








Introducción a JavaScript

Dando vida a tus sitios web

Fundamentos de la Web - Clase 7

Objetivos de Hoy

¿Qué aprenderemos?

-  Entender qué es JavaScript y para qué sirve
-  Crear y usar variables para guardar información
-  Conocer los tipos de datos básicos en JavaScript
-  Usar condicionales para tomar decisiones en el código
-  Dominar los bucles para repetir tareas
-  Trabajar con arreglos para organizar datos
-  Darle lógica e interactividad a nuestros sitios web

🌟 ¿Qué es JavaScript?

🌟 ¿Qué es JavaScript?

¿Cuándo necesito esto?

Escenario: "Mi sitio web se ve bonito con HTML y CSS, pero no hace nada interactivo"

JavaScript es el lenguaje que da vida a tu sitio web:

- 🎮 **Interactividad:** Hacer clic en botones y que pasen cosas
- 🎨 **Dinamismo:** Cambiar contenido sin recargar la página
- 🧮 **Lógica:** Realizar cálculos y tomar decisiones
- ✨ **Funcionalidad:** Validar formularios, crear animaciones, y mucho más

🌟 HTML, CSS y JavaScript

Piensa en construir una casa:



HTML → Es la estructura (paredes, techo, puertas)



CSS → Es el estilo (pintura, decoración, muebles)



JavaScript → Es la funcionalidad (luz, agua, electrodomésticos)

Juntos crean una experiencia completa:

- HTML: El contenido
- CSS: Cómo se ve
- JavaScript: Cómo se comporta




Node.js - Nuestra Herramienta

Node.js

¿Qué es Node.js?

Node.js nos permite ejecutar JavaScript fuera del navegador, directamente en nuestra computadora.

¿Para qué lo necesitamos?

-  Probar nuestro código JavaScript
-  Encontrar y corregir errores
-  Aprender sin necesitar un navegador

Instalación:






Descarga desde <https://nodejs.org> y ejecuta el instalador.

 **Tip:** Ya deberías tenerlo instalado de clases anteriores

Ejecutar Archivos JavaScript

¿Cómo probar mi código?

Pasos para ejecutar un archivo .js:

1.  Crear un archivo con extensión `.js`
2.  Escribir código JavaScript en el archivo
3.  Guardar el archivo
4.  Abrir la terminal
5.  Ejecutar con Node.js

¡Vamos a verlo paso a paso!




Paso 1: Crear un Archivo JavaScript

En VS Code:

1. **Clic derecho** en la carpeta donde quieres crear el archivo
2. **New File** (Nuevo archivo)
3. **Nombrar** el archivo con extensión `.js`
 - Ejemplo: `prueba.js`
 - Ejemplo: `mi-primer-programa.js`

```
// prueba.js
console.log("¡Hola, JavaScript!");
console.log("Mi primer programa funciona");
var nombre = "Ana";
console.log("Hola, " + nombre);
```

 ¡No olvides guardar! (Ctrl + S o Cmd + S)

Paso 2: Abrir la Terminal

Tres formas de abrir la terminal en VS Code:

Opción 1: Menú superior

- `Terminal` → `New Terminal`

Opción 2: Atajo de teclado

- **Windows/Linux:** `Ctrl + Shift + ñ` o `Ctrl + '`
- **Mac:** `Cmd + '`

Opción 3: Clic derecho

- Clic derecho en el archivo `.js` → `Open in Integrated Terminal`

 La terminal aparecerá en la parte inferior de VS Code

Paso 3: Navegar a la Carpeta

Ubicarte en la carpeta correcta:

Ver dónde estás:

```
pwd          # En Mac/Linux
cd           # En Windows
```

Ir a la carpeta del archivo:

```
cd nombre-carpeta    # Entrar a una carpeta
cd ..                # Salir a la carpeta anterior
cd ejercicios         # Ejemplo: entrar a "ejercicios"
```

Ver archivos en la carpeta:

```
ls            # En Mac/Linux
dir           # En Windows
```

Paso 4: Ejecutar con Node.js

El comando mágico:

```
node nombre-archivo.js
```

Ejemplos:

```
node prueba.js  
node mi-primer-programa.js  
node ejercicio-01.js
```

¡Presiona Enter y verás el resultado! ⚡

Ejemplo Completo Paso a Paso

Vamos a ejecutar nuestro primer programa:

1. Crear archivo `hola.js` :

```
console.log("¡Hola desde JavaScript!");  
var edad = 25;  
console.log("Tengo " + edad + " años");
```

2. Abrir terminal (Ctrl + Shift + ñ)

3. Ejecutar:

```
node hola.js
```

4. Ver resultado:

```
¡Hola desde JavaScript!  
Tengo 25 años
```

Tips para Ejecutar Código

Consejos importantes:

- ✓ Guarda el archivo antes de ejecutar (Ctrl + S)
- ✓ Verifica el nombre del archivo (respeta mayúsculas/minúsculas)
- ✓ Asegúrate de estar en la carpeta correcta antes de ejecutar
- ✓ Si cambias el código, guarda y vuelve a ejecutar el comando

⚠ Si hay errores:

- Lee el mensaje de error (te dice en qué línea está el problema)
- Revisa tu código
- Guarda y ejecuta de nuevo

Errores Comunes al Ejecutar

Problema 1: "No se encuentra el archivo"

Error: Cannot find module 'prueba.js'

Solución: Verifica que estés en la carpeta correcta

Problema 2: "node no se reconoce"

```
'node' no se reconoce como un comando
```

Solución: Node.js no está instalado o no está en el PATH

Problema 3: Error en el código

SyntaxError: Unexpected token

Solución: Hay un error de sintaxis en tu código (revisa comillas, paréntesis, punto y coma)




Variables - Cajas para Guardar Información

Variables

¿Cuándo necesito esto?

