Javascript

Javascript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript.
Es imperativo, débilmente tipado, dinámico, orientado a objetos, basado en prototipos.

(Fuente: Wikipedia)

Un archivo Javscript es texto.

Javascript se puede usar del lado del cliente (en el navegador web) y del lado del servidor (ej: Node.js (https://nodejs.org)).

Del lado del cliente los programas JS se incluyen en los documentos HTML, y se encargan de realizar acciones como pedir datos, confirmaciones, mostrar mensajes, animaciones, comprobar campos, interactuar con servicios web, etc.

Compatibilidad

En el pasado había muchos problemas de compatibilidad entre los navegadores.

Hoy día casi no hay problemas de compatibilidad, los navegadores modernos soportan casi todo lo nuevo de JS.

Ante la duda: https://caniuse.com

Incluir Javascript

Embebido en el HTML:

```
<script>
// código Javscript
</script>
```

En un archivo externo:

```
<script src="js/app.js"></script>
```

Constantes, variables y valores

- Constantes: una vez declaradas y definidas no se puede cambiar su valor. Por lo general sus nombres se escriben en mayúsculas

```
const PI = 3.141592654;
```

- Variables: se pueden declarar, o declarar y definir. Una vez definidas no solo pueden cambiar su valor sino también su tipo (débilmente tipado)

```
var saludo = "Hola mundo";
let a = x + 3;
let miSuperVariable;
```

No es obligatorio declarar las variables, aunque es aconsejable.

Para entender mejor las diferencias entre `let` y `var`:
https://stackoverflow.com/questions/762011/whats-the-difference-between-using-let-and-var

Nombres de las variables

- Deben comenzar con una letra o el `_` (guión bajo); los caracteres siguientes pueden ser números o letras
- No se pueden utilizar caracteres especiales como espacios o signos
- No se pueden utilizar nombres reservados (ej.: `return`, `for`, `if`)

Tipos de datos - Numéricos

- Se pueden almacenar números enteros y números reales (de coma flotante). Se puede trabajar con números en base 10 (decimal), base 8 (octal), y base 16 (hexadecimal)
- `Infinity` representa el concepto de infinito matemático (división por cero):
  - `Number.POSITIVE_INFINITY` = ∞
  - `Number.NEGATIVE_INFINITY` = -∞
  - `3 / 0` = Infinity
- `NaN` (*Not A Number*) es un valor que no se puede representar con un número definido, asignado o resultante de una operación matemática:
  - `0 / 0` = `NaN`
  - `Number.POSITIVE_INFINITY / Number.POSITIVE_INFINITY` = `NaN`
  - `10 / "Hola Mundo"` = `NaN`

Tipos de datos - Cadena de caracteres (String)

- Almacena texto
- Puede contener números, letras, signos y cualquier otro tipo de caracteres
- Los valores se pueden escribir entre comillas simples o dobles
- Ciertos caracteres se deben “escapar”, ej.: tabulador: `\t` contra-barra: `\\`, comilla simple: `\'` comilla doble: `\"`

Tipos de datos - Nulos (null)

- Son datos “vacíos”
- Por lo general, tienen un significado especial, para significar “nada”

Tipos de datos - Indefinidos (undefined)

- Variables declaradas cuyo valor aún no se definió con el operador de asignación
- También se puede encontrar `undefined` al intentar ejecutar una función que no está definida (las funciones en JS son “ciudadanos de 1er nivel”, como las variables)

Tipos de datos - Lógicos o Booleanos

- Valor de verdad: `true` (verdadero), o `false` (falso)
- El valor booleano de los operandos con tipos de datos `null` o `undefined` y el número `0` es `false`

Tipos de datos - Objetos

- Variables y funciones definidos previamente por el lenguaje o el navegador (objetos predefinidos como `Number`, `Math`, `Window`, `Document`, etc.), o por el usuario
- Se usan para almacenar tipos de datos complejos (ej.: registros de bases de datos), y también pueden encapsular funcionalidad

Operadores - De asignación

- Asigna el valor de la expresión de la derecha al objeto de la izquierda

```
y = 3 * x + 5;
geoCABA = {lat: -34.61315, long: -58.37723};
```

- Suma, resta, multiplica, o divide el valor de la expresión de la derecha a la variable de la izquierda

```
y += 8;
y -= 37 * x;
y *= y;
y /= 2;
```

Operadores - Aritméticos

- Suma, `+`; resta, `-`; multiplicación, `*`; y división, `/`
- Resto, `%`; devuelve el resto de la división entera entre dos números. Siempre toma el signo del dividendo
- Incremento, `++`; y decremento, `--`; suma o resta 1 al valor de una variable y asigna el resultado a dicha variable

```
let a = 10;
a++;
console.log(a); // Debe imprimir 11
```

Operadores - De relación

- Igual a, `==` ; `true` si los operandos son iguales
- Estrictamente igual a, `===` ; `true` si los operandos del mismo tipo son iguales
- No igual a, `!=` ; `true` si los operandos no son iguales
- Estrictamente no igual a, `!==` ; `true` si los operandos del mismo tipo no son iguales
- Mayor que, `>` ; `true` si el operando izquierdo es mayor que el derecho
- Mayor o igual que, `>=` ; `true` si el operando izquierdo es mayor o igual que el derecho
- Menor que, `<` ; `true` si el operando izquierdo es menor que el derecho
- Menor o igual que, `<=` ; `true` si el operando izquierdo es menor o igual que el derecho

