



códigofacilito

Proyecto con Decoradores

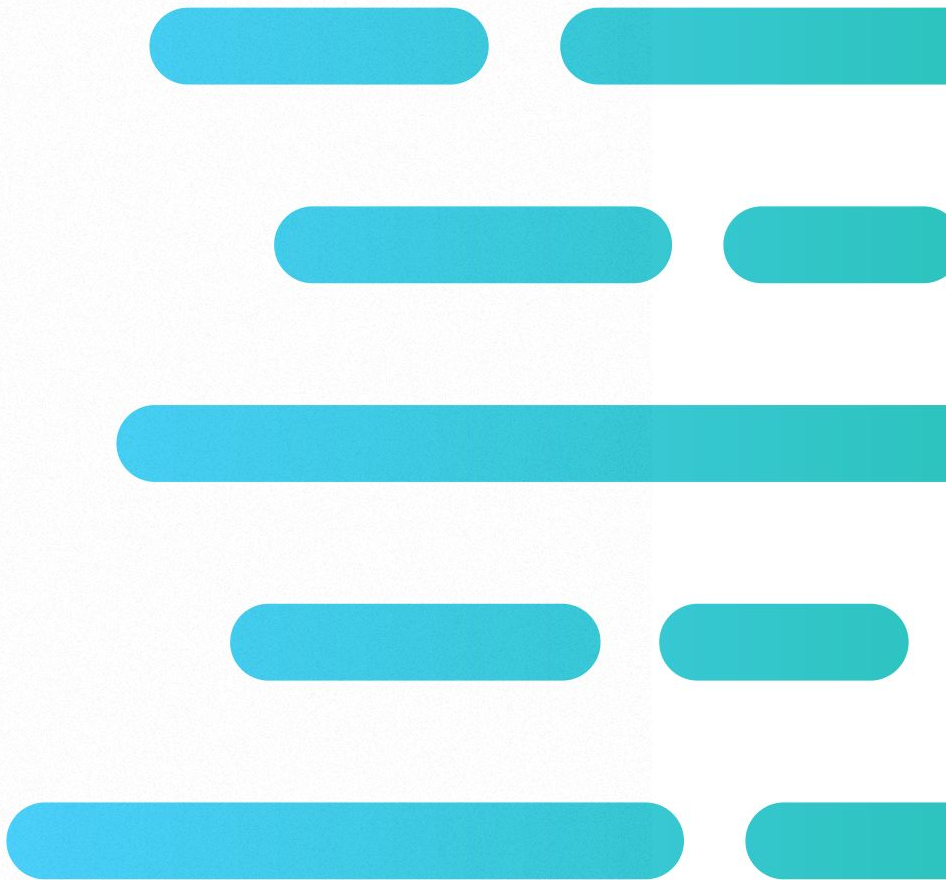
Bootcamp Python Avanzado

Carolina - Backend Developer





Código de la clase 🦆





- >_ Patrones de diseño
- >_ Clasificación de los Patrones de Diseño
- >_ Patrón Decorator
- >_ Proyecto





¿Qué son los
patrones de diseño?





Los patrones de diseño son soluciones habituales a problemas que ocurren con frecuencia en el diseño de software. Son como planos prefabricados que se pueden personalizar para resolver un problema de diseño recurrente en tu código.



Recurso recomendado:

<https://refactoring.guru/es/design-patterns/what-is-pattern>



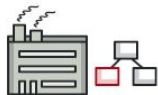
Clasificación de los Patrones de Diseño





Patrones creacionales

Los patrones creacionales proporcionan varios mecanismos de creación de objetos que incrementan la flexibilidad y la reutilización del código existente.



Factory Method

Proporciona una interfaz para la creación de objetos en una superclase, mientras permite a las subclases alterar el tipo de objetos que se crearán.



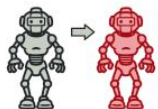
Abstract Factory

Permite producir familias de objetos relacionados sin especificar sus clases concretas.



Builder

Permite construir objetos complejos paso a paso. Este patrón nos permite producir distintos tipos y representaciones de un objeto empleando el mismo código de construcción.



Prototype

Permite copiar objetos existentes sin que el código dependa de sus clases.



Singleton

Permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.



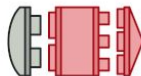
Recurso recomendado:

<https://refactoring.guru/es/design-patterns/creational-patterns>



Patrones estructurales

Los patrones estructurales explican cómo ensamblar objetos y clases en estructuras más grandes, a la vez que se mantiene la flexibilidad y eficiencia de estas estructuras.



Adapter

Permite la colaboración entre objetos con interfaces incompatibles.



Bridge

Permite dividir una clase grande o un grupo de clases estrechamente relacionadas, en dos jerarquías separadas (abstracción e implementación) que pueden desarrollarse independientemente la una de la otra.



Composite

Permite componer objetos en estructuras de árbol y trabajar con esas estructuras como si fueran objetos individuales.



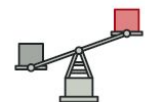
Decorator

Permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.



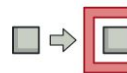
Facade

Proporciona una interfaz simplificada a una biblioteca, un framework o cualquier otro grupo complejo de clases.