Escenario: "Necesito guardar información para usarla después en mi programa"

Las variables son como cajas donde guardamos información:

-  Tienen un **nombre** para identificarlas
-  Guardan un **valor** que podemos cambiar
-  Pueden cambiar de contenido cuando queramos

```
var edad = 25;  
var nombre = "Ana";  
var estudiante = true;
```


Creando Variables

Sintaxis básica:

```
var nombreVariable = valor;
```

Ejemplos prácticos:

```
// Variables con diferentes tipos de datos
var edad = 50;                // Número
var apellido = "Santos";     // Texto (string)
var meGustaJS = true;        // Booleano (verdadero/falso)
var ciudad;                  // Variable sin valor (undefined)
```




 **Importante:** Usamos `var` para declarar variables

Cambiando Variables

Las variables son "variables" porque pueden cambiar:

```
var edad = 25;  
console.log(edad);      // Muestra: 25  
  
edad = 30;              // Cambiamos el valor  
console.log(edad);      // Muestra: 30  
  
edad = "Treinta";       // ¡Incluso podemos cambiar el tipo!  
console.log(edad);      // Muestra: Treinta
```

¿Qué es console.log()?

-  Muestra información en la consola
-  Como una "linterna" para ver qué pasa en nuestro código
-  Herramienta esencial para depurar

+ Operadores Matemáticos

+ Operadores Básicos

Símbolos para hacer operaciones:

Operador	Qué hace	Ejemplo
+	Suma números, une texto	12 + 3 → 15
-	Resta números	12 - 3 → 9
*	Multiplica	12 * 3 → 36

```
var suma = 10 + 5;           // 15
var resta = 10 - 5;          // 5
var multiplicacion = 10 * 5; // 50
var division = 10 / 5;       // 2
```

+ Operaciones con Texto

El símbolo + también une cadenas de texto:

```
var nombre = "Juan";  
var edad = 25;  
  
console.log("¡Hola, " + nombre + "! Tienes " + edad + " años.");  
// Resultado: ¡Hola, Juan! Tienes 25 años.
```

Esto se llama concatenación:

- Unir textos con el signo +
- Puedes mezclar texto y variables
- Útil para crear mensajes dinámicos

+ Orden de Operaciones

JavaScript sigue las reglas matemáticas:

```
var resultado1 = 2 + 4 * 10;  
console.log(resultado1);    // 42 (primero multiplica, luego suma)  
  
var resultado2 = (2 + 4) * 10;  
console.log(resultado2);    // 60 (paréntesis primero)  
  
var resultado3 = 5 * 3 + 10 / 2 - 1;  
console.log(resultado3);    // 19
```

Regla PEMDAS:

1. Paréntesis
2. Multiplicación y División (de izquierda a derecha)
3. Suma y Resta (de izquierda a derecha)

+ Operadores Combinados

Abreviando operaciones comunes:

```
var total = 10;  
  
total += 5;      // Equivale a: total = total + 5 → 15  
total -= 3;      // Equivale a: total = total - 3 → 12  
total *= 2;      // Equivale a: total = total * 2 → 24  
total /= 4;      // Equivale a: total = total / 4 → 6
```

También funciona con texto:





```
var mensaje = "¡Hola, ";  
mensaje += "mundo!";    // "¡Hola, mundo!"
```

Tipos de Datos

Tipos de Datos en JavaScript

Diferentes categorías de información:

Tipos básicos que usaremos hoy:

1.  **Number (Número):** 25 , 3.14 , -10
2.  **String (Texto):** "Hola" , 'JavaScript'
3.  **Boolean (Booleano):** true , false
4.  **Undefined:** Variable sin valor asignado

```
var edad = 25;           // Number
var nombre = "Ana";      // String
var esEstudiante = true; // Boolean
var ciudad;             // Undefined
```

? Pregunta Rápida

¿Qué mostrará este código?

```
var a = "5";  
var b = 3;  
var resultado = a + b;  
console.log(resultado);
```

- A) 8
- B) "53"
- C) 53
- D) Error

Piensa unos segundos... 🤔

✓ Respuesta

```
var a = "5";           // Texto (string)
var b = 3;              // Número
var resultado = a + b;
console.log(resultado); // "53"
```

Respuesta correcta: B) "53"

¿Por qué?

- `a` es un texto (string), no un número
- Cuando usas `+` con un string, concatena en vez de sumar
- `"5" + 3` → `"53"` (texto)
- Si quisieras sumar: `var a = 5;` (sin comillas)

¡Excelente si acertaste! 🎉





Booleanos y Operadores de Comparación

Booleanos

¿Cuándo necesito esto?

