Grupo 1 — Fundamentos y sintaxis (10 ejercicios)

1) Hola, consola

Enunciado: Muestra en la consola `"Hola, JS"` y en una alerta `"Bienvenidos"`. **Resultado esperado**: La consola imprime *Hola, JS* y aparece una alerta con *Bienvenidos*. **Ayuda de código**:

```
console.log("Hola, JS");
alert("Bienvenidos");
```

2) Variables y tipos

Enunciado: Declara una variable `nombre` (string) y otra `edad` (number). Imprime: `"Me llamo <nombre> y tengo <edad> años"`. **Resultado esperado**: Cadena interpolada correcta. **Ayuda de código**:

```
const nombre = "Ana";
let edad = 19;

console.log(`Me llamo ${nombre} y tengo ${edad} años`);
```

3) Operadores aritméticos

Enunciado: Pide dos números con `prompt` y muestra su suma, resta, producto y división. **Resultado esperado**: 4 líneas con los resultados. **Ayuda de código**:

```
const a = Number(prompt("A:"));
const b = Number(prompt("B:"));

console.log("Suma:", a + b);
console.log("Resta:", a - b);
console.log("Producto:", a * b);
console.log("División:", a / b);
```

4) Template literals

Enunciado: Con `producto` y `precio` genera: `"El <producto> cuesta <precio>€"` con dos decimales. **Resultado esperado**: Formato `precio.toFixed(2)`. **Ayuda de código**:

```
const producto = "Teclado";
const precio = 19.9;

console.log(`El ${producto} cuesta ${precio.toFixed(2)}€`);
```

5) Conversión y NaN

Enunciado: Convierte `"123abc"` a número de forma segura; si no es válido, muestra `"Entrada no numérica"`. **Resultado esperado**: Manejo de `Number.isNaN`. **Ayuda de código**:

```
const entrada = "123abc";
const n = Number(entrada);

if (Number.isNaN(n)) {
  console.log("Entrada no numérica");
} else {
  console.log(n);
}
```

6) Operadores lógicos

Enunciado: Dado `nota` entre 0 y 10, indica si **aprobado** (>=5) y **excelente** (>=9). **Resultado esperado**: Dos mensajes según condiciones. **Ayuda de código**:

```
const nota = 8;

if (nota >= 5) {
  console.log("Aprobado");
}

if (nota >= 9) {
  console.log("Excelente");
}
```

7) Incremento y asignación

Enunciado: Partiendo de `contador=0`, simula 3 clics sumando 1 por clic e imprime el valor final. **Resultado esperado**: `3`. **Ayuda de código**:

```
let contador = 0;

contador += 1;
contador += 1;
contador += 1;

console.log(contador);
```

8) Math básico

Enunciado: Genera un entero aleatorio entre 1 y 6 (dado). **Resultado esperado**: Nº entre 1 y 6. **Ayuda de código**:

```
const dado = Math.floor(Math.random() * 6) + 1;

console.log(dado);
```

9) Strings útiles

Enunciado: Normaliza un email: pásalo a minúsculas y recorta espacios. **Resultado esperado**: `" usuario@correo.com "` -> `"usuario@correo.com"`. **Ayuda de código**:

```
const email = " Usuario@Correo.com ";

console.log(email.trim().toLowerCase());
```

10) Comentarios y estilo

Enunciado: Reescribe un pequeño bloque con comentarios y nombres claros. **Resultado esperado**: Código más legible. **Ayuda de código**:

```
// Cálculo de área de rectángulo
const base = 5; // en cm
const altura = 3; // en cm

const area = base * altura;

console.log(`Área: ${area} cm²`);
```

Grupo 2 — Control de flujo y bucles (10 ejercicios)

1) If/else básico

Enunciado: Pide edad y muestra si es **mayor** o **menor** de edad. **Resultado esperado**: Mensaje correcto. **Ayuda de código**:

```
const edad = Number(prompt("Edad:"));

if (edad >= 18) {
  console.log("Mayor de edad");
} else {
  console.log("Menor de edad");
}
```

2) Múltiples condiciones

Enunciado: Dado un número, indica si es negativo, cero o positivo. **Resultado esperado**: Uno de los tres mensajes. **Ayuda de código**:

```
const n = Number(prompt("Número:"));

if (n < 0) {
  console.log("Negativo");
} else if (n === 0) {
  console.log("Cero");
} else {
  console.log("Positivo");
}
```

3) Switch de menús

Enunciado: Crea un menú (1: sumar, 2: restar). Pide dos números y ejecuta la opción. **Resultado esperado**: Resultado correcto. **Ayuda de código**:

```
const op = Number(prompt("1 Sumar / 2 Restar"));
const a = Number(prompt("A:"));
const b = Number(prompt("B:"));

switch (op) {
  case 1: {
    console.log(a + b);
    break;
  }
  case 2: {
    console.log(a - b);
    break;
  }
  default: {
    console.log("Opción inválida");
  }
}
```

4) For contador

Enunciado: Imprime los números del 1 al 10 en una sola línea separados por comas. **Resultado esperado**: `1,2,3,...,10`. **Ayuda de código**:

```
let salida = "";

for (let i = 1; i <= 10; i++) {
  salida += i + (i < 10 ? "," : "");
}
```

```
console.log(salida);
```

5) While adivina el número

Enunciado: Genera un número secreto 1–5 y pide intentos hasta acertar. **Resultado esperado**: Mensaje de acierto y nº de intentos. **Ayuda de código**:

```
const secreto = Math.floor(Math.random() * 5) + 1;
let intentos = 0;
let x;

while (x !== secreto) {
  x = Number(prompt("Adivina 1-5"));
  intentos++;
}

console.log(`¡Acertaste en ${intentos} intentos!`);
```

