Métodos mágicos

terça-feira, 1 de novembro de 2022 14:47

Os metodos mágicos modificam o comportamento da sua classe

Método __new__

O método __new__ funciona como um construtor assim como o __init__, quando criamos um método new o init "para" de funcionar.

Use o ***__new__*** quando você precisar controlar a criação de uma nova instancia da classe. Use o ***__init__*** quando você precisar controlar a inicialização de uma nova instancia.

Exemplo:

```
class A:
    def __new__(cls, *args, **kwargs):
        print("Eu sou o new")
        return object().__new__(cls)

    def __int__(self):
        print("Estou funcionando")


a = A()
```

```
C:\Users\geron\AppData\Local\Programs\Python\Python310\python.ex
Eu sou o new

Process finished with exit code 0
```

para criar atributos dentro do new devemos usar o cls.

Exemplo:

```
class A:
    def __new__(cls, *args, **kwargs):
        cls.nome = "Geronimo"

        return object().__new__(cls)
```

```
a = A()
print(a.nome)
```

```
C:\Users\geron\AppData\Local\Programs\Python\Python310\py
Geronimo

Process finished with exit code 0
```

a partir disso também podemos criar metodos dentro do método new.

Exemplo:

quando instanciamos a função no a e chamamos ela, é como se tivesse um self dentro do a.mensagem(), logo devemos usar os args e kwargs.

quando queremos usar um design patterns chamado singleton:

"Singleton é um padrão de projeto de software. Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao seu objeto. Nota linguística: O termo vem do significado em inglês para um conjunto que contenha apenas um elemento."

Logo quando instanciamos um objeto a classe, os proximos objetos que forem instânciados serão iguais aos primeiro, sendo apenas uma cópia deste.

Exemplo:

estamos usando o design pattern hasattr para criar cópias da instâncias iguais as outras.

Método __call__

Outro método importante é o método __call__ que serve usando a class para funcionar como uma função, dessa forma podemos chamar a instância apenas com os argumentos dentro dela, podendo fazer tudo que quiser dentro dela, a classe continua funcionando do mesmo jeito como classe porém pode ser usado como função apenas se chamada como função pela instância a().

```python
class A:
    def __init__(self):
        print("Estou funcionando")

    def __call__(self, *args, **kwargs):
        print(args)
        print(kwargs)

a = A()
a(1,2,3,4,5,nome = "luiz")
```

```
C:\Users\geron\AppData\Local\Programs\Python\Py
Estou funcionando
(1, 2, 3, 4, 5)
{'nome': 'luiz'}

Process finished with exit code 0
```

Podendo usar também para multiplicar e fazer outras coisas mais.

```python
class A:
    def __init__(self):
        print("Estou funcionando")

    def __call__(self, *args, **kwargs):
        resultado = 1

        for i in args:
            resultado *= i
        print(resultado)
a = A()
a(1,2,3,4,5,nome = "luiz")
```

podendo usar o print ou então usar um objeto para a função.

```python
class A:
    def __init__(self):
        print("Estou funcionando")

    def __call__(self, *args, **kwargs):
        resultado = 1

        for i in args:
            resultado *= i
        return resultado
a = A()
valor = a(1,2,3,4,)
print(valor)
```

a classe ainda pode ser usada também da forma convencional podendo ser chamada os métodos dentro da classe.

Ex:

```python
class A:
    #construtor
    def __init__(self):
        pass

    def __call__(self, *args, **kwds):
        #Criando variável dentro do metodo call
        valor = 1

        #iterando sobre os parametros args que vou passar
        for v in args:
            valor += v
        return valor

    #Criando um metodo normal
    def falando(self):
        print("Estou funcionando tbm da mesma forma.")


a = A()
valor = a(1,2,3,4,5,96)
a.falando()
print(valor)
```

O método falando funciona da mesma forma igualmente.

__setattr__

O método mágico setattr toda vez que configurar um atributo novo na sua classe ele vai ser chamado.

Ex:

```python
class Teste:
    def __init__(self):
        pass

    def __setattr__(self, key, value):
        #O setattr dessa forma armazena o valor que foi criado pelo atributo a classe.
        self.__dict__[key] = value

    def falar(self):
        print(self.nome)

a = Teste()
#Criando atributo pela instância
a.nome = "Geronimo"
a.falar()
```

O setattr armazena o atributo criado pela instância a classe, podendo ser usado dentro da classe com o self.

__del__

O método __del__ é na teoria chamado sempre que chamamos a classe e quanto encerra é coletado o objeto, no caso a instância.

Ex:

```python
# O método __del__ na teoria funciona da seguinte forma, quando chega ao fim do uso do objeto
# ele é coletado.

class Teste:

    def __init__(self):
        pass

    def __del__(self):
        print("Objeto Coletado.")


test = Teste()
#Aqui quando encerrar o objeto test é coletado.
```

__str__

O método mágico __str__ é usado quando queremos chamar nossa classe como string, podemos então modificar o comportamento.

Ex:

```python
#O método __str__ é chamado sempre que eu tentar chamar minha classe como uma string

class Teste:

    def __init__(self):
        pass

    def __str__(self):
        return "Estou usando esse método pois vou usar o print"


objeto = Teste()

#Imprimindo a instância da classe, se o metodo __str__ não estivesse na classe o resultado seria assim:
#<__main__.Teste object at 0x0000019A9B75B100>
print(Teste())
#Acontece o mesmo com a instância
print(objeto)
```

__len__

O método len é executado sempre que precisar contar algo.

```python
#Sempre que usamos a função len com uma classe para fazer a contagem
# é como se tivessemos usando o método mágico __len__.

class Teste:

    def __init__(self):
        pass


    def __len__(self):
        return 55


a = Teste('nome',12,132)

#O valor do __len__ vai ser o valor que estiver dentro do método, porém é usado
#quando temos objetos dentro desse objeto a como por exemplo, listas, mostrando o valor
#de quantos objetos tem dentro da classe;

print(len(a))
```