Aula 11

Resumo

Os decoradores no Python são uma poderosa ferramenta que permite a modificação ou o aprimoramento do comportamento de funções, métodos, classes ou módulos existentes. Eles são amplamente utilizados para adicionar funcionalidades extras a essas estruturas sem a necessidade de modificar o código-fonte original.

Os decoradores são implementados como funções que envolvem a definição da estrutura que será decorada. Eles recebem como argumento a função ou classe alvo e retornam uma função ou classe modificada. Dessa forma, quando a função ou classe é chamada, o decorador é automaticamente aplicado.

Existem diferentes maneiras de definir um decorador no Python. A forma mais comum é usando a sintaxe do "@" antes da definição da função decoradora. Por exemplo, o código abaixo demonstra a aplicação de um decorador a uma função:

```python
def decorador(func):
    """função decoradora"""
    def wrapper(*args, **kwargs):
        print('antes da função')
        func(*args, **kwargs)
        print('depois da função')
    return wrapper

# uso de decorador de forma braçal
def outra_funcao():
    """descrição da função"""
outra_funcao = decorador(outra_funcao)
outra_funcao()

# uso de decorador de forma automática
@decorador
def uma_funcao():
    """descrição da função"""
```

Os decoradores podem ser usados para diversas finalidades, como:

- Adicionar funcionalidades extras: Um decorador pode adicionar comportamentos adicionais a uma função ou método, como logging, validação de parâmetros, caching, tempo de execução, entre outros. Isso permite separar essas funcionalidades da lógica principal do código, tornando-o mais modular e de fácil manutenção;
- Modificar o comportamento: Um decorador pode modificar o comportamento de uma função ou método existente, alterando o valor de retorno, os argumentos recebidos ou até mesmo substituindo completamente a função original por outra. Isso possibilita adaptar o código para diferentes contextos ou requisitos específicos.
- Controle de acesso: Decoradores também podem ser usados para controlar o acesso a funções, métodos ou classes, verificando permissões ou autenticação antes de executar a estrutura decorada.
- Registrar informações: Decoradores podem ser utilizados para coletar informações sobre as funções, métodos ou classes em tempo de execução, possibilitando a criação de sistemas de metaprogramação ou ferramentas de análise de código.

Os decoradores são uma funcionalidade avançada do Python, mas podem ser uma ferramenta extremamente útil para melhorar a modularidade, a reutilização de código e a flexibilidade de uma aplicação. É possível adicionar ou modificar comportamentos de forma elegante e eficiente.

Métodos de Classe

Em Python, o método de classe é uma função definida dentro de uma classe que opera nos atributos e objetos dessa classe. É usado para definir o comportamento específico que as instâncias da classe devem ter. Os métodos de classe têm acesso aos atributos da classe e podem ser chamados tanto pelas instâncias da classe quanto pela própria classe.

Para criar um método de classe em Python, você deve definir uma função dentro da classe e decorá-la com o decorador @classmethod. O primeiro parâmetro de um método de classe é sempre a própria classe, por convenção chamado de cls (embora possa ser qualquer nome válido em Python). O método de classe pode então acessar os atributos da classe usando cls e pode criar novas instâncias da classe, se necessário.

Aqui está um exemplo que ilustra o conceito:

```python
class Carro:
    fabricante = "General Motors"  # Atributo de classe

    def __init__(self, modelo):
        self.modelo = modelo  # Atributo de instância

    @classmethod
    def info_fabricante(cls):
        print("Fabricante:", cls.fabricante)

    def info_modelo(self):
        print("Modelo:", self.modelo)

# Usando o método de classe
Carro.info_fabricante()  # Saída: Fabricante: General Motors

# Criando instâncias da classe
carro1 = Carro("Sedan")
carro1.info_modelo()  # Saída: Modelo: Sedan
carro1.info_fabricante()  # Saída: Fabricante: General Motors

carro2 = Carro("SUV")
carro2.info_modelo()  # Saída: Modelo: SUV
```

No exemplo acima, a classe Carro tem um atributo de classe chamado fabricante, que é compartilhado por todas as instâncias da classe. O método de classe info_fabricante acessa esse atributo de classe usando cls.fabricante e imprime seu valor.

