Repaso POO

Programación Orientada a Objetos

También conocido como POO

**Qué tenés que saber:**

* ¿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 principales de la Programación Orientada a Objetos?

Programación Orientada a Objetos

Es un paradigma de programación que utiliza objetos y clases para organizar el código. Proporciona una estructura que permite a los objetos interactuar entre sí mediante propiedades (atributos) y acciones (métodos).

Clase

**Qué tenés que saber:**

* ¿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?

**Clase**

Es una plantilla para crear objetos. Define las propiedades (atributos) y acciones (métodos) que tendrá el objeto.

Objeto

También conocido como Instancia

**Qué tenés que saber:**

* ¿Qué es un objeto en programación?
* ¿Cómo se crea un objeto a partir de una clase en Python?
* ¿Qué diferencia hay entre una clase y un objeto?

Objeto

Es una instancia de una clase. Un objeto representa una entidad con estado (atributos) y comportamiento (métodos) en el programa.

Atributo

**Qué tenés que saber:**

* ¿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?

Atributo

Es una variable que se define dentro de una clase y representa una propiedad o característica de los objetos creados a partir de esa clase.

Método

**Qué tenés que saber:**

* ¿Qué es un método en programación orientada a objetos?
* ¿Cómo se define y se llama a un método en Python?
* ¿Qué diferencia hay entre un método de clase, un método estático y un método de instancia?

Método

Es una función definida dentro de una clase que puede operar sobre los datos (atributos) de la instancia de esa clase.

self

**Qué tenés que saber:**

* ¿Qué representa `self` en Python y cuál es su propósito?
* ¿Por qué es necesario incluir `self` como el primer parámetro de los métodos de instancia?
* ¿Cómo se diferencian las variables que se acceden con `self` de las variables locales en un método?

self

En Python, `self` es una referencia a la instancia actual del objeto. Se utiliza para acceder a los atributos y métodos de la instancia en el contexto de la clase.

Método \_\_str\_\_

**Qué tenés que saber:**

* ¿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 \_\_str\_\_

Es un método especial en Python que retorna una representación en forma de cadena de texto del objeto, y se invoca cuando se utiliza la función `print()` o `str()`.

Método \_\_repr\_\_

**Qué tenés que saber:**

* ¿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\_\_`?

Método \_\_repr\_\_

Es un método especial en Python que retorna una representación formal del objeto, ideal para desarrollo y debugging. Se invoca cuando se ingresa el objeto en el intérprete o se utiliza la función `repr()`.

El método \_\_repr\_\_ se utiliza para definir una representación legible del objeto, generalmente con el objetivo de poder reproducirlo. Se recomienda utilizarlo cuando deseamos obtener una descripción detallada de nuestro objeto que sea fácilmente entendible por otros programadores. Esto puede ser útil en situaciones como depuración, log de eventos o cuando queremos imprimir el objeto en la consola.

Claro, aquí tienes un ejemplo de cómo se podría utilizar el método \_\_repr\_\_:  
  
class Persona:  
 def \_\_init\_\_(self, nombre, edad):  
 self.nombre = nombre  
 self.edad = edad  
  
 def \_\_repr\_\_(self):  
 return f"Persona(nombre={self.nombre}, edad={self.edad})"  
  
persona1 = Persona("Juan", 25)  
print(persona1)  
  
  
En este ejemplo, hemos definido una clase "Persona" con un método \_\_repr\_\_ que devuelve una cadena que representa al objeto de manera legible. Al imprimir el objeto "persona1", se utilizará esa representación definida en el método \_\_repr\_\_ y se mostrará en la consola. En este caso, el resultado sería: "Persona(nombre=Juan, edad=25)".

Comportamiento

**Qué tenés que saber:**

* ¿Qué se entiende por comportamiento en programación orientada a objetos?
* ¿Cómo se define y modifica el comportamiento de un objeto?
* ¿Cómo se relacionan los métodos de una clase con el comportamiento de sus instancias?

Hace referencia a cómo un objeto opera o actúa, generalmente definido por sus métodos. En programación orientada a objetos, el comportamiento se refiere a las acciones que un objeto puede llevar a cabo.

Relaciones entre Clases:

Composición

**Qué tenés que saber:**

* ¿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?

Composición

Es una forma de relación entre clases en la que una clase se compone de una o más instancias de otras clases. Si la clase principal se destruye, también lo hacen las clases que la componen.

Agregación

**Qué tenés que saber:**

* ¿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?

Agregación

Es otra forma de relación entre clases donde una clase contiene instancias de otra(s) clase(s). Sin embargo, a diferencia de la composición, si la clase principal se destruye, las clases agregadas no necesariamente lo hacen.

Asociación

**Qué tenés que saber:**

* ¿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?

Es un tipo de relación entre dos clases donde una clase se asocia con otra sin tener una dependencia fuerte como en el caso de la composición o agregación.

Comportamiento

Método mágico

**Qué tenés que saber:**

* ¿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?

Método mágico

Son métodos especiales en Python que tienen doble guion bajo al inicio y al final de su nombre (por ejemplo, `\_\_init\_\_`, `\_\_str\_\_`). Estos métodos son invocados automáticamente en ciertas situaciones, como la inicialización de un objeto o su representación como cadena de texto.

