositivos externos.

Entrada	Algoritmo	Salida
Inicio de programa: datos	Proceso de programa: cálculos	Fin de programa: resultados

2. Organización de un programa informático

2.2 Puntos de entrada

En lenguajes de programación el Punto de entrada (Entry Point en inglés) es el procedimiento de inicio de un programa, en muchos lenguajes de programación, el inicio de un programa se establece por el procedimiento `main`.

2.3 Procedimientos y funciones

Uno de los métodos fundamentales para resolver un problema es dividirlo en problemas más pequeños, llamados subproblemas. Estos problemas pueden a su vez dividirse repetidamente en problemas más pequeños hasta que los problemas sean de fácil solución. Divide y vencerás ... Cada subproblema es deseable que sea independiente de los demás y se denomina módulo. El problema original se resuelve con un programa principal (llamado también driver o main), y los subproblemas (módulos) mediante subprogramas: procedimientos y funciones.

3. JavaScript como primer lenguaje de programación

JavaScript es un robusto lenguaje de programación que se puede aplicar a un documento HTML y usarse para crear interactividad dinámica en los sitios web. Fue inventado por Brendan Eich, cofundador del proyecto Mozilla, Mozilla Foundation y la Corporación Mozilla.

Puedes hacer casi cualquier cosa con JavaScript. Puedes empezar con pequeñas cosas como carruseles, galerías de imágenes, diseños fluctuantes, y respuestas a las pulsaciones de botones. Con más experiencia, serás capaz de crear juegos, animaciones 2D y gráficos 3D, aplicaciones integradas basadas en bases de datos ¡y mucho más!
JavaScript por sí solo es bastante compacto aunque muy flexible, y los desarrolladores han escrito gran cantidad de herramientas encima del núcleo del lenguaje JavaScript, desbloqueando una gran cantidad de funcionalidad adicional con un mínimo esfuerzo. Esto incluye:

3. JavaScript como primer lenguaje de programación

- Interfaces de Programación de Aplicaciones del Navegador (APIs) – APIs construidas dentro de los navegadores que ofrecen funcionalidades como crear dinámicamente contenido HTML y establecer estilos CSS, hasta capturar y manipular un vídeo desde la cámara web del usuario, o generar gráficos 3D y muestras de sonido.

- APIs de terceros, que permiten a los desarrolladores incorporar funcionalidades en sus sitios de otros proveedores de contenidos como Twitter o Facebook.

- Marcos de trabajo y librerías de terceros que puedes aplicar a tu HTML para que puedas construir y publicar rápidamente sitios y aplicaciones

3. JavaScript como primer lenguaje de programación

¿ Qué es JavaScript ?

JavaScript (JS) es un lenguaje ligero e interpretado, orientado a objetos con funciones de primera clase, generalmente usado para el desarrollo web pero también usado en muchos entornos sin navegador, tales como node.js.

- Basado en prototipos
- Dinámico
- Multi-paradigma, soporta estilo de programación funcional y orientado a objetos entre otros
- Su estándar es ECMAScript

En 2015 se publicó la sexta versión de ECMAScript, llamada ECMAScript 2015, ECMAScript 6 o ES6. Actualmente se lanza una versión de forma anual añadiendo nuevas características.

JavaScript no debe ser confundido con el lenguaje de programación Java.

3. JavaScript como primer lenguaje de programación

1. Los punto y coma o semicolon (;) son opcionales sólo en algunas situaciones gracias a ASI
   Si queremos evitar problemas, puede ser una buena práctica escribirlos siempre.

2. Si se quiere introducir código JS directamente en HTML, debemos hacerlo con las etiquetas <script></script>.

3. Si se produce un error en la programación y la excepción no es capturada, la ejecución se interrumpe inmediatamente.

4. Los comentarios de una sola línea con // y de múltiples líneas con /* Comentario */

5. Primera posición de un string o array es 0.

6. Los strings o cadenas de caracteres, se definen entre ‘ o “. También se pueden utilizar las comillas ` si queremos mezclarlas con otros tipos de datos.

7. Podemos hacer una depuración básica a través de console.log();

4. Variables y tipos de datos

Números

Dentro de los datos numéricos encontramos los enteros y los reales. Ambos pueden estar precedidos del signo + o -. Los enteros no tienen parte decimal y los reales sí. En JS, ambos son de tipo “number” sin distinción.

Ejemplos en JavaScript y en C++:

```javascript
// JavaScript
number1 = 5

number2 = 3.14

number3 = -10

number4 = -984.5
```

```cpp
// C++
int age = 1;
float height = 1.75;
```

4. Variables y tipos de datos

Strings (Cadenas de texto)

Las cadenas de texto o strings se utilizan para guardar un conjunto de carácteres alfanuméricos, incluyendo letras del abecedario (a, b, c, ..., z), dígitos (0, 1, 9) o símbolos especiales (#, $, ^, etc)

```
// JavaScript
address = "Calle Larios nº5"

lastname = "Díaz"

postTitle = "Teletrabajo, ¿sí o no?"
```

Aunque en algunos lenguajes existe un tipo de dato “char” que sólo guarda un único carácter, en JavaScript y PHP no es así. Así que lo pasaremos por alto.

4. Variables y tipos de datos

Boolean (Lógico)

Este tipo de datos se conoce como booleano y sólo puede contener 2 valores:

- true (verdadero)
- false (falso)

```
isAlive = true

examPassed = false
```

4. Variables y tipos de datos

Identificadores

Los datos a procesar por un ordenador tienen que almacenarse en celdas de memoria para su posterior utilización.

Estas celdas de memoria tienen un nombre que permite su identificación. Para definir estos nombres se utilizan ciertas reglas (varían en función del lenguaje de programación):

- Pueden ser letras, dígitos o el símbolo especial _

Adicionalmente, se aplican las siguientes convenciones:

- nombres camelCase
- significado semántico de todos los identificadores

```
number1 = 5
PI = 3.14159;
address = "Calle Larios nº5"
lastname = "Díaz“
isAlive = true
examPassed = false
```

4. Variables y tipos de datos

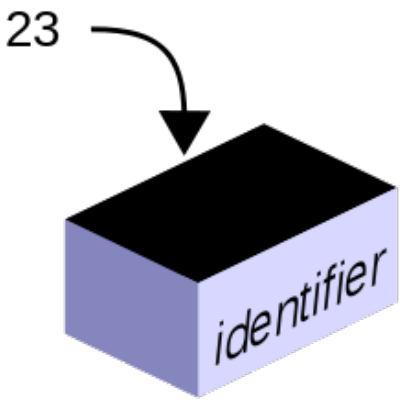
Constantes

Las constantes son datos que no cambian durante la ejecución de un programa. Para nombrar una contante se utiliza un identificador expresado anteriormente. Las constantes pueden ser de cualquier tipo de datos: número, strings, booleanos, etc..

En JS, se usa la palabra reservada const.

Variables

Las variables pueden cambiar su valor durante la ejecución de un programa.
En JavaScript se denota con la palabra reservada let.



NOTA: Tanto en los constantes como en las variables los identificadores deben ser representativos. Ejemplo: dayOfTheMonth en lugar de sólo d

```
const height = 1.75
let age = 23
```

4. Variables y tipos de datos

Reglas para crear variables y constantes:

- No pueden empezar con un número.
- No pueden contener espacio o guión (-).
- Distingue entre mayúsculas y minúsculas.
- No pueden ser una palabra reservada.
- Los nombres deben tener significado.
- Convención de camelCase.

