**Aplicando lo aprendido 1** - Juan Pablo Agüero

# **Ejercicio 1**

## **Parte A**

Considera el lenguaje JavaScript acotado al paradigma de programación estructurada y analízalo en términos de [los cuatro componentes de un paradigma](https://www.notion.so/Paradigmas-lenguajes-y-programas-f36d432c55274b93913dc289446f424d?pvs=21) mencionados por Kuhn.

1. Generalización simbólica: ¿Cuáles son las reglas escritas del lenguaje?
2. Creencias de los profesionales: ¿Qué características particulares del lenguaje se cree que sean "mejores" que en otros lenguajes?
3. Valores: ¿Qué pensamiento o estilo de programación consideraron mejor los creadores?
4. Ejemplares: ¿Qué clase de problemas pueden resolverse más fácilmente en el lenguaje?

### **1. Generalización simbólica (Reglas escritas del lenguaje)**

En JavaScript estructurado incluyen:

* **Sintaxis básica**: variables (let, const, var), operadores aritméticos y lógicos, expresiones.
* **Control de flujo**: if/else, switch, for, while, do...while.
* **Funciones**: declaración (function nombre(...) { ... }) y uso como bloques de código reutilizables.
* **Tipado dinámico**: no es necesario declarar el tipo de una variable.
* **Ámbito léxico**: variables con alcance de bloque o función.

### **2. Creencias de los profesionales (Qué creen que tiene de mejor)**

En la comunidad se suele destacar que JavaScript, incluso de forma estructurada:

* Es **simple y flexible**: no exige tipos rígidos ni estructuras muy formales.
* Es **interpretado y multiplataforma**: corre en cualquier navegador sin necesidad de compilación.
* Es **fácil de aprender** para principiantes, en comparación con C o Java.
* Tiene una **comunidad y ecosistema amplísimo**, con abundante documentación y soporte.

### **3. Valores (Estilo de programación privilegiado por sus creadores)**

En su origen (Brendan Eich, 1995), JavaScript fue diseñado para:

* Favorecer la **sencillez y accesibilidad** para programadores no expertos.
* Impulsar un estilo **imperativo y estructurado**, con instrucciones secuenciales y funciones pequeñas.
* Priorizó la **rapidez en el desarrollo** sobre la rigurosidad tipada.
* En el contexto de los 90, se buscó **facilitar scripts cortos y claros** que manipulen páginas web sin necesidad de un lenguaje complejo como Java o C++.

### **4. Ejemplares (Qué problemas resuelve más fácilmente)**

Con JavaScript estructurado se pueden resolver bien problemas como:

* **Automatización de interacciones simples** en páginas web (validación de formularios, menús desplegables).
* **Manipulación básica del DOM** (mostrar/ocultar elementos, animaciones simples).
* **Procesamiento de datos sencillo**: cálculos matemáticos, conversión de datos, operaciones con cadenas.
* **Prototipos rápidos** de algoritmos o simulaciones por su sintaxis ligera.

## **Parte B**

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 ([¿Cómo elegir un lenguaje?](https://www.notion.so/C-mo-elegir-un-lenguaje-818a7dd066514c119d8a97857443ed4c?pvs=21) ) y responde:

1. ¿Tiene una sintaxis y una semántica bien definida? ¿Existe documentación oficial?
2. ¿Es posible comprobar el código producido en ese lenguaje?
3. ¿Es confiable?
4. ¿Es ortogonal?
5. ¿Cuáles son sus características de consistencia y uniformidad?
6. ¿Es extensible? ¿Hay subconjuntos de ese lenguaje?
7. El código producido, ¿es transportable?

### **1.** ¿Tiene una sintaxis y una semántica bien definida? ¿Existe documentación oficial? (**Sintaxis y semántica bien definida**)

* Sí, la sintaxis y semántica de JavaScript están **formalizadas en el estándar ECMAScript** (ECMA-262).
* Existe **documentación oficial** (ECMA y Mozilla MDN como referencia principal).

### **2.** ¿Es posible comprobar el código producido en ese lenguaje? (**Comprobabilidad**)

* Sí, el código puede **ejecutarse directamente en navegadores** o entornos como Node.js.
* No existe un compilador estricto, pero hay **herramientas de análisis estático** (linters, TypeScript) que permiten detectar errores antes de ejecutar.

### **3.** ¿Es confiable?(**Confiabilidad**)

* Es confiable en cuanto a estabilidad: lleva más de 25 años en uso.
* Sin embargo, su **tipado dinámico** puede generar errores en tiempo de ejecución si no se tiene cuidado.
* La confiabilidad aumenta usando buenas prácticas o extensiones como TypeScript.

### **4.** ¿Es ortogonal?(**Ortogonalidad**)

* **No del todo.**
* La ortogonalidad implica que un conjunto reducido de conceptos se combine de forma coherente en todas partes.
* En JavaScript hay excepciones y particularidades (por ejemplo: diferencias entre null y undefined, coerción de tipos, el uso especial de this), lo que le resta ortogonalidad.

### **5.** Características de consistencia y uniformidad (**Consistencia y Uniformidad**)

* **Consistencia parcial**: las estructuras de control (if, for, while) siguen reglas conocidas y uniformes.
* **Inconsistencias notorias**:
  + Diferencia entre == y ===.
  + El manejo irregular de valores falsy (0, null, undefined, "").
* Esto afecta la uniformidad del lenguaje.

### **6.** ¿Es extensible? ¿Hay subconjuntos del lenguaje? (**Extensibilidad y Subconjuntos**)

* Sí, es extensible mediante **librerías y frameworks** (p. ej., jQuery, React, D3).
* Existen **subconjuntos prácticos** como *"JavaScript estricto"* ("use strict") y también *subset languages* como TypeScript (que compila a JS).

### **7.** El código producido, ¿es transportable? (**Transportabilidad**)

* Sí, el código estructurado en JavaScript es **altamente portable**, ya que corre en cualquier navegador moderno y en múltiples entornos (Node.js, Deno).
* No depende del sistema operativo, sino de la presencia de un motor JavaScript.