Escenario: "Necesito que mi código tome decisiones según condiciones"

Los booleanos solo tienen dos valores:

-  `true` (verdadero)
-  `false` (falso)

```
var esLunes = true;  
var esFinDeSemana = false;  
var mayorDeEdad = true;  
var estaLloviendo = false;
```

 Son como interruptores: encendido o apagado

Operadores de Comparación

Comparan valores y devuelven true o false:

Operador	Significado	Ejemplo	Resultado
<code>==</code>	Igual a	<code>5 == 5</code>	<code>true</code>
<code>!=</code>	Diferente de	<code>5 != 3</code>	<code>true</code>
<code>></code>	Mayor que	<code>10 > 5</code>	<code>true</code>
<code><</code>	Menor que	<code>3 < 8</code>	<code>true</code>
<code>>=</code>	Mayor o igual	<code>5 >= 5</code>	<code>true</code>
<code><=</code>	Menor o igual	<code>4 <= 3</code>	<code>false</code>

Ejemplos de Comparaciones

```
var numero = 10;

console.log(numero > 5);      // true (10 es mayor que 5)
console.log(numero < 5);      // false (10 no es menor que 5)
console.log(numero == 10);    // true (10 es igual a 10)
console.log(numero != 8);     // true (10 es diferente de 8)

var edad = 20;
console.log(edad >= 18);      // true (20 es mayor o igual a 18)
```

Resultado: Cada comparación devuelve `true` o `false`



Condicionales - Tomando Decisiones



El Condicional IF

¿Cuándo necesito esto?

Escenario: "Quiero que mi código haga diferentes cosas según la situación"

Sintaxis básica:

```
if (condición) {  
    // Código que se ejecuta SI la condición es verdadera  
}
```

Ejemplo:

```
var esDeDia = true;  
if (esDeDia) {  
    console.log("¡Explora el mundo!");  
}  
// Se muestra: ¡Explora el mundo!
```

IF y ELSE

Para cuando la condición NO se cumple:

```
var clima = "soleado";

if (clima == "soleado") {
    console.log("¡Es un día perfecto para un picnic!");
} else {
    console.log("¡Mejor quedémonos en casa!");
}
```

Estructura:

- `if` : Si la condición es `true` , ejecuta este bloque
- `else` : Si la condición es `false` , ejecuta este otro bloque



ELSE IF - Múltiples Condiciones

Para verificar varias opciones:

```
var clima = "lluvioso";

if (clima == "soleado") {
    console.log("¡Es un día perfecto para un picnic!");
} else if (clima == "lluvioso") {
    console.log("¡Hora de saltar en charcos!");
} else if (clima == "nublado") {
    console.log("Lleva un paraguas por si acaso");
} else {
    console.log("¡Vamos a tirarnos en trineo en la nieve!");
}
```


Se ejecuta solo el primer bloque cuya condición sea verdadera



Operadores Lógicos: Combinar condiciones

AND (&&) - "Y":  Ambas condiciones deben ser verdaderas

```
var temperatura = 25;  
var estaLloviendo = false;  
if (temperatura >= 20 && !estaLloviendo) {  
    console.log("¡Este es un buen día para dar un paseo!");  
}
```

OR (||) - "O":  Al menos una condición debe ser verdadera

```
if (temperatura < 5 || estaLloviendo) {  
    console.log("¡Mejor quedarse en casa!");  
}
```

Tabla de Verdad

Cómo funcionan AND y OR:

AND (&&) - Ambas deben ser verdaderas:

```
true  && true   → true
true  && false  → false
false && true    → false
false && false   → false
```

OR (||) - Al menos una debe ser verdadera:

```
true  || true   → true
true  || false  → true
false || true    → true
false || false   → false
```


1234 Operador Módulo (%)

¿Cuándo necesito esto?

Escenario: "Quiero saber si un número es par, impar, o divisible por otro"

El módulo (%) devuelve el residuo de una división:

```
console.log(10 % 2);    // 0 (10 ÷ 2 = 5, residuo 0)
console.log(10 % 3);    // 1 (10 ÷ 3 = 3, residuo 1)
console.log(17 % 5);    // 2 (17 ÷ 5 = 3, residuo 2)
```

Analogía del chocolate:

- Tienes 5 chocolates y 2 amigos
- $5 \% 2 = 1$ → Sobra 1 chocolate (es impar)
- Si tienes 6 chocolates: $6 \% 2 = 0$ → No sobra nada (es par)

12 34 Par o Impar

Usando el operador módulo:

```
var numero = 7;

if (numero % 2 == 0) {
    console.log(numero + " es PAR");
} else {
    console.log(numero + " es IMPAR");
}
// Resultado: 7 es IMPAR
```

¿Cómo funciona?

