Unidad 2

Resumen Unidad 2: Introducción a la programación en JavaScript

1. Introducción

La sintaxis de JavaScript es muy **parecida a Java y C++**. Cualquier programador que sepa programar en Java, PHP u otro lenguaje con sintaxis similar será capaz de comprender la sintaxis de JavaScript de una manera rápida.

No obstante, al ser JavaScript un lenguaje de programación del lado del cliente, es importante conocer sus aspectos básicos como:

- Los eventos
- El control de los elementos HTML

---

2. 10 reglas básicas de JavaScript

1. **Las instrucciones terminan en punto y coma** (`;`)
2. **Uso de decimales**: Los números con decimales utilizan el punto (`.`) como separador
3. **Literales**: Se pueden escribir entre comillas dobles (`"`) o simples (`'`)
4. **Declaración de variables**: Se utiliza la palabra reservada `var`
5. **Operador de asignación**: Es el símbolo igual (`=`)
6. **Operadores aritméticos**: `+`, `-`, `*`, `/`
7. **Variables en expresiones**: Las variables también se pueden utilizar en expresiones
8. **Comentarios en JavaScript**:
    - Comentarios en línea: `//`

- Comentarios de varias líneas: `/* */`
- Comentarios hashbang: `#!`

9. **Identificadores**: Comienzan por una letra, barra baja (`_`) o el símbolo del dólar (`$`)
10. **Case-sensitive**: JavaScript distingue entre mayúsculas y minúsculas

> ⚠️ **Importante**: `Mielemento`, `miElemento`, `Mielemento` y `MIELEMENTO` son cuatro identificadores completamente distintos para JavaScript.

---

3. Palabras reservadas de JavaScript

Las palabras reservadas más utilizadas son:

Palabra	Descripción
`var`	Utilizada para declarar una variable
`if ... else`	Estructura condicional
`for`	Estructura de repetición
`do ... while`	Estructura de repetición
`switch`	Serie de sentencias que van a ser ejecutadas dependiendo de diferentes circunstancias
`break`	Termina un switch o un bucle
`continue`	Sale del bucle y se coloca al comienzo de este
`function`	Declara una función
`return`	Sale de una función
`try ... catch`	Utilizadas para el manejo de excepciones

🔗 Listado completo de palabras reservadas en JS

---

4. Identificadores

Para definir identificadores en JavaScript hay que seguir estas reglas:

- ✅ Debe empezar obligatoriamente por una letra, guion bajo (`_`) o el símbolo del dólar (`$`)
- ✅ Pueden seguirle más letras, números o guiones bajos
- ✅ Distingue entre mayúsculas y minúsculas

- ✅ Puede usar todas las letras que están definidas en UNICODE

---

5. Comentarios

Existen 3 tipos de comentarios en JavaScript:

1. Comentarios en línea

Se utiliza `//` para iniciar el comentario. Todo lo que aparezca a continuación hasta el final de la línea se considera comentario.

```
// Controla el primer bucle (número de iteraciones)
var contador = 0;
```

2. Comentarios de varias líneas o comentarios de bloques

Se utiliza `/* comentario */` para comentarios más extensos.

```
/*
Almacena la cantidad de elementos actuales de la estructura.
Solo se incluyen los positivos.
Debe coincidir con los dispositivos válidos.
*/
let numElementos = 0;
```

3. Comentarios hashbang

Comentarios especiales que se utilizan para indicar la ruta a un motor de JavaScript específico que debe ejecutar el script. Se utiliza `#!`

```
#!/usr/bin/env node
```

---

6. Variables

VARIABLE: Posición de memoria a la que se le asigna un nombre y en la que se guarda un valor.

El nombre de la variable permitirá a JS poder ubicar y localizar dicho espacio cuando interprete los scripts.

```
var miVariable;
miVariable = 8;
```

> 📝 **Nota**: Solo es necesario declarar las variables una vez. Después pueden utilizarse tantas veces como se quiera simplemente usando su identificador.