6) For...of en strings

Enunciado: Cuenta cuántas vocales tiene un texto. **Resultado esperado**: Número de vocales. **Ayuda de código**:

```
const texto = "Programación".toLowerCase();
let c = 0;

for (const ch of texto) {
  if ("aeiou".includes(ch)) {
    c++;
  }
}

console.log(c);
```

7) Break/continue

Enunciado: Recorre 1–20, salta múltiplos de 3 y detente al llegar a 17. **Resultado esperado**: Secuencia sin múltiplos de 3 hasta 17. **Ayuda de código**:

```
for (let i = 1; i <= 20; i++) {
  if (i % 3 === 0) {
    continue;
  }

  if (i === 17) {
    break;
```

```
  }

  console.log(i);
}
```

8) Do...while menú

Enunciado: Muestra un menú que se repite hasta que elija «Salir». **Resultado esperado**: Bucle controlado por opción. **Ayuda de código**:

```
let opcion;

do {
  opcion = prompt("1 Info / 2 Ayuda / 0 Salir");

  if (opcion === "1") {
    console.log("Info");
  } else if (opcion === "2") {
    console.log("Ayuda");
  }
} while (opcion !== "0");
```

9) Validación de entrada

Enunciado: Pide un número hasta que la entrada sea válida. **Resultado esperado**: Solo termina al introducir un número. **Ayuda de código**:

```
let n;

do {
  n = Number(prompt("Número válido:"));
} while (Number.isNaN(n));

console.log("OK:", n);
```

10) Mini-quiz

Enunciado: Haz 3 preguntas tipo test y cuenta aciertos. **Resultado esperado**: `"Aciertos: X/3"`. **Ayuda de código**:

```
let aciertos = 0;

const p1 = prompt("2+2=? 4/5");
if (p1 === "4") {
  aciertos++;
}
```

```
const p2 = prompt("Capital de España? Madrid/Paris");
if (p2.toLowerCase() === "madrid") {
  aciertos++;
}

const p3 = prompt("HTML es: lenguaje de marcado? si/no");
if (p3.toLowerCase() === "si") {
  aciertos++;
}

console.log(`Aciertos: ${aciertos}/3`);
```

Grupo 3 — Arrays y objetos (10 ejercicios)

1) Crear y acceder

Enunciado: Crea un array de 5 lenguajes y muestra el primero y el último. **Resultado esperado**: Dos líneas con valores. **Ayuda de código**:

```
const langs = ["HTML", "CSS", "JS", "PHP", "SQL"];

console.log(langs[0]);
console.log(langs[langs.length - 1]);
```

2) Push/Pop/Shift/Unshift

Enunciado: Simula una cola de impresión (push para añadir, shift para atender). **Resultado esperado**: Muestra el orden de atención. **Ayuda de código**:

```
const cola = [];

cola.push("Trabajo1", "Trabajo2");
console.log("Atendido:", cola.shift());

cola.push("Trabajo3");
console.log("Atendido:", cola.shift());

console.log("Pendientes:", cola);
```

3) Map

Enunciado: Dado `[1,2,3,4]` obtén sus cuadrados. **Resultado esperado**: `[1,4,9,16]`. **Ayuda de código**:

```
const nums = [1, 2, 3, 4];

const cuadrados = nums.map((n) => n * n);

console.log(cuadrados);
```

4) Filter

Enunciado: De una lista de precios, quédate con los > 20. **Resultado esperado**: Nuevo array con filtrados. **Ayuda de código**:

```
const precios = [9, 25, 12, 40, 7];

const mayores = precios.filter((p) => p > 20);

console.log(mayores);
```

5) Reduce

Enunciado: Suma total de un carrito `[10, 5, 8]`. **Resultado esperado**: `23`. **Ayuda de código**:

```
const carrito = [10, 5, 8];

const total = carrito.reduce((acc, n) => {
  return acc + n;
}, 0);

console.log(total);
```

6) Find/Some/Every

Enunciado: Con edades `[12,18,20,15]`, encuentra la primera >=18; ¿hay algún menor? ¿son todos mayores? **Resultado esperado**: `18`, `true`, `false`. **Ayuda de código**:

```
const edades = [12, 18, 20, 15];

const primeraMayor = edades.find((e) => e >= 18);
const hayMenor = edades.some((e) => e < 18);
const todosMayores = edades.every((e) => e >= 18);

console.log(primeraMayor);
console.log(hayMenor);
console.log(todosMayores);
```

7) Ordenar y copiar

Enunciado: Ordena un array de nombres sin mutar el original. **Resultado esperado**: Original intacto; copia ordenada. **Ayuda de código**:

```
const nombres = ["Lucía", "Ana", "Pedro"];

const copiaOrdenada = [...nombres].sort();

console.log(nombres);
console.log(copiaOrdenada);
```

8) Objetos básicos

Enunciado: Crea un objeto `alumno` con `nombre`, `curso`, `nota`. Imprime una frase con esos datos. **Resultado esperado**: String interpolado. **Ayuda de código**:

```
const alumno = {
  nombre: "Mar",
  curso: "SMR",
  nota: 8.2,
};

console.log(`${alumno.nombre} (${alumno.curso}) tiene un ${alumno.nota}`);
```

9) Array de objetos

Enunciado: Filtra los productos con `stock>0` y luego mapea sus nombres. **Resultado esperado**: Lista de nombres disponibles. **Ayuda de código**:

```
const productos = [
  { nombre: "Ratón", stock: 3 },
  { nombre: "Teclado", stock: 0 },
  { nombre: "Monitor", stock: 5 },
];

const disponibles = productos
  .filter((p) => p.stock > 0)
  .map((p) => p.nombre);

console.log(disponibles);
```