- Si `numero % 2 == 0` → El número es par (residuo 0)
- Si `numero % 2 == 1` → El número es impar (residuo 1)

12 34 Divisibilidad

Verificar si un número es divisible por otro:

```
var numero = 78;

if (numero % 3 == 0) {
    console.log(numero + " es divisible por 3");
} else {
    console.log(numero + " NO es divisible por 3");
}
// Resultado: 78 es divisible por 3
```

Aplicación práctica:

- $78 \% 3 = 0$ → Divisible perfectamente
- $79 \% 3 = 1$ → No es divisible (sobra 1)

? Pregunta Rápida

¿Qué mostrará este código?

```
var edad = 16;  
  
if (edad >= 18) {  
    console.log("Puedes votar");  
} else {  
    console.log("Aún no puedes votar");  
}
```

- A) "Puedes votar"
- B) "Aún no puedes votar"
- C) No muestra nada
- D) Error

Piensa unos segundos... 🤔

✓ Respuesta

```
var edad = 16;

if (edad >= 18) {                                // 16 >= 18 es false
  console.log("Puedes votar");
} else {
  console.log("Aún no puedes votar"); // Este se ejecuta
}
```

Respuesta correcta: B) "Aún no puedes votar" ¿Por qué?

- `edad` es 16
- La condición `16 >= 18` es `false`
- Se ejecuta el bloque `else`

¡Perfecto si acertaste! 🎉

Pensamiento Algorítmico



¿Qué es un Algoritmo?

¿Cuándo necesito esto?

Escenario: "Necesito que la computadora resuelva un problema paso a paso"

Un algoritmo es una receta o instrucciones paso a paso para resolver un problema:

- Serie de pasos ordenados
- Cada paso debe ser claro y específico
- Debe tener un inicio y un fin
- Se puede repetir cada vez que lo necesites



¡Ya usas algoritmos todos los días sin darte cuenta!



Algoritmos en la Vida Diaria

Ejemplo 1: Hacer un sándwich 🥪

1. Tomar dos rebanadas de pan
2. Untar mantequilla en una rebanada
3. Colocar jamón y queso
4. Cubrir con la otra rebanada
5. ¡Listo para comer!

Esto es un algoritmo: Pasos ordenados para lograr un objetivo



Algoritmos en Programación

Ejemplo: Encontrar el número más grande

Problema: Tengo 5 números, ¿cuál es el más grande?

Algoritmo (en palabras):

1. Tomo el primer número y lo marco como "el más grande"
2. Comparo el segundo número con "el más grande"
3. Si es mayor, lo marco como "el nuevo más grande"
4. Repito con el tercer número, cuarto y quinto
5. Al final tengo el número más grande

Esto es exactamente lo que haremos en JavaScript ✨



Pensamiento Algorítmico

¿Cómo desarrollarlo?

Antes de escribir código, pregúntate:

1. 🎯 ¿Qué problema quiero resolver?
2. 📋 ¿Qué pasos necesito seguir?
3. 🔄 ¿Hay pasos que debo repetir?
4. ⚖️ ¿Necesito tomar decisiones?
5. ✅ ¿Cómo sé que terminé?



Pensamiento Algorítmico

Ejemplo: Sumar números del 1 al 10

- Inicio con suma = 0
- Repito: agregar cada número a la suma
- Termino cuando llegue a 10

Bucles - Repitiendo Tareas

¿Por qué usar Bucles?

¿Cuándo necesito esto?

Escenario: "Necesito hacer la misma tarea muchas veces"

Sin bucles (tedioso):





```
console.log("Mensaje 1");  
console.log("Mensaje 2");  
console.log("Mensaje 3");  
// ... repetir 100 veces 😓
```

Con bucles (eficiente):

```
for (var i = 1; i <= 100; i++) {  
    console.log("Mensaje " + i);  
}
```

Bucle FOR

¿Para qué sirve?

-  Ejecuta código un número específico de veces
-  Procesa listas de elementos uno por uno
-  Automatiza tareas repetitivas
-  Realiza cálculos iterativos

Sintaxis:

```
for (inicio; condición; incremento) {  
    // Código a repetir  
}
```

Ejemplo: Para repetir un número específico de veces:

Ejemplo:

```
for (var i = 0; i < 5; i++) {  
    console.log("Número: " + i);  
}
```

Resultado:

```
Número: 0  
Número: 1  
Número: 2  
Número: 3  
Número: 4
```

Anatomía del Bucle FOR

```
for (var i = 0; i < 3; i++) {  
    console.log(i);  
}
```

Desglose:

1. `var i = 0` - Inicialización: Empieza desde 0
2. `i < 3` - Condición: Continúa mientras i sea menor que 3
3. `i++` - Incremento: Suma 1 a i en cada vuelta
4. `{ ... }` - Código que se repite

Desglose del Bucle FOR (cont.)

```
for (var i = 0; i < 3; i++) {  
    console.log(i);  
}
```

Flujo:




- Vuelta 1: $i = 0$, imprime 0, i pasa a ser 1
- Vuelta 2: $i = 1$, imprime 1, i pasa a ser 2
- Vuelta 3: $i = 2$, imprime 2, i pasa a ser 3
- Termina porque i ya no es menor que 3



El Misterioso "i"

¿Por qué usamos "i"?