4. Variables y tipos de datos

Bloques de asignación

Un bloque de asignación se utiliza para asignar valores o expresiones a una variable. La asignación es una operación destructiva, lo que significa que si la variable tenía un valor asignado, éste se destruye, conservando ahora el nuevo valor, es decir, se sobrescribe. Para la asignación utilizamos el =

Variable = expresión o valor

Ejercicio. Determinar el tipo de dato y el valor de las siguientes variables:
Tipos: number, string, boolean

a) variable = 838 < 542
b) variable = 17 / 5
c) variable = 12 * 6
d) variable = 35 % 8
e) variable = 1 == 3 || 5 > 7 || 10*5 > 40
f) variable = 13 % 2 == 0 && “Releevant” == “Academia”
g) variable = “Relee” + “vant”
h) variable = (21 / 2) % 3
i) variable = 5 == “5”
j) variable = !(1 == 1) || “PHP” === “PHP”

4. Variables y tipos de datos (avanzados)

Conviene separarlos en primitivos y objetos.

Primitivos: string, number, boolean, undefined y null.

```
let str = String("I'm a String");
let str2 = "I'm another String";
console.log(typeof str, typeof str2, typeof "");
// output: string string

let num = Number(5);
let num2 = 5;
console.log(typeof num, typeof num2, typeof 5);
// output: number number

let bool = Boolean(true);
let bool2 = true;
console.log(typeof bool, typeof bool2, typeof true);
// output: boolean boolean
```

```
let undef;
console.log(typeof undef);
// output: undefined

num = null;
console.log(typeof num, num);
// output: object, null
```

4. Variables y tipos de datos (avanzados)

Objetos
- Array
- Function
- ...

Ya definidos
- Date
- Error
- ...

```
    const array = Array(1, 2, 3);
    const array2 = [1, 2, 3];
    console.log(typeof array, typeof array2)
    // output: object object

console.log(typeof (() => {}), typeof function () {})
    // output: function function

console.log(typeof new Date("2021-02-02"));
    console.log(typeof new Error("Mensaje de error"));
    // output: object
```

4. Variables y tipos de datos (avanzados)

Objects

Son datos estructurados que tienen el mismo significado que podría tener un objeto en la vida real. De esta forma relacionamos dentro de un mismo dato variables y funciones que comparten alguna relación.

Podemos acceder a los campos del objeto de dos formas:

- A través de un punto.
- Con corchetes.

```
const person = {
    name: "John",
    age: 30,
}

person.name = "Jane";

const propertyName = "name";
person[propertyName] = "Peter";
```

4. Variables y tipos de datos (Avanzado)

Arrays

En muchas ocasiones no es posible resolver un problema con una estructura simple de datos y es necesario utilizar tipos de datos estructurados. Como definimos anteriormente, los datos estructurados ocupan más de 1 posición de memoria.

Arrays unidimensionales

Una array es un colección finita, homogénea y ordenada de elementos (no se cumple en todos los lenguajes de programación, ya que en algunos permite que no sean homogéneos).

Un array posee dos partes:

- El índice: Es el identificador de un componente individual del array.
- El elemento: El dato en sí dentro de un índice

4. Variables y tipos de datos (Avanzado)

Arrays unidimensionales

Ejemplos con diferentes tipos de datos:

booleanos

0	1	2	3	4
True	False	False	True	True

Índice

strings

0	1	2	3	4
PHP	Javascript	C	Kotlin	Python

Elemento

numbers

0	1	2	3	4
65	12	5	73619	512

4. Variables y tipos de datos (Avanzado)

Arrays unidimensionales

Un array se define con un identificador, como cualquier otra variable, de la forma:

```
indentificadorArray = [1, 2, 3, 4];
```

- Pregunta: ¿Cuál es el último índice de un array de 10 posiciones?

- Un array puede ser asociativo o no asociativo.
  - Asociativo: El índice es una cadena de caracteres personalizadas:
    animales[“bufalo”] = 14
  - No asociativo o indexado: Los índices son numéricos
    lenguajeDeProgramacion[1] = “Javascript”

Para acceder a cualquier elemento de un array, se usan los corchetes [ ].

4. Variables y tipos de datos (Avanzado)

Operaciones sobre arrays

Acceder a elementos del array
miArray[1], miArray[40]

Modificar elementos del array
miArray[1] = “nuevo valor”
miArray[40] = “otros caracteres”

4. Variables y tipos de datos (Avanzado)

Ordenar Arrays

La Ordenación de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento.

Se trata de ir revisando cada elemento del array con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo que significaría que la lista ya está ordenada.

Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo uno de los más sencillos de implementar.

Más algoritmos de ordenación de forma visual.

4. Variables y tipos de datos (Avanzado)

Ordenar Arrays

Ejemplo: ordenar el array = [3, 9, 8, 1, 4].

0	1	2	3	4
3	**9**	8	1	4
3	**9**	**8**	1	4
3	8	**9**	**1**	4
3	8	1	**9**	**4**
3	8	1	4	**9**

4. Variables y tipos de datos (Avanzado)

Ordenar Arrays

Ejemplo: ordenar el array = [3, 9, 8, 1, 4].

0	1	2	3	4
3	**9**	8	1	4
3	**9**	**8**	1	4
3	8	**9**	**1**	4
3	8	1	**9**	**4**
3	8	1	4	**9**

```
for (let i = 0; i < size - 2; i++) {
    for (let j = 0; j < size - 1; j++) {
        if (array[j] < array[j+1]) {
            aux = array[j];
            array[j] = array[j+1];
            array[j+1] = aux;
        }
    }
}
```

5. Estructuras de control (condicionales)

Las estructuras lógicas selectivas se encuentra en la solución algorítmica de cualquier problema. Se utilizan cuando en un problema de debe tomar una decisión para establecer un proceso o determinar el camino a seguir.

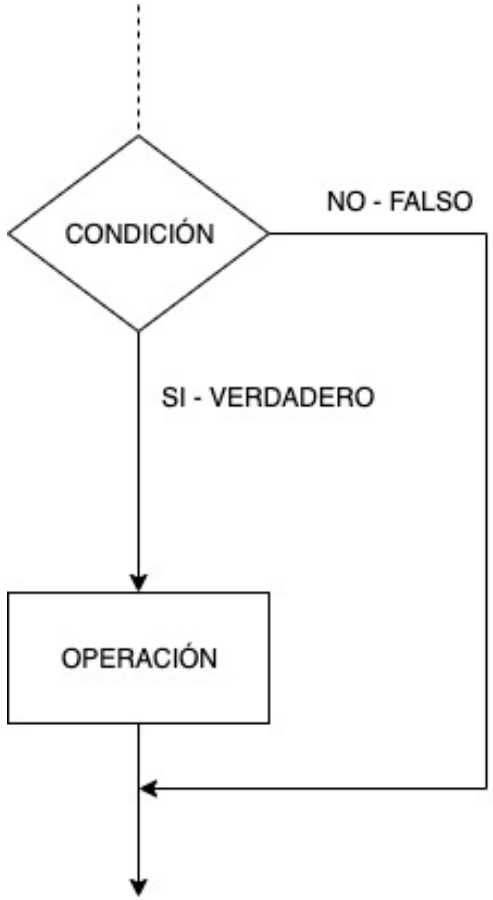
Se pueden clasificar de la siguiente forma:

1. if (Estructura selectiva simple)
2. if-else (Estructura selectiva doble)
3. if-else if / switch (Estructura selectiva múltiple)

