
Desvendando Decoradores
Aprimore seu Código em Python

Introdução

Decoradores em Python são uma maneira incrível de modificar e melhorar suas funções e métodos sem alterar seu código original. Neste ebook, vamos explorar como usar decoradores para adicionar funcionalidades, otimizar seu código e torná-lo mais elegante. Prepare-se para descobrir como esses truques podem transformar a forma como você programa!

Principais Erros e Dúvidas ao se Trabalhar com Decoradores em Python

- 1. Esquecer de Usar functools.wraps

Erro Comum: Quando você cria um decorador, é fácil esquecer de usar functools.wraps, o que pode fazer você perder informações importantes da função original, como seu nome e docstring.

Ao não utilizar functools.wraps, além da perda de informações, pode também se tornar um problema ao depurar ou documentar seu código.

Exemplo de código:

```python
import functools

def meu_decorador(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        print("Função decorada!")
        return func(*args, **kwargs)
    return wrapper

@meu_decorador
def diga_oi():
    """Diz oi"""
    print("Oi!")

diga_oi()
print(diga_oi.__name__)  # Deve imprimir 'diga_oi'
```

- 2. Não Entender Como Funciona a Ordem de Execução

Erro Comum: Muita gente se confunde com a ordem de execução dos decoradores, especialmente quando há múltiplos decoradores em uma função. Lembre-se de que o decorador mais interno é executado primeiro.

Exemplo de código:

```python
def decorador1(func):
    def wrapper(*args, **kwargs):
        print("Decorador 1")
        return func(*args, **kwargs)
    return wrapper

def decorador2(func):
    def wrapper(*args, **kwargs):
        print("Decorador 2")
        return func(*args, **kwargs)
    return wrapper

@decorador1
@decorador2
def minha_funcao():
    print("Função original")

minha_funcao()
# Ordem de execução: decorador2, depois decorador1, depois função original.
```

- 3. Passar Argumentos para Decoradores

Erro Comum: Esquecer como passar argumentos para decoradores ou confundir a sintaxe. Quando você precisa passar argumentos para decoradores, a sintaxe pode ser um pouco complexa. É necessário criar uma função adicional para receber os argumentos e retornar o decorador.

Exemplo de código:

```python
def repetir(n):
    def decorador(func):
        def wrapper(*args, **kwargs):
            for _ in range(n):
                func(*args, **kwargs)
        return wrapper
    return decorador

@repetir(3)
def diga_ola():
    print("Olá!")

diga_ola()  # Vai imprimir "Olá!" três vezes
```

- 4. Decoradores com Classes

Dúvida Comum: Como usar decoradores com métodos de classe e como isso afeta self?
Quando você usa decoradores em métodos de classe, é crucial garantir que eles tratem corretamente o parâmetro self para não interferir na instância da classe. O decorador deve ser projetado para funcionar com o método da instância sem modificar a lógica esperada.

Exemplo de código:

```python
def meu_decorador(func):
    def wrapper(*args, **kwargs):
        print("Método decorado!")
        return func(*args, **kwargs)
    return wrapper

class MinhaClasse:
    @meu_decorador
    def meu_metodo(self):
        print("Método original")

obj = MinhaClasse()
obj.meu_metodo()
```

- 5. Decoradores de Classe

Dúvida Comum: Como criar decoradores que funcionam em classes inteiras, não apenas em métodos?

Decoradores que afetam classes inteiras podem adicionar novos atributos ou modificar comportamentos de todos os métodos da classe, não apenas métodos específicos. Isso é útil para aplicar funcionalidades gerais, como adicionar logging ou validação a todos os métodos da classe de uma só vez, sem alterar cada método individualmente. Por exemplo, um decorador pode adicionar um novo atributo a todas as instâncias da classe ou alterar o comportamento padrão dos métodos existentes.

Exemplo de código:

python

```python
def decorador_de_classe(cls):
    cls.novo_atributo = "Atributo adicionado!"
    return cls

@decorador_de_classe
class MinhaClasse:
    def __init__(self):
        pass

obj = MinhaClasse()
print(obj.novo_atributo)  # Imprime "Atributo adicionado!"
```

Conclusão

Decoradores são como superpoderes para suas funções e classes em Python. Eles podem ser um pouco complicados no início, mas com prática e atenção aos detalhes, você vai dominar essa ferramenta poderosa. Lembre-se de sempre usar functools.wraps, entender a ordem de execução e como passar argumentos corretamente.

Pronto para elevar o nível do seu código?

#Python #Decoradores #Programação #CódigoLimpo #DevLife