

Python Orientado a Objetos

Smailyn G B.

Programación orientada a objetos

La programación orientada a objetos (POO) es un enfoque de desarrollo de software que organiza el código alrededor de objetos, los cuales combinan datos y funciones relacionadas en una única entidad. Las clases actúan como plantillas para crear objetos, definiendo sus atributos (datos) y métodos (funciones). La POO fomenta la modularidad y la reutilización de código, permitiendo crear sistemas más flexibles y mantenibles.

Los conceptos fundamentales de la POO incluyen la encapsulación, que oculta la complejidad interna de los objetos, la herencia, que permite la creación de jerarquías de clases para compartir funcionalidades, y el polimorfismo, que facilita el tratamiento uniforme de objetos diferentes a través de una misma interfaz. Esta metodología se ha vuelto presente en la industria del software debido a su capacidad para modelar problemas de manera intuitiva y para producir sistemas más robustos y escalables.

Las clases

Una clase es una plantilla que define la estructura y el comportamiento de un tipo de objeto. Las clases son como moldes que especifican qué datos pueden contener los objetos (llamados atributos) y qué operaciones pueden realizar (llamadas métodos). Una clase describe las propiedades y acciones que los objetos de ese tipo pueden tener.

Por ejemplo, si estuviéramos creando un sistema de gestión de biblioteca, podríamos tener una clase llamada "Libro" que define los atributos de un libro (como título, autor y número de páginas) y los métodos que pueden actuar sobre esos atributos (como "prestar" o "devolver").

Las clases son esenciales en la POO porque permiten organizar y estructurar el código de manera modular y reutilizable, lo que facilita el desarrollo y mantenimiento de sistemas complejos.

Atributos

En la programación orientada a objetos, los atributos son variables que forman parte de un objeto y que representan sus características o estado interno. También se les conoce como propiedades o datos miembro. Estos atributos describen las características específicas de cada instancia de la clase a la que pertenecen.

En la programación orientada a objetos, los atributos son variables que forman parte de un objeto y que representan sus características o estado interno. También se les conoce como propiedades o datos miembro. Estos atributos describen las características específicas de cada instancia de la clase a la que pertenecen.

Constructor

El constructor es un método especial que se utiliza para inicializar los atributos de un objeto cuando se crea una nueva instancia de la clase. En Python, el constructor se define con el método mágico `__init__(self, ...)`.

Tienen el mismo nombre que la clase a la que pertenecen y se invocan automáticamente al crear un nuevo objeto. Su función principal es asignar valores iniciales a los atributos del objeto o realizar otras tareas de inicialización necesarias. Los constructores pueden tener diferentes parámetros para permitir la personalización de la inicialización del objeto y una clase puede tener múltiples constructores para adaptarse a diferentes formas de creación de objetos.

Métodos

Los métodos son funciones asociadas a una clase que pueden realizar acciones específicas sobre los objetos de esa clase o manipular sus datos. Estas funciones definen el comportamiento de los objetos y pueden acceder a los atributos de la clase a través de parámetros o directamente si tienen visibilidad apropiada. Los métodos pueden realizar tareas como cálculos, manipulación de datos, interacción con otros objetos o cualquier otra operación relacionada con el propósito de la clase. En resumen, los métodos son las acciones que un objeto puede realizar y son fundamentales para definir su comportamiento en un sistema orientado a objetos.

Métodos Mágicos

Los métodos mágicos, métodos especiales o métodos dunder (double underscore), son métodos que proporcionan funcionalidades específicas en clases. Estos métodos se llaman automáticamente en ciertas circunstancias según el contexto en el que se utilizan.

Tienen nombres que comienzan y terminan con doble guion bajo, como `__init__` para el constructor, `__str__` para representar una cadena legible de un objeto, `__len__` para obtener la longitud de un objeto, etc. Permiten que las clases se comporten de manera similar a los tipos de datos integrados de Python y proporcionan una forma estándar de sobrecargar operadores y comportamientos específicos.

Estos métodos permiten personalizar el comportamiento de las clases en relación con operaciones como la creación, comparación, asignación, acceso a atributos, entre otros.

