Conceptos Iniciales

Que es un IDE y di al menos 3 de los más usados.

Un IDE (Entorno de Desarrollo Integrado) es un software que proporciona un entorno completo para el desarrollo de aplicaciones. Combina diversas herramientas del desarrollador en una sola interfaz gráfica, facilitando la escritura, depuración y compilación de código. Los IDEs son utilizados para mejorar la eficiencia en el desarrollo de software, permitiendo a los programadores trabajar de manera más organizada y efectiva.

Los tres IDEs más usados son:
Visual Studio Code (VS Code): Ideal para desarrolladores web y aquellos que buscan un editor rápido y flexible, con una amplia biblioteca de extensiones.

PyCharm: Excelente opción para desarrolladores que trabajan exclusivamente con Python, gracias a sus herramientas avanzadas y refactorización.

IntelliJ IDEA: Popular entre los desarrolladores de Java, conocido por sus características avanzadas y facilidad de uso.

Estos IDEs son altamente recomendados por su versatilidad y funcionalidad en el desarrollo de software.

Ciclo de vida de un software.

El ciclo de vida de un software se compone de varias etapas estructuradas que guían el desarrollo de aplicaciones desde su concepción hasta su retiro. Las etapas principales son:

Requisitos: Definición de objetivos y funcionalidades que debe cumplir el software.

Diseño: Creación de la arquitectura del software y definición de componentes.

Desarrollo: Implementación del software siguiendo el diseño establecido.

Pruebas: Verificación de que el software cumple con los requisitos.

Implementación: Despliegue del software en el entorno de producción.

Mantenimiento: Actualizaciones y correcciones necesarias después de la implementación.

Retiro: Proceso de descontinuación del software cuando ya no es necesario.

Este proceso estructurado asegura que el software se desarrolle de manera eficiente y cumpla con los estándares requeridos.

Fases en el desarrollo y ejecución del software.

Fases en el desarrollo y ejecución de un software
El desarrollo y ejecución de un software se llevan a cabo a través de varias fases, cada una con su propio conjunto de actividades y objetivos. Estas fases son fundamentales para garantizar la calidad y eficiencia del software final. Aquí se detallan las fases clave del ciclo de vida del desarrollo de software:
Planificación: Definición de objetivos, requisitos del usuario y elaboración de un plan detallado para el desarrollo del software.

Análisis de Requerimientos: Traslado de la visión del negocio a especificaciones técnicas.

Diseño de la Solución: Definición de la arquitectura del sistema, diseño de la base de datos, interfaz de usuario (UI) y algoritmos.

Implementación: Escribe el código real del software siguiendo los diseños y especificaciones previamente definidos.

Pruebas: Realización de pruebas exhaustivas para identificar y corregir errores y asegurar que el software funcione correctamente.

Despliegue: Lleva el producto a un entorno accesible para usuarios.

Mantenimiento y Evolución: Después del lanzamiento, el ciclo continúa con el mantenimiento y la evolución del software.

Estas fases son esenciales para crear software de alta calidad, eficiente y que satisfaga plenamente las necesidades del cliente. Dominar estas fases es esencial para el éxito de un proyecto tecnológico.

Diferencia entre código fuente, código objeto y código ejecutable.

El código fuente es el conjunto de instrucciones escritas por un programador en un lenguaje de programación legible por humanos. El código objetivo es una versión intermedia del código fuente que ha sido compilada a un formato binario, pero aun no es ejecutable directamente. El código ejecutable es un archivo binario que puede ser ejecutado directamente por el sistema operativo. Estos términos son fundamentales en el desarrollo de software, ya que permiten entender como un programa pasa de ser un conjunto de instrucciones escritas por un programador a algo que el ordenador puede ejecutar.

Diferencia entre algoritmo, pseudocódigo y código.

Algoritmo: Es un conjunto ordenado de pasos o instrucciones que se siguen para resolver un problema específico. Es independiente del lenguaje de programación y se puede implementar en cualquier lenguaje.

Pseudocódigo: Es una representación simplificada de un algoritmo que utiliza un lenguaje similar al lenguaje humano, permitiendo una comprensión más accesible. No es ejecutable por computadora, sino que se utiliza para planificar y comunicar la lógica de un algoritmo.

Código: Es la representación real de un algoritmo en un lenguaje de programación específico, como Java, Python o C++. Se traduce directamente en acciones que la computadora puede ejecutar.

En resumen, el algoritmo es la idea o plan, el pseudocódigo es una forma de expresarlo de manera más clara, y el código es la implementación en un lenguaje de programación.

¿Qué es el DISEÑO ESTRUCTURADO y cuáles son los 3 tipos de construcciones en las que se basa?

El diseño estructurado es una metodología utilizada principalmente en ingeniería de software para planificar y organizar sistemas complejos. Consiste en descomponer un problema global en subproblemas más manejables, llamados módulos, cada uno con una función específica

Tipos de Módulos o Construcciones del Diseño Estructurado
En función de su función y ubicación dentro del sistema, los módulos se pueden clasificar en tres tipos básicos dentro de un diseño estructurado

Módulo coordinador (Cm): Actúa como gestor del flujo principal del sistema, coordinando los módulos subordinados y la ejecución general del programa. Gestiona los procesos globales y supervisa la interacción entre distintos submódulos.

Módulo controlador o de procesamiento de información de llegada (Ce): Administra y valida la entrada de datos o eventos del sistema, determinando cuáles procesos deben activarse en respuesta a señales o información externa.