5. Estructuras de control (condicionales)

Estructura selectiva simple: if

```
if (expresionLogica || valorLogico) {
    // Hago algunas instrucciones
    // ...
}
```

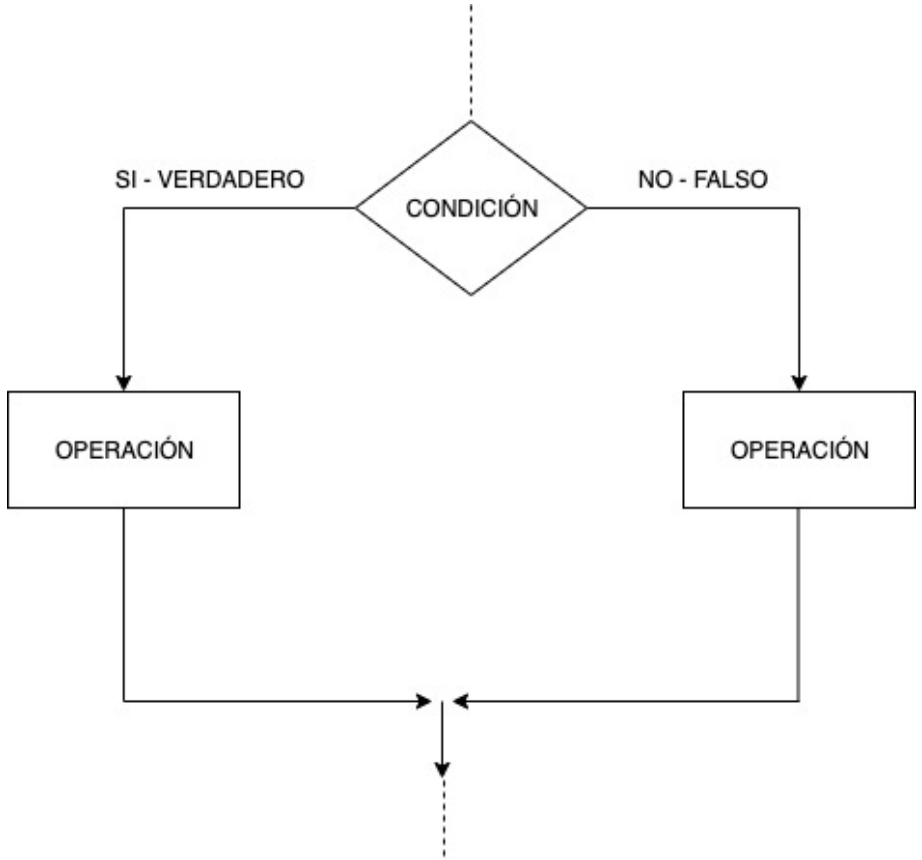


Ejemplo: Condicional que indique “aprobado” si un alumno tiene la nota mayor que 5.

5. Estructuras de control (condicionales)

Estructura selectiva simple: if / else

```
if (userGuess === randomNumber) {
    //operaciones 1
} else {
    //operaciones 2
}
```

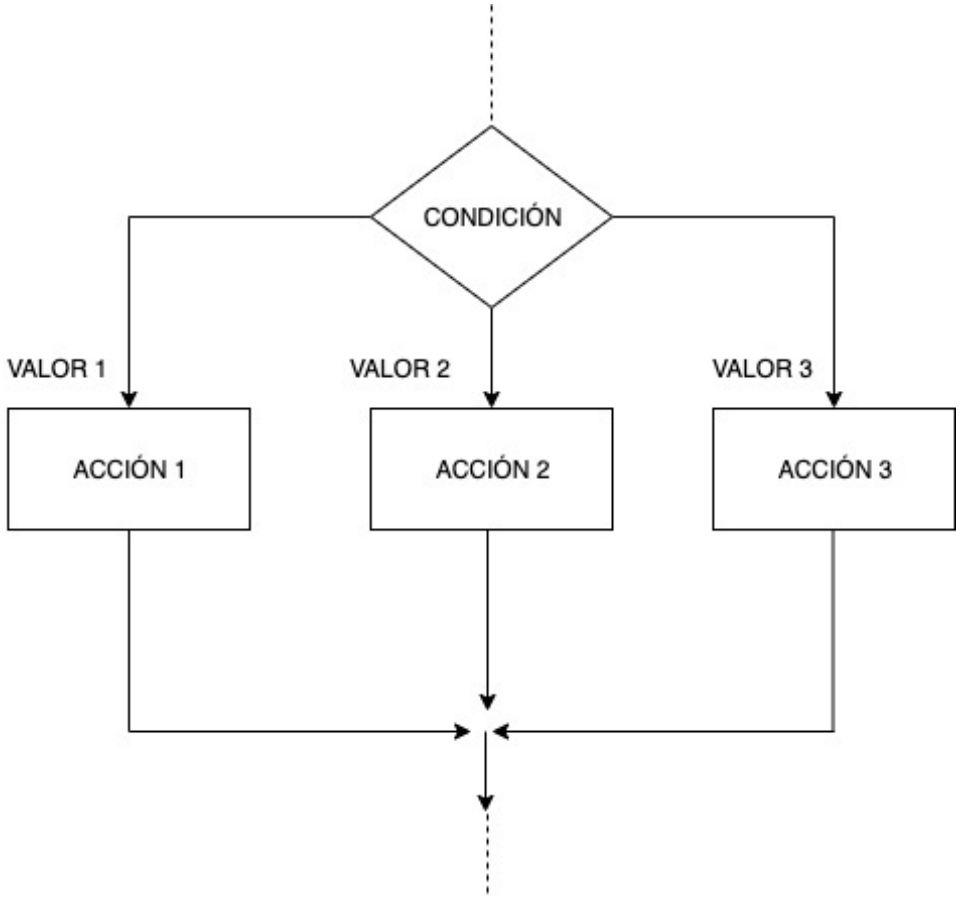


Ejemplo: Construir condicional que escriba “aprobado” si un alumno ha aprobado y ”suspenso” si ha suspendido.