Algunos ejemplos de métodos mágicos son __init__, __str__, __eq__, __len__, entre otros.

Miembros Privados

Los miembros privados son aquellos atributos y métodos de una clase que no pueden ser accedidos directamente desde fuera de la clase. Esta característica permite encapsular y proteger el estado interno del objeto, asegurando que solo pueda ser modificado a través de métodos específicos de la clase, lo que facilita el mantenimiento y la integridad de los datos.

Ventajas de usar miembros privados:

Encapsulamiento:

Protege los datos y garantiza que solo se accedan de manera controlada

Mantenimiento:

Facilita el mantenimiento del código al reducir el riesgo de que cambios internos afecten a otras partes del programa.

Integridad de datos:

Ayuda a mantener la integridad de los datos al controlar cómo se modifican.

Modularidad:

Fomenta la creación de módulos de código independientes y reutilizables.

Propiedades

Las propiedades son características o datos de una clase que almacenan el estado de sus objetos. Se accede y modifica a través de métodos específicos, llamados getters y setters.

Esto permite:

Encapsulamiento: Protege los datos internos.

Control de acceso: Permite añadir lógica para validaciones y otros procesos al leer o modificar los datos.

Mantenibilidad: Mejora la claridad y el mantenimiento del código.

Herencia

La herencia es un mecanismo que permite a una clase (denominada clase hija o subclase) adquirir las propiedades y comportamientos (atributos y métodos) de otra clase (denominada clase padre o superclase).

La herencia facilita la reutilización del código y la creación de una jerarquía de clases que refleja las relaciones del mundo real.

Tipos de Herencia

Herencia simple: Una subclase hereda de una sola superclase.

Herencia múltiple: Una subclase hereda de más de una superclase (soportada por lenguajes como C++ pero no por Java).

Herencia jerárquica: Una superclase es heredada por múltiples subclases.

Herencia multinivel: Una subclase hereda de otra subclase, formando una cadena.

Sobrecarga de Métodos

La sobrecarga de métodos es una característica en la programación orientada a objetos (POO) que permite definir múltiples métodos con el mismo nombre en la misma clase, pero con diferentes parámetros (ya sea en número, tipo o ambos). Esta técnica proporciona flexibilidad y mejora la legibilidad del código, permitiendo que un método realice diferentes tareas según los argumentos que reciba.

Permitiendo que los métodos manejen múltiples casos de uso basados en diferentes conjuntos de parámetros. Esta capacidad es especialmente útil para crear interfaces intuitivas y fáciles de usar.

Herencia multinivel

La herencia multinivel es un tipo de herencia en donde una clase deriva de otra clase que a su vez deriva de otra clase. En otras palabras, es una cadena de herencia en la que una subclase se convierte en una superclase para otra subclase. Este mecanismo permite que una clase herede características y comportamientos de más de una clase a lo largo de una cadena de herencia, formando una jerarquía de clases.

poderoso mecanismo en la programación orientada a objetos que permite crear una jerarquía de clases, donde cada clase hereda características de la clase anterior y puede añadir o modificar funcionalidades. Este enfoque mejora la reutilización del código, la organización y la extensibilidad del sistema.

Clases abstractas

Las clases abstractas son aquellas que por sí mismas no se pueden identificar con algo 'concreto' (no existen como tal en el mundo real), pero sí poseen determinadas características que son comunes en otras clases que pueden ser creadas a partir de ellas.

Una forma de definir comportamientos comunes y forzar la implementación de métodos específicos en las subclases. Facilitan la organización del código, promueven la reutilización y aseguran consistencia en la estructura de la jerarquía de clases. Sin embargo, deben usarse con cuidado para evitar la complejidad y rigidez innecesarias en el diseño del software.

Polimorfismo

El polimorfismo consiste en diseñar objetos para compartir comportamientos, lo que nos permite procesar objetos de diferentes maneras. Es la capacidad de presentar la misma interfaz para diferentes formas subyacentes o tipos de datos. Al utilizar la herencia, los objetos pueden anular los comportamientos principales compartidos, con comportamientos secundarios específicos. Se refiere a la capacidad de un objeto de tomar diferentes formas o comportamientos según el contexto en el que se utilice. Permite que objetos de diferentes clases respondan de manera distinta a un mismo mensaje o método. Esto facilita la reutilización de código y mejora la flexibilidad y extensibilidad del sistema.