---

7. Tipos de datos

Todas las variables son plenamente funcionales cuando se les asocia un valor y todos los valores pertenecen a un tipo concreto.

En JavaScript tenemos los siguientes tipos de datos:

1. STRING (Cadena de texto)

Secuencia de caracteres. Hay 3 formas de escribir una cadena de texto en JS:

```
miString_1 = "con comillas dobles";
miString_2 = 'con comillas simples';
miString_3 = `con comillas invertidas o backticks`;
```

Operador de concatenación (+):

```
subcadena_1 = "Primera parte, ";
subcadena_2 = "parte concatenada";
subcadena_3 = ". Última parte añadida";
cadenaFinal = "Cadena: " + subcadena_1 + subcadena_2 + subcadena_3 + ".";
```

Usando backticks (simplifica la concatenación):

```
cadenaFinal = `Cadena: ${subcadena_1}${subcadena_2}${subcadena_3}.`;
```

Secuencias de escape más utilizadas:

Secuencia	Uso
`\'`	Comillas simples
`\"`	Comillas dobles

Secuencia	Uso
`\\`	Barra invertida o backslash
`\b`	Retroceso
`\f`	Salto de página
`\n`	Salto de línea
`\r`	Retorno de carro
`\t`	Tabulador horizontal
`\v`	Tabulador vertical

2. NUMBER (Número)

Único tipo de dato numérico que recoge cualquier formato de número con o sin decimales.

```
alturaCm = 182;
pesoKg = 71.6;
diametroTransistor = 2e-9; // Notación científica
```

Sistemas numéricos:

```
numeroBinario = 0b100110;
numeroOctal = 0o46;
numeroHexadecimal = 0x26;
```

Curiosidades de JavaScript:

- Es capaz de operar con el valor infinito: `Infinity`
- Devuelve `NaN` (Not a Number) cuando opera con tipos de datos distintos

3. BOOLEAN (Booleano)

Dato lógico que solo puede tomar 2 valores: `true` (verdadero) o `false` (falso).

Al evaluar expresiones lógicas se considera:

- Todo lo que sea igual a cero es **FALSO**: `0`, `""`, `null`, `undefined`, `NaN`
- Todo lo que sea distinto de cero es **VERDADERO**

```
console.log(Boolean(1)); // true
console.log(Boolean(0)); // false
console.log(Boolean("texto")); // true
console.log(Boolean("")); // false
```

4. ARRAY (Veremos posteriormente)

5. OBJECT (Veremos posteriormente)

Otros tipos de datos:

- **UNDEFINED**: Valor que se obtiene cuando todavía no se le ha asignado un valor a una variable
- **NULL**: Representa el valor vacío o nulo, de forma intencionada
- **BigInt**: Para números enteros muy grandes

🔗 Tipos de datos en JS

---

8. Conversión entre tipos de datos

JS es un **lenguaje débilmente tipado**, es decir, que es capaz de realizar operaciones donde se mezclan tipos de datos distintos sin lanzar errores constantemente.

La conversión de tipos en JS se puede realizar de dos formas: **Automáticamente** o **Manualmente**.

Funciones de conversión:

1. **String()**: Convierte un número a string (cadena)
2. **Number()**: Convierte una cadena en número
3. **parseInt()**: Convierte una cadena a un número entero
4. **parseFloat()**: Convierte una cadena a un número decimal
5. **toDateString()**: Convierte fechas a string
6. **toUTCString()**: Convertir la hora a formato UTC

```
// Ejemplo conversión manual
console.log('3. 12+"5"= ${12+Number("5")}'); // Suma
console.log('6+7 Sin forzar: ${6+7}'); // 13
console.log('6+7 Forzando: ${String(6)+String(7)}'); // 67 (concatena)
```

---

9. Declaración e inicialización de variables

En JS se pueden declarar las variables de 3 formas: **var**, **let** y **const**.

1. VAR

Es la forma tradicional de declarar una variable.

- El ámbito de uso es el contexto de ejecución en el que se encuentra la palabra `var`

```
var miVariable;
miVariable = 8;
```

2. LET

La gran diferencia entre `let` y `var` es que el ámbito de `let` es más local que el de `var`.