Es una tradición en programación:

-  Viene de "index" (índice)
-  Es corto y fácil de escribir
-  Todos los programadores lo entienden

Pero puedes usar cualquier nombre:

```
for (var contador = 1; contador <= 5; contador++) {  
    console.log("Vuelta " + contador);  
}  
  
for (var numero = 0; numero < 10; numero++) {  
    console.log(numero);  
}
```

Incremento y Decremento

Operadores especiales:

Incremento (++) - Suma 1:

```
var x = 5;  
x++;           // Equivale a: x = x + 1  
console.log(x); // 6
```

Decremento (--) - Resta 1:

```
var y = 10;  
y--;           // Equivale a: y = y - 1  
console.log(y); // 9
```

Operadores especiales (cont.):

En bucles:

```
// Contar hacia adelante
for (var i = 1; i <= 5; i++) {
    console.log(i); // 1, 2, 3, 4, 5
}

// Contar hacia atrás
for (var i = 5; i >= 1; i--) {
    console.log(i); // 5, 4, 3, 2, 1
}
```

Bucle WHILE

Cuando no sabes cuántas veces repetir:

Sintaxis:

```
while (condición) {  
    // Código a repetir  
}
```

Ejemplo:

```
var i = 0;  
while (i < 5) {  
    console.log("Número: " + i);  
    i++;  
}
```

Diferencia entre WHILE y FOR:

- `for` : Sabes exactamente cuántas veces repetir
- `while` : Repites mientras se cumpla una condición

FOR vs WHILE

Comparando ambos bucles:

Mismo resultado, diferente estructura:

```
// Con FOR
for (var i = 0; i < 3; i++) {
    console.log(i);
}

// Con WHILE
var i = 0;
while (i < 3) {
    console.log(i);
    i++;
}
```

Ambos muestran: 0, 1, 2

💡 **Tip:** Usa `for` cuando sabes cuántas veces, `while` cuando depende de una condición



Ejemplo Creativo con WHILE

```
var inicio = 0;
var fin = 10;

while (inicio <= fin) {
  console.log("inicio: " + inicio + ", fin: " + fin);
  inicio++;    // Sube de 0 a 10
  fin--;       // Baja de 10 a 0
}
```

Resultado:

```
inicio: 0, fin: 10
inicio: 1, fin: 9
inicio: 2, fin: 8
inicio: 3, fin: 7
inicio: 4, fin: 6
inicio: 5, fin: 5
```


⚠ Cuidado con Bucles Infinitos

¡Nunca olvides la condición de salida!

✗ Mal - Bucle infinito:

```
var i = 0;
while (i < 5) {
  console.log(i);
  // ¡Olvidamos incrementar i!
  // Se repetirá forever mostrando 0
}
```

✓ Bien - Con salida:

```
var i = 0;
while (i < 5) {
  console.log(i);
  i++;    // ¡No olvidar esto!
}
```

🛑 Un bucle infinito bloqueará tu programa

? Pregunta Rápida

¿Cuántas veces se ejecutará este bucle?

```
for (var i = 2; i <= 10; i += 2) {  
    console.log(i);  
}
```

- A) 4 veces
- B) 5 veces
- C) 10 veces
- D) Infinitas veces

Piensa unos segundos... 🤔



Respuesta







```
for (var i = 2; i <= 10; i += 2) {  
    console.log(i);  
}
```

Respuesta correcta: B) 5 veces ¿Por qué?

- Empieza en `i = 2`
- Incrementa de 2 en 2: `i += 2`
- Se ejecuta mientras `i <= 10`

```
for (var i = 2; i <= 10; i += 2) {  
    console.log(i);  
}
```

Valores que se imprimen:

- i = 2 
- i = 4 
- i = 6 
- i = 8 
- i = 10 
- i = 12  (ya no cumple i <= 10)



Ejercicio: Primeros Pasos (00-primeros-pasos.js)

1. ☒ Variables y operaciones básicas
2. ☒ Trabajar con texto
3. ☒ Tu primera condición (if-else)
4. ☒ Comparar números
5. ☒ Tu primer bucle
6. ☒ Bucles con mensajes
7. ☒ Sumar con bucles
8. ☒ Bucles con condiciones
9. ☒ Contar números positivos
10. ☒ Desafío: encontrar el mayor

¡Ejecuta el archivo y observa cómo funciona cada parte!

Práctica: Ahora hazlo tú

Ejercicios para completar (00-practica-primeros-pasos.js):

1. Crear tus propias variables
2. Tu calculadora básica
3. Comparar edades
4. Cuenta regresiva (5 a 1)
5. Mostrar números impares
6. Sumar tus números favoritos
7. Encontrar el número más pequeño

 **RETO FINAL:** ¡Crear una tabla de multiplicar!

 **Tiempo:** 15 minutos

Ejercicio: FizzBuzz Clásico

Crea un archivo `fizzbuzz.js` con:

Imprime los números del 1 al 30, pero:

- Si el número es divisible por 3 → Imprime "Fizz"
- Si el número es divisible por 5 → Imprime "Buzz"
- Si es divisible por 3 Y por 5 → Imprime "FizzBuzz"
- Si no → Imprime el número

Pistas:

- Usa un bucle `for` del 1 al 30
- Usa el operador módulo `%` para verificar divisibilidad
- Revisa primero si es divisible por ambos (3 y 5)




 **Tiempo:** 15 minutos

Depurando Código




¿Qué es Depurar?