Duck type

Duck typing es una filosofía de la programación orientada a objetos que prioriza el comportamiento de los objetos sobre su tipo explícito. En lenguajes como Python, Ruby y JavaScript, esta técnica permite determinar la validez de un objeto según sus métodos y propiedades, no su clase. Por ejemplo, un objeto con un método "sound" puede ser considerado como un "pato", independientemente de su clase.

Esto simplifica la escritura de código genérico y reutilizable, al permitir que funciones y métodos trabajen con cualquier objeto que tenga los métodos necesarios, reduciendo dependencias en tipos específicos. Aunque ofrece flexibilidad, puede conducir a errores en tiempo de ejecución si los objetos no tienen los métodos esperados, lo que demanda una gestión cuidadosa del código y documentación clara para garantizar la comprensión y mantenibilidad del software.

Dataclase

Una dataclase es un concepto que se encuentra en Python. Se utiliza para definir clases que están diseñadas principalmente para almacenar datos y no incluyen lógica adicional más allá de la inicialización y la representación de datos.

Permiten crear clases con campos de datos (atributos) que son automáticamente inicializados en el momento de la creación del objeto.

El objetivo principal de las dataclases es reducir la cantidad de código boilerplate necesario para definir clases simples de almacenamiento de datos, mejorando así la legibilidad y la mantenibilidad del código.

API

Una API (Interfaz de Programación de Aplicaciones) basada en programación orientada a objetos (POO) consiste en un conjunto de clases y métodos que ofrecen una forma organizada y coherente de comunicarse con un sistema o biblioteca de software. Estas clases encapsulan la funcionalidad y los datos relacionados con una tarea específica, como el manejo de usuarios o la gestión de datos. Los métodos de estas clases permiten realizar operaciones específicas, como crear, leer, actualizar y eliminar datos, proporcionando así una interfaz estructurada para interactuar con el sistema. Una API en Python POO facilita la comunicación entre diferentes partes de una aplicación o entre aplicaciones diferentes al definir claramente cómo acceder y manipular los recursos ofrecidos por el sistema o la biblioteca.

Principios de diseño.

SOLID es un acrónimo que representa cinco principios de diseño de objetos que fueron propuestos por Robert C. Martin (también conocido como Uncle Bob) a principios de la década de 2000.

Estos principios son:

S - Principio de Responsabilidad Única (SRP): Una clase debe tener una sola razón para cambiar, es decir, debe tener una sola responsabilidad.

O - Principio de Abierto/Cerrado (OCP): Las entidades de software (clases, módulos, funciones, etc.) deben estar abiertas para extensión pero cerradas para modificación. Esto significa que deberíamos poder extender su comportamiento sin necesidad de modificar su código fuente.

L - Principio de Sustitución de Liskov (LSP): Los objetos de un programa deberían ser sustituibles por instancias de sus subtipos sin alterar la corrección del programa.

I - Principio de Segregación de la Interfaz (ISP): Los clientes no deberían verse obligados a depender de interfaces que no utilicen.

D - Principio de Inversión de Dependencia (DIP): Las entidades de nivel superior no deberían depender de entidades de nivel inferior. Ambos deberían depender de abstracciones. Además, los detalles concretos deben depender de abstracciones, no al revés.

KISS es un acrónimo que significa "Keep It Simple, Stupid" (Mantenlo Simple, Estúpido). Este principio aboga por mantener las cosas simples y evitar la complejidad innecesaria en el diseño y la implementación del software. Se centra en la simplicidad como una virtud y promueve la idea de que la simplicidad conduce a una mayor comprensión, mantenibilidad y eficiencia.

DRY es un acrónimo que significa "Don't Repeat Yourself" (No Te Repitas). Este principio enfatiza la reutilización del código y la eliminación de la duplicación. Se sugiere que cada pieza de conocimiento debe tener una representación única y no ambigua dentro de un sistema, evitando así la repetición de código y reduciendo la complejidad y el riesgo de errores.