A1.1 Conceptos iniciales

Investiga y responde a las siguientes preguntas/conceptos:

- Qué es un IDE y di al menos 3 de los más usados.
- Ciclo de vida de un software.
- Fases en el desarrollo y ejecución del software.
- Diferencia entre código fuente, código objeto y código ejecutable.
- Diferencia entre algoritmo, pseudocódigo y código.
- ¿Qué es el DISEÑO ESTRUCTURADO y cuáles son los 3 tipos de construcciones en las que se basa?
- 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.
- Diferencia entre compilador e intérprete.
- ¿QUÉ ES BYTECODE?
- 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.

Pd: Se debe crear una rama en github llamada conceptosInicialesTuapellido

1. Que es un IDE?

IDE significa **"Entorno de Desarrollo Integrado"** (en inglés *Integrated Development Environment*).
Es una **aplicación que reúne en un solo lugar todas las herramientas necesarias para programar**.

- Los mas usados son:
  - Visual studio code (Técnicamente es un editor con extensiones IDE)
  - IntelliJ IDEA (Java y Kotlin)
  - Eclipse (Java, C++)
  - PyCharm (Python)

2. ¿Que es el ciclo de vida de un software?

- Es el **proceso completo que sigue un software desde que se idea hasta que deja de usarse**.
  Incluye todas las fases necesarias: **planificación, desarrollo, pruebas, mantenimiento**, etc.

  Piensa que un programa no se hace solo "escribiendo código":
  antes hay que analizar qué necesita el cliente, diseñar la solución, probarla y mantenerla.

3. Ciclo de vida del desarrollo y ejecución del software.

- El ciclo de vida del software representa todas las fases que atraviesa una aplicación desde su planificación inicial hasta su mantenimiento una vez puesta en marcha.
  Se divide en **fases de desarrollo** y **fases de ejecución**, que permiten garantizar la calidad y continuidad del proyecto.

Fases de desarrollo

1. **Análisis de requisitos**
   En esta fase se recopila información sobre las necesidades del cliente o usuario.
   El objetivo es definir claramente qué debe hacer el software, qué datos manejará y qué limitaciones existen.
   El resultado es un documento de requisitos funcionales y técnicos.

2. **Diseño del sistema y del software**
   Se planifica la estructura del sistema: arquitectura, módulos, base de datos, interfaces y flujo de información.
   Esta etapa actúa como el “plano” del software antes de programarlo.

3. **Codificación o implementación**
   Los programadores desarrollan el código siguiendo el diseño establecido.
   Se utilizan lenguajes de programación y herramientas adecuadas al proyecto.
   Es la fase donde el software comienza a tomar forma real.

4. **Pruebas (testing)**
   Se comprueba que el sistema funcione correctamente y cumpla los requisitos.
   Se realizan pruebas unitarias, de integración, de sistema y de aceptación, para detectar y corregir errores antes del lanzamiento.

Fases de ejecución

5. **Implantación o despliegue**
   El software se instala en el entorno real de trabajo.
   Puede implicar configurar servidores, importar datos, o capacitar a los usuarios.
   Es la fase en la que el sistema pasa a estar operativo.

6. **Mantenimiento y operación**
   Una vez en funcionamiento, se monitoriza el rendimiento del sistema.
   Se corrigen errores detectados, se añaden mejoras y se adapta el software a nuevos requerimientos o entornos.
   Es una fase continua que garantiza la evolución y estabilidad del producto.

4. Diferencias entre código fuente, objeto y ejecutable.

Tipo de código	***Descripción***	***Características principales***
Código fuente	Es el programa escrito por el desarrollador en un lenguaje de programación.	Legible por humanos. No puede ejecutarse directamente.
Código objeto	Resultado de la compilación del código fuente.	Traducido a lenguaje máquina, pero todavía no ejecutable.
Código ejecutable	Programa final obtenido tras enlazar los códigos objeto con las librerías necesarias.	Listo para ser ejecutado por el sistema operativo.

5. Diferencias entre algoritmo, pseudocódigo y código.

Concepto	***Descripción***	***Nivel de abstracción***	***Ejecución***
Algoritmo	Conjunto de pasos lógicos para resolver un problema.	Muy alto (idea general).	No se puede ejecutar.
Pseudocódigo	Representación del algoritmo con una estructura parecida a un lenguaje de programación.	Intermedio.	No se ejecuta, se usa como guía.
Código fuente	Implementación real del algoritmo en un lenguaje de programación.	Bajo (cercano a la máquina).	Sí se puede ejecutar.

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

- El **diseño estructurado** es una **metodología para crear programas de forma clara, ordenada y fácil de entender**.
  Su objetivo es **dividir un problema grande en partes más pequeñas** (subprogramas o módulos) y usar **estructuras de control simples** que faciliten el desarrollo, la lectura y el mantenimiento del código.

  Se basa en el principio de que **todo programa puede construirse combinando tres tipos básicos de estructuras de control**, sin necesidad de usar instrucciones como *goto* o saltos incontrolados.

