Unidad 1 - Fundamentos

Características básicas

- Es un lenguaje interpretado.
- Distingue entre mayúsculas y minúsculas.
- Las sentencias se separan por punto y coma, pero no es obligatorio si están en distintas líneas.
- Comentarios:
  - `// Comentario de una línea`
  - `/* Comentario de varias líneas */`

Identificadores (variables, constantes)

- Pueden estar formados por letras, dígitos, `_` y `$`.
- No pueden empezar por un dígito.
- No pueden ser palabras reservadas.
- Variables:
  - Suelen empezar por minúsculas.
  - Se usa estilo `camelCase` para nombres con varias palabras.
- Constantes:
  - Se suelen escribir en mayúsculas.
  - Se utiliza el estilo `PALABRA1_PALABRA2` para nombres con varias palabras.

```
let v;
let v = val1, ...;
```

- Declaración de variables: pueden inicializarse o no.
- Las variables declaradas con `let` son locales al bloque de código donde se declaran (`{}`).

```
var v;
var v1 = val1, ...;
```

- `var` es un método más antiguo de declaración de variables que `let`.
- Las variables declaradas con `var` son globales o locales a la función en la que aparecen.

```
const ID = valor;
```

- Declaración de constantes.

Tipos de datos

Existen 8 tipos básicos: `number`, `string`, `boolean`, `undefined`, `null`, `object`, `symbol` y `bigint`.

Tipo `number`

- Se utilizan para enteros y números de punto flotante.

- Valores especiales: `NaN`, `Infinity`, `Infinity`.

Tipo `string`

- Se pueden usar comillas dobles `"`, comillas simples `'`, o comillas invertidas `` ` ``.
  - Con comillas invertidas, se pueden insertar expresiones encerradas entre `${ … }`.

Tipo `null`

- Tiene un único valor: `null`.

Tipo `undefined`

- Tiene un único valor: `undefined`.
- Si una variable se declara pero no se inicializa, su valor será `undefined`.

Operador `typeof`

El operador `typeof` devuelve una cadena que indica el tipo de dato de la expresión:

```
typeof alert // "function"
typeof null  // "object"
```

- En algunos casos no devuelve lo que se espera, por compatibilidad con versiones antiguas de JavaScript.

Función `isNaN()`

- `isNaN(exp)` devuelve `true` si `exp` es `NaN`, y `false` en caso contrario.

Funciones `alert()`, `confirm()`, `prompt()`

- `alert(msg)` muestra una ventana modal con un mensaje.
- `confirm(msg)` muestra una ventana modal con un mensaje y dos botones (Aceptar/Cancelar), devolviendo `true` o `false`.
- `prompt(msg [,txt])` muestra una ventana modal con un mensaje, un cuadro de texto y dos botones. Devuelve el texto del cuadro, o `null` si se pulsa Cancelar.

Conversiones de tipos

Implícita

Cuando se espera un tipo específico, la expresión se convierte automáticamente a ese tipo.

Explícita

Utilizando funciones específicas para la conversión:

- `String(valor)` convierte `valor` en una cadena.
- `Number(valor)` convierte `valor` a número:

Valor	Devuelve
`true`	`1`
`false`	`0`

Valor	Devuelve
`undefined`	`NaN`
`null`	`0`
Cadena	Intenta convertir la cadena a número, `NaN` si no es posible

- `Boolean(valor)` convierte `valor` a booleano:

Valor	Devuelve
`undefined`	`false`
`null`	`false`
`0, NaN`	`false`
`""`	`false`
Otros valores	`true`

Operadores matemáticos

Operador	Ejemplo
`+` Unario	`+a`
`-` Unario	`-a`
`+` Binario	`a + b`
`-` Binario	`a - b`
`*` Producto	`a * b`
`/` División	`a / b`
`%` Resto	`a % b`
`**` Exponenciación	`a ** b`
`=` Asignación	`a = 3`
`+=`, `-=`, etc.	`a += 3`
`,` Coma	`a = 3, b = 2`
`++`, `--`	Incremento/decremento de variable

Concatenación de cadenas

- Se usa el operador `+`. Si uno de los operandos es de tipo cadena, el otro se convierte automáticamente a cadena.

Operadores de comparación

Operador	Ejemplo
`>`, `<`	`a < 4`
`>=`, `<=`	`a <= 4`
`==`	`a == 4`
`!=`	`a != 4`
`===`	`a === 4`

- Las cadenas se comparan alfabéticamente.

- No se puede comparar `NaN` directamente; se usa `isNaN()`.
- Comparar valores de distintos tipos convierte ambos valores a número, excepto:
  - `null` solo es igual a `undefined`.
  - `undefined` solo es igual a `null` y a sí mismo.

Operadores lógicos

Operador	Ejemplo
`||`	`a < 4 || b==2`
`&&`	`a <= 4 && b == 2`
`!`	`!(a == 4)`

- `||`: Devuelve el primer valor verdadero o el último evaluado si todos son falsos.
- `&&`: Devuelve el primer valor falso o el último evaluado si todos son verdaderos.

Sentencia `if`

```
if (exp) {
    // comandos
} else if (exp) {
  // comandos
} else {
  // comandos
}
```

- Las partes `else if` y `else` son opcionales.

Operador ternario `?:`

```
(exp) ? valor1 : valor2;
```

- Si `exp` es verdadera, devuelve `valor1`. Si es falsa, devuelve `valor2`.

Sentencia `switch`

```
switch (exp) {
  case val1:
      // comandos
      break;
  case val2:
      // comandos
      break;
  default:
      // comandos
      break;
}
```

- Utiliza igualdad estricta (tipo y valor deben coincidir).

Bucles

`while`

```
while (exp) {
    // comandos
}
```

`do...while`

```
do {
    // comandos
} while (exp);
```

`for`

```
for (exp1; condición; exp2) {
    // comandos
}
```

`break` y `continue`

- `break` termina el bucle o el `switch`.
- `continue` salta a la siguiente iteración del bucle.

Declaración de funciones

```
function nombre(par1, par2, …) {
    [return [valor];]
}
```

- Si no hay `return`, o el `return` no tiene valor, la función devuelve `undefined`.
- Estas funciones pueden usarse antes de ser declaradas.

Valores predeterminados de los parámetros

- Se puede asignar un valor por defecto a los parámetros:

```
function nombre(par1 = val1, …)
```

- También se puede asignar un valor usando `||`:

```
par1 = par1 || val1;
```

Expresiones de función

```
nombre = function(par1, par2, ...) {
    [return [valor];]
}
```

- No se pueden usar antes de declararlas.

Funciones flecha

```
nombre = (par1, par2, …) => {
    [return [valor];]
}

nombre = (par1) => valor;
```

- Si la función solo tiene una línea, no necesita llaves ni `return`.
- Si no tiene parámetros, se deben incluir paréntesis. Si tiene un solo parámetro, los paréntesis son opcionales.