Método \_\_init\_\_

**Qué tenés que saber:**

* ¿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?

Método \_\_init\_\_

Es el método constructor en Python y se invoca automáticamente cuando se crea una nueva instancia de una clase. Sirve para inicializar atributos y configurar el estado inicial del objeto.

Atributo público

**Qué tenés que saber:**

* ¿Qué es un atributo público en programación orientada a objetos?
* ¿Cómo se define y se accede a un atributo público en Python?
* ¿Cuándo es adecuado usar atributos públicos en lugar de atributos privados?

Atributo público

En Python, un atributo público es aquel que se puede acceder desde fuera de la clase. No tiene restricciones de acceso y generalmente se puede leer y modificar directamente.

Atributo privado

**Qué tenés que saber:**

* ¿Qué es un atributo privado y cómo se diferencia de un atributo público?
* ¿Cómo se define un atributo privado en Python?
* ¿Por qué y cuándo usaríamos atributos privados?

Atributo privado

En Python, un atributo privado es aquel que no se puede acceder directamente desde fuera de la clase. Se define utilizando dos guiones bajos al inicio de su nombre (por ejemplo, `\_\_atributo\_privado`).

Método getter

**Qué tenés que saber:**

* ¿Qué es un método getter y para qué se utiliza?
* ¿Cómo se implementa un método getter en Python?
* ¿Por qué es útil tener métodos getter en lugar de acceder directamente a los atributos?

Método getter

Es un método que se utiliza para obtener el valor de un atributo privado. Permite el acceso controlado a dicho atributo.

Método setter

**Qué tenés que saber:**

* ¿Qué es un método setter y cuál es su propósito?
* ¿Cómo se implementa un método setter en Python?
* ¿Por qué es útil tener métodos setter en lugar de modificar directamente los atributos?

Método setter

Es un método que se utiliza para establecer o modificar el valor de un atributo privado. Permite el acceso controlado y posiblemente la validación del valor antes de establecerlo.

Propiedad @property

**Qué tenés que saber:**

* ¿Qué es el decorador `@property` y cómo se utiliza en Python?
* ¿Cómo se define una propiedad usando `@property`?
* ¿Cuál es la ventaja de usar `@property` en lugar de un método regular?

Propiedad @property

En Python, `@property` es un decorador que convierte un método en una propiedad de solo lectura. Esto permite acceder a un método como si fuera un atributo sin paréntesis.

Método \_\_del\_\_

**Qué tenés que saber:**

* ¿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?

Método \_\_del\_\_

Es el destructor en Python y se invoca automáticamente antes de que un objeto sea destruido. Es útil para liberar recursos o realizar limpieza.

Atributo de clase

**Qué tenés que saber:**

* ¿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?

Atributo de clase

Es un atributo que pertenece a la clase y no a una instancia específica de la clase. Todos los objetos de la clase comparten el mismo valor para un atributo de clase.

Atributo de instancia

**Qué tenés que saber:**

* ¿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?

Atributo de instancia

Es un atributo que se define para una instancia específica de una clase. Cada objeto tiene su propio valor para un atributo de instancia.

Método estático

**Qué tenés que saber:**

* ¿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?

En Python, un método estático es un método que pertenece a la clase y no a una instancia específica de la clase. Se define usando el decorador `@staticmethod` y no recibe argumento `self`.

Herencia

**Qué tenés que saber:**

* ¿Qué es la herencia en programación orientada a objetos?
* ¿Cómo se implementa la herencia en Python?
* ¿Cuáles son las ventajas y desventajas de usar herencia?

Herencia

Es un mecanismo en la programación orientada a objetos que permite definir una nueva clase basada en una clase existente. La nueva clase hereda atributos y comportamientos de la clase base, y puede añadir o modificar características.

Superclase

También conocido como Clase Base, Clase Padre

**Qué tenés que saber:**

* ¿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?

Superclase

También conocida como clase base o clase padre, es la clase de la cual otras clases heredan atributos y comportamientos.

Subclase

**Qué tenés que saber:**

* ¿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?

Subclase

También conocida como clase derivada, es una clase que hereda atributos y comportamientos de una superclase. Puede añadir o modificar características de la clase base.

Método super()

**Qué tenés que saber:**

* ¿Qué es el método `super()` y cuál es su propósito?
* ¿Cómo se utiliza el método `super()` en Python para llamar a métodos de la superclase?
* ¿Por qué es útil usar `super()` en lugar de invocar directamente el método de la superclase?

Método super()

Es una función en Python que se utiliza para invocar un método de la superclase desde una subclase. Es especialmente útil cuando se quiere extender o modificar el comportamiento heredado.

Clase abstracta

**Qué tenés que saber:**

* ¿Qué es una clase abstracta y cuál es su propósito en programación orientada a objetos?
* ¿Cómo se define y se utiliza una clase abstracta en Python?
* ¿Cuál es la diferencia entre una clase abstracta y una interfaz?

Es una clase que no puede ser instanciada y se utiliza como base para otras clases. Las clases abstractas pueden tener métodos abstractos, los cuales deben ser implementados por las subclases.