- El ámbito es el bloque donde fue declarada y no puede utilizarse fuera

```
{
   var miVariable = "Soy visible fuera del bloque.";
   let otraVariable = "No soy visible fuera del bloque.";
}
console.log(miVariable); // ✅ Funciona
console.log(otraVariable); // ❌ Error
```

3. CONST

Se utiliza para definir **constantes**.

CONSTANTE: Variable a la que se le asigna un valor que no cambiará nunca.

```
const PI = 3.141592;
```

> 💡 **Convención**: Se suelen escribir en mayúsculas para distinguirlas del resto de las variables.

---

10. Entrada y salida del navegador

1. Mensajes en consola

```
console.log("Versión LOG de un mensaje por consola.");
console.info("%c Versión INFO del mensaje", "font-weight:bold");
console.warn("Versión WARN de un mensaje por consola.");
console.error("Versión ERROR de un mensaje por consola.");
```

Métodos útiles adicionales:

Método	Descripción
`console.dir()`	Muestra un listado interactivo de las propiedades de un objeto
`console.group ()`	Crea un nuevo grupo que tabula los mensajes
`console.table()`	Muestra los datos en forma de tabla
`console.time()`	Inicia un temporizador
`console.trace()`	Muestra una traza del error

2. Mensajes de entrada

```
let mensaje = "Para eliminar escribe ELIMINAR";
let respuesta = prompt(mensaje);
console.log(`El usuario escribió: ${respuesta}.`);
```

> 📌 **Comportamiento del** `prompt()`:

> - Si el usuario pulsa **Cancelar** → devuelve `null`

> - Si el usuario pulsa **Aceptar** sin escribir nada → devuelve cadena vacía

> - Si escribe algo y pulsa **Aceptar** → recibe la cadena introducida

3. Mensajes de alerta

```
const PI = 3.14159;
alert(`Recuerda usar esta aproximación de Π en tus cálculos: ${PI}`);
```

> ⚠️ **Importante**: Si bien la utilización de mensajes de alerta, cuadros de confirmación y cuadros de entrada de texto son muy útiles desde el punto de vista del aprendizaje, es importante destacar que son cosas del pasado. En la actualidad ninguna aplicación web sería utiliza estas herramientas del navegador para interactuar con el usuario.

---

11. Operadores

Tipos de operadores en JS:

1. Operadores aritméticos

Operador	Descripción
`+`	Suma
`-`	Resta
`*`	Multiplicación
`/`	División
`%`	Módulo
`++`	Incremento
`--`	Decremento
`**`	Potencia

Operadores especiales:

- **Incremento (`++`)**: Preincremento (`++a`) y Posincremento (`a++`)
- **Decremento (`--`)**: Predecremento (`--a`) y Posdecremento (`a--`)

2. Operadores de asignación

Operador	Ejemplo	Equivalente a
`=`	`x = y`	`x = y`
`+=`	`x += y`	`x = x + y`

Operador	Ejemplo	Equivalente a
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

3. Operadores de comparación

Operador	Significado	Ejemplo
`==`	Igualdad	`5 == 5` → true
`!=`	Distinto	`5 != 9` → true
`===`	Igualdad estricta	`5 === 5` → true
`!==`	Desigualdad estricta	`5 !== "5"` → true
`>`	Mayor que	`9 > 5` → true
`>=`	Mayor o igual que	`9 >= 3` → true
`<`	Menor que	`3 < 9` → true
`<=`	Menor o igual que	`5 <= 9` → true

4. Operadores lógicos

Operador	Uso	Descripción
AND lógico (`&&`)	`expr1 && expr2`	Devuelve `true` si ambos operandos son `true`
OR lógico (`\	\	`)
NOT lógico (`!`)	`!expr`	Devuelve `false` si su operando se puede convertir a `true`

Tablas de verdad:

AND (`&&`):

- `0 && 0` → `0`
- `0 && 1` → `0`
- `1 && 0` → `0`
- `1 && 1` → `1`

OR (`||`):

- `0 || 0` → `0`
- `0 || 1` → `1`
- `1 || 0` → `1`
- `1 || 1` → `1`

5. Operadores bit a bit

Tratan a sus operandos como una ristra de 32 bits y operan en binario bit a bit.