  - *Estos son los tres tipos de construcciones básicas.*

Secuencia

- Las instrucciones se ejecutan **una detrás de otra**, en el orden en que aparecen.
- Ejemplo:

```
Leer A
Leer B
Sumar A + B
Mostrar resultado
```

Selección (o decisión)

- Permite **elegir entre dos o más caminos** según una condición.
- Ejemplo (estructura *if*):

```
Si A > B Entonces
    Escribir "A es mayor"
Sino
    Escribir "B es mayor"
FinSi
```

Iteración (o repetición/bucle)

- Permite **repetir** un conjunto de instrucciones **mientras** se cumpla una condición.
- Ejemplo (estructura *mientras*):

```
Mientras contador <= 10 Hacer
    Escribir contador
    contador ← contador + 1

FinMientras
```

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

- Un **lenguaje de programación** es un **conjunto de instrucciones y reglas** que permiten al programador **comunicar órdenes a un ordenador** para que ejecute determinadas tareas.

  En otras palabras:
  Es el “idioma” que usamos para decirle al ordenador **qué hacer y cómo hacerlo**.

Nivel	***Descripción***	***Ejemplos***
Lenguajes de bajo nivel	Están **muy cerca del lenguaje máquina**. Son difíciles de leer y entender, pero muy rápidos.	*Lenguaje máquina*, *ensamblador (Assembly)*
Lenguajes de nivel medio	Combinan características de bajo y alto nivel. Permiten cierto control del hardware sin perder legibilidad.	*C*, *C++*
Lenguajes de alto nivel	Son los más **similares al lenguaje humano**. Fáciles de leer, entender y mantener.	*Python*, *Java*, *JavaScript*, *PHP*, *C#*

Clasificación según la forma de ejecución

Esto depende de **cómo se transforma el código fuente en instrucciones que entiende el ordenador**.

Tipo	***Descripción***	***Ejemplos***
Lenguajes compilados	El código fuente se **traduce completamente** a código máquina antes de ejecutarse. Produce un **archivo ejecutable**.	*C*, *C++*, *Go*, *Rust*
Lenguajes interpretados	El código se **traduce y ejecuta línea a línea** por un **intérprete** (no se genera ejecutable).	*Python*, *JavaScript*, *PHP*
Lenguajes híbridos	Se **compilan parcialmente** y luego se **interpretan** por una máquina virtual.	*Java* (usa la JVM), *C#* (usa el CLR de .NET)

9.Diferencia entre compilador e intérprete.

- Tanto el **compilador** como el **intérprete** sirven para **traducir el código fuente** (escrito por el programador en un lenguaje de alto nivel) a un **lenguaje que entienda la máquina (código máquina o binario)**.

Característica	***Compilador***	***Intérprete***
Forma de traducción	Traduce **todo el programa de una vez**.	Traduce **línea por línea** durante la ejecución.
Resultado	Genera un **archivo ejecutable**.	No genera archivo; ejecuta directamente.
Velocidad de ejecución	**Alta** (ya está traducido).	**Más lenta** (traduce en tiempo real).
Gestión de errores	Muestra **todos los errores** tras la compilación.	Se detiene **en el primer error** encontrado.
Ejemplos de lenguajes	C, C++, Java (fase de compilación).	Python, JavaScript, PHP.

9.¿Qué es el bytecode?

El **bytecode** es un **código intermedio** entre el **código fuente** (el que escribes tú) y el **código máquina** (el que entiende directamente el procesador).

Cuando programas en lenguajes como **Java o C#**, el proceso es así:

1. Escribes tu **código fuente** en Java (por ejemplo `HolaMundo.java`).

2. El **compilador** lo traduce a **bytecode** (archivo `.class` en Java).

3. Ese bytecode **no se ejecuta directamente por el procesador**, sino por una **máquina virtual** (JVM en Java o CLR en C#).
   → La máquina virtual interpreta o compila el bytecode en tiempo de ejecución, adaptándolo al sistema operativo.

10. Define los conceptos fundamentales de la programación orientada a objetos (POO)

- En la programación orientada a objetos existen tres conceptos fundamentales: **clase**, **objeto** y **método**.

  Una **clase** es una plantilla o modelo que define las características y comportamientos que tendrán los objetos. En ella se describen los atributos (propiedades) y los métodos (acciones) comunes a todos los objetos que se crean a partir de ella.

  Un **objeto** es una instancia concreta de una clase, es decir, un ejemplar real que posee los atributos definidos en la clase con valores propios. Cada objeto puede operar de manera independiente dentro del programa.

  Por último, un **método** es una función o procedimiento definido dentro de una clase que describe el comportamiento o las acciones que los objetos de esa clase pueden realizar.

  Estos tres elementos permiten estructurar el código de manera modular, reutilizable y más fácil de mantener, favoreciendo una programación más clara y organizada.

Ejemplo:

```
// Definición de una clase

class Coche {

  constructor(marca, modelo) {

    this.marca = marca;

    this.modelo = modelo;

  }
```

```
  // Método: acción del objeto

  arrancar() {

    console.log(`El ${this.marca} ${this.modelo} está
arrancando...`);

  }

}



// Creación de un objeto (instancia de la clase)

const miCoche = new Coche("Toyota", "Corolla");



// Uso de un método del objeto

miCoche.arrancar();  // Salida: El Toyota Corolla está arrancando...
```