5. Estructuras de control (condicionales)

Estructura selectiva múltiple if-else if / switch

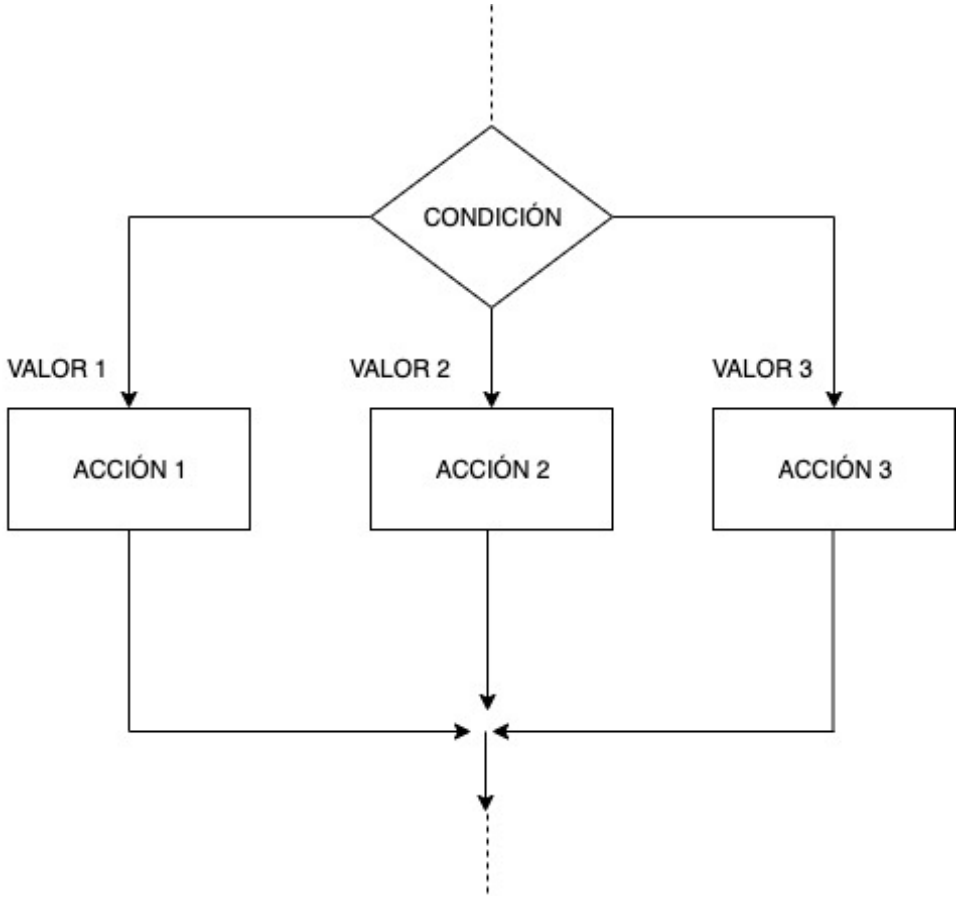
```
if (userGuess === randomNumber) {
    //operaciones 1
} else if (userGuess > randomNumber) {
    //operaciones 2
} else if(userGuess < randomNumber) {
    //operaciones 3
} else {
    //operaciones 4
}
```



5. Estructuras de control (condicionales)

Estructura selectiva múltiple if-else if / switch

```
switch(option) {
    case 1:
        // code block
        break;
    case 2:
        // code block
        break;
    case 3:
        // code block
        break;
    default:
        // code block
}
```

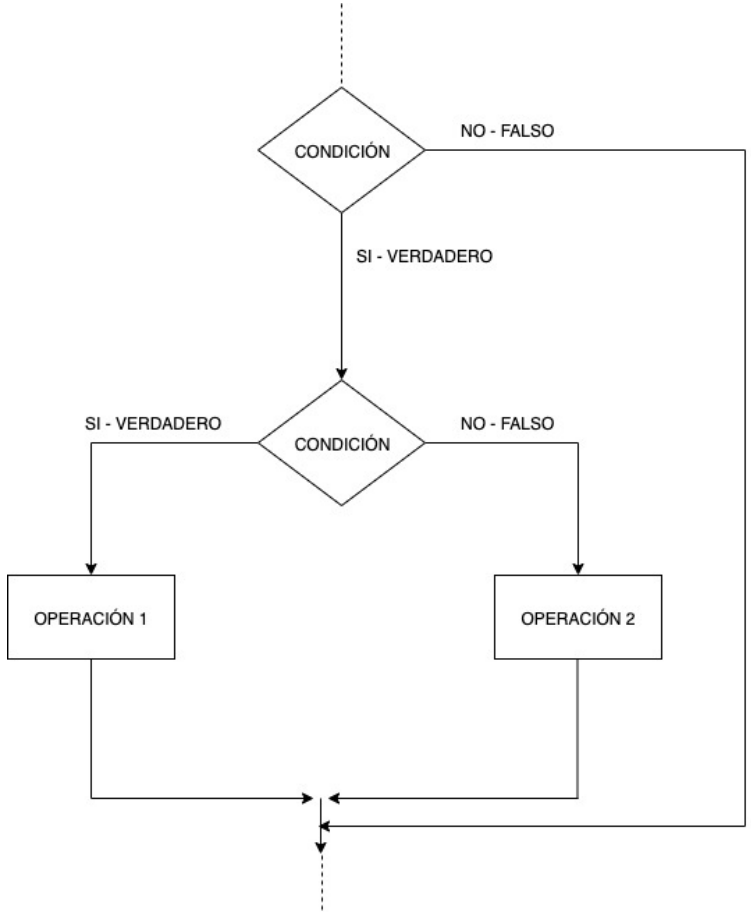


5. Estructuras de control (condicionales)

Estructura selectiva en cascada (anidadas)

Generalmente para la solución de un problema será necesario tomar múltiples decisiones, por lo que necesitaremos anidar las estructuras selectivas anteriores.

```
if (expresionLogica || valorLogico) {
    // Hago algunas instrucciones
    // ...
    if (otraCondicion) {
        // Otra Operación
    } else {
        // Otra Operación
    }
}
```



5. Estructuras de control (bucles)

En la práctica encontramos muchas veces algoritmos cuya estructura se tiene que ejecutar un número repetido de veces. Si bien las instrucciones son las mismas, los datos sobre los que se operan varían. El conjunto de instrucciones que se ejecuta repetidamente se llama ciclo o bucle.

Todo ciclo tiene que terminar tras un número de iteraciones, por lo que es necesario que en cada iteración se evalúen las condiciones para determinar si se tiene que seguir ejecutando o parar. IMPORTANTE: En todo ciclo tiene que existir una condición de parada o fin de ciclo.

En función de si previamente el número de iteraciones de un ciclo es conocido o no, podemos clasificar los algoritmos repetitivos en:

1) while
2) for

5. Estructuras de control (bucles)

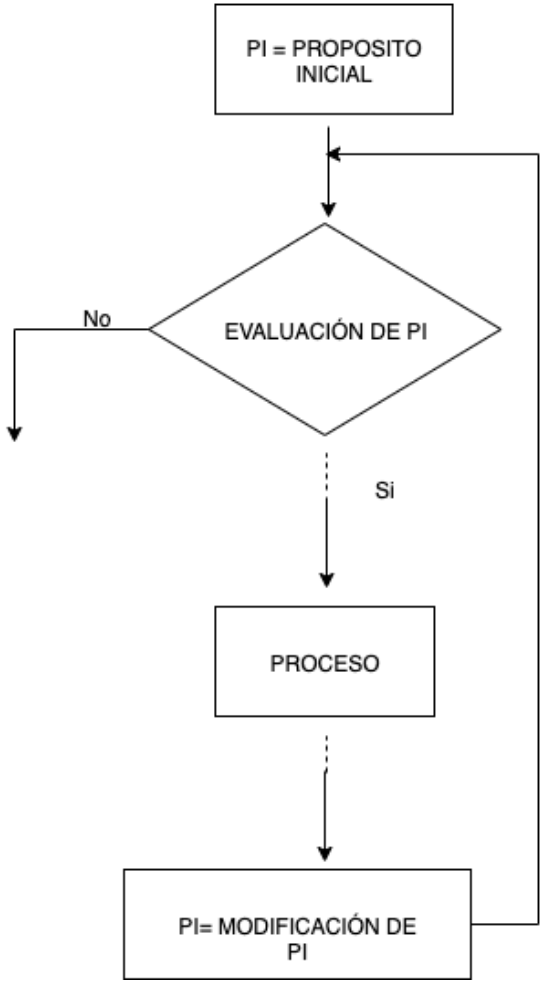
El bucle **WHILE** se utiliza para ejecutar la estructura cuando el número de ciclos no es conocido con anterioridad.