Para chamar um método de classe, você pode usá-lo diretamente na classe, como Carro.info_fabricante(), ou em uma instância da classe, como carro1.info_fabricante(). Quando chamado em uma instância, a classe do objeto é automaticamente passada como o primeiro argumento para o método (ou seja, o parâmetro cls).

Métodos de Classe como Fábricas

Os métodos de classe são comumente usados para criar métodos de fábrica, que são métodos usados para criar e retornar novas instâncias da classe. Eles fornecem uma forma conveniente de criar objetos sem precisar chamar explicitamente o construtor.

Aqui estão dois exemplos de métodos de classe que funcionam como métodos de fábrica:

```python
class Retangulo:
    def __init__(self, comp, larg):
        self.comprimento = comp
        self.largura = larg

    @classmethod
    def criar_quadrado(cls, lado):
        return cls(lado, lado)

    @classmethod
    def criar_ret_proporcional(cls, ret_base, escala):
        return cls(ret_base.comprimento * escala, ret_base.largura *
            escala)

# Usando o método de fábrica criar_quadrado()
quadrado = Retangulo.criar_quadrado(5)
print(quadrado.comprimento, quadrado.largura)  # Saída: 5 5

# Usando o método de fábrica criar_retângulo_proporcional()
ret_base = Retangulo(4, 6)
ret_prop = Retangulo.criar_ret_proporcional(ret_base, 2)
print(ret_prop.comprimento, ret_prop.largura)  # Saída: 8 12
```

No exemplo acima, a classe Retangulo possui dois métodos de classe que funcionam como métodos de fábrica. O método criar_quadrado() cria um quadrado recebendo o valor de um lado e retorna uma nova instância da classe Retângulo com os valores de comprimento e largura iguais ao lado fornecido.

O método criar_ret_proporcional() cria um novo retângulo proporcional a partir de um retângulo base e um fator de escala. Ele recebe uma instância de Retangulo e um valor de escala, multiplica o comprimento e a largura do retângulo base pelo fator de escala e retorna uma nova instância da classe Retangulo com os valores atualizados.

Os métodos de classe também podem ser usados para criar métodos de inicialização alternativos, fornecendo diferentes formas de construir objetos da classe. Isso é útil quando você deseja permitir que os usuários da classe forneçam diferentes conjuntos de parâmetros para inicializar os objetos.

Aqui estão dois exemplos de métodos de classe que funcionam como métodos de inicialização alternativos:

```python
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    @classmethod
    def criar_com_ano_nasc(cls, nome, ano_nasc):
        idade = 2023 - ano_nasc
        return cls(nome, idade)

# Usando o método de inicialização alternativo
# criar_com_ano_nascimento()
pessoa1 = Pessoa.criar_com_ano_nasc("João", 1985)
print(pessoa1.nome, pessoa1.idade)  # Saída: João 38
```

No exemplo acima, a classe Pessoa possui um método de inicialização padrão __init__() que requer os parâmetros nome e idade. Além disso, ela possui um método de classe criar_com_ano_nasc() que permite criar uma instância

de Pessoa fornecendo o nome e o ano de nascimento. Esse método calcula a idade com base no ano de nascimento e o ano de 2023 e, em seguida, chama o construtor __init__() para criar o objeto.

Os métodos de classe também podem ser usados para fornecer funcionalidades relacionadas à classe, que são métodos que operam em nível de classe e não requerem acesso direto aos atributos de instância. Eles podem ser úteis para realizar cálculos, retornar informações ou executar ações específicas relacionadas à classe em si.

