**Decoradores**

**Decoradores** em Python são uma maneira poderosa de modificar o comportamento de funções ou métodos, sem alterar diretamente seu código. Eles permitem "envolver" uma função com outra, adicionando funcionalidades extras. Em termos simples, um decorador é uma função que recebe outra função como argumento, faz algo com essa função e retorna uma nova função ou a própria função modificada.

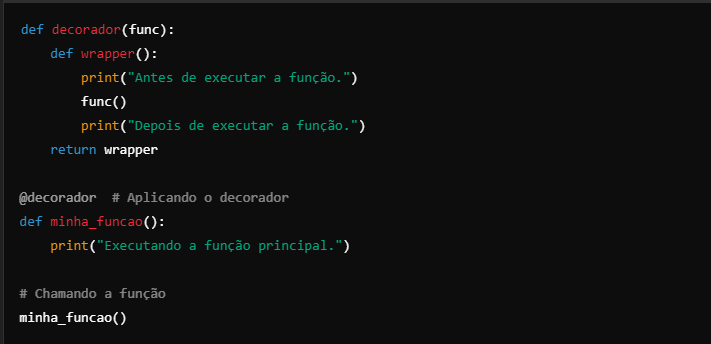
**Estrutura Básica de um Decorador:**

Um decorador é uma função que envolve outra função para adicionar ou modificar seu comportamento.

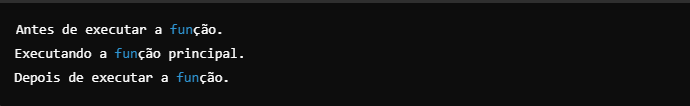
**Decoradores** em Python são uma maneira poderosa de modificar o comportamento de funções ou métodos, sem alterar diretamente seu código. Eles permitem "envolver" uma função com outra, adicionando funcionalidades extras. Em termos simples, um decorador é uma função que recebe outra função como argumento, faz algo com essa função e retorna uma nova função ou a própria função modificada.

### Estrutura Básica de um Decorador:

Um decorador é uma função que envolve outra função para adicionar ou modificar seu comportamento.

****

Saída:

****

### Explicação:

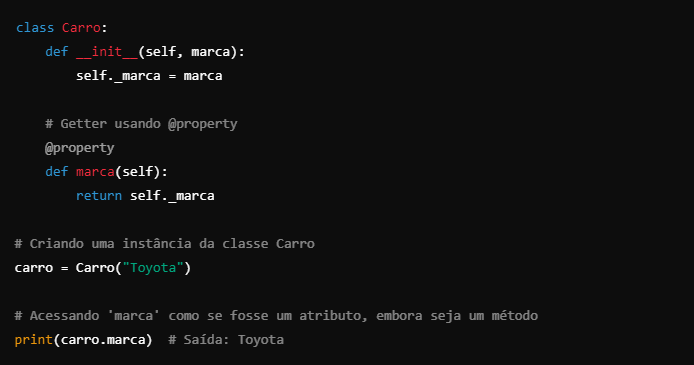
* decorador(func) é uma função que recebe func (outra função) como argumento.
* A função interna wrapper() envolve a execução da função original, permitindo adicionar comportamento antes e/ou depois de func().
* O @decorador é aplicado diretamente sobre minha\_funcao(). Isso faz com que, ao chamar minha\_funcao(), o decorador seja automaticamente aplicado.

### Decoradores em Métodos de Classe

Decoradores são amplamente usados em Programação Orientada a Objetos (POO) para manipular o comportamento de métodos de classe. Python possui alguns decoradores embutidos, como @property, @classmethod, e @staticmethod.

#### Exemplo com @property:

O decorador @property transforma um método em um **getter** para que ele possa ser acessado como se fosse um atributo.

****

Aqui, @property transforma o método marca() em um getter que pode ser acessado como um atributo.

#### Exemplo com @classmethod e @staticmethod:

* @classmethod: Permite que um método seja chamado diretamente pela classe, e não pela instância, recebendo cls como primeiro argumento (referência à classe, não à instância).
* @staticmethod: Indica que um método não depende nem da instância (self) nem da classe (cls). É basicamente uma função dentro de uma classe.

****

**Benefícios dos Decoradores**

* **Reutilização de código**: Com decoradores, você pode adicionar comportamentos comuns a várias funções ou métodos sem duplicar código.
* **Organização**: Eles ajudam a separar a lógica adicional da lógica principal da função/método.
* **Limpeza**: Tornam o código mais limpo e legível.

Em resumo, decoradores permitem adicionar funcionalidade de forma clara e eficiente, sem modificar diretamente a função original, tornando-os uma ferramenta poderosa na programação Python.

**Anotações:**

*Esses decoradores fazem uma ligação direta com os atribultos da classe.*

*Por exemplo: Quando faço uma atribuição a nome, a atribuição primeiro passa pelo nome.setter, recebe o tratamento, e apenas depois retorna a atribuição.*

*Da mesma forma quando recebe a chamada para a exibição, primeiro passa pelo metodo getter, de forma que pode receber algum tratamento para depois retornar.*