Operadores - Lógicos

- Y (*AND*), `&&`; `true` si ambos operandos son `true`
- O (*OR*), `||`; `true` si cualquiera de los operandos es `true`
- No (*NOT*), `!`; `true` si el operando es `false`

Operadores - De cadenas

- Todos los de relación para comparar cadenas de caracteres
- Concatenación, `+`; “une” dos cadenas de caracteres en una nueva cadena de caracteres

Funciones

- Promueven la estructuración y reutilización del código
- Deben estar definidas antes de ser usadas (llamadas)
- Pueden, o no, retornar un valor u otra función
- Definición:

```
function f() {
  // instrucciones de la función
}
```

- Llamada:

```
f();
```

Funciones - Parámetros

- Son opcionales
- Son los valores de entrada que recibe una función
- Pueden ser de cualquier tipo, incluso otras funciones
- Pueden ser más de uno (separados por comas), se interpretan según el orden
- Pueden tener un valor predeterminado que se usa si la llamada no proporciona dicho valor (en lugar de ser `undefined`):

```
function f(a,b,c=3) {
  // código de la función
}

f(3,4); // c vale 3 porque no se pasó el parámetro
f(3,4,5); // c vale 5
```

Estructuras condicionales - if

- `if` . Se ejecuta el bloque de código si la expresión evalúa a `true` :

```
if (expresión) {
  /* código a ejecutar */
}
```

Ej:

```
let adivinar = Math.round(Math.random() * 10);
if (a > adivinar) {
  alert("El número es menor");
}
```

Estructuras condicionales - if ... else

- `if ... else`. Permite ejecutar código también cuando la expresión evalúa a `false`:

```
if (expresión) {
  /* código si true */
} else {
  /* código si false */
}
```

Ej:

```
let adivinar = Math.round(Math.random() * 10);
if (a > adivinar) {
  alert("El número es menor");
} else if (a < adivinar) {
  alert("El número es mayor");
} else {
  alert("¡Adivinó!");
}
```

Estructuras condicionales - switch

- `switch`. Útil para expresiones con más de dos resultados de evaluación:

```
switch (expresión) {
  case valor1:
  case valor2:
    // código si expresión es valor1 o es valor2
    break;
  case valor3:
    // código si expresión es valor3
    break;
  default:
    // código si expresión no vale ninguno de los anteriores
}
```

Estructuras repetitivas - while

- `while` . Evalúa una expresión y ejecuta el bloque mientras dicha evaluación sea `true` . El bloque puede no ejecutarse nunca

```
while (expresión) {
  // código a ejecutar en cada iteración
}
```

Ej:

```
let a = 0;
while (a < 10) {
  a++;
  console.log(a);
}
```

Estructuras repetitivas - do ... while

- `do ... while`. Ejecuta el bloque y evalúa una expresión para determinar si repite la ejecución. El bloque ejecuta al menos una vez

```
do {
  // código a ejecutar en cada iteración
} while (expresión)
```

Ej:

```
let a = 0;
do {
  a++;
  console.log(a);
} while (a < 10);
```

Estructuras repetitivas - for

- `for`. El más versátil, también puede funcionar como `while` y como `do ... while`, dependiendo de la condición de inicio y de la actualización. El bloque puede ejecutarse nunca, una, o más veces

```
for (inicialización; expresión; actualización) {
  // código a ejecutar en cada iteración
}
```

Ej:

```
for (let a = 1; a <= 10; a++) {
  console.log(a);
}
```

Objetos del navegador

- `window`. Raíz de la jerarquía de objetos del navegador. Representa a la ventana del navegador.
- `document`. Es el documento (página HTML) cargado en el navegador
- `document.forms`. Es la lista de formularios del documento cargado en el navegador

- Algunos métodos de window
- `alert(texto)`. Muestra un pop-up con el texto
- `open()`. Abre una nueva ventana del navegador
- `close()`. Cierra la ventana del navegador
- `print()`. Imprime la pantalla
- `prompt()`. Muestra un pop-up con un campo de entrada de datos y devuelve el valor ingresado por el usuario

Eventos en Javascript

- Sirven para definir un comportamiento de la página en respuesta a ciertas acciones del usuario (clic, entrada de texto), o a ciertas interacciones con sistemas externos (recepción de datos de un servicio web, desconexión de la red)
- Para definir las acciones a realizar cuando se produce el evento se usan los manejadores de eventos (*event handlers*), ej: `onclick`
- El manejador del evento es una función
- Cada elemento HTML tiene su propia lista de eventos soportados

Algunos eventos de Javascript

- `onfocus` / `onblur`. Foco, fuera de foco en el elemento
- `onchange`. Cambio (ej: cambia el valor de un `<input>` o de un `<select>`)
- `onclick`. Clic en el elemento
- `onkeydown` / `onkeypress` / `onkeyup`. Comienzo, presión, soltado de una tecla del teclado
- `onmousedown` / `onmouseup`. Presioné o solté un botón del mouse
- `onresize`. El elemento cambia de tamaño (ej: cuando cambio el tamaño de la ventana del navegador)