Flyweight

Permite mantener más objetos dentro de la cantidad disponible de memoria RAM compartiendo las partes comunes del estado entre varios objetos en lugar de mantener toda la información en cada objeto.



Proxy

Permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.



Recurso recomendado:

<https://refactoring.guru/es/design-patterns/structural-patterns>



Patrones de comportamiento

Los patrones de comportamiento tratan con algoritmos y la asignación de responsabilidades entre objetos.



Chain of Responsibility

Permite pasar solicitudes a lo largo de una cadena de manejadores. Al recibir una solicitud, cada manejador decide si la procesa o si la pasa al siguiente manejador de la cadena.



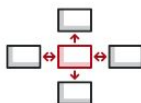
Command

Convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. Esta transformación te permite parametrizar los métodos con diferentes solicitudes, retrasar o poner en cola la ejecución de una solicitud y soportar operaciones que no se pueden realizar.



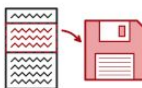
Iterator

Permite recorrer elementos de una colección sin exponer su representación subyacente (lista, pila, árbol, etc.).



Mediator

Permite reducir las dependencias caóticas entre objetos. El patrón restringe las comunicaciones directas entre los objetos, forzándolos a colaborar únicamente a través de un objeto mediador.



Memento

Permite guardar y restaurar el estado previo de un objeto sin revelar los detalles de su implementación.



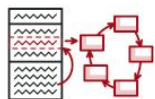
Observer

Permite definir un mecanismo de suscripción para notificar a varios objetos sobre cualquier evento que le suceda al objeto que están observando.



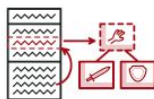
Recurso recomendado:

<https://refactoring.guru/es/design-patterns/behavioral-patterns>



State

Permite a un objeto alterar su comportamiento cuando su estado interno cambia. Parece como si el objeto cambiara su clase.



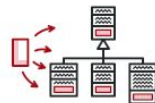
Strategy

Permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer sus objetos intercambiables.



Template Method

Define el esqueleto de un algoritmo en la superclase pero permite que las subclases sobrescriban pasos del algoritmo sin cambiar su estructura.



Visitor

Permite separar algoritmos de los objetos sobre los que operan.



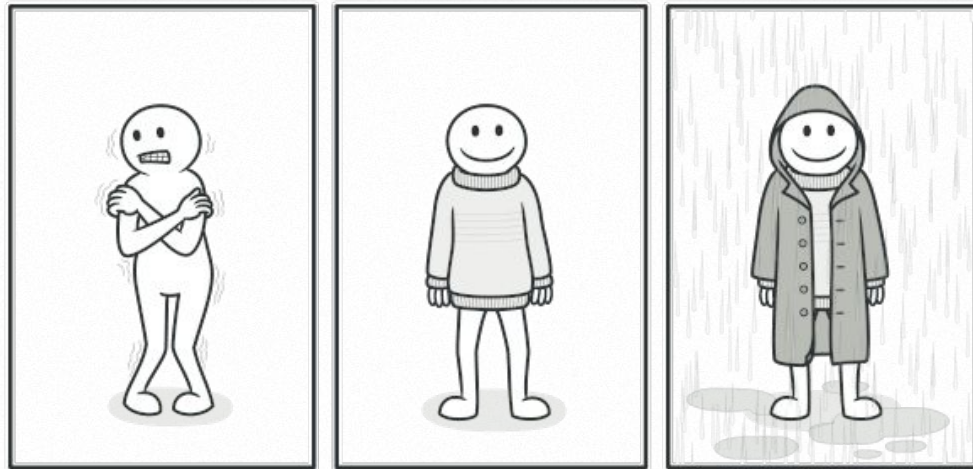


Patrón Decorator





Decorator es un patrón de diseño estructural que te permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades.



Recurso recomendado:

<https://refactoring.guru/es/design-patterns/decorator>



Proyecto





examples of a project with decorators using classes in python



View other drafts ▾



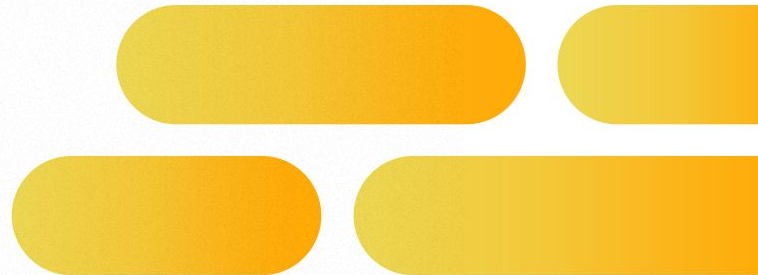
Sure, here are some examples of projects with decorators using classes in Python:

- **Logging:** A decorator can be used to add logging functionality to a class. This can be useful for tracking the execution of a class's methods.
- **Authentication:** A decorator can be used to add authentication functionality to a class. This can be used to restrict access to a class's methods to authorized users.
- **Caching:** A decorator can be used to add caching functionality to a class. This can be used to improve the performance of a class by storing the results of expensive computations in memory.
- **Rate limiting:** A decorator can be used to add rate limiting functionality to a class. This can be used to prevent a class from being overloaded with requests.





¿Preguntas?





¡Gracias!

