O Poder dos __métodos_mágicos__

Olá!
Eu sou o Douglas Silva

Sou formado em Ciência da Computação pela UFRPE.

Atualmente sou Backend na Olist.

E tenho trabalhado com Python a mais de 6 anos.

O que são métodos?

- Basicamente são funções internas a uma classe, definidas pelo programador, que modelam o comportamento de seus objetos.

```
import collections
import random

Carta = collections.namedtuple('Carta', ['valor', 'naipe'])

class Baralho:
    def __init__(self):
        self._cartas = [
            Carta(valor, naipe)
            for naipe in ["ouros", "espadas", "copas", "paus"]
            for valor in ["2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"]
        ]

    def seleciona_uma_carta(self):
        """Método que seleciona uma carta aleatória do baralho instanciado."""
        return random.choice(self._cartas)

>>> baralho = Baralho()

>>> baralho.seleciona_uma_carta()
Carta(valor='4', naipe='hearts')
```

Certo, mas o que são métodos mágicos?

- São métodos especiais pré definidos pela API do Python e que são chamados diretamente pelo interpretador para executar operações básicas.
- Geralmente são acionados por sintaxe especial (açúcares sintáticos).

Nomenclatura dos métodos mágicos

- Os nomes dos métodos mágicos sempre são escritos com duplo underlines a esquerda e a direita, por isso podem também ser chamados de **dunder methods**:
  1 - __add__(dunder add)
  2 - __nome__ (dunder nome)

```
>>> inteiro = 1

>>> type(inteiro)
int

>>> dir(inteiro)
[
    '__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
    ... # Vários outros metodos especiais!
    'to_bytes'
]
```

Cuidado

- Você não pode criar métodos especiais. O máximo que você pode é fazer a sobrecarga dos métodos já pré definidos pela API do Python.
- Desta forma, não é uma boa prática criar métodos que comecem e terminem com duplo underlines.

Como os métodos mágicos são invocados ?

- Geralmente são chamado por sintaxe especial (açúcares sintáticos) tais como:
  1. Operadores aritméticos: +, -, *, /, ...
  2. Operadores lógicos: <, >, ==, !=, ...
  3. Slice listas: baralho[1:10]
  4. Funções Built-in: abs(), len(), delattr(), str(), repr(), ...
- Esses açúcares sintáticos tem por finalidade tornar as construções mais fáceis de serem lidas e expressas.

Como os métodos mágicos são invocados ?

obj1 + obj2	obj1.__add__(obj2)
obj1 - obj2	obj1.__sub__(obj2)
obj1 * obj2	obj1.__mul__(obj2)
obj1 == obj2	obj1.__eq__(obj2)
len(obj)	obj.__len__()
str(obj)	obj.__str__()
obj[1]	obj.__getitem__(1)

Quantidade de assentos no avião 👎

```
class Aviao:
    def __init__(self):
        self.assentos = [
            f"{numero}{letra}" for numero in range(10) for letra in "ABCDEF"
        ]

>>> sum(1 for _ in Aviao().assentos) # 1ª opção :/
60

>>> count = 0  # 2ª opção :/
>>> for _ in Aviao().assentos: count += 1
>>> count
60

>>> len(Aviao().assentos)
60
```

Quantidade de assentos no avião 👋

```python
class Aviao:
    def __init__(self):
        self.assentos = [
            f"{numero}{letra}" for numero in range(10) for letra in "ABCDEF"
        ]

    def __len__(self):
        """Sobrecarga do método mágico __len__ para que possamos usar a função built-in len()"""
        return len(self.assentos)

>>> len(Aviao()) # :)
60
```

Selecionando um assento no avião 😕

```
class Aviao:
     def __init__(self):
        self.assentos = [
            f"{numero}{letra}" for numero in range(10) for letra in "ABCDEF"
        ]

>>> aviao = Aviao()

>>> aviao.assentos[10]
'1E'

>>> aviao.assentos[:10]
['0A', '0B', '0C', '0D', '0E', '0F', '1A', '1B', '1C', '1D']

>>> for assento in aviao.assentos: print(assento)
0A
0B
...
```