Aqui está um exemplo de método de classe que fornece funcionalidade relacionada à classe:

```python
class Matematica:
    @classmethod
    def soma(cls, a, b):
        return a + b

    @classmethod
    def multiplicacao(cls, a, b):
        return a * b

    @classmethod
    def fatorial(cls, n):
        if n < 0:
            raise ValueError("O fatorial não está definido para números
            negativos.")
        if n == 0 or n == 1:
            return 1
        return n * cls.fatorial(n - 1)

# Usando os métodos de classe
resultado_soma = Matematica.soma(5, 3)
print(resultado_soma)  # Saída: 8

resultado_multiplicacao = Matematica.multiplicacao(4, 6)
print(resultado_multiplicacao)  # Saída: 24

resultado_fatorial = Matematica.fatorial(5)
print(resultado_fatorial)  # Saída: 120
```

No exemplo acima, a classe Matematica possui três métodos de classe: soma(), multiplicacao() e fatorial(). Esses métodos fornecem funcionalidades matemáticas relacionadas à classe sem precisar acessar os atributos de instância.

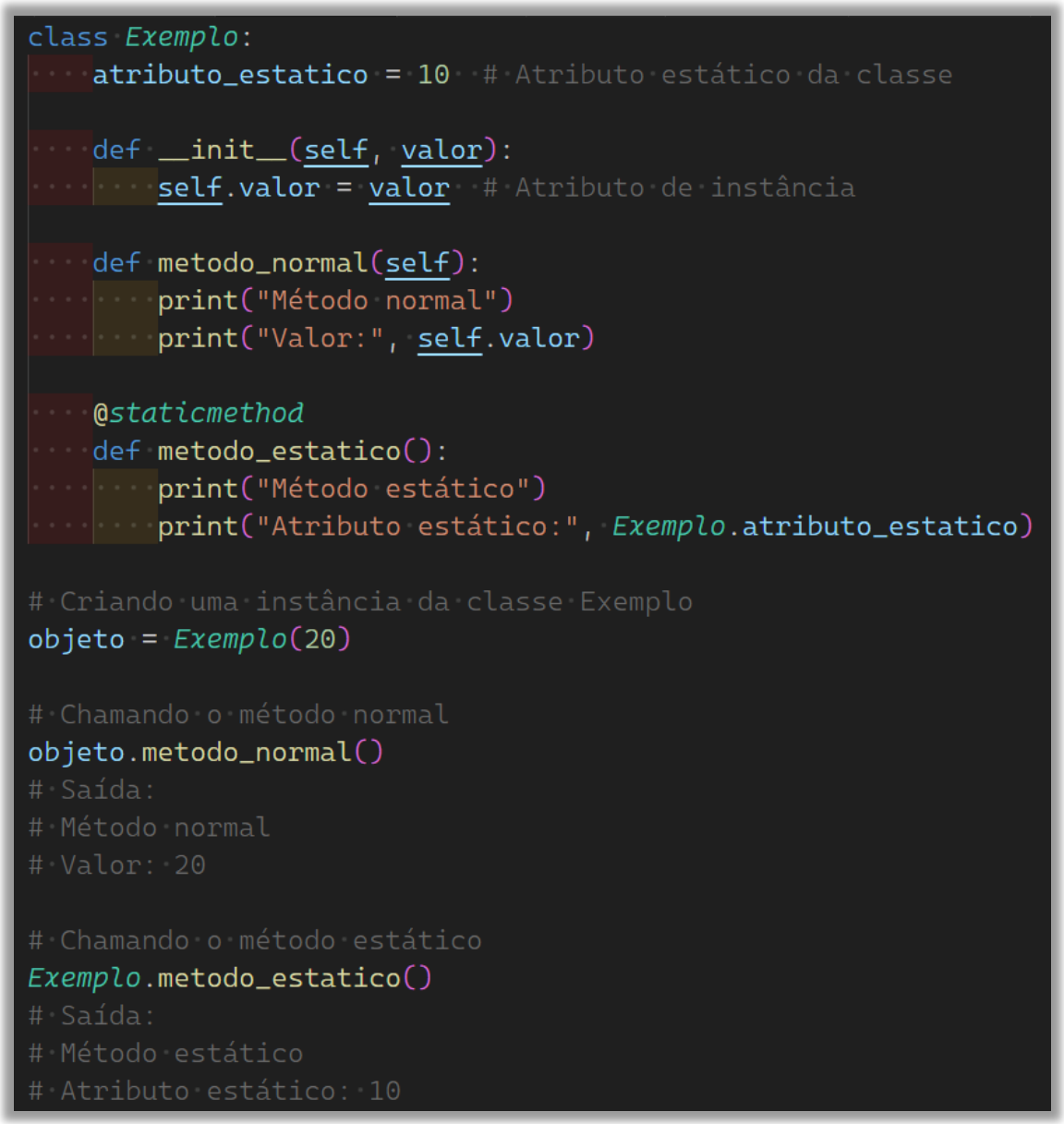
O método soma() recebe dois valores a e b e retorna a soma deles. O método multiplicacao() recebe dois valores a e b e retorna o resultado da multiplicação. O método fatorial() calcula o fatorial de um número n usando recursão.