10) Desestructuración y rest/spread

Enunciado: Extrae `nombre` de `alumno` y agrupa el resto en `otros`. **Resultado esperado**: `nombre` y objeto `otros`. **Ayuda de código**:

```
const alumno2 = { nombre: "Iker", curso: "DAW", nota: 7.5 };

const { nombre, ...otros } = alumno2;

console.log(nombre);
console.log(otros);
```

Grupo 4 — Funciones y lógica (10 ejercicios)

1) Declaración vs expresión

Enunciado: Crea `function suma(a,b)` y `const resta = (a,b)=>a-b`. Pruébalas. **Resultado esperado**: Resultados correctos. **Ayuda de código**:

```
function suma(a, b) {
  return a + b;
}

const resta = (a, b) => {
  return a - b;
};

console.log(suma(2, 3));
console.log(resta(5, 2));
```

2) Parámetros por defecto

Enunciado: Función `saluda(nombre="mundo")` que devuelva `"Hola, <nombre>"`. **Resultado esperado**: `Hola, mundo` y `Hola, Ana`. **Ayuda de código**:

```
function saluda(nombre = "mundo") {
  return `Hola, ${nombre}`;
}

console.log(saluda());
console.log(saluda("Ana"));
```

3) Funciones puras

Enunciado: Escribe una función pura `iva(precio, tipo=0.21)` que calcule el total. **Resultado esperado**: Mismo input -> mismo output, sin efectos colaterales. **Ayuda de código**:

```
const iva = (precio, tipo = 0.21) => {
  return precio * (1 + tipo);
};
```

```
console.log(iva(100));
console.log(iva(100, 0.1));
```

4) Higher-order (map con función)

Enunciado: Crea `doble(n)` y úsala dentro de `map`. **Resultado esperado**: Nueva lista con valores *2. **Ayuda de código**:

```
const doble = (n) => n * 2;

console.log([1, 2, 3].map(doble));
```

5) Closures simples

Enunciado: Implementa `contador()` que devuelva una función que incremente y devuelva un número interno. **Resultado esperado**: Llamadas sucesivas: 1, 2, 3... **Ayuda de código**:

```
function contador() {
  let x = 0;
  return () => {
    x += 1;
    return x;
  };
}

const c = contador();

console.log(c());
console.log(c());
console.log(c());
```

6) Composición básica

Enunciado: Crea `compose(f,g)` tal que `compose(f,g)(x) = f(g(x))`. **Resultado esperado**: Composición funcionando. **Ayuda de código**:

```
const compose = (f, g) => {
  return (x) => {
    return f(g(x));
  };
};

const inc = (x) => x + 1;
const dup = (x) => x * 2;

console.log(compose(dup, inc)(3)); // 8
```

7) Recursión

Enunciado: Calcula factorial de `n` recursivamente. Maneja `0! = 1`. **Resultado esperado**: Factorial correcto. **Ayuda de código**:

```
function fact(n) {
  if (n <= 1) {
    return 1;
  }
  return n * fact(n - 1);
}

console.log(fact(5));
```

8) Validaciones con funciones

Enunciado: Escribe `esEmail(str)` que valide patrón simple `@` y `.`. **Resultado esperado**: `true/false` según string. **Ayuda de código**:

```
function esEmail(s) {
  return /.+@.+\..+/.test(s);
}

console.log(esEmail("a@b.com"));
console.log(esEmail("abc"));
```

9) Inmutabilidad ligera

Enunciado: Función `actualizaNota(alumno, nuevaNota)` que devuelva **nuevo** objeto sin mutar el original. **Resultado esperado**: Original intacto. **Ayuda de código**:

```
function actualizaNota(alumno, nuevaNota) {
  return { ...alumno, nota: nuevaNota };
}

const a = { nombre: "Nora", nota: 6 };
const b = actualizaNota(a, 9);

console.log(a);
console.log(b);
```

10) TDD mini (pensar antes de codificar)

Enunciado: Define casos de prueba (como comentarios) para `esPar(n)` y luego impleméntala. **Resultado esperado**: Función que cumple los casos. **Ayuda de código**:

```
// Casos:
// esPar(0) -> true
// esPar(1) -> false
// esPar(-2) -> true

const esPar = (n) => {
  return n % 2 === 0;
};

console.log(esPar(0));
console.log(esPar(1));
console.log(esPar(-2));
```

Grupo 5 — DOM y eventos (10 ejercicios)

1) Seleccionar y modificar contenido

Enunciado: Selecciona un elemento con id `titulo` y cambia su texto a `"Hola DOM"`. **Resultado esperado**: El `<h1 id="titulo">` muestra *Hola DOM*. **Ayuda de código**:

```
<h1 id="titulo">Título original</h1>
<script>
  const h1 = document.getElementById("titulo");
  h1.textContent = "Hola DOM";
</script>
```

2) Crear y añadir nodos

Enunciado: Crea un `<li>` con texto `"Nuevo"` y añádelo a `<ul id="lista">`. **Resultado esperado**: La lista tiene un nuevo elemento al final. **Ayuda de código**:

```
<ul id="lista">
  <li>A</li>
</ul>
<script>
  const ul = document.getElementById("lista");
  const li = document.createElement("li");
  li.textContent = "Nuevo";
  ul.appendChild(li);
</script>
```

3) Clases y estilos

Enunciado: Al pulsar un botón, alterna la clase `activo` en un cuadro. **Resultado esperado**: El cuadro cambia aspecto con cada clic. **Ayuda de código**:

```
<style>
  .box { width: 80px; height: 80px; border: 2px solid #333; }
  .activo { transform: scale(1.1); box-shadow: 0 0 10px #999; }
</style>
<div id="box" class="box"></div>
<button id="btn">Toggle</button>
<script>
  const box = document.getElementById("box");
  const btn = document.getElementById("btn");

  btn.addEventListener("click", () => {
    box.classList.toggle("activo");
  });
</script>
```