Ejemplos:

- Pedir entrada de datos hasta que ésta sea válida

El diagrama de flujo de la estructura while es:

```
while (condition) {
    // Repito la operación
    // actualizo la condición
}

// Termino cuando la condición
// deja de cumplirse
```



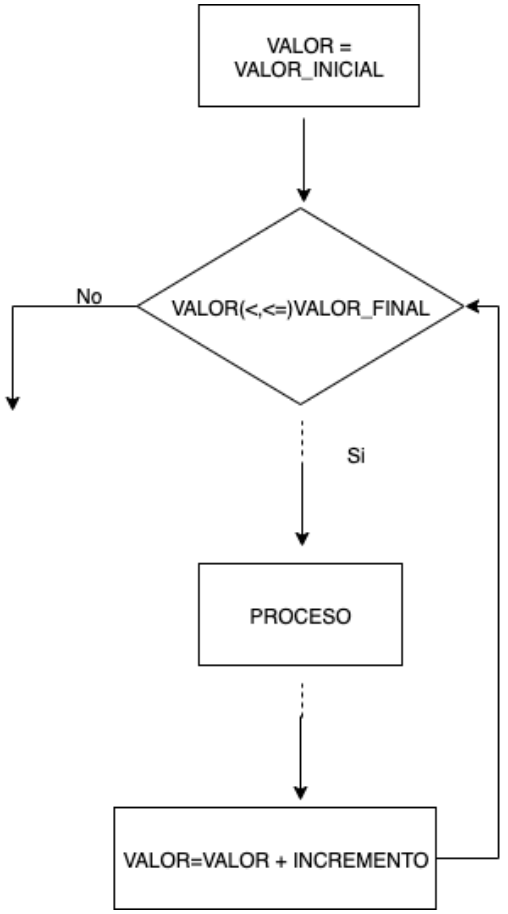
5. Estructuras de control (bucles)

El bucle FOR se utiliza para ejecutar la estructura un número de veces definido.

Ejemplos:
- Sumar las nóminas de N empleados de una empresa.
- Enviar un email a todos los empleados de una empresa.

El diagrama de flujo de la estructura for es:

```
for (alumno = 0; alumno < 15; alumno++) {
    console.log("Alumno nº ", alumno);
}
```



5. Estructuras de control (bucles)

Ejemplo del bucle FOR anidado:

```
for (contador1 = 0; contador1 < 3; contador1++) {

    console.log("Iteración", contador1, "del bucle externo");

    for (contador2 = 0; contador2 < 2; contador2++) {
        console.log("Iteración", contador2, "del bucle interno");
    }

    console.log();
}
```

```
Iteración 0 del bucle externo
Iteración 0 del bucle interno
Iteración 1 del bucle interno
Iteración 2 del bucle interno

Iteración 1 del bucle externo
Iteración 0 del bucle interno
Iteración 1 del bucle interno
Iteración 2 del bucle interno

Iteración 2 del bucle externo
Iteración 0 del bucle interno
Iteración 1 del bucle interno
Iteración 2 del bucle interno
```

5. Estructuras de control (bucles)

For

```
for (let i = 1 ; i < 21 ; i+
+) {
    console.log(i) ;
}
```

For ... of

```
for (const item of array) {
    console.log(item);
}
```

For each

```
array.forEach(myFunction);

function myFunction (item, index) {
    console.log(index, ": ", item);
}
```

While

```
while (condition) {
    //code block to be executed
}
```

Elementos de control de bucles:

```
break
continue
```

6. Operadores

Para poder realizar operaciones aritméticas son necesarios los operadores aritméticos. Las operaciones se pueden realizar con números, variables o constantes y el resultado es un número.

Operador aritmético	Operación	Ejemplo	Resultado
**	Potencia	4**3	64
*	Multiplicación	20.21*17	360,57
/	División	15/4	3.75
+	Suma / concat.	10+5	15
-	Resta	20-5	15
%	Módulo o residuo	15 % 2	1

6. Operadores

Jerarquía

Se cumple la jerarquía matemática. Por tanto por orden de ejecución es:

1. Lo que está entre paréntesis
2. Las potencias **
3. *, /, %
4. +, -

Ejemplos:

A) 7 * 5**3 / 4 % 3

6. Operadores

Jerarquía

Se cumple la jerarquía matemática. Por tanto por orden de ejecución es:

1. Lo que está entre paréntesis
2. Las potencias **
3. *, /, %
4. +, -

Ejemplos:

A) 7 * 5**3 / 4 % 3

7*125 / 4 % 3

875 / 4 % 3

218.75 % 3

2.75

6. Operadores

Jerarquía

B) 15 / 2 * (7 + (68 – 15 * 33 + (45**2 / 16) /3) / 15) + 19

6. Operadores

Jerarquía

B) 15 / 2 * (7 + (68 – 15 * 33 + (45**2 / 16) /3) / 15) + 19
15 / 2 * (7 + (68 – 15 * 33 + (2025 / 16) / 3) /15) + 19
15 / 2 * (7 + (18 - 15 * 33 + 126.5625 / 3) / 15) + 19
15 / 2 * (7 + (68 – 495 + 126.5625 / 3) / 15) + 19
15 / 2 * (7 + (68 – 495 + 42.1875) / 15) + 19
15 / 2 * (7 + (-427 + 42.1875) / 15) + 19
15 / 2 * (7 + (-)384.8125 / 15) + 19
15 / 2 * (7 + (-)25.6541) + 19
15 / 2 * (-)18.6541 + 19
7.5 * (-)18.6541 + 19
-139.9062 + 19
-120.90625

6. Operadores

Expresiones lógicas

Permiten formular condiciones complejas a partir de condiciones simples. Los operadores lógicos son de conjunción (y), disyunción (o) y negación (no).

```
5 === "5" //false
5 == "5" //true
5 !== "5" // true
5 != "5" // false
5 >= 7 // false (>, <, <=, >=)
!true // false
false && true // false
false || true // true
```

P	Q	!P	!Q	P && Q	P II Q
TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	TRUE	TRUE	FALSE	FALSE

Operador	Operación
!	No
&&	Y
I I	O

6. Operadores

Expresiones lógicas

El resumen de jerarquía es:

1. lo que está entre ()
2. la potencia **
3. *, /, %
4. +, -
5. ==, !=, < ,>, <=, >=, ===, !== ...
6. !
7. &&
8. ||

7. Excepciones

Definición

Errores ocasionados durante la ejecución de un programa informático. Cuando ocurre cierto tipo de error, el sistema reacciona ejecutando un fragmento de código que resuelve la situación, por ejemplo retornando un mensaje de error o devolviendo un valor por defecto.

Su función es separar el código del manejo de errores de la lógica de aplicación del programa. En aquellos lenguajes que incluyen soporte para el manejo de excepciones.

7. Excepciones

Capturar excepciones

try ... catch corresponde a un tipo de estructura de control con la que distribuir el flujo de un programa en función a posibles errores.

La primera parte, try, ejecutará el código susceptible de contener errores. Si el error a controlar efectivamente ocurre, el flujo del programa parará en esa línea y seguirá en el bloque de catch.

```
try {
        // Código que puede contener errores
} catch( error ) {
        // Controlo el error y actúo
}
```

7. Excepciones

Lanzar excepciones

También podemos lanzar una excepción cuando detectamos un comportamiento programado. Por ejemplo, si en una acción es necesario que el usuario esté logueado y no lo está, puedo lanzar una excepción.

