PYTHON
DESARROLLO ORIENTADO A OBJETOS

IAC MÁS QUE COMPUTACIÓN

DESARROLLO EN PYTHON

INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

- ¿Qué es la Programación Orientada a Objetos y cómo se diferencia de otros paradigmas?
- ¿Qué ventajas ofrece la Programación Orientada a Objetos sobre otros enfoques?
- ¿Cuáles son los componentes básicos de la POO?
- ¿Qué es una clase en programación y para qué se utiliza?
- ¿Cómo se define una clase en Python?
- ¿Qué relación tiene una clase con los objetos?
- ¿Qué es un objeto en programación?
- ¿Cómo se crea un objeto a partir de una clase en Python?
- ¿Qué es el método __init__ y cuál es su propósito?
- ¿Cómo se define el método __init__ en una clase en Python?
- ¿Por qué es útil tener un método constructor en una clase?

ATRIBUTOS Y MÉTODOS BÁSICOS

- ¿Qué es un atributo en programación orientada a objetos?
- ¿Cómo se define y accede a un atributo en Python?
- ¿Cuál es la diferencia entre un atributo de clase y un atributo de instancia?
- ¿Qué es un método en programación orientada a objetos?
- ¿Cómo se define y se llama a un método en Python?
- ¿Cuál es la diferencia entre un método de clase, un método estático y un método de instancia?

MÉTODOS ESTÁTICOS Y DINÁMICOS

- ¿Qué es un método estático y cómo se diferencia de un método de instancia o un método de clase?
- ¿Cómo se define y se llama a un método estático en Python?
- ¿Cuándo es adecuado usar un método estático en lugar de otros tipos de métodos?
- ¿Qué es la herencia en programación orientada a objetos?

ATRIBUTO DE INSTANCIA Y COMPOSICIÓN

- ¿Qué es un atributo de instancia y cómo se diferencia de un atributo de clase?
- ¿Cómo se define y se accede a un atributo de instancia en Python?
- ¿Por qué es útil tener atributos de instancia en una clase?
- ¿Qué es la composición en programación orientada a objetos?
- ¿Cómo se implementa la composición en Python?
- ¿En qué situaciones es adecuado usar composición en lugar de herencia?

- ¿Cómo se implementa la herencia en Python?
- ¿Cuáles son las ventajas y desventajas de usar herencia?

SUPERCLASE, SUBCLASE Y MÉTODO SUPER()

- ¿Qué es una superclase y cómo se relaciona con la herencia?
- ¿Cómo se define y se utiliza una superclase en Python?
- ¿Qué diferencias hay entre una superclase y una subclase?
- ¿Qué es una subclase y cómo se relaciona con la herencia?
- ¿Cómo se define y se utiliza una subclase en Python?
- ¿En qué situaciones es útil definir subclases?

MÉTODO MÁGICO Y MÉTODO __STR__

- ¿Qué son los métodos mágicos en Python y cómo se reconocen?
- ¿Cómo se utilizan los métodos mágicos y en qué situaciones son invocados automáticamente?
- ¿Cuál es la diferencia entre un método mágico y un método regular?
- ¿Qué es el método `__str__` y para qué se utiliza?
- ¿Cómo se define el método `__str__` en una clase?
- ¿Qué diferencia hay entre `__str__` y `__repr__`?

MÉTODO __REPR__ Y ATRIBUTO DE CLASE

- ¿Qué es el método `__repr__` y cuál es su propósito?
- ¿Cómo se define el método `__repr__` en una clase y cómo se invoca?
- ¿Por qué es importante definir una representación adecuada con `__repr__`?
- ¿Qué es un atributo de clase y cómo se diferencia de un atributo de instancia?
- ¿Cómo se define y se accede a un atributo de clase en Python?
- ¿Cuándo es útil usar atributos de clase en lugar de atributos de instancia?

PYTHON

DESARROLLO ORIENTADO A OBJETOS

IAC MÁS QUE COMPUTACIÓN

AGREGACIÓN, ASOCIACIÓN Y MÉTODO __DEL__

- ¿Qué es la agregación en programación orientada a objetos?
- ¿Cuál es la principal diferencia entre composición y agregación?
- ¿Cómo se representa la agregación en Python?
- ¿Qué es la asociación en programación orientada a objetos?
- ¿Cómo se diferencia la asociación de la composición y la agregación?
- ¿Cómo se implementa la asociación en Python?
- ¿Qué es el método `__del__` y cuál es su propósito?
- ¿Cómo se define el método `__del__` en una clase en Python?
- ¿En qué situaciones es útil definir un destructor?

FLASK - DESARROLLO WEB Y MODULARIDAD AVANZADA

- ¿Qué es Flask y cómo se utiliza para desarrollar aplicaciones web?
- Principios básicos de enrutamiento y vistas en Flask.
- Estrategias avanzadas para implementar modularidad en proyectos web.
- Proyecto práctico intermedio integrando Flask.

TKINTER - INTERFACES GRÁFICAS Y PROYECTO INTERMEDIO

- Conceptos básicos de Tkinter para la creación de interfaces gráficas.
- Diseño y estructura de ventanas y widgets en Tkinter.
- Desarrollo de un proyecto intermedio que incluya Flask y Tkinter.
- Evaluación y retroalimentación intermedia.

REQUESTS - CONSUMO DE APIS Y PROYECTO INTERMEDIO

- ¿Cómo utilizar la librería Requests para consumir APIs?
- Integración de APIs en proyectos Python.
- Continuación del proyecto intermedio, incorporando consumo de APIs.
- Evaluación y retroalimentación intermedia.

IMPLEMENTACIÓN INTEGRAL Y PROYECTO FINAL

- Implementación Integral de librerías.
- Integración de Tkinter, Flask y Requests en proyectos.
- Ejemplos prácticos de proyectos modulares.
- Inicio del desarrollo del proyecto final, integrando todas las librerías aprendidas.
- Revisión de la estructura modular del proyecto.