4) Delegación de eventos

Enunciado: Detecta cuál `<li>` se pulsó dentro de `<ul id="menu">` usando *delegación*. **Resultado esperado**: Consola imprime el texto del `<li>` pulsado. **Ayuda de código**:

```
<ul id="menu">
  <li>Home</li>
  <li>Productos</li>
  <li>Contacto</li>
</ul>
<script>
  const menu = document.getElementById("menu");

  menu.addEventListener("click", (e) => {
    if (e.target.tagName === "LI") {
      console.log("Click en:", e.target.textContent);
    }
  });
</script>
```

5) Input en vivo (contador de caracteres)

Enunciado: Muestra cuántos caracteres lleva un `<input>`. **Resultado esperado**: Un `<span>` actualiza la cuenta mientras escribes. **Ayuda de código**:

```
<input id="nombre" placeholder="Escribe tu nombre">
<span id="contador">0</span>
<script>
  const input = document.getElementById("nombre");
  const contador = document.getElementById("contador");

  input.addEventListener("input", () => {
    contador.textContent = input.value.length.toString();
```

```
  });
</script>
```

6) `preventDefault` en enlaces

Enunciado: Evita que un enlace navegue y muestra un aviso. **Resultado esperado**: No cambia de página; aparece mensaje en consola. **Ayuda de código**:

```
<a id="enlace" href="https://ejemplo.com">Ir</a>
<script>
  const a = document.getElementById("enlace");

  a.addEventListener("click", (e) => {
    e.preventDefault();
    console.log("Navegación cancelada");
  });
</script>
```

7) Temporizador con `setInterval`

Enunciado: Incrementa y muestra un número cada segundo; botón para parar. **Resultado esperado**: Contador sube 1/s hasta pulsar *Parar*. **Ayuda de código**:

```
<div id="out">0</div>
<button id="parar">Parar</button>
<script>
  const out = document.getElementById("out");
  const btn = document.getElementById("parar");

  let x = 0;
  const id = setInterval(() => {
    x += 1;
    out.textContent = x.toString();
  }, 1000);

  btn.addEventListener("click", () => {
    clearInterval(id);
  });
</script>
```

8) Animación con `requestAnimationFrame`

Enunciado: Mueve un cuadrado de izquierda a derecha suavemente. **Resultado esperado**: Animación fluida del elemento. **Ayuda de código**:

```
<style>
  #wrap { position: relative; height: 100px; border: 1px dashed #aaa; }
```

```
  #sq { position: absolute; width: 30px; height: 30px; background: #4caf50; }
</style>
<div id="wrap">
  <div id="sq"></div>
</div>
<script>
  const sq = document.getElementById("sq");
  let x = 0;

  function tick() {
    x += 2;
    sq.style.transform = `translateX(${x}px)`;
    if (x < 300) {
      requestAnimationFrame(tick);
    }
  }

  requestAnimationFrame(tick);
</script>
```

9) Accesibilidad básica

Enunciado: Un botón alterna mostrar/ocultar un panel y actualiza `aria-expanded`. **Resultado esperado**: Panel visible/oculto y atributo aria coherente. **Ayuda de código**:

```
<button id="toggle" aria-controls="panel" aria-expanded="false">Mostrar</
button>
<div id="panel" hidden>
  Contenido accesible
</div>
<script>
  const btn = document.getElementById("toggle");
  const panel = document.getElementById("panel");

  btn.addEventListener("click", () => {
    const expanded = btn.getAttribute("aria-expanded") === "true";
    btn.setAttribute("aria-expanded", (!expanded).toString());
    panel.hidden = expanded;
  });
</script>
```

10) Propagación y `stopPropagation`

Enunciado: Muestra mensajes al clicar caja y botón interno; evita que el clic del botón burbujee. **Resultado esperado**: Clic en caja -> mensaje caja. Clic en botón -> solo botón. **Ayuda de código**:

```
<div id="caja" style="padding: 20px; border: 1px solid #333;">
  Caja
  <button id="inner">Botón</button>
```

```
</div>
<script>
  document.getElementById("caja").addEventListener("click", () => {
    console.log("Click en caja");
  });

  document.getElementById("inner").addEventListener("click", (e) => {
    e.stopPropagation();
    console.log("Click en botón interno");
  });
</script>
```

Grupo 6 — Formularios y validación (10 ejercicios)

1) Capturar submit

Enunciado: Intercepta el envío de un `<form>` y muestra los valores sin recargar. **Resultado esperado**: Consola con pares `nombre=valor`. **Ayuda de código**:

```
<form id="f">
  <input name="usuario" placeholder="Usuario">
  <input name="email" placeholder="Email">
  <button>Enviar</button>
</form>
<script>
  const form = document.getElementById("f");

  form.addEventListener("submit", (e) => {
    e.preventDefault();
    const data = new FormData(form);
    for (const [k, v] of data.entries()) {
      console.log(k, v);
    }
  });
</script>
```

2) Validación de email

Enunciado: Marca en rojo el input si el email no cumple un patrón básico. **Resultado esperado**: Borde rojo si inválido; verde si válido. **Ayuda de código**:

```
<style>
  .ok { border: 2px solid #4caf50; }
  .err { border: 2px solid #f44336; }
</style>
<input id="em" placeholder="email@dominio.com">
<script>
```

```
  const em = document.getElementById("em");
  const re = /.+@.+\..+/;

  em.addEventListener("input", () => {
    em.classList.toggle("ok", re.test(em.value));
    em.classList.toggle("err", !re.test(em.value));
  });
</script>
```