La forma más sencilla para lanzar errores es utilizando throw. Este comando permite enviar al navegador un evento similar al que se produce cuando ocurre algún imprevisto o nos encontramos ante un tipo inesperado de datos. El lenguaje permite enviar todo tipo de elementos, incluyendo texto, números, valores booleanos o incluso objetos. Sin embargo, la opción más usual es enviar el objeto nativo Error:

```
throw new Error( "Something bad happened." );
```

8. Funciones y procedimientos

Las funciones son un elemento muy utilizado en la programación. Empaquetan y aíslan una parte de código que realiza una tarea específica del resto del programa.

Son un conjunto de instrucciones que ejecutan una tarea determinada y que hemos encapsulado en un formato estándar para que nos sea muy sencillo de manipular y reutilizar.

```
function sum(num1, num2) {
    const result = num1 + num2;
    return result;
}

console.log(sum(2, 5)); // 7
```

8. Funciones y procedimientos

Declaración de funciones

Corresponde al bloque:

```
function sum(num1, num2) {
    // proceso
}
```

Podemos distinguir los siguientes elementos en este bloque de código que define la función:

- La palabra reservada 'function'
- El nombre que queremos darle a dicha función, en este caso: sum.
- Unos paréntesis en el que escribimos los parámetros (si los necesitamos).
- El bloque de instrucciones con la lógica rodeada de las llaves {}.

8. Funciones y procedimientos

Llamar a la función

Lo hacemos con la instrucción: nombreDeLaFuncion();

La función puede ser llamada con o sin parámetros, como fuera definida. Los parámetros se definen incluyéndolos dentro de los paréntesis, y dependiendo de estos, la función devolverá resultados distintos.

Parámetros

Las funciones puede poseer, ninguno, uno o múltiples parámetros y se definen entre los (). Los argumentos serán los valores de dichos parámetros cuando se ejecute la función.

Los argumentos no tienen por qué tener los mismos nombres que los parámetros, aunque sí deben tener el mismo número y orden. Es decir, si una función tiene dos parámetros, tendremos que “pasarle” dos argumentos.

8. Funciones y procedimientos

```
function saludar(nombre, apellido, dia) {

    console.log("Hola,", nombre, apellido);

    if (dia === "viernes") {
        console.log("Ya es viernes, ¡buen finde!");
    } else {
        console.log("¡A seguir con la semana!");
    }
}
```

```
const name = "John";
const lastname = "Doe";
const day = "friday";

sayHello(name, lastname, day);
sayHello("Jane", "Doe", "monday");
```

8. Funciones y procedimientos

En la mayoría de los lenguajes hay dos formas de pasar las variables a una función, por valor o por referencia.

- Por valor significa que la función recibe sólo una copia del valor que tiene la variable, o sea, que no la puede modificar.

- Por referencia significa que se pasa la posición de memoria donde esta guardada la variable, por lo que la función puede saber cuánto vale, pero además puede modificarla de cualquier manera.

8. Funciones y procedimientos

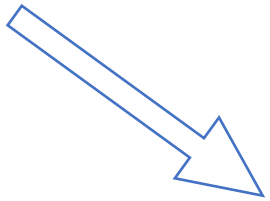
Repasemos los tipos de datos:

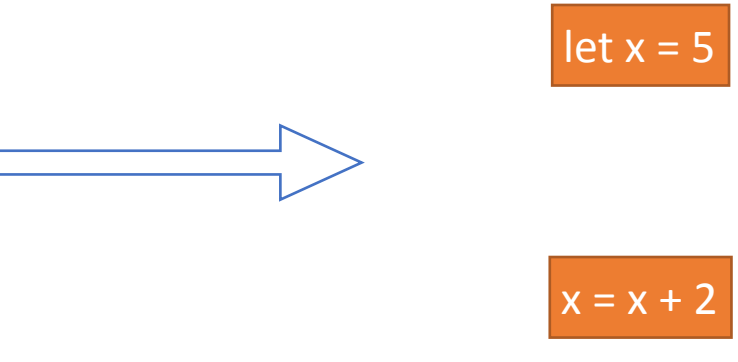
PRIMITIVOS	ESTRUCTURALES (TIPO OBJETO)
Number	Object
String	Array
Boolean	Function
Null	
Undefined	

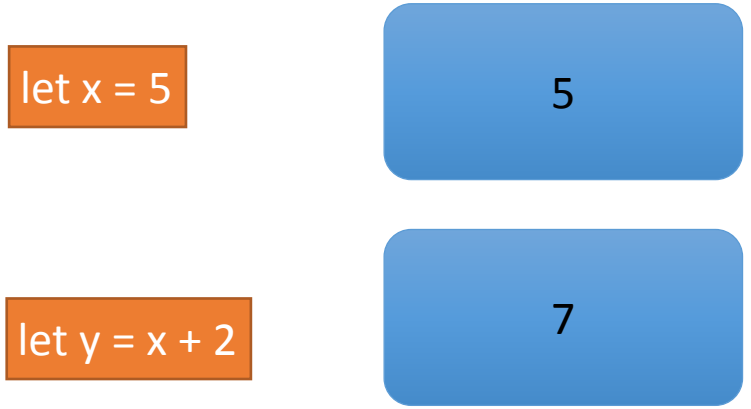
8. Funciones y procedimientos

Primitivos

- Inmutables (no se pueden alterar)
- No tienen propiedades ni métodos
- Se copian por valor

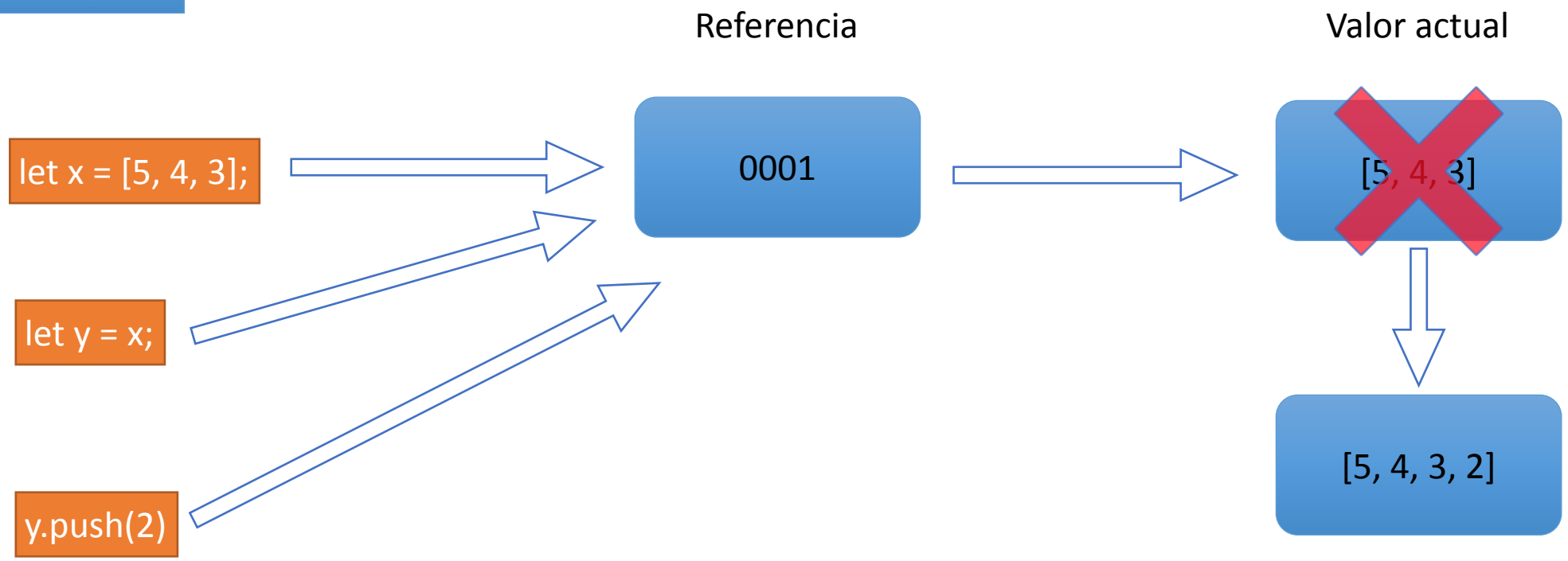






8. Funciones y procedimientos

Tipo - objeto



8. Funciones y procedimientos

Return: devolución de un resultado

Las funciones pueden o no devolver un resultado.