Encontrar y corregir errores en el código

Depurar es como ser detective:

-  Buscar dónde está el problema
-  Examinar qué está pasando en cada línea
-  Corregir el error




Herramientas en VS Code:

-  Puntos de interrupción (breakpoints)
-  Ejecutar paso a paso
-  Ver valores de variables en tiempo real

Puntos de Interrupción

Pausar la ejecución en una línea específica:

Cómo colocar un breakpoint:






1.  Abre tu archivo `.js` en VS Code
2.  Haz clic en el margen izquierdo (al lado del número de línea)
3. Aparecerá un círculo rojo 

Ahora cuando ejecutes en modo depuración:

- El código se detendrá en esa línea
- Podrás ver el valor de todas las variables
- Avanzar línea por línea

Ejecutar en Modo Depuración

Pasos para depurar en VS Code:

1.  Abre tu archivo JavaScript
2.  Coloca puntos de interrupción donde quieras pausar
3.  Haz clic en "Run" → "Start Debugging"
4.  Elige "Node.js" si te lo pide
5.  Usa los controles:
 - **F5** - Continuar hasta el siguiente breakpoint
 - **F10** - Ejecutar la siguiente línea
 - **Shift+F5** - Detener depuración

 Verás los valores de las variables en tiempo real



Ejemplo de Depuración

```
function calcularDoble(numero) {  
    var resultado = numero * 2;  
    return resultado;  
}  
  
var x = 5;  
var doble = calcularDoble(x);  
console.log(doble);
```

Coloca breakpoints en:

- Línea 2: Ver el valor de `numero`
- Línea 3: Ver el `resultado` antes de retornarlo
- Línea 7: Ver el valor final de `doble`

🔍 Así descubrirás si hay errores en la lógica



Desafío de Bucles

Crea un archivo `desafio-bucles.js` con:

1. **Pares del 1 al 30:** Imprime solo los números pares
2. **Múltiplos de 4 descendente:** Del 100 al 0, solo múltiplos de 4
3. **Secuencia especial:** Imprime 10, 7, 4, 1, -2, -5
4. **Suma de pares:** Suma todos los pares del 1 al 50 y muestra el resultado
5. **Factorial:** Multiplica $1 \times 2 \times 3 \times \dots \times 20$ y muestra el producto

Pistas:

- Usa `for` para conteos específicos
- Usa `%` para verificar divisibilidad
- Crea variables acumuladoras para sumas y productos



Tiempo: 20 minutos

Arreglos - Organizando Datos

¿Qué son los Arreglos?

¿Cuándo necesito esto?

Escenario: "Necesito guardar muchos datos relacionados, como una lista de compras"

Sin arreglos (tedioso):

```
var compra1 = 1500;  
var compra2 = 2300;  
var compra3 = 890;  
var total = compra1 + compra2 + compra3; // ¡Y si hay 100 compras!
```




Con arreglos (eficiente):

```
var compras = [1500, 2300, 890, 1200, 3400];  
// ¡Todos los datos juntos y organizados!
```

Creando Arreglos

```
var nombreArreglo = [elemento1, elemento2, elemento3];
```

Características:

-  Guardas múltiples valores en una sola variable
-  Pueden ser números, textos, booleanos, ¡o incluso otros arreglos!
-  Todo organizado en un solo lugar

Ejemplos:

```
var numeros = [1, 2, 3, 4, 5];  
var nombres = ["Ana", "Luis", "María"];  
var mixto = [42, "Hola", true, 3.14];  
var compras = [1500, 2300, 890];
```


Índices - La Dirección de Cada Elemento

¡Los índices comienzan en 0!

```
var frutas = ["🍏 Manzana", "🍌 Banana", "🍊 Naranja", "🍇 Uvas"];
```

Posiciones:

Índice:	0	1	2	3
Valor:	"🍏 Manzana"	"🍌 Banana"	"🍊 Naranja"	"🍇 Uvas"

Acceder a elementos:

```
console.log(frutas[0]); // "🍏 Manzana"  
console.log(frutas[1]); // "🍌 Banana"  
console.log(frutas[3]); // "🍇 Uvas"
```




 **Importante:** El primer elemento siempre está en la posición 0

Propiedad .length

Saber cuántos elementos tiene el arreglo:

```
var compras = [1500, 2300, 890, 1200];  
console.log(compras.length); // 4
```

Útil para:

-  Saber la cantidad de elementos
-  Recorrer todo el arreglo
-  Acceder al último elemento

Acceder al último elemento:

```
var ultimaCompra = compras[compras.length - 1];  
console.log(ultimaCompra); // 1200  
  
// ¿Por qué -1? Porque los índices empiezan en 0  
// Si hay 4 elementos, el último está en posición 3
```

Recorrer Arreglos con Bucles

Procesar todos los elementos:

```
var compras = [1500, 2300, 890, 1200, 3400];
var total = 0;

for (var i = 0; i < compras.length; i++) {
    console.log("Compra " + (i+1) + ": $" + compras[i]);
    total += compras[i];
}

console.log("Total gastado: $" + total);
```

```
var compras = [1500, 2300, 890, 1200, 3400];  
var total = 0;  
  
for (var i = 0; i < compras.length; i++) {  
    console.log("Compra " + (i+1) + ": $" + compras[i]);  
    total += compras[i];  
}  
  
console.log("Total gastado: $" + total);
```

Resultado:

```
Compra 1: $1500  
Compra 2: $2300  
Compra 3: $890  
Compra 4: $1200  
Compra 5: $3400  
Total gastado: $9290
```