3) Campos requeridos

Enunciado: Si `usuario` está vacío al enviar, muestra un mensaje bajo el campo. **Resultado esperado**: `span` con error hasta que se rellene. **Ayuda de código**:

```
<input id="user" placeholder="Usuario">
<span id="eUser"></span>
<button id="go">Enviar</button>
<script>
  const user = document.getElementById("user");
  const eUser = document.getElementById("eUser");
  const go = document.getElementById("go");

  go.addEventListener("click", () => {
    if (!user.value.trim()) {
      eUser.textContent = "El usuario es obligatorio";
    } else {
      eUser.textContent = "";
      console.log("OK");
    }
  });
</script>
```

4) Validación en tiempo real

Enunciado: Muestra `min 6` mientras la contraseña tenga menos de 6 caracteres. **Resultado esperado**: Mensaje desaparece al cumplir longitud. **Ayuda de código**:

```
<input id="pwd" type="password" placeholder="Contraseña">
<small id="hpwd"></small>
<script>
  const pwd = document.getElementById("pwd");
  const hpwd = document.getElementById("hpwd");

  pwd.addEventListener("input", () => {
    hpwd.textContent = pwd.value.length < 6 ? "min 6" : "";
  });
</script>
```

5) Reglas compuestas

Enunciado: Contraseña válida si tiene min 8, una mayúscula y un número. **Resultado esperado**: Mensaje `Fuerte` o lista de requisitos faltantes. **Ayuda de código**:

```
<input id="pw2" type="password" placeholder="Contraseña">
<div id="msg"></div>
<script>
  const pw2 = document.getElementById("pw2");
  const msg = document.getElementById("msg");

  function faltas(s) {
    const errs = [];
    if (s.length < 8) errs.push("8+ caracteres");
    if (!/[A-Z]/.test(s)) errs.push("una mayúscula");
    if (!/\d/.test(s)) errs.push("un número");
    return errs;
  }

  pw2.addEventListener("input", () => {
    const e = faltas(pw2.value);
    msg.textContent = e.length ? `Falta: ${e.join(", ")}` : "Fuerte";
  });
</script>
```

6) Checkbox de aceptación

Enunciado: Deshabilita el botón *Enviar* hasta marcar *Acepto términos*. **Resultado esperado**: Botón activo solo con checkbox marcado. **Ayuda de código**:

```
<input id="ok" type="checkbox"> Acepto
<button id="enviar" disabled>Enviar</button>
<script>
  const ok = document.getElementById("ok");
  const enviar = document.getElementById("enviar");

  ok.addEventListener("change", () => {
    enviar.disabled = !ok.checked;
  });
</script>
```

7) `<select>` y dependencias

Enunciado: Al elegir una categoría, carga opciones de subcategoría. **Resultado esperado**: Segundo `<select>` se rellena dinámicamente. **Ayuda de código**:

```
<select id="cat">
  <option>HW</option>
```

```
  <option>SW</option>
</select>
<select id="sub"></select>
<script>
  const mapa = {
    HW: ["CPU", "GPU", "RAM"],
    SW: ["SO", "IDE", "Navegador"],
  };

  const cat = document.getElementById("cat");
  const sub = document.getElementById("sub");

  function cargar() {
    sub.innerHTML = "";
    for (const s of mapa[cat.value]) {
      const opt = document.createElement("option");
      opt.textContent = s;
      sub.appendChild(opt);
    }
  }

  cat.addEventListener("change", cargar);
  cargar();
</script>
```

8) Serializar a objeto

Enunciado: Convierte un formulario en un objeto `{campo: valor}`. **Resultado esperado**: Objeto mostrado por consola. **Ayuda de código**:

```
<form id="login">
  <input name="user" placeholder="User">
  <input name="pass" type="password" placeholder="Pass">
  <button>OK</button>
</form>
<script>
  const login = document.getElementById("login");

  login.addEventListener("submit", (e) => {
    e.preventDefault();
    const fd = new FormData(login);
    const obj = Object.fromEntries(fd.entries());
    console.log(obj);
  });
</script>
```

9) `FormData` + fetch (demo sin backend)

Enunciado: Prepara el `fetch` POST de un formulario (no se enviará realmente). **Resultado esperado**: Muestra el body creado y la configuración. **Ayuda de código**:

```
<form id="reg">
  <input name="name" placeholder="Nombre">
  <input name="email" placeholder="Email">
  <button>Enviar</button>
</form>
<script>
  const reg = document.getElementById("reg");

  reg.addEventListener("submit", async (e) => {
    e.preventDefault();
    const body = JSON.stringify(Object.fromEntries(new FormData(reg)));

    const req = {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body,
    };

    console.log("Listo para enviar a /api/registro:", req);
    // await fetch("/api/registro", req);
  });
</script>
```

10) Auto-guardado en `localStorage`

Enunciado: Guarda el contenido de un textarea cada 500ms y recupéralo al cargar. **Resultado esperado**: El texto persiste entre recargas. **Ayuda de código**:

```
<textarea id="nota" rows="4" cols="30"></textarea>
<script>
  const nota = document.getElementById("nota");
  const KEY = "nota-borrador";

  nota.value = localStorage.getItem(KEY) ?? "";

  let id;
  nota.addEventListener("input", () => {
    clearTimeout(id);
    id = setTimeout(() => {
      localStorage.setItem(KEY, nota.value);
    }, 500);
  });
</script>
```

Grupo 7 — Fetch y async/await (10 ejercicios)

1) GET básico con `async/await`

Enunciado: Pide un recurso JSON y muéstralo en consola. **Resultado esperado**: Objeto impreso o error capturado. **Ayuda de código**:

```
async function getPost() {
  try {
    const res = await fetch("https://jsonplaceholder.typicode.com/posts/1");
    if (!res.ok) throw new Error(res.statusText);
    const data = await res.json();
    console.log(data);
  } catch (e) {
    console.error("Error:", e.message);
  }
}

getPost();
```