En el primer ejemplo de la función sayHello(), la función ejecutaba un console log, es decir mostraban una frase en pantalla, con lo que ya nos daba un resultado. Pero ¿qué ocurre, si en vez de esto, lo que calculan es una valor o una serie de valores que queremos utilizar en el resto del programa?

En estos casos, para devolver el valor calculado por la función se suele utilizar la palabra reservada 'return'.

8. Funciones y procedimientos

Variables locales y globales

Las variables definidas dentro de las funciones son denominadas variables locales y no pueden ser accedidas desde el exterior de la función.

Esta es una práctica muy recomendable que debemos seguir para que no entren en conflicto con otras variables del mismo nombre que puedan existir en el resto del script, incluyendo las que pueden estar dentro de otras funciones que también hayamos definido en dicho programa.

La contraposición a esto están las variables globales, aquellas que definimos fuera de las funciones que tienen efecto en todo el programa. Se dice por tanto que una variable puede tener ámbito (scope) local o global.

8. Funciones y procedimientos

Beneficios de utilizar funciones

- Las funciones permiten crear programas mejor estructurados y más claros, evitando repeticiones innecesarias de código y facilitando su mantenimiento.
- Las funciones 'empaquetan' y aislan del resto del programa una serie de variables e instrucciones de código que realizan alguna tarea específica.
- Los valores de las variables internas (locales) no entran en conflicto con el resto del programa.

8. Funciones y procedimientos

Tipos de funciones

Pueden ser funciones con nombre o sin él (anónimas).

Nombradas:

```
function suma(num1, num2) {
    return num1 + num2;
}

suma(2, 2);
```

Anónima

```
const saludar = function () { console.log("Hello"); };
```

Funciones de flecha:

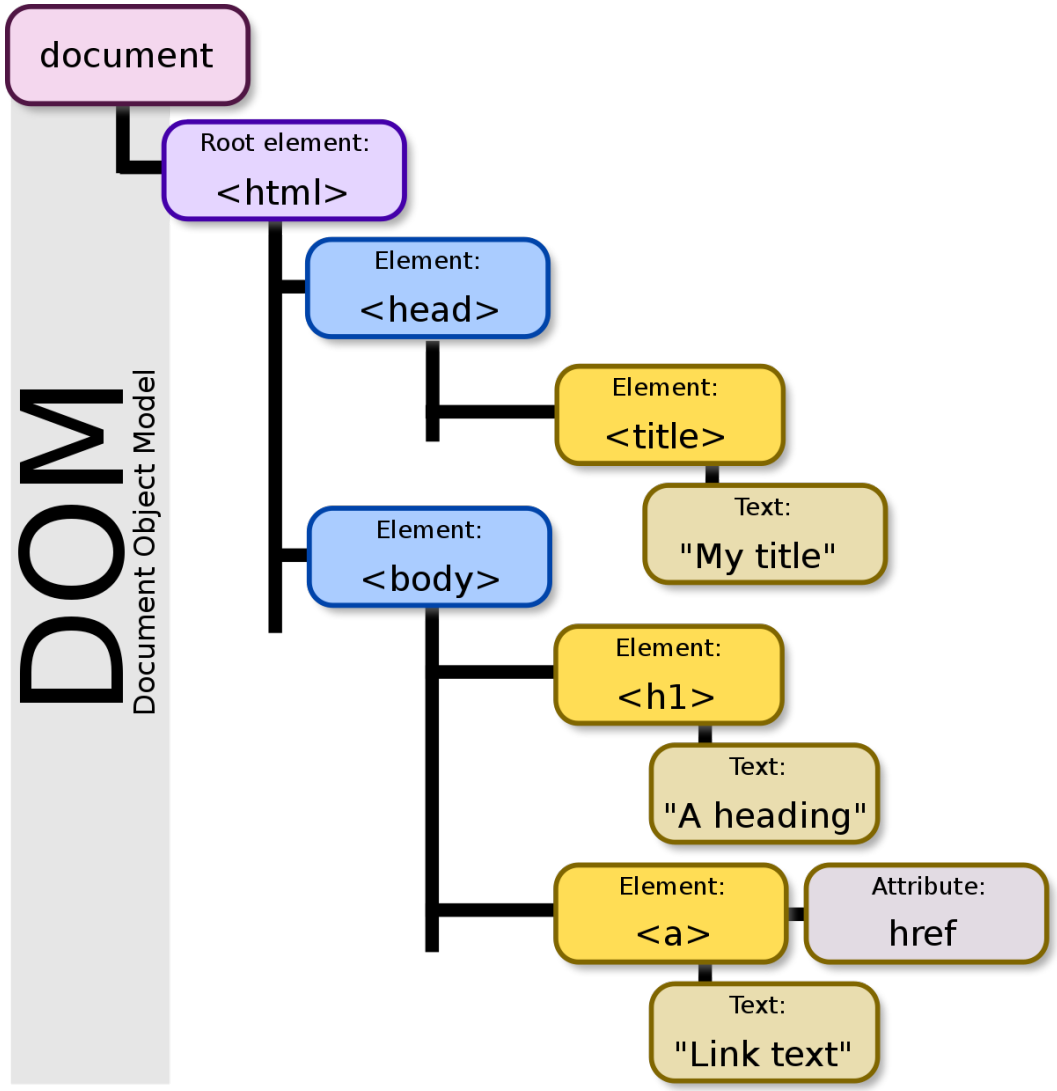
```
const perimeterOfSquare = (side) => 4*side;
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

9. Acceso al DOM

Document Object Model

- Árbol de elementos / nodos creado por el navegador.
- Podemos leer / escribir / manipular el DOM.



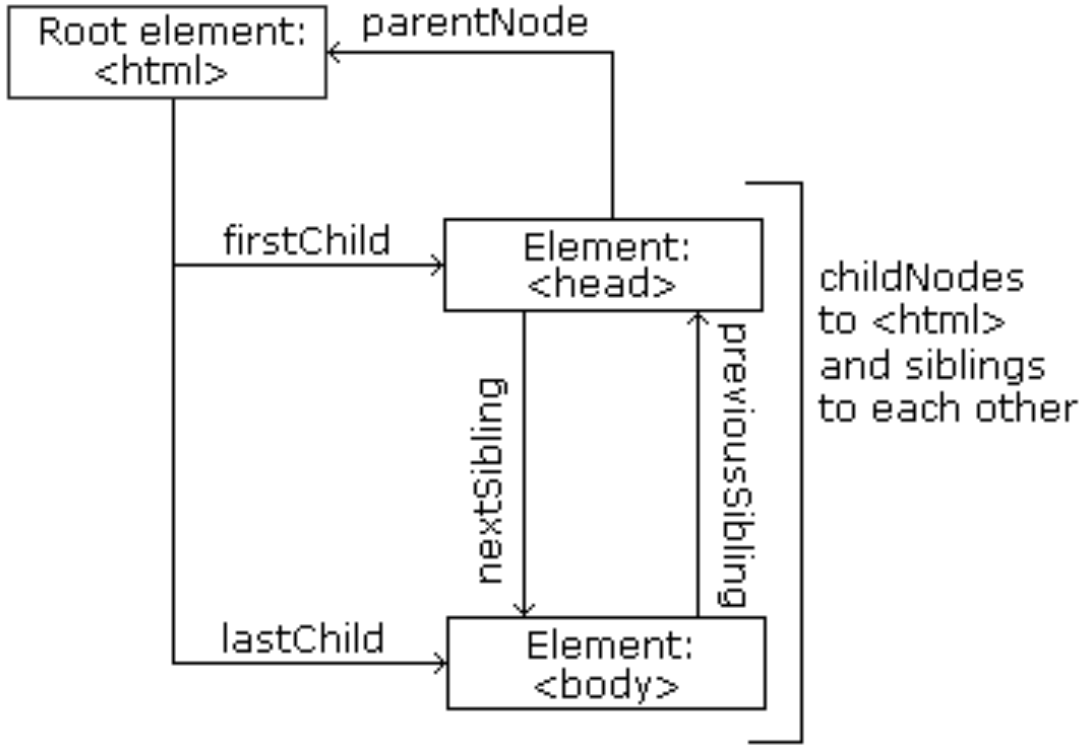
9. Acceso al DOM

Selectores