Operador	Uso	Descripción
AND (`&`)	`a & b`	Devuelve 1 en cada posición donde ambos bits son 1
OR (`\	`)	`a \
XOR (`^`)	`a ^ b`	Devuelve 0 donde ambos bits son iguales, o 1 si son distintos
NOT (`~`)	`~a`	Invierte los bits
`<<`	`a << b`	Desplazamiento a la izquierda
`>>`	`a >> b`	Desplazamiento a la derecha (con propagación de signo)
`>>>`	`a >>> b`	Desplazamiento a la derecha sin signo

6. Operadores de cadena

En JS se utilizan los operadores `+` y `+=` para concatenar cadenas.

```
console.log("Así " + " concatenamos " + 4 + " cadenas");
let cadena = "nombre";
cadena += " y apellidos";
```

7. Operador condicional (ternario)

Sintaxis: `condición ? valor1 : valor2`

```
let edad = 21;
let mensaje = (edad >= 18) ? "mayor" : "menor";
console.log(`El usuario es ${mensaje} de edad.`);
```

Precedencia de los operadores

De mayor a menor precedencia:

1. Miembro (`.[]`)
2. Llamar / crear instancia (`() new`)
3. Negación / incremento (`! ~ - + ++ -- typeof void delete`)
4. Multiplicar / dividir (`* / %`)
5. Adición / sustracción (`+ -`)
6. Desplazamiento bit a bit (`<< >> >>>`)
7. Relacional (`< <= > >= in instanceof`)
8. Igualdad (`== != === !==`)
9. AND bit a bit (`&`)
10. XOR bit a bit (`^`)
11. OR bit a bit (`|`)
12. AND lógico (`&&`)
13. OR lógico (`||`)
14. Condicional (`?:`)
15. Asignación (`= += -= *= /= %= <<= >>= >>>= &= &= |= ||= ??=`)
16. Coma (`,`)

🔗 Operadores JavaScript

---

12. Estructuras de control

Son instrucciones que permiten **elegir alternativas de ejecución**, **plantear bifurcaciones** o **realizar tareas repetitivas** en un punto dado, es decir, son instrucciones que permiten alterar el flujo de ejecución de un programa.

El elemento que determina estas trayectorias es la **condición**, que normalmente está construida con expresiones lógicas.

1. Estructuras condicionales

En función de una condición, permiten elegir un camino u otro.

1.1) IF

Es la estructura condicional más simple.

Sintaxis:

```
// IF SIMPLE
if (condición) {
   /* código a ejecutar si la condición es verdad */
}

// IF CON ELSE
if (condición) {
   /* código a ejecutar si la condición es verdad */
} else {
   /* código a ejecutar si la condición es falsa */
}

// IF ANIDADOS
if (condición1) {
   /* código si condición1 es verdad */
} else if (condición2) {
   /* código si condición1 es falsa y condición2 es verdad */
} else {
   /* código si todas las condiciones son falsas */
}
```

> 💡 **Importante**: En las estructuras `if`, y en general en todas las estructuras que utilicen bloques, si el contenido del bloque lo forma una única instrucción, no es necesario escribir las llaves.

1.2) SWITCH

Se utiliza para expresiones cuyo resultado puede ser variado y necesita distintas vías de ejecución. Es una solución mucho más sencilla que los `if` anidados.

Sintaxis:

```
switch (expresión) {
   case valor_1:
      instrucciones_1
```

```
    [break;]
  case valor_2:
    instrucciones_2
    [break;]
  ...
  default:
    instrucciones_predeterminadas
    [break;]
}
```

> 📝 **Nota**: Se utiliza la sentencia `break` para salir de cada bloque del `switch`. La sentencia `default` se ejecutará si ninguna de las opciones anteriores ha sido ejecutada con las opciones `case`.