2) Manejo de `response.ok`

Enunciado: Si la respuesta no es 2xx, lanza error con código. **Resultado esperado**: Mensaje `HTTP <status>` en errores. **Ayuda de código**:

```
async function load(url) {
  const res = await fetch(url);
  if (!res.ok) {
    throw new Error(`HTTP ${res.status}`);
  }
  return res.json();
}

load("https://jsonplaceholder.typicode.com/posts/
999999").catch(console.error);
```

3) Pintar lista en el DOM

Enunciado: Descarga 5 posts y pinta sus títulos en `<ul id="out">`. **Resultado esperado**: Lista con 5 `<li>`. **Ayuda de código**:

```
<ul id="out"></ul>
<script>
  async function listar() {
    const res = await fetch("https://jsonplaceholder.typicode.com/posts?
_limit=5");
```

```
    const data = await res.json();
    const out = document.getElementById("out");
    out.innerHTML = "";
    for (const p of data) {
      const li = document.createElement("li");
      li.textContent = p.title;
      out.appendChild(li);
    }
  }

  listar();
</script>
```

4) Cancelar petición con `AbortController`

Enunciado: Inicia una petición y permite cancelarla con un botón. **Resultado esperado**: Al cancelar, se muestra mensaje de abort. **Ayuda de código**:

```
<button id="cancel">Cancelar</button>
<script>
  const controller = new AbortController();

  async function demo() {
    try {
      const res = await fetch("https://jsonplaceholder.typicode.com/photos",
{
        signal: controller.signal,
      });
      console.log("OK", await res.json());
    } catch (e) {
      if (e.name === "AbortError") {
        console.log("Petición cancelada");
      } else {
        console.error(e);
      }
    }
  }

  document.getElementById("cancel").onclick = () => controller.abort();
  demo();
</script>
```

5) Paralelizar con `Promise.all`

Enunciado: Pide 3 recursos en paralelo y muestra tiempos. **Resultado esperado**: Array con tres resultados. **Ayuda de código**:

```
async function multi() {
  const urls = [1, 2, 3].map((n) => `https://jsonplaceholder.typicode.com/
```

```
users/${n}`);
  console.time("paralelo");
  const data = await Promise.all(urls.map((u) => fetch(u).then((r) =>
r.json())));
  console.timeEnd("paralelo");
  console.log(data);
}

multi();
```

6) Secuencial con `await`

Enunciado: Descarga 3 recursos uno tras otro y compara tiempos con el anterior. **Resultado esperado**: Tiempos mayores que en paralelo. **Ayuda de código**:

```
async function secuencial() {
  console.time("secuencial");
  const a = await fetch("https://jsonplaceholder.typicode.com/users/
1").then((r) => r.json());
  const b = await fetch("https://jsonplaceholder.typicode.com/users/
2").then((r) => r.json());
  const c = await fetch("https://jsonplaceholder.typicode.com/users/
3").then((r) => r.json());
  console.timeEnd("secuencial");
  console.log([a, b, c]);
}

secuencial();
```

7) Timeout con `Promise.race`

Enunciado: Falla si una petición tarda más de 2s. **Resultado esperado**: Error `Timeout` si excede el tiempo. **Ayuda de código**:

```
function timeout(ms) {
  return new Promise((_, rej) => setTimeout(() => rej(new Error("Timeout")),
ms));
}

async function conTimeout(url, ms = 2000) {
  return Promise.race([fetch(url), timeout(ms)]);
}

conTimeout("https://jsonplaceholder.typicode.com/comments", 2000)
  .then((r) => r.json())
  .then(console.log)
  .catch(console.error);
```

8) Reintentos simples (backoff)

Enunciado: Reintenta una petición hasta 3 veces con espera creciente. **Resultado esperado**: Éxito o error tras 3 intentos. **Ayuda de código**:

```
async function fetchRetry(url, n = 3) {
  let intento = 0;
  while (intento < n) {
    try {
      const res = await fetch(url);
      if (!res.ok) throw new Error("HTTP " + res.status);
      return res.json();
    } catch (e) {
      intento += 1;
      if (intento === n) throw e;
      await new Promise((r) => setTimeout(r, 300 * intento));
    }
  }
}

fetchRetry("https://jsonplaceholder.typicode.com/todos/
1").then(console.log).catch(console.error);
```

9) Cargar imagen con `blob()`

Enunciado: Descarga una imagen y muéstrala en `<img id="pic">`. **Resultado esperado**: Imagen renderizada desde blob URL. **Ayuda de código**:

```
<img id="pic" alt="demo" />
<script>
  async function carga() {
    const res = await fetch("https://via.placeholder.com/150");
    const blob = await res.blob();
    document.getElementById("pic").src = URL.createObjectURL(blob);
  }
  carga();
</script>
```

10) Paginación básica

Enunciado: Botones *Anterior/Siguiente* cambian la página de resultados. **Resultado esperado**: Lista se actualiza al navegar. **Ayuda de código**:

```
<button id="prev">Anterior</button>
<button id="next">Siguiente</button>
<ul id="list"></ul>
<script>
  let page = 1;
```

```
  async function load() {
    const res = await fetch(`https://jsonplaceholder.typicode.com/posts?
_limit=5&_page=${page}`);
    const data = await res.json();
    const ul = document.getElementById("list");
    ul.innerHTML = "";
    for (const p of data) {
      const li = document.createElement("li");
      li.textContent = `${p.id}. ${p.title}`;
      ul.appendChild(li);
    }
  }

  document.getElementById("prev").onclick = () => {
    if (page > 1) {
      page -= 1;
      load();
    }
  };

  document.getElementById("next").onclick = () => {
    page += 1;
    load();
  };

  load();
</script>
```

Grupo 8 — Módulos y Storage (10 ejercicios)