Esses métodos de classe fornecem funcionalidades relacionadas à classe Matematica (e as vistas logo acima) e podem ser usados sem a necessidade de criar instâncias da classe. Eles encapsulam operações matemáticas específicas e tornam mais conveniente realizar cálculos relacionados dentro do contexto da classe.

Métodos Estáticos

Em Python, os métodos estáticos são funções definidas dentro de uma classe que não requerem uma instância da classe para serem chamados. Eles são úteis para agrupar funções que estão logicamente relacionadas à classe, mas não dependem de nenhum atributo de instância específico. Os métodos estáticos são declarados usando o decorador @staticmethod antes da definição do método.

Aqui está um exemplo básico para ilustrar como os métodos estáticos funcionam em Python:



```python
class Exemplo:
    atributo_estatico = 10  # Atributo estático da classe

    def __init__(self, valor):
        self.valor = valor  # Atributo de instância

    def metodo_normal(self):
        print("Método normal")
        print("Valor:", self.valor)

    @staticmethod
    def metodo_estatico():
        print("Método estático")
        print("Atributo estático:", Exemplo.atributo_estatico)

# Criando uma instância da classe Exemplo
objeto = Exemplo(20)

# Chamando o método normal
objeto.metodo_normal()
# Saída:
# Método normal
# Valor: 20

# Chamando o método estático
Exemplo.metodo_estatico()
# Saída:
# Método estático
# Atributo estático: 10
```

Neste exemplo, temos a classe Exemplo com um atributo estático chamado atributo_estatico e dois métodos: metodo_normal e metodo_estatico.

O método metodo_normal é um método de instância, pois possui o parâmetro self e pode acessar os atributos específicos de cada instância. Ao chamar esse método em uma instância objeto, ele imprime o valor do atributo valor específico dessa instância.

O método metodo_estatico, por outro lado, é um método estático. Ele não possui o parâmetro self e não pode acessar atributos de instância diretamente. No entanto, ele pode acessar o atributo estático atributo_estatico da classe usando o nome da classe (Exemplo.atributo_estatico). Esse método pode ser chamado diretamente na classe Exemplo sem criar uma instância e imprime o valor do atributo estático.

Os métodos estáticos são úteis em situações em que você precisa definir funções relacionadas à classe, mas que não dependem de atributos específicos de instância. Eles podem ser usados para agrupar funções de utilidade, implementar operações independentes de estado ou até mesmo criar fábricas de objetos. No entanto, é importante observar que os métodos estáticos não têm acesso aos atributos de instância e não podem ser substituídos por subclasses.