2. Estructuras repetitivas

Permiten la ejecución repetitiva de una o varias instrucciones mientras se cumpla una condición.

2.1) WHILE

El cuerpo del bucle `while` se repite mientras la condición sea verdadera. Si no se cumple la condición, el bucle no se ejecuta nunca.

Sintaxis:

```
while (condición) {
  instrucciones;
}
```

> ⚠️ **Importante**: Asegurarse de que la variable que controla la condición de continuación se modifique dentro del bucle. De no hacerlo, es muy probable que nunca se dé una condición de salida y se produzca un **bucle infinito**.

2.2) DO WHILE

Variante del bucle `while`. La condición se sitúa al final del bucle, por tanto, el bucle `do while` **siempre se ejecuta al menos una vez**.

Sintaxis:

```
do {
   instrucciones;
} while (condición);
```

2.3) FOR

El bucle se repite hasta que la condición especificada sea falsa. El bucle `for` es **determinado**, es decir, que antes de la ejecución ya sabemos cuántas veces se va a ejecutar.

Sintaxis:

```
for (inicio; condición; incremento/decremento) {
   // código a repetir mientras la condición sea cierta
}
```

Dentro del paréntesis del FOR tenemos tres zonas separadas por `;`:

1. **Inicio del bucle**: Valor inicial de la variable de control del bucle
2. **Condición del bucle**: Lo que hace que el bucle se ejecute (la condición)
3. **Incremento/decremento**: Cómo cambia la variable de control para que el bucle llegue a finalizar

3. Estructuras de salto

Permiten romper el flujo de ejecución establecido por los bucles cuando se dan situaciones excepcionales que quieren controlarse.

3.1) BREAK

Se utiliza para terminar un bucle `while`, `do while` o `for` o una sentencia `switch` y transferir el control a la siguiente instrucción.

```
for (let i = 0; i < 5; i++) {
   if (i == 3) break;
   text += i + "<br>";
}
// Resultado: 0, 1, 2
```

3.2) CONTINUE

Se utiliza para, estando en medio de una iteración de un bucle, descartar el resto de las instrucciones del bloque y volver a evaluar la condición.

```
for (let i = 0; i < 5; i++) {
   if (i === 3) continue;
   text += i + "<br>";
}
// Resultado: 0, 1, 2, 4 (omite el 3)
```

3.3) LABELED

Completa la funcionalidad de las dos instrucciones anteriores. Establece puntos en el programa a los que se les asigna un nombre (etiqueta) al que hacer referencia cuando se desea efectuar un salto.

```
let primero = segundo = 1;
buclePrincipal: while (true) {
   console.log(`(Bucle principal) Iteración ${primero}`);
   primero++;
   while (true) {
      console.log(`(Bucle secundario) Iteración ${segundo}`);
      segundo++;
      if (segundo == 5)
         break;
      else if (primero == 3)
         break buclePrincipal;
   }
}
```

> ⚠️ **Para saber más**: Las instrucciones de salto son muy útiles en situaciones muy concretas, pero no debe abusarse de ellas. Un exceso de instrucciones de salto termina convirtiendo el programa en **código espagueti**, o lo que es lo mismo, un galimatías en el que nadie es capaz de seguir el flujo de ejecución del programa.

Conclusión

Esta unidad introduce los fundamentos de la programación en JavaScript, cubriendo desde la sintaxis básica hasta las estructuras de control más avanzadas. Los conceptos clave incluyen:

- Las 10 reglas básicas de JavaScript
- Tipos de datos: String, Number, Boolean
- Declaración de variables con `var`, `let` y `const`
- Operadores aritméticos, lógicos, de comparación y bit a bit
- Estructuras condicionales: `if`, `else`, `switch`
- Estructuras repetitivas: `while`, `do while`, `for`
- Estructuras de salto: `break`, `continue`, `labeled`

Estos conocimientos son fundamentales para comenzar a programar aplicaciones web interactivas con JavaScript.