Practico 1 ✍️

Paradigmas de Programación 💻

Facundo Guzmán - Ing. en Sistemas de Información

JavaScript:

> 💡 **Considera el lenguaje JavaScript acotado al paradigma de programación estructurada y analízalo en términos de los cuatro componentes de un paradigma mencionados por Kuhn.**

> 🤔 **¿Cuáles son las reglas escritas del lenguaje?**

- **Identificadores:** nombres para variables y funciones. Deben comenzar con letra, `$` o `_`, y pueden contener letras, dígitos o `_`. Es sensible a mayúsculas y minúsculas.
- **Variables:** se declaran con `var`, `let` o `const`.
- **Funciones:** bloques de código reutilizables definidos con `function nombre(parámetros){...}`.
- **Bloques de código:** delimitados por `{}`.
- **Comentarios:**
  - Una línea: `// comentario`

    - Multilínea: `/* comentario */`
- **Palabras reservadas:** `if`, `else`, `while`, `for`, `switch`, `return`, `break`, `continue`, `function`, `let`, `const`, entre otras.
- **Tipos de datos:**
    - Primitivos: `number`, `string`, `boolean`, `undefined`, `null`, `bigint`, `symbol`.
    - Objetos: aunque existen, se restringe el uso a lo mínimo para mantener el paradigma estructurado.
- **Operadores:** aritméticos (`+ - * / %`), lógicos (`&& || !`), relacionales (`== != === !== > < >= <=`), de asignación (`= += -= *= /=`), condicional ternario (`cond ? a : b`).
- **Control de flujo:**
    - Condicionales: `if`, `if-else`, `switch`.
    - Bucles: `for`, `while`, `do...while`.
    - Sentencias de salto: `break`, `continue`, `return`.
- **Entrada/salida:** a través de `console.log` o funciones del entorno (navegador, Node.js).

⌨️ ¿Qué características particulares del lenguaje se cree que sean "mejores" que en otros lenguajes?

- **Universalidad y accesibilidad:** JavaScript se ejecuta en cualquier navegador sin instalación adicional, lo que lo hace omnipresente en aplicaciones web.
- **Facilidad de uso:** al no ser tipado estrictamente, resulta más rápido para prototipar que C o Java.
- **Integración inmediata:** se cree mejor que otros porque se combina naturalmente con HTML y CSS para desarrollo frontEnd.
- **Interactividad:** permite construir interfaces dinámicas de manera sencilla.

- **Curva de aprendizaje baja:** muchos desarrolladores lo eligen porque es fácil empezar a programar con él.
- **Flexibilidad:** se valora que un mismo programa pueda ejecutarse tanto en el navegador como en entornos de servidor (Node.js).
- **Ejecución inmediata:** al no requerir compilación, se pueden probar cambios rápidamente.

💡 **Considera el lenguaje JavaScript acotado al paradigma de programación estructurada y analízalo en términos de los ejes propuestos para la elección de un lenguaje de programación y responde:**

📚 **¿Tiene una sintaxis y una semántica bien definida? ¿Existe documentación oficial?**

▼ ***Sintaxis**:*

Sí, JavaScript tiene una sintaxis clara y estandarizada. Se basa en **bloques** `{}`**, punto y coma (opcional en algunos casos), palabras clave (** `if` **,** `for` **,** `function` **, etc.)**.

▼ ***Semántica**:*

Está definida en la **especificación ECMAScript (ECMA-262)**, que determina el comportamiento de cada instrucción.

▼ **Documentación**:

- Sí, el estándar ECMA-262 es la referencia principal. Además, existe documentación reconocida (MDN Web Docs de Mozilla, W3C, WHATWG).

💻 ¿Es posible comprobar el código producido en ese lenguaje?

- Sí, se puede comprobar directamente **ejecutándolo en navegadores o en Node.js**.
- Existen herramientas que detectan errores de sintaxis y malas prácticas, como **linters (ESLint, JSHint)**.
- La verificación se hace por **intérprete** y pruebas en tiempo de ejecución.

✅ ¿Es confiable?

JavaScript tiene algunas limitaciones en cuanto a confiabilidad

▼ **Pros**:

- Tiene manejo automático de memoria (recolección de basura), lo que evita fugas comunes de lenguajes como C.
- Maneja errores con excepciones (`try...catch`).

▼ **Contras**:

- Tipado dinámico: puede generar errores difíciles de detectar (ej. `5 + "5"` → `"55"`).
- El intérprete no impide operaciones peligrosas hasta tiempo de ejecución.

❓ ¿Es ortogonal?

no es completamente ortogonal:

- Muchas construcciones pueden combinarse de forma flexible (ej. funciones dentro de condicionales, bucles con distintos tipos de expresiones).
- Sin embargo, hay **excepciones**:
  - El valor de `null` y `undefined` no se comporta igual en todas las operaciones.
  - Conversión implícita de tipos (coerción) rompe la ortogonalidad.

👨‍🎓 ¿Cuáles son sus características de consistencia y uniformidad?

▼ **Características de consistencia y uniformidad en JavasCript**

- Uso uniforme de `{}` para bloques (`if`, `for`, `while`, `function`).
- Todas las estructuras de control (`if`, `switch`, `for`, `while`) usan paréntesis para condiciones.
- Operadores aritméticos y lógicos funcionan igual sobre los tipos válidos.

▼ **Inconsistencias:**

- No requiere `;` siempre (inserción automática de punto y coma → puede dar errores confusos).
- Diferencias entre `==` y `===`.
- El manejo de `NaN`, `null` y `undefined` es inconsistente en algunos contextos.

📖 ¿Es extensible? ¿Hay subconjuntos de ese lenguaje?

- Sí es extensible, se pueden añadir nuevas funcionalidades mediante librerías, APIs del navegador, o frameworks.
- **Subconjuntos:**
  - **Strict Mode (`"use strict"`)**: restringe algunas prácticas para mayor seguridad.
  - **TypeScript** (superset con tipado estático, muy usado en la industria).
  - También existen subconjuntos para entornos embebidos o motores reducidos (ej. Espruino para microcontroladores).

🚗 El código producido, ¿es transportable?

Sí, con condiciones:

- El mismo código JavaScript suele correr en **todos los navegadores modernos y en Node.js**, siempre que se respeten estándares ECMAScript.
- **Limitaciones:** algunas APIs son específicas de entornos (ej. `document` en navegador no existe en Node.js).

  👉 En general, el **código estructurado puro (sin dependencias del entorno)** es altamente transportable.