Live de Python #13

dunders

Roteiro

- Objetos em Python
- A função **dir()**
- Que raios é __dunder__

- - -

- Criando alguns objetos

Objetos em Python (Uma introdução inicial ao datamodel)

Todos os objetos em python (e vamos dizer quem tudo são objetos) contém 3 propriedade básicas.

- Identidade - **id()** - obj_1 **is** obj_2 - **IMUTÁVEL**
- Tipo - **type() - IMUTÁVEL**
- Valor - **MUTÁVEL/IMUTÁVEL** - (Referente ao seu tipo)

Exemplo de tipos mutáveis x imutáveis

Mutáveis

- Listas
- Dicionários
- Conjuntos

Imutáveis

- Tuplas
- Números - int, float, complex
- Strings
- Conjuntos congelados

Códigos de exemplo [0]

```
In [1]: # Definição do objeto 'set'

In [2]: conjunto = set()

In [3]: id(conjunto)  # Identidade
Out[3]: 140309515145000

In [4]: type(conjunto)  # tipo
Out[4]: set

In [5]: conjunto  # valor
Out[5]: set()
```

Códigos de exemplo [1]

```
In [6]: conjunto.add(4)  # modificação do valor

In [7]: id(conjunto)
Out[7]: 140309515145000

In [8]: conjunto  # valores
Out[8]: {4}
```

Códigos de exemplo [2]

```
In [12]: string = 'Live de Python'

In [13]: id(string)
Out[13]: 140309498722416

In [14]: type(string)
Out[14]: str

In [15]: string + ' #13'
Out[15]: 'Live de Python #13'

In [16]: string
Out[16]: 'Live de Python'
```

Códigos de exemplo [3]

```
In [12]: string = 'Live de Python'

In [13]: id(string)
Out[13]: 140309498722416

In [14]: type(string)
Out[14]: str

In [15]: string + ' #13'
Out[15]: 'Live de Python #13'

In [16]: string
Out[16]: 'Live de Python'
```

Códigos de exemplo [4]

```
In [17]: id(string)
Out[17]: 140309498722416

In [18]: id(string + ' Live #13')
Out[18]: 140309533647976
```

A função dir()

- Se chamada sozinha, retorna as variáveis do escopo local;
- Caso seja chamada com um objeto, retorna os métodos/atributos do objeto chamado.

A função dir()

```
In [23]: class teste:
    ...:     pass
```

```
In [24]: dir(teste)
Out[24]:
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
```

```
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__']
```

De onde vem e o que são esse _métodos_?

De onde vem esses métodos?

Esse, muitos, métodos são incorporados de um objeto comum em python chamado ‘object’ e eles definem uma interface padrão para que um objeto seja um objeto.

```
In [28]: set(dir(object)) - set(dir(teste))
Out[28]: set()
```

De onde vem esses métodos?

Esse, muitos, métodos são incorporados
python chamado ‘object’ e eles definem
para que um objeto seja um objeto.

Não existe diferença entre os métodos/atributos de object e de nossa classe inicial

```
In [28]: set(dir(object)) - set(dir(teste))
Out[28]: set()
```

Tá bom, mas o que são esses _?

Métodos especiais ou dunders

- São métodos invocados (callable) pelo próprio interpretador python para que os objetos criados possam seguir/usar os artifícios usados pela própria linguagem.
- Não são questões de contingência para que o objeto manipulado se comporte como um objeto nativo, todas as implementações do python também usam esse métodos (como vimos em object (ele é default para objetos nativos também)

Tipos de dunders

___

- Customização básica
  - `__new__, __init__, __del__, __str__, __repr__, __bool__, ...`
- Customização de acesso aos atributos
  - `__getattr__, __getattribute__, __dir__, ...`
- Emulação de objetos invocáveis
  - `__call__`
- Emulação de containers
  - `__len__, __getitem__, __setitem__, __contains__, ...`
- Emulação de tipos numéricos
  - `__add__, __sub__, __mul__, __div__, ...`
- E muitos outros ...
  - https://docs.python.org/3/reference/datamodel.html#special-method-names

Tipos de dunders

___

- Customização básica
  - __new__, **__init__**, __del__, __str__, **__repr__**, __bool__, ...
- Customização de acesso aos atributos
  - __getattr__, __getattribute__, __dir__, ...
- Emulação de objetos invocáveis
  - **__call__**
- Emulação de containers
  - **__len__**, **__getitem__**, __setitem__, __contains__, ...
- Emulação de tipos numéricos
  - **__add__**, __sub__, __mul__, __div__, ...
- E muitos outros ...
  - https://docs.python.org/3/reference/datamodel.html#special-method-names

CODEEEEEEEEEEEEEE!!!