1) Export/Import básicos

Enunciado: Crea un módulo `maths.js` con `suma` y úsalo en `main.js`. **Resultado esperado**: Consola muestra el resultado de `suma(2,3)`. **Ayuda de código**:

```
// maths.js
export function suma(a, b) {
  return a + b;
}
```

```
<!-- index.html -->
<script type="module">
  import { suma } from "./maths.js";
  console.log(suma(2, 3));
</script>
```

2) `export default` vs nombrados

Enunciado: Define `export default` para `resta` y nombrado para `mul`. **Resultado esperado**: Importación mixta funcionando. **Ayuda de código**:

```
// ops.js
export default function resta(a, b) {
  return a - b;
}
export function mul(a, b) {
  return a * b;
}
```

```
<script type="module">
  import resta, { mul } from "./ops.js";
  console.log(resta(5, 2));
  console.log(mul(3, 4));
</script>
```

3) Barrel (re-export)

Enunciado: Reexporta todo desde `index.js` y centraliza importaciones. **Resultado esperado**: Importas desde una sola ruta. **Ayuda de código**:

```
// maths.js
export const pi = 3.14;
export const sq = (x) => x * x;
```

```
// index.js
export * from "./maths.js";
```

```
<script type="module">
  import { pi, sq } from "./index.js";
  console.log(pi, sq(3));
</script>
```

4) `localStorage` JSON

Enunciado: Guarda un objeto usuario y recupéralo al recargar. **Resultado esperado**: Objeto mostrado correctamente. **Ayuda de código**:

```
<script>
  const key = "user";
  const user = { nombre: "Ana", rol: "SMR" };
```

```
  localStorage.setItem(key, JSON.stringify(user));
  const loaded = JSON.parse(localStorage.getItem(key) ?? "null");
  console.log(loaded);
</script>
```

5) `sessionStorage`

Enunciado: Cuenta cuántas veces se ha recargado la página en esta sesión. **Resultado esperado**: Número aumenta en cada refresh. **Ayuda de código**:

```
<script>
  const KEY = "reloads";
  const n = Number(sessionStorage.getItem(KEY) ?? 0) + 1;
  sessionStorage.setItem(KEY, String(n));
  console.log("Recargas sesión:", n);
</script>
```

6) Evento `storage` entre pestañas

Enunciado: Escucha cambios en `localStorage` desde otra pestaña. **Resultado esperado**: Consola muestra cambios detectados. **Ayuda de código**:

```
<script>
  window.addEventListener("storage", (e) => {
    console.log("Storage cambiado:", e.key, e.newValue);
  });
</script>
```

7) Patrón módulo (IIFE)

Enunciado: Crea un contador con estado interno y métodos `inc/get`. **Resultado esperado**: `get()` devuelve 1, luego 2, etc. **Ayuda de código**:

```
const Counter = (() => {
  let x = 0;
  return {
    inc() {
      x += 1;
    },
    get() {
      return x;
    },
  };
})();
```

```
Counter.inc();
console.log(Counter.get());
```

8) Tema oscuro persistente

Enunciado: Guarda elección de tema y aplícala al cargar. **Resultado esperado**: Alterna `dark` y recuerda preferencia. **Ayuda de código**:

```
<button id="theme">Toggle tema</button>
<script>
  const KEY = "theme";

  function apply() {
    document.documentElement.classList.toggle(
      "dark",
      localStorage.getItem(KEY) === "dark",
    );
  }

  apply();

  document.getElementById("theme").onclick = () => {
    const next = localStorage.getItem(KEY) === "dark" ? "light" : "dark";
    localStorage.setItem(KEY, next);
    apply();
  };
</script>
```

9) `import()` dinámico

Enunciado: Carga un módulo solo cuando se pulsa un botón. **Resultado esperado**: Función del módulo usada tras la carga. **Ayuda de código**:

```
<button id="load">Cargar util</button>
<script type="module">
  document.getElementById("load").onclick = async () => {
    const mod = await import("./util.js");
    console.log(mod.hola());
  };
</script>
```

```
// util.js
export function hola() {
  return "Hola desde util";
}
```

10) Validador reutilizable como módulo

Enunciado: Exporta `esEmail` y úsalo en un formulario. **Resultado esperado**: Borde verde si el email es válido. **Ayuda de código**:

```
// validator.js
export const esEmail = (s) => /.+@.+\..+/.test(s);
```

```
<input id="mail" placeholder="email">
<script type="module">
  import { esEmail } from "./validator.js";
  const el = document.getElementById("mail");
  el.addEventListener("input", () => {
    el.style.border = esEmail(el.value) ? "2px solid green" : "2px solid
red";
  });
</script>
```

Grupo 9 — Errores y pruebas (10 ejercicios)

1) `try/catch/finally`

Enunciado: Captura un error y ejecuta `finally` siempre. **Resultado esperado**: Consola muestra *catch* y *finally*. **Ayuda de código**:

```
try {
  JSON.parse("{");
} catch (e) {
  console.error("Error parseando:", e.message);
} finally {
  console.log("Siempre se ejecuta");
}
```

2) Lanzar errores personalizados

Enunciado: Función `divide(a,b)` lanza si `b===0`. **Resultado esperado**: Error `División por cero`. **Ayuda de código**:

```
function divide(a, b) {
  if (b === 0) throw new Error("División por cero");
  return a / b;
}

try {
  console.log(divide(10, 0));
```

```
} catch (e) {
  console.error(e.message);
}
```

3) Validar tipos con `TypeError`

Enunciado: `suma(a,b)` debe recibir números o lanzar `TypeError`. **Resultado esperado**: Lanza con mensaje claro al fallar. **Ayuda de código**:

```
function suma(a, b) {
  if (typeof a !== "number" || typeof b !== "number") {
    throw new TypeError("suma requiere números");
  }
  return a + b;
}
```