Método .push() - Agregar al Final

Añadir nuevos elementos:

```
var frutas = ["🍏 Manzana", "🍌 Banana"];  
  
console.log(frutas); // ["🍏 Manzana", "🍌 Banana"]  
  
frutas.push("🍊 Naranja");  
  
console.log(frutas); // ["🍏 Manzana", "🍌 Banana", "🍊 Naranja"]  
  
frutas.push("🍇 Uvas");  
  
console.log(frutas); // ["🍏 Manzana", "🍌 Banana", "🍊 Naranja", "🍇 Uvas"]
```

.push() agrega el elemento al FINAL del arreglo

Método .pop() - Eliminar del Final

Quitar el último elemento:

```
var frutas = ["🍏 Manzana", "🍌 Banana", "🍊 Naranja"];

console.log(frutas); // ["🍏 Manzana", "🍌 Banana", "🍊 Naranja"]

var ultimaFruta = frutas.pop();

console.log(ultimaFruta); // "🍊 Naranja"
console.log(frutas);      // ["🍏 Manzana", "🍌 Banana"]

frutas.pop();

console.log(frutas); // ["🍏 Manzana"]
```

.pop() elimina y devuelve el último elemento

? Pregunta Rápida

¿Qué mostrará este código?

```
var numeros = [10, 20, 30];  
numeros.push(40);  
numeros.pop();  
numeros.push(50);  
console.log(numeros[2]);
```

- A) 30
- B) 40
- C) 50
- D) Error

Piensa unos segundos... 🤔

✓ Respuesta

```
var numeros = [10, 20, 30];    // [10, 20, 30]
numeros.push(40);              // [10, 20, 30, 40]
numeros.pop();                 // [10, 20, 30] (quitó el 40)
numeros.push(50);              // [10, 20, 30, 50]
console.log(numeros[2]);        // Posición 2 → 30
```

Respuesta correcta: A) 30 ¿Por qué?

- `push(40)` agregó 40 al final
- `pop()` eliminó el 40
- `push(50)` agregó 50 al final
- Posición 2 sigue siendo 30
- El arreglo final: [10, 20, 30, 50]

Operaciones con Arreglos



Actualizar Elementos

Cambiar el valor de una posición:

```
var userData = ["Lewis", "Hamilton", "l.hamilton@email.com", "piloto"];

console.log(userData);
// ["Lewis", "Hamilton", "l.hamilton@email.com", "piloto"]

// Lewis ahora es fotógrafo
userData[3] = "Fotógrafo";

console.log(userData);
// ["Lewis", "Hamilton", "l.hamilton@email.com", "Fotógrafo"]
```

Sintaxis: `nombreArreglo[índice] = nuevoValor`



Ejemplo Completo de Operaciones

```
var empleado = ["Ana", "García"];

// Ver datos iniciales
console.log(empleado); // ["Ana", "García"]

// Agregar email
empleado.push("ana.garcia@empresa.com");
console.log(empleado); // ["Ana", "García", "ana.garcia@empresa.com"]

// Agregar puesto
empleado.push("Desarrolladora");
console.log(empleado); // ["Ana", "García", "ana.garcia@empresa.com", "Desarrolladora"]

// Cambiar puesto
empleado[3] = "Desarrolladora Senior";
console.log(empleado); // ["Ana", "García", "ana.garcia@empresa.com", "Desarrolladora Senior"]

// Ver cantidad de datos
console.log("Datos guardados: " + empleado.length); // 4
```

Ejercicio: Gestión de Lista

¡Hora de practicar arreglos!

Crea un archivo `gestion-lista.js` con:

1. Crea un arreglo `tareas` con 3 tareas pendientes
2. Agrega 2 tareas más con `.push()`
3. Muestra cuántas tareas hay en total
4. Recorre el arreglo e imprime cada tarea numerada
5. Elimina la última tarea con `.pop()`
6. Muestra el arreglo final

Ejemplo de salida esperada:

```
Total de tareas: 5  
1. Estudiar JavaScript  
2. Hacer ejercicio  
...
```

 **Tiempo:** 10 minutos

Desafíos con Arreglos

Desafío 1: Siempre Aburrido

Buscar en un arreglo:

Objetivo: Buscar "ver TV" en un arreglo de actividades.

```
var actividades = ["cantar", "correr", "salir", "ver TV"];

for (var i = 0; i < actividades.length; i++) {
  if (actividades[i] === "ver TV") {
    console.log("¡Entretenido!");
  } else {
    console.log("¡Estoy Aburrido!");
  }
}

// Resultado:
// ¡Estoy Aburrido!
// ¡Estoy Aburrido!
// ¡Estoy Aburrido!
// ¡Entretenido!
```


Desafío 2: Número de Corte

Filtrar números menores:

Objetivo: Guardar solo los números menores que un valor de corte.

```
var numeros = [1, 2, 8, 4, 5, 7, 6];
var valorCorte = 4;
var menores = []; // Arreglo para guardar resultados

for (var i = 0; i < numeros.length; i++) {
    if (numeros[i] < valorCorte) {
        menores.push(numeros[i]);
    }
}

console.log(menores);
// Resultado: [1, 2]
```

