TP DE PdeP N°1

Alumno:Ernesto Godoy

RESPUESTAS

EJERCICIO 1

Parte A

1. Generalización simbólica

(Equivale a las reglas formales del lenguaje, es decir, su sintaxis, semántica y estructuras básicas.)

Bajo el paradigma estructurado, JavaScript puede analizarse en términos de:

Sintaxis básica:

Uso de estructuras de control como if, else, while, for, switch, break, continue, etc.

Declaración de funciones con function, que permite modularizar el código.

Variables declaradas (clásicamente) con var, y más modernamente con let y const.

Bloques de código:

Estructuras delimitadas por llaves {}, que indican el inicio y fin de bloques, fundamentales en la programación estructurada.

Flujo secuencial y controlado:

El código sigue una lógica de ejecución de instrucciones de forma lineal, con bifurcaciones (if, else) y repeticiones (for, while) bien definidas.

Evita el uso del goto:

Aunque JavaScript no tiene un goto como tal, su ausencia respalda el paradigma estructurado, donde se promueve un flujo de control claro y predecible.

Resumen: La generalización simbólica en JavaScript estructurado se basa en reglas sintácticas que promueven un control de flujo claro mediante funciones, condicionales y bucles, respetando la lógica secuencial del paradigma estructurado.

2. Creencias de los profesionales

(Equivale a las preferencias y suposiciones compartidas sobre lo que hace "bueno" al lenguaje o paradigma.)

Dentro del uso estructurado de JavaScript, los desarrolladores suelen compartir algunas creencias comunes:

Flexibilidad sintáctica:

\*Muchos creen que JavaScript es más "ligero" y accesible que otros lenguajes como Java o C, especialmente para principiantes o desarrollo web rápido.

Facilidad para estructurar código modular:

\*Aunque JavaScript permite programación funcional y orientada a objetos, muchos aprecian su capacidad para organizar código en funciones simples y reutilizables, siguiendo la lógica estructurada.

Rapidez de prototipado:

\*Se cree que JavaScript permite desarrollar soluciones funcionales de forma rápida gracias a su sintaxis poco rígida y a la ejecución inmediata en navegadores.

Interactividad directa con el entorno web:

\*Aunque no es exclusivo del paradigma estructurado, muchos valoran cómo JavaScript permite estructurar lógica de control que interactúa directamente con el DOM (Document Object Model), usando funciones y estructuras condicionales/bucles de manera clara.

Legibilidad y mantenibilidad:

\*Desde la perspectiva estructurada, se cree que el uso de funciones pequeñas y control de flujo explícito mejora la legibilidad del código frente a soluciones altamente orientadas a objetos o basadas en callbacks anidados.

Resumen: Los profesionales creen que JavaScript estructurado es accesible, fácil de modularizar, ideal para prototipado rápido y bien adaptado a tareas web interactivas, lo que lo hace "mejor" en contextos donde se busca simplicidad y rapidez sin mucha carga estructural.

PARTE B

1. ¿Tiene una sintaxis y una semántica bien definida? ¿Existe documentación oficial?

✅ Sí.

Sintaxis: JavaScript tiene una sintaxis formalmente definida en el estándar ECMAScript, cuya especificación es mantenida por ECMA International (ECMA-262).

Semántica: También está especificada formalmente, aunque con algunos matices (por ejemplo, el hoisting y el type coercion pueden ser contraintuitivos si no se entienden bien).

Documentación oficial: Existe. El estándar ECMAScript es oficial, y además existe documentación extensa y mantenida por la comunidad, como:

MDN Web Docs

ECMAScript Language Specification

2. ¿Es posible comprobar el código producido en ese lenguaje?

✅ Sí, en términos generales.

Aunque JavaScript no es un lenguaje compilado de forma estricta, puede comprobarse con:

Linter (e.g., ESLint) para verificar errores de estilo y sintaxis.

Herramientas de análisis estático para encontrar errores comunes.

Type checking con herramientas como TypeScript.

No obstante, la comprobación está limitada debido a que JavaScript es dinámicamente tipado.

3. ¿Es confiable?

🟡 Moderadamente.

JavaScript puede ser confiable si se emplean buenas prácticas, pero:

Al ser dinámico y con tipado débil, es propenso a errores difíciles de detectar.

La estructura del lenguaje permite escribir código poco claro o ambiguo si no se controla.

En un estilo estructurado (sin abusar de funciones anidadas o asincronía), puede seR razonablemente confiable.

4. ¿Es ortogonal?

🔴 No del todo.

Ortogonalidad implica que las construcciones del lenguaje se puedan combinar libremente sin interferencias.

En JavaScript:

Hay muchas excepciones y reglas implícitas (e.g., coerción de tipos, diferencias entre == y ===, el comportamiento del this incluso en código estructurado).

No todos los tipos y estructuras combinan de forma predecible.

Su diseño histórico ha acumulado inconsistencias que le restan ortogonalidad.

5. ¿Cuáles son sus características de consistencia y uniformidad?

🟡 Parcialmente consistente.

En su núcleo estructurado (sentencias if, for, while, bloques {}, etc.), es consistente y fácil de seguir.

Sin embargo:

Hay inconsistencias históricas (e.g., funciones que retornan undefined si no tienen return, coerciones automáticas, comportamiento de null y undefined).

Algunas estructuras tienen sintaxis redundante o poco intuitiva (e.g., var vs let vs const).

6. ¿Es extensible? ¿Hay subconjuntos de ese lenguaje?

✅ Sí.

Extensible: Se pueden crear bibliotecas, módulos y, con ciertas herramientas (como Babel), incluso nuevos dialectos o transpilaciones.

Subconjuntos:

Se puede usar un subconjunto estructurado del lenguaje (evitando clases, promesas, async/await, etc.).

Algunos entornos o herramientas usan subconjuntos como "Strict Mode" ('use strict'), que eliminan ciertas ambigüedades.

Herramientas como ESLint pueden restringir el uso a un subconjunto definido por reglas.

7. El código producido, ¿es transportable?

✅ Sí.

JavaScript es uno de los lenguajes más portables actualmente.

Funciona en:

Navegadores (de forma casi universal).

Servidores (Node.js).

Dispositivos móviles (React Native).

Embebidos (e.g., Espruino).

Para código estructurado en particular, es aún más portable porque evita dependencias complejas del entorno.