4) Rechazos de promesas

Enunciado: Captura errores de una `Promise` con `await` dentro de `try/catch`. **Resultado esperado**: Mensaje de error controlado. **Ayuda de código**:

```
async function falla() {
  return Promise.reject(new Error("ups"));
}

(async () => {
  try {
    await falla();
  } catch (e) {
    console.log("capturado", e.message);
  }
})();
```

5) `window.onerror` básico

Enunciado: Registra errores globales en la consola. **Resultado esperado**: Al producir un error, se intercepta el evento. **Ayuda de código**:

```
<script>
  window.onerror = (msg, src, line, col, err) => {
    console.log("Global error:", msg, "en", src, line, col);
    return true; // evita el log por defecto (opcional)
  };
  // Fuerza un error
  setTimeout(() => noExiste(), 0);
</script>
```

6) `assert` casero

Enunciado: Implementa `assert(cond, msg)` y úsalo. **Resultado esperado**: Lanza si la condición es falsa. **Ayuda de código**:

```
function assert(cond, msg = "Assert failed") {
  if (!cond) throw new Error(msg);
}

assert(2 + 2 === 4, "Math roto");
```

7) Mini framework de pruebas

Enunciado: Escribe `test(nombre, fn)` que muestre ✔/❌ **Resultado esperado**: Reporte de 3 casos sobre `esPar`. **Ayuda de código**:

```
function test(nombre, fn) {
  try {
    fn();
    console.log("✔", nombre);
  } catch (e) {
    console.error("❌", nombre, "-", e.message);
  }
}

function assert(cond, msg) {
  if (!cond) throw new Error(msg);
}

const esPar = (n) => n % 2 === 0;

test("0 es par", () => assert(esPar(0)));

test("1 no es par", () => assert(!esPar(1)));

test("-2 es par", () => assert(esPar(-2)));
```

8) Medir rendimiento

Enunciado: Compara tiempo de dos funciones con `console.time`. **Resultado esperado**: Tiempos mostrados. **Ayuda de código**:

```
function a() {
  let s = 0;
  for (let i = 0; i < 1e6; i++) s += i;
  return s;
}
```

```
function b() {
  return Array.from({ length: 1e6 }, (_, i) => i).reduce((x, y) => x + y, 0);
}

console.time("a");
a();
console.timeEnd("a");

console.time("b");
b();
console.timeEnd("b");
```

9) Programación defensiva

Enunciado: Accede de forma segura a `user.profile.email`. **Resultado esperado**: Usa `?.` y `??` para valores por defecto. **Ayuda de código**:

```
const user = {};
const email = user.profile?.email ?? "sin correo";
console.log(email);
```

10) `debugger;`

Enunciado: Inserta `debugger;` en una función y observa el *pause* en DevTools. **Resultado esperado**: La ejecución se detiene permitiendo inspección. **Ayuda de código**:

```
function calc(n) {
  debugger; // Abre DevTools y verás la pausa aquí
  return n * 2;
}

calc(5);
```

Grupo 10 — Mini-proyectos (10 ejercicios)

1) To-Do mínimo

Enunciado: Añade tareas a una lista; al pulsar una tarea se marca como hecha. **Resultado esperado**: Tareas con clase `done` al hacer clic. **Ayuda de código**:

```
<input id="t" placeholder="Nueva tarea">
<button id="add">Añadir</button>
<ul id="todos"></ul>
<style>
  .done { text-decoration: line-through; opacity: 0.6; }
```

```
</style>
<script>
  const t = document.getElementById("t");
  const add = document.getElementById("add");
  const ul = document.getElementById("todos");

  add.onclick = () => {
    if (!t.value.trim()) return;
    const li = document.createElement("li");
    li.textContent = t.value;
    li.onclick = () => li.classList.toggle("done");
    ul.appendChild(li);
    t.value = "";
  };
</script>
```

2) Reloj digital

Enunciado: Muestra la hora actual HH\:MM\:SS actualizada cada segundo. **Resultado esperado**: Texto cambia cada 1s. **Ayuda de código**:

```
<div id="clock"></div>
<script>
  function pad(n) { return String(n).padStart(2, "0"); }
  function tick() {
    const d = new Date();
    clock.textContent = `${pad(d.getHours())}:${pad(d.getMinutes())}:$
{pad(d.getSeconds())}`;
  }
  setInterval(tick, 1000);
  tick();
</script>
```

3) Pomodoro simple (25/5)

Enunciado: Temporizador 25 minutos trabajo / 5 descanso. **Resultado esperado**: Cuenta atrás que alterna estados. **Ayuda de código**:

```
<div id="timer"></div>
<script>
  let work = true;
  let secs = 25 * 60;

  function tick() {
    timer.textContent = (work ? "Trabajo" : "Descanso") +
      `: ${Math.floor(secs/60)}:${String(secs%60).padStart(2,"0")}`;
    secs -= 1;
    if (secs < 0) {
      work = !work;
```

```
      secs = work ? 25 * 60 : 5 * 60;
    }
  }

  setInterval(tick, 1000);
  tick();
</script>
```

4) Buscador en vivo

Enunciado: Filtra una lista según el texto tecleado. **Resultado esperado**: Solo se muestran elementos que coinciden. **Ayuda de código**:

```
<input id="q" placeholder="Buscar">
<ul id="items">
  <li>CPU</li>
  <li>GPU</li>
  <li>RAM</li>
  <li>SSD</li>
</ul>
<script>
  const q = document.getElementById("q");
  const items = Array.from(document.querySelectorAll("#items li"));

  q.addEventListener("input", () => {
    const s = q.value.toLowerCase();
    for (const li of items) {
      li.style.display = li.textContent.toLowerCase().includes(s) ? "" :
"none";
    }
  });
</script>
```

5) Galería con lightbox básico

Enunciado: Al clicar miniatura se abre imagen grande en overlay. **Re