PARTE A

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

1. Generalización simbólica: ¿Cuáles son las reglas escritas del lenguaje?

En el contexto de la programación estructurada en JavaScript, las reglas o "generalizaciones simbólicas" se refieren a las estructuras de control y las convenciones del lenguaje que se deben seguir para que el código sea ejecutable y coherente. En JavaScript estructurado, el enfoque está en el uso de las estructuras de control como:

Secuencialidad: El flujo de ejecución sigue una secuencia de instrucciones lineales, donde cada instrucción se ejecuta de manera secuencial, uno después del otro.

Condicionales: Las estructuras como if, else y switch permiten bifurcar el flujo de control según ciertas condiciones.

Bucles: for, while y do...while son fundamentales para repetir acciones de manera controlada.

Funciones: Las funciones en JavaScript permiten la modularización del código. Las funciones reciben argumentos, ejecutan una secuencia de instrucciones y devuelven un resultado.

Ámbito: El concepto de alcance o "scope" en JavaScript permite que las variables sean accesibles solo dentro de un contexto determinado, lo cual es crucial para el diseño estructurado de programas.

Estas reglas constituyen el conjunto básico de "sintaxis" y "semántica" que conforman la base de la programación estructurada en JavaScript. La programación estructurada rechaza las prácticas que implican saltos incontrolados en el flujo del programa (como el uso de goto en lenguajes más antiguos).

2. Creencias de los profesionales: ¿Qué características particulares del lenguaje se cree que sean "mejores" que en otros lenguajes?

Desde la perspectiva de los profesionales que trabajan con JavaScript, algunas características particulares que se consideran "mejores" en comparación con otros lenguajes son:

Sintaxis accesible y flexible: JavaScript tiene una sintaxis relativamente simple y flexible, lo que lo hace accesible para programadores novatos sin sacrificar la potencia de la programación estructurada. Los desarrolladores suelen apreciar la facilidad con la que se pueden escribir secuencias de código claras y directas.

Ambiente de ejecución dinámico: A diferencia de lenguajes de compilación más rígidos, JavaScript es interpretado y ejecutado en tiempo real, lo que lo hace muy adecuado para aplicaciones web interactivas y el desarrollo rápido de prototipos.

Interoperabilidad con HTML/CSS: La capacidad de integrarse estrechamente con HTML y CSS en la creación de aplicaciones web es vista como una gran ventaja. Esto permite manipular directamente la estructura del DOM (Document Object Model) y manejar eventos de usuario de manera eficiente.

Librerías y frameworks: Aunque esto es más relevante para la programación orientada a objetos, los desarrolladores estructurados también valoran el ecosistema de librerías y frameworks como jQuery, que permiten una programación más eficiente sin abandonar las bases de la programación estructurada.

Acceso a múltiples plataformas: JavaScript, al ser el lenguaje dominante para el desarrollo web, se cree que tiene una ventaja frente a otros lenguajes al ser soportado en casi todos los navegadores, y, más recientemente, en entornos de servidor (como Node.js).

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 (Sintaxis y semántica bien definidas, Comprobabilidad, Confiabilidad, Ortogonalidad, Generalidad, Consistencia y Uniformidad, Subconjuntos, Extensibilidad, Transportabilidad) y responde:

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

Sí, JavaScript tiene una sintaxis y semántica bien definidas, aunque la evolución del lenguaje ha agregado complejidad con el tiempo. Su sintaxis es bastante consistente, basada en el uso de palabras clave y estructuras conocidas como condicionales, bucles y funciones. La semántica está definida en el estándar ECMAScript, el cual describe de manera detallada cómo se comporta el lenguaje en diferentes contextos.

Existen varias fuentes de documentación oficial, siendo la principal el ECMAScript Language Specification, que detalla la sintaxis y semántica de JavaScript. Además, recursos como MDN Web Docs proporcionan documentación accesible y actualizada.