Módulo centro de transformación (Ct): Contiene las funciones esenciales o núcleo del sistema, realizando el procesamiento de datos y transformaciones necesarias para cumplir con los requerimientos del sistema. Es independiente de detalles de entrada/salida específicos, enfocándose en la lógica central del problema.

Qué es un lenguaje de programación y clasificación de los lenguajes en función de su nivel de abstracción y según la forma de ejecución. Pon ejemplos

Un lenguaje de programación es un conjunto de instrucciones y reglas que permiten a un programador comunicarse con una computadora para realizar tareas específicas. Estos lenguajes permiten traducir ideas y algoritmos a un formato que una máquina puede interpretar y ejecutar. Los lenguajes de programación varían en sintaxis, funcionalidad y nivel de cercanía al hardware, lo que influye en su complejidad y eficiencia.

Los lenguajes de programación se pueden clasificar principalmente en bajo nivel y alto nivel, dependiendo de qué tan cercanos estén al lenguaje máquina o al hardware de la computadora. A continuación se detallan las principales categorías:

- Lenguajes de bajo nivel:

Estos lenguajes están muy cerca del hardware y ofrecen control directo sobre los recursos de la computadora como memoria y registros. Su abstracción es mínima, lo que requiere mayor conocimiento técnico del funcionamiento interno del sistema.

Lenguaje máquina: Compuesto por instrucciones binarias (0 y 1) que la computadora interpreta directamente.

Lenguaje ensamblador (Assembly): Utiliza mnemónicos en lugar de códigos binarios, facilitando la lectura y escritura de programas respecto al lenguaje máquina, pero sigue siendo muy cercano al hardware.

- Lenguajes de alto nivel:

Estos lenguajes permiten escribir programas de forma más comprensible y cercana al lenguaje humano, ocultando complejidades del hardware. Generalmente requieren compiladores o intérpretes para traducirse a código máquina.

Ejemplos comunes: Python, Java, C++, JavaScript.

Ventajas: Mayor productividad, facilidad de mantenimiento y portabilidad entre plataformas.

- Lenguajes de muy alto nivel:

Diseñados para tareas muy concretas (por ejemplo, SQL para bases de datos o MATLAB para cálculos matemáticos).

Ofrecen un nivel de abstracción aún mayor, permitiendo centrarse en la lógica del problema más que en la gestión del hardware

Diferencia entre compilador e intérprete

En programación, compilador e intérprete son herramientas utilizadas para convertir un programa escrito en un lenguaje comprensible por humanos a instrucciones que la computadora pueda ejecutar, pero se diferencian en la forma en que realizan esta traducción.
Compilador: Su función es traducir todo el código fuente de un programa a código máquina o a un lenguaje intermedio antes de que el programa se ejecute. Esto genera un archivo ejecutable que puede ejecutarse de manera independiente, sin necesidad de recompilar continuamente. Ejemplos de lenguajes que suelen usar compiladores son C, C++, Rust y Go. Entre sus ventajas se encuentran el mayor rendimiento durante la ejecución y la detección de errores de sintaxis antes de ejecutar el programa. Sin embargo, la compilación puede ser lenta en programas grandes y cambios frecuentes requieren recompilar todo el código.
Intérprete: Este traduce y ejecuta el código línea por línea en el momento en que se escribe o se corre el programa, sin generar un archivo ejecutable permanente. Ejemplos de lenguajes interpretados son Python, JavaScript, Ruby y PHP. Sus ventajas incluyen la facilidad para probar y depurar código, ya que cualquier cambio se refleja de manera inmediata, lo que lo hace ideal para scripting o desarrollo rápido. La desventaja principal es que la ejecución suele ser más lenta y puede consumir más recursos, debido a que la traducción ocurre en tiempo real..

¿QUÉ ES BYTECODE?

El bytecode es un código de bajo nivel generado por los compiladores y utilizado por las máquinas virtuales para ejecutar programas. Es un conjunto de instrucciones que puede ejecutar el procesador del ordenador.

En la programación orientada a objetos (POO), existen tres conceptos fundamentales: clase, objeto y método. Defínelos y pon un ejemplo práctico en un lenguaje que conozcas.

La Programación Orientada a Objetos (POO) se basa en modelos que representan entidades del mundo real mediante objetos y su interacción. Los tres conceptos fundamentales son clase, objeto y relación entre objetos. A continuación se explica cada uno:

1. Clase
Una clase es un molde o plantilla que define las propiedades (atributos) y comportamientos (métodos) de un tipo de objeto.
Sirve para crear instancias concretas (objetos) con las mismas características definidas en la clase.
Ejemplo:

```
# Python
class Coche:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def arrancar(self):
        print(f"El coche {self.marca} {self.modelo} ha arrancado.")
```

2. Objeto
Un objeto es una instancia de una clase, es decir, un ejemplar concreto que posee los atributos y métodos definidos por la clase.
Cada objeto tiene estado (valores de sus atributos) y puede comportarse según los métodos de su clase.
Ejemplo:

```
mi_coche = Coche("Toyota", "Corolla")
mi_coche.arrancar()  # Output: El coche Toyota Corolla ha arrancado.
```

3. Relación entre objetos
Los objetos y clases no existen de forma aislada; interactúan mediante relaciones:
Asociación: Relación general donde un objeto usa otro.

```
class Conductor:
    def __init__(self, nombre):
        self.nombre = nombre

mi_conductor = Conductor("Carlos")
# Asociación: el conductor "usa" el coche
```

Herencia: Una clase puede heredar atributos y métodos de otra, facilitando la reutilización y extensión.
Composición y Agregación: Define relaciones “tiene un/a” donde un objeto contiene o gestiona a otros objetos como parte de su estructura.