```
    // Selección por ID
    document.getElementById("mainContainer");

// Selección por etiqueta HTML
    document.getElementsByTagName("table");

// Selección por clase
    document.getElementsByClassName("rounded");

// Selección con un selector completo
    document.querySelector("div.container#mainContainer");
```

9. Acceso al DOM

- parentNode
- children[nodenumber]
- firstElementChild
- lastElementChild
- nextElementSibling
- previousElementSibling



9. Acceso al DOM

Cambiar los elementos existentes:

Propiedad	**Descripción**
elemento.innerHTML = *nuevo contenido html*	Cambia el contenido interno de un elemento
elemento.attribute = nuevo valor	Cambia el valor de un atributo
elemento.style.*property = estilo*	Cambia el estilo de un elemento
Method	**Description**
elemento.setAttribute*(atributo, valor)*	Establece el atributo de un elemento

9. Acceso al DOM

Crear y eliminar elementos

Metodo	Descripción
document.createElement(*elemento*)	Crea un elemento
document.removeChild(*elemento*)	Elimina elemento
document.appendChild(*elemento*)	Añade un elemento hijo
document.replaceChild(*nuevo, antiguo*)	Reemplaza un elemento
insertBefore()	Inserta antes de un elemento

10. Eventos

Existen varias maneras de ser notificado de los eventos del DOM. Dos enfoques comunes son *addEventListener*() y los manejadores (*handlers*) específicos de un evento.

Desde JS

```
document.getElementById(id).onclick = function(){
    //nuestro código
}
```

Desde HTML

```
<button onclick="nombreFuncion()">Al hacer click</button>
```

Eventos
https://developer.mozilla.org/en-US/docs/Web/Events

Manejadores específicos de eventos
https://developer.mozilla.org/en-US/docs/Web/Guide/Events/Event_handlers

10. Eventos

Event Listeners

Se utilizan para escuchar un evento, es decir, que esté pendiente de dicho evento. Se pueden añadir múltiples "event listener" sobre un elemento, tanto del mismo evento como de eventos distintos:

```
element.addEventListener("eventoQueQuieroControlar", function(){});
```

En este caso los eventos no llevan "on", por ejemplo:

```
element.addEventListener("click", function(){});
```

Para eliminar un event listener se usa:

```
element.removeEventListener("mousemove", myFunction);
```

11. Operadores avanzados

Spread operator / Operador de propagación

Utilizado sobre un array u objeto, podemos copiarlos por valor, fusionarlos, expandirlos, o separar sus elementos para un uso individual. Suele ser recomendado desde que se incluyó por su simplicidad.

Arrays

```
// Copiar
const paradigms = ["OOP", "Functional"];
let copy = [...paradigms]; // Copia superficial
// copy = paradigms; NO es una copia
```

```
// Separación de elementos
const fullStack = ["frontend", "backend"];
myFunction(...fullStack);
// myFunction("frontend", "backend")
```

```
// Expandir y fusionar
let array = [0, 1, 2];
let anotherArray = [4, 5, 6];
let number = 3;
numberStore = [...numberStore, number,  ...otherArray]; // [0, 1, 2, 3, 4, 5, 6]
```

11. Operadores avanzados

Spread operator / Operador de propagación

Objetos

```
// Copia superficial
const student= {
    name: "Alberto",
    age: 37
};

let copy = {...student};

// copy = student;
// NO sería una copia
```

```
// Expandir y fusionar
const user = {name: "Alberto", age: 35, address: "Calle Larios"};
const contactInfo = {phone: 654897341, email: "albertito13@yahoo.es"};
const completeInfo = {...user, ...contactInfo, address: "Carlos Haya"};
```

11. Operadores avanzados

```
const car = {
    brand: "Audi",
    model: "Q7",
    plateNumber: "0818BHG",
    color: "white"
};

const {brand, model, ...rest} = car;

console.log(brand, model); // Audi Q7
console.log(rest); // { plateNumber: '0818BHG', color: 'white' }
```

Desestructuración de objetos

A partir de un objeto, podemos crear variables con el nombre de propiedades existentes y con su valor cómodamente, el orden es irrelevante.

Desestructuración de arrays

A partir de un array, podemos crear tantas variables como elementos tenga, en orden.

```
const languajes = ["Go", "TypeScript", "Rust", "C#"];

const [lang1, lang2, ...rest] = ["Go", "TypeScript", "Rust", "C#"];

console.log(languaje1, languaje2); // Go TypeScript
console.log(rest); // ["Rust", "C#"]
```

11. Operadores avanzados

Encadenamiento opcional con “?”

Evita errores si propiedades embebidas son undefined

```
const adventurer = {
    name: 'Alice',
    cat: {
        name: 'Dinah'
    }
};

const dogName = adventurer.dog?.name;
console.log(dogName);
// expected output: undefined

console.log(adventurer.someNonExistentMethod?.());
// expected output: undefined
```

Cortocircuito con “&&” y “||”

Comprobación sencilla en una línea.

```
// La segunda instrucción será evaluada sólo
// si la primera es true
isAuthenticated && doSomePrivateOperations();
```

```
// La segunda será asignada SÓLO si la
// primera no existe.
// Útil para configurar valores por
// defecto que depende de datos existentes.
const PORT = existingPort || 5000;
```

11. Operadores avanzados

Valores truthy / falsy

En JavaScript, hay valores similares a true / false a efectos de una comprobación, pero que no son booleanos.

```
if (true) // true
if ({}) // true
if ([]) // true
if (42) // true
if ("0") // true
if ("false") // true
if (new Date()) // true
if (-42) // true
if (3.14) // true
if (-3.14) // true
if (Infinity) // true
```

```
if (false) // false
if (null) // false
if (undefined) // false
if (0) // false
if (-0) // false
if (NaN) // false
if ("") // false
```

Introducción a la Programación con JavaScript

Nacho Viano