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

Sí, es posible comprobar el código en JavaScript mediante herramientas de análisis estático y pruebas unitarias. Herramientas como JSHint, ESLint y Prettier permiten verificar la calidad y conformidad del código con las reglas del lenguaje, ayudando a detectar errores de sintaxis, estilo y posibles problemas de lógica.

Además, las pruebas unitarias con frameworks como Jest, Mocha y Chai permiten comprobar el comportamiento del código de manera efectiva. Sin embargo, debido a su naturaleza interpretada y la falta de un sistema de tipos estáticos (como en otros lenguajes), JavaScript no permite una comprobación exhaustiva de errores en tiempo de compilación.

3. ¿Es confiable?

JavaScript es generalmente confiable en su funcionamiento, pero su naturaleza dinámica (tipos de datos débiles y evaluación en tiempo de ejecución) puede generar incertidumbre si no se maneja con cuidado. Aunque los navegadores modernos y entornos como Node.js proporcionan una plataforma sólida, el código JavaScript puede ser propenso a errores si no se siguen buenas prácticas, especialmente en aplicaciones grandes o en entornos con baja visibilidad de errores.

La confiabilidad también depende de las herramientas que se usen para asegurar la calidad del código, como pruebas automatizadas, linters y la aplicación de patrones de diseño adecuados

4. ¿Es ortogonal?

En términos de programación estructurada, JavaScript es parcialmente ortogonal. Si bien las estructuras de control como los bucles, condicionales y funciones son independientes entre sí, JavaScript no es completamente ortogonal porque existen interacciones y dependencias entre algunos de sus componentes. Por ejemplo, el manejo de variables globales y locales o la forma en que se manejan los tipos de datos dinámicamente no es tan completamente independiente.

La ortogonalidad, en este contexto, se refiere a la ausencia de dependencias entre las características del lenguaje. En JavaScript, algunas características como los tipos dinámicos y la conversión implícita de tipos pueden generar comportamientos no deseados, lo que reduce su ortogonalidad.

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

JavaScript tiene una consistencia moderada, pero debido a su evolución, algunos aspectos del lenguaje presentan inconsistencias. Por ejemplo, la manera en que se manejan los valores null y undefined, o el comportamiento de == (comparación no estricta) frente a === (comparación estricta), genera confusión a menudo. Sin embargo, ECMAScript ha ido corrigiendo y estandarizando muchas de estas inconsistencias con cada nueva versión.

En cuanto a uniformidad, JavaScript sigue principios bastante uniformes en su mayoría, especialmente en la sintaxis básica (funciones, condicionales, bucles). A pesar de las diferencias en su evolución, se busca una coherencia al agregar nuevas funcionalidades al lenguaje.

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

Sí, JavaScript es un lenguaje extensible. Existen numerosas bibliotecas, frameworks y módulos que amplían su funcionalidad, como React, Vue.js, Node.js, y otros. La extensibilidad también es posible mediante la creación de nuevas funciones y objetos dentro del propio lenguaje.

TypeScript es un superconjunto de JavaScript que agrega tipos estáticos. Aunque TypeScript no es estrictamente parte del estándar JavaScript, es una forma popular de extenderlo y mejorar la comprobación del código.

JavaScript no tiene subconjuntos formales en el sentido de que no existen versiones del lenguaje que sean una "versión reducida" o "subconjunto" oficial de JavaScript.

7. El código producido, ¿es transportable?

Sí, el código JavaScript es generalmente transportable, especialmente en el contexto de la web, donde JavaScript se ejecuta en todos los navegadores modernos y entornos de servidor compatibles con Node.js. El código que sigue los estándares ECMAScript es compatible entre diferentes plataformas y sistemas operativos, lo que lo hace altamente transportable.

Sin embargo, ciertas características del lenguaje, como dependencias de API específicas de un navegador o una versión particular de Node.js, pueden afectar la portabilidad. El uso de herramientas como Babel también ayuda a hacer que el código sea más compatible entre versiones más antiguas de JavaScript.