INFORME DE LENGUAJES, PARADIGMAS Y ESTÁNDARES DE PROGRAMACIÓN

I. INTRODUCCIÓN

Breve descripción sobre la importancia de los lenguajes, paradigmas y estándares, en el mundo de la programación.

a. **Los lenguajes paradigmas** de programación son una herramienta útil para clasificar los lenguajes de programación y elegir el paradigma adecuado para cada proyecto. La elección del paradigma adecuado puede afectar significativamente la calidad, eficiencia, mantenibilidad y escalabilidad del software.

b. **Los lenguajes estándares** son aquellos que siguen unas normas o reglas comunes para facilitar la comunicación, el intercambio y la comprensión entre diferentes personas, dispositivos o sistemas. Los lenguajes estándares pueden ser naturales, como el español o el inglés, o artificiales, como el HTML o el CSS.

II. TIPOS DE LENGUAJES DE PROGRAMACIÓN.

1. Descripción general de los lenguajes de alto nivel, medio nivel y bajo nivel:

a. **Los lenguajes de alto nivel**, son los que más se parecen al lenguaje humano y los que más se usan para desarrollar aplicaciones. Algunos ejemplos son Python, Java, JavaScript, etc. Estos lenguajes son portables, es decir, que se pueden ejecutar en diferentes plataformas con pocos o ningún cambio. También son legibles, es decir, que se pueden entender fácilmente por los programadores. Sin embargo, estos lenguajes requieren

de un intérprete o un compilador que los traduzca al lenguaje de máquina, lo que implica una pérdida de eficiencia y velocidad.

b. **Los lenguajes de medio nivel**, son los que combinan características de los lenguajes de alto y bajo nivel. Algunos ejemplos son C, C++, Rust, etc. Estos lenguajes son más eficientes y rápidos que los de alto nivel, ya que se comunican directamente con el hardware. También permiten un mayor control sobre la gestión de la memoria y los recursos del sistema. Sin embargo, estos lenguajes son menos portables que los de alto nivel, ya que dependen del tipo de procesador y del sistema operativo. También son menos legibles que los de alto nivel, ya que usan una sintaxis más compleja y abstracta.

c. **Los lenguajes de bajo nivel**, son los que más se acercan al lenguaje de máquina y los que menos se usan para desarrollar aplicaciones. Algunos ejemplos son el ensamblador, el Cobol, el Pascal, etc. Estos lenguajes son los más eficientes y rápidos de todos, ya que se ejecutan directamente en el procesador. También ofrecen un control total sobre el funcionamiento interno de la máquina. Sin embargo, estos lenguajes son los menos portables de todos, ya que varían según el modelo y la arquitectura del procesador. También son los menos legibles de todos, ya que usan una sintaxis muy difícil de entender y memorizar.

III. PARADIGMAS DE PROGRAMACIÓN:

Un paradigma de programación es un enfoque o estilo fundamental de desarrollo de software. Define cómo se estructuran y organizan las tareas para construir un programa.

1. **Descripción detallada de los principales paradigmas: Imperativo, Declarativo, Orientada a objetos, Funcional, Lógica**

a. **Programación Imperativa (o Procedimental):**

- Se basa en dar instrucciones al computador sobre cómo realizar una tarea paso a paso.
- Se centra en describir "cómo" hacer las cosas.
- Ejemplo de lenguajes: C, Pascal, Fortran.

b. **Programación Orientada a Objetos (OOP):**

- Organiza el software como una colección de objetos que contienen tanto datos como comportamientos.
- Introduce conceptos como herencia, encapsulamiento y polimorfismo.
- Ejemplo de lenguajes: Java, C++, Python, Ruby.

c. **Programación Funcional:**

- Se basa en la evaluación de funciones matemáticas y enfatiza la inmutabilidad (evitar cambiar el estado).
- Los programas son tratados como una evaluación de funciones compuestas.
- Ejemplo de lenguajes: Haskell, Lisp, Erlang, Scala.

d. **Programación Lógica:**

- Los programas son vistos como un conjunto de afirmaciones y reglas lógicas.
- Se utiliza principalmente para inferencia y consultas sobre datos.
- Ejemplo de lenguaje: Prolog.

2. Lenguajes representativos de cada paradigma y sus características distintivas.

Los paradigmas de programación son formas de clasificar los lenguajes de programación según sus características, principios y estilo. Algunos de los paradigmas más comunes son:

a. **Imperativo:** Se basa en la ejecución secuencial de instrucciones que modifican el estado del programa. Los lenguajes representativos de este paradigma son FORTRAN, COBOL, BASIC, PASCAL, C, ADA.

b. **Orientado a objetos:** Se basa en la definición de entidades llamadas objetos, que tienen atributos y métodos, y que se relacionan mediante herencia y polimorfismo. Los lenguajes representativos de este paradigma son Java, C++, C#, Eiffe, Python.

c. **Funcional**: Se basa en la evaluación de expresiones matemáticas que representan funciones, sin efectos secundarios ni cambios de estado. Los lenguajes representativos de este paradigma son LISP, Haskell, Erlang, Scala.

d. **Lógico:** Se basa en la definición de hechos y reglas que permiten inferir nuevos conocimientos mediante la lógica matemática. El lenguaje representativo de este paradigma es PROLOG.

IV. ESTÁNDARES DE PROGRAMACIÓN

1. Introducción a la importancia de seguir estándares.

Los estándares de programación son para escribir código fuente en ciertos lenguajes de programación.

Los estándares de programación son importantes para los desarrolladores por una amplia gama de razones. Permiten el entendimiento mutuo de la calidad del código entre los miembros del equipo, facilitando una mayor comunicación e incluso mejorando la calidad del software.

Los estándares de código son una serie de reglas definidas para un lenguaje de programación, o bien, un estilo de programación específico. El estilo garantiza que todos los ingenieros que contribuyen a un proyecto tengan una forma única de diseñar su código, lo que da como resultado una base de código coherente, asegurando una fácil comprensión y mantenimiento del mismo.

Por otro lado, no adherirse a los estándares de programación puede tener consecuencias negativas para los desarrolladores. El código no estandarizado puede ser difícil de leer y mantener, lo que puede llevar a errores y retrasos en el desarrollo. Además, el código no estandarizado puede ser difícil de entender para otros desarrolladores, lo que puede dificultar la colaboración y el trabajo en equipo.