Desafío 3: Números bajo el promedio

Objetivo: Encontrar números por debajo del promedio.

```
var numeros = [1, 20, 3, 4, 15, 6, 27];

// Paso 1: Calcular el promedio
var total = 0;
for (var i = 0; i < numeros.length; i++) {
    total += numeros[i];
}
var promedio = total / numeros.length;

// Paso 2: Encontrar números bajo el promedio
var bajos = [];
for (var i = 0; i < numeros.length; i++) {
    if (numeros[i] < promedio) {
        bajos.push(numeros[i]);
    }
}

console.log("Promedio: " + promedio); // 10.86
console.log("Números bajo promedio: " + bajos); // [1, 3, 4, 6]
```

Desafío 4: Conteo de Pares

Contar números pares:

```
var numeros = [1, 2, 3, 4, 5, 6, 10, 11, 13, 14, 16, 18];
var contador = 0;

for (var i = 0; i < numeros.length; i++) {
  if (numeros[i] % 2 === 0) {
    contador++;
  }
}

console.log("Cantidad de números pares: " + contador);
// Resultado: 7 (hay 7 números pares)
```

Desafío 5: Fibonacci

Secuencia de Fibonacci:

Objetivo: Generar 10 números de la secuencia de Fibonacci.

```
var fibonacci = [0, 1]; // Empezamos con los dos primeros

// Generamos los siguientes 8 números
for (var i = 2; i < 10; i++) {
    var siguiente = fibonacci[i - 1] + fibonacci[i - 2];
    fibonacci.push(siguiente);
}

console.log(fibonacci);
// [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Cada número es la suma de los dos anteriores:

- $0 + 1 = 1$
- $1 + 1 = 2$
- $1 + 2 = 3$ etc



Práctica Final: Desafíos Completos

Crea un archivo `desafios-arreglos.js` con:

Resuelve los 5 desafíos usando variables, bucles y arreglos:

1. **Siempre Aburrido** - Buscar "ver TV" en actividades
2. **Número de Corte** - Filtrar números menores que 4
3. **Bajo el Promedio** - Números menores al promedio
4. **Conteo de Pares** - Contar números pares
5. **Fibonacci** - Generar secuencia de 10 números

Usa solo lo aprendido: variables, bucles, condicionales, arreglos



Tiempo: 25 minutos



Resumen y Recursos



Conceptos Clave de Hoy

Lo más importante:

- ✓ **Variables:** Cajas para guardar información (`var nombre = "Ana"`)
- ✓ **Tipos de datos:** Number, String, Boolean, Undefined
- ✓ **Operadores:** `+` , `-` , `*` , `/` , `%` , `==` , `!=` , `>` , `<`
- ✓ **Condicionales:** `if` , `else if` , `else` para tomar decisiones
- ✓ **Bucles:** `for` y `while` para repetir tareas
- ✓ **Arreglos:** `[]` para organizar múltiples datos






Lo Más Importante

Recuerda:

- 💡 `console.log()` es tu mejor amigo para ver qué pasa en tu código
- 💡 Los índices comienzan en 0 en los arreglos
- 💡 El operador módulo (%) sirve para verificar divisibilidad
- 💡 Los bucles evitan repetir código manualmente
- 💡 Los arreglos organizan datos relacionados
- 💡 Depurar con breakpoints te ayuda a encontrar errores

Recursos para Seguir Aprendiendo






Documentación y herramientas:

-  [MDN - JavaScript](#) - Documentación oficial
-  [JavaScript.info](#) - Tutorial interactivo
-  [Codecademy JavaScript](#) - Curso interactivo
-  [JSFiddle](#) - Editor online para probar código
-  [Eloquent JavaScript](#) - Libro gratuito online

Próximos Pasos

¿Qué viene después?

En las próximas clases veremos:







-  **Funciones** - Organizar código en bloques reutilizables
-  **Objetos** - Estructuras de datos más complejas
-  **DOM** - Manipular HTML desde JavaScript
-  **Eventos** - Hacer sitios web interactivos
-  **Callbacks** - Funciones dentro de funciones

¡JavaScript apenas comienza!

Práctica para Casa

Proyecto sugerido: Calculadora de Gastos

Crea un programa que:

1.  Tenga un arreglo con gastos de la semana
2.  Calcule el total gastado
3.  Muestre el promedio de gasto por día
4.  Identifique el gasto más alto
5.  Muestre cuántos gastos fueron mayores al promedio
6.  Imprima un resumen completo

Recursos: Usa todo lo aprendido hoy: arreglos, bucles, condicionales

Consejos Finales

Buenas prácticas:

- ✓ Usa nombres descriptivos para variables (`totalCompras` mejor que `x`)
- ✓ Comenta tu código para explicar partes complejas
- ✓ Prueba en partes pequeñas - No escribas todo de una vez
- ✓ Usa `console.log()` para ver qué pasa en cada paso
- ✓ No temas a los errores - Son parte del aprendizaje
- ✓ Practica todos los días - La programación se aprende haciendo
- ✓ Pide ayuda cuando la necesites - Todos empezamos desde cero

 ¡Felicidades!

Ya conoces los fundamentos de JavaScript

Ahora puedes crear programas con lógica, decisiones y datos organizados 

? Preguntas

¿Alguna duda sobre JavaScript, variables, condicionales, bucles o arreglos?

 ¡Excelente Trabajo!

¡Nos vemos en la siguiente clase con Funciones y Objetos!

No olvides completar todos los ejercicios 