Mais alguns exemplos:

```python
class Matematica:
    @staticmethod
    def soma(a, b):
        return a + b

    @staticmethod
    def multiplicacao(a, b):
        return a * b

# Chamando os métodos estáticos diretamente na classe
print(Matematica.soma(5, 3))  # Saída: 8
print(Matematica.multiplicacao(4, 6))  # Saída: 24
```

Nesse exemplo, a classe Matematica define métodos estáticos soma e multiplicacao que realizam operações matemáticas simples. Eles não dependem de nenhum atributo de instância e podem ser chamados diretamente na classe, sem a necessidade de criar uma instância.

```python
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    def apresentar(self):
        print(f"Olá, meu nome é {self.nome} e tenho {self.idade} anos.")

    @staticmethod
    def criar_pessoa(nome, idade):
        return Pessoa(nome, idade)

# Criando uma instância usando o método estático como uma fábrica
pessoa = Pessoa.criar_pessoa("João", 25)
pessoa.apresentar()  # Saída: Olá, meu nome é João e tenho 25 anos.
```

Nesse exemplo, a classe Pessoa tem um método estático chamado criar_pessoa que serve como uma fábrica para criar objetos Pessoa. Em vez de chamar o construtor diretamente, chamamos o método estático criar_pessoa passando os parâmetros desejados, que retorna uma nova instância da classe Pessoa.

```python
class Utilidades:
    @staticmethod
    def inverter_string(texto):
        return texto[::-1]

# Chamando o método estático para inverter uma string
string_original = "Python é divertido!"
string_invertida = Utilidades.inverter_string(string_original)
print(string_invertida)  # Saída: "!otnivrid é nohtyP"
```

Nesse exemplo, a classe Utilidades possui um método estático inverter_string que recebe uma string e retorna a string invertida. Esse método é útil para tarefas comuns, como inverter uma sequência de caracteres, e pode ser chamado diretamente na classe.

Exercícios para Praticar

1. Crie um decorador que imprima "Iniciando função" antes da execução de uma função e "Finalizando função" após a execução.
2. Implemente um decorador que calcule o tempo de execução de uma função e exiba-o após a execução.
3. Crie um decorador que aceite um argumento inteiro "n" e execute a função decorada "n" vezes.
4. Desenvolva um decorador que cacheie o resultado de uma função, armazenando o valor de retorno e retornando-o diretamente nas próximas chamadas, evitando a execução da função novamente.
5. Implemente um decorador que verifique se todos os argumentos passados para uma função são do tipo inteiro.
6. Crie um decorador que converta automaticamente o valor de retorno de uma função para uma string em letras maiúsculas.
7. Desenvolva um decorador que verifique se um usuário está autenticado antes de executar uma função, exibindo uma mensagem de erro caso contrário.
8. Implemente um decorador que limite o tempo máximo de execução de uma função, interrompendo-a e exibindo uma mensagem de timeout se o tempo limite for excedido.
9. Crie um decorador que registre o histórico de chamadas de uma função, armazenando os argumentos e valores de retorno.

10. Crie uma classe chamada Pessoa com um método de classe que retorne a quantidade total de pessoas criadas.
11. Implemente uma classe Círculo com um método de classe que calcule a área de um círculo com base no raio fornecido.
12. Crie uma classe Banco com um método de classe que retorne a taxa de juros atual do banco.
13. Implemente uma classe Calculadora com métodos de classe para adição, subtração, multiplicação e divisão de dois números.
14. Crie uma classe Data com um método de classe que valide se uma data fornecida é válida ou não.
15. Implemente uma classe Triângulo com um método de classe que verifique se um triângulo é equilátero, isósceles ou escaleno.
16. Crie uma classe Animal com um método de classe que conte o número total de animais criados.
17. Implemente uma classe Tempo com um método de classe que converta horas em minutos.
18. Crie uma classe Lista com um método de classe que ordene uma lista de números fornecidos.
19. Implemente uma classe Estudante com um método de classe que retorne a média das notas de um grupo de estudantes.
20. Crie uma classe Funcionario com um método de classe que calcule o salário líquido com base no salário bruto e nos descontos.
21. Implemente uma classe Matriz com um método de classe que multiplique duas matrizes.
22. Crie uma classe Conversor com um método de classe que converta uma temperatura de Celsius para Fahrenheit.
23. Implemente uma classe GeradorSenha com um método de classe que gere uma senha aleatória de tamanho especificado.
24. Crie uma classe Vetor com um método de classe que calcule o produto escalar entre dois vetores.
25. Implemente uma classe Pessoa com um método de classe que verifique se uma pessoa é maior de idade com base na data de nascimento fornecida.
26. Crie uma classe CarrinhoCompras com um método de classe que calcule o valor total das compras com base nos itens adicionados.
27. Implemente uma classe Geometria com um método de classe que calcule o perímetro de um polígono regular com base no número de lados e no comprimento dos lados.
28. Crie uma classe Texto com um método de classe que conte o número de palavras em um texto fornecido.
29. Implemente uma classe CalculadoraFinanceira com um método de classe que calcule o valor futuro de um investimento com base no valor presente, taxa de juros e período.