Selecionando um assento no avião 🥰

```
class Aviao:
     def __init__(self):
        self.assentos = [
            f"{numero}{letra}" for numero in range(10) for letra in "ABCDEF"
        ]

    def __getitem__(self, posicao):
        """
        Sobrecarga do método mágico __getitem__ para que o Aviao se comporte como uma lista
        """
        return self.assentos[posicao]

>>> aviao = Aviao()

>>> aviao[10]
'1E'

>>> aviao[:10]
['0A', '0B', '0C', '0D', '0E', '0F', '1A', '1B', '1C', '1D']

>>> for assento in aviao: print(assento)
0A
0B
...
```

Pagamento confuso 

```
class Pagamento:
    conversao = {"$": 0.2, "R$": 5}

    def __init__(self, valor, moeda):
        self.valor = valor
        self.moeda = moeda

    def __repr__(self):
        """Sobrecarga do método mágico __repr__ para representar objeto amigavelmente."""
        return f"Pagamento({self.moeda} {self.valor})"

>>> pagamento1, pagamento2, pagamento3 = Pagamento(10, "R$"), Pagamento(10, "R$"),  Pagamento(10, "$")

>>> Pagamento(pagamento1.valor + pagamento2.valor, "R$") # Complicado né ?!?!
Pagamento(R$ 20)

>>> Pagamento(pagamento1.valor + pagamento3.valor * Pagamento.conversao["R$"]  , "R$") # OOO QUÊÊÊ ???
Pagamento(R$ 60)
```

Pagamento stonks 

```
class Pagamento:
    ...
    def __add__(self, pagamento):
        """Sobrecarga do método mágico __add__ para somarmos 2 pagamentos com logica de currency."""
        if self.moeda == pagamento.moeda:
            return Pagamento(self.valor + pagamento.valor, self.moeda)
        return Pagamento(self.valor + self.conversao[self.moeda] * pagamento.valor, self.moeda)

>>> pagamento1, pagamento2, pagamento3 = Pagamento(10, "R$"), Pagamento(10, "R$"),  Pagamento(10, "$")

>>> pagamento1 + pagamento2  # MAR QUE COISA LINDA!!
Pagamento(R$ 20)

>>> pagamento1 + pagamento3  # HEIN!? PRA ONDE FOI A COMPLEXIDADE???
Pagamento(R$ 60)

>>> pagamento3 + pagamento1  # EITA JA CONVERTEU DE ACORDO COM O PRIMEIRO OPERANDO!!!!
Pagamento($ 12.0)
```

Comparação confusa 😢

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

>>> pessoa1 = Pessoa(nome = "Antonio Nunes", cpf="08410663596")

>>> pessoa2 = Pessoa(nome = "Antonio Nunes", cpf="08410663596")

>>> pessoa1 == pessoa2
False

>>> id(pessoa1), id(pessoa2)
(139646104189488, 139646102943488)

>>> pessoa1.cpf == pessoa2.cpf
True
```

Comparação easy peasy 🍋

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

    def __eq__(self, obj):
        """Sobrecarga do método mágico __eq__ para fornecer comparação de objetos"""
        return self.cpf == obj.cpf

>>> pessoa1 = Pessoa(nome = "Antonio Nunes", cpf="08410663596")

>>> pessoa2 = Pessoa(nome = "Antonio Nunes", cpf="08410663596")

>>> id(pessoa1), id(pessoa2)
(139646017014800, 139646104105024)

>>> pessoa1 == pessoa2
True
```

Mais Alguns métodos mágicos

```
alguns_metodos_magicos = [
    __pos__, __neg__, __abs__, __invert__, __round__, __floor__, __ceil__, __trunc__,
    __iadd__, __isub__, __imul__, __ifloordiv__, __idiv__, __itruediv__, __imod__, __ipow__,
    __ilshift__, __irshift__, __iand__, __ior__, __ixor__, __int__, __float__, __complex__,
    __oct__, __hex__, __index__, __trunc__, __str__, __repr__, __unicode__, __format__,
    __hash__, __nonzero__, __dir__, __sizeof__, __getattr__, __setattr__, __delattr__,
    __add__, __sub__, __mul__, __floordiv__, __div__, __mod__, __pow__, __lt__, __le__,
    __eq__, __ne__, __ge__ # E MUITOS OUTROS!
]
```

Documentação de todos métodos mágicos

Considerações Finais

Obrigado !

Alguma pergunta ? :)

Referências

- Python Data Model Documentação oficial
- Python Built-in Functions
- Luciano Ramalho Fluent Python 2ª edição (Capitulo 1 The Python Data Model)
- Rafael Henrique Magic Methods