30. Crie uma classe chamada Calculadora com um método estático soma que recebe dois números como parâmetros e retorna a soma deles.
31. Crie uma classe chamada StringUtils com um método estático contar_vogais que recebe uma string como parâmetro e retorna o número de vogais presentes nela.
32. Crie uma classe chamada Conversor com um método estático metro_para_km que recebe um valor em metros como parâmetro e retorna o valor equivalente em quilômetros.

33. Crie uma classe chamada DataUtils com um método estático eh_bissexto que recebe um ano como parâmetro e verifica se é um ano bissexto. O método deve retornar True se for bissexto e False caso contrário.
34. Crie uma classe chamada Matematica com um método estático maior_numero que recebe uma lista de números como parâmetro e retorna o maior número presente na lista.
35. Crie uma classe chamada EmailUtils com um método estático validar_email que recebe um endereço de email como parâmetro e verifica se é um email válido. O método deve retornar True se for válido e False caso contrário.
36. Crie uma classe chamada IOUtils com um método estático ler_arquivo que recebe o nome de um arquivo como parâmetro e retorna o conteúdo desse arquivo.
37. Crie uma classe chamada Geometria com um método estático area_retangulo que recebe a largura e altura de um retângulo como parâmetros e retorna a área desse retângulo.
38. Crie uma classe chamada ArrayUtils com um método estático inverter_lista que recebe uma lista como parâmetro e retorna uma nova lista com os elementos invertidos.
39. Crie uma classe chamada ConversorTemperatura com um método estático celsius_para_fahrenheit que recebe uma temperatura em graus Celsius como parâmetro e retorna o valor equivalente em graus Fahrenheit.
40. Crie uma classe chamada ValidadorCPF com um método estático validar_cpf que recebe um CPF como parâmetro e verifica se é um CPF válido. O método deve retornar True se for válido e False caso contrário.
41. Crie uma classe chamada Estatisticas com um método estático media que recebe uma lista de números como parâmetro e retorna a média desses números.
42. Crie uma classe chamada Palindromo com um método estático verificar_palindromo que recebe uma palavra como parâmetro e verifica se é um palíndromo. O método deve retornar True se for um palíndromo e False caso contrário.
43. Crie uma classe chamada GeradorSenha com um método estático gerar_senha que gera uma senha aleatória com 8 caracteres. A senha deve conter letras maiúsculas, letras minúsculas e números.
44. Crie uma classe chamada ConversorMoeda com um método estático real_para_dolar que recebe um valor em reais como parâmetro e retorna o valor equivalente em dólares.
45. Crie uma classe chamada ListaUtils com um método estático elemento_repetido que recebe uma lista como parâmetro e retorna o primeiro elemento que se repete.
46. Crie uma classe chamada MatrizUtils com um método estático soma_matrizes que recebe duas matrizes como parâmetros e retorna a matriz resultante da soma das duas.
47. Crie uma classe chamada ValidadorSenha com um método estático validar_senha que recebe uma senha como parâmetro e verifica se é uma senha válida. A senha deve conter pelo menos 8 caracteres, incluindo letras maiúsculas, letras minúsculas e números.
48. Crie uma classe chamada Fatorial com um método estático calcular_fatorial que recebe um número como parâmetro e retorna o fatorial desse número.
49. Crie uma classe chamada ConversorPeso com um método estático quilo_para_libra que recebe um valor em quilos como parâmetro e retorna o valor equivalente em libras.