Classes e Objetos

Exemplo de uma classe

Criamos uma classe para representar contas bancárias, com os seguintes atribuitos: * titular - nome da pessoa titular da conta * saldo - saldo da conta * id - o número da conta

Inserimos ainda métodos chamados `deposita` e `descreve`.

O atributo `titular` vai ser de objetos da classe *Pessoa*

```
class Conta(object):
    def __init__(self, dono: Pessoa,  numero: str, saldo = 0):
        self.titular = dono
        self.saldo = saldo
        self.id = numero

    def deposita(self, valor):
        if valor > 0:
            self.saldo += valor

    def descreve(self):
        print(f"Conta {self.id} da titular {self.titular} com saldo de
        {self.saldo}")



class Pessoa(object):
    def __init__(self, nome, RG):
        self.nome = nome
        self.RG = RG

    def __str__(self):
        return self.nome

class ContaInvestimento(Conta):
    def __init__(self):
        __super__.__init__()
```

Como usar a classe

Criando objetos

```
maria = Pessoa("Maria 1", "581176173491")
gabriel = Pessoa("Gabriel 3", "98181998")

c1 = Conta(maria,  "7878-7", 1000)
c2 = Conta(gabriel, "1539-9")
```

Acessando atributos

Para acessar atributos usamos o operador · :

```
c1.saldo = 700


c2.saldo -=50
```

Chamando métodos

As funções que pertencem às classes são chamadas de *métodos*

```
c1.descreve()
```

```
Conta 7878-7 da titular Maria 1 com saldo de 1000
```

Dicas

dir

O comando 'dir' retorna uma lista de tudo que existe na classe

```
c1.deposita(150)

print(c1.saldo)
```

```
1150
```

```
dir(c1)
```

```
['__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__',
 'banana',
 'deposita',
 'id',
 'saldo',
 'titular']
```

help

O comando `help()` explica brevemente a classe:

```
help(Conta)
```

```
Help on class Conta in module __main__:

class Conta(builtins.object)
 |  Conta(dono: __main__.Pessoa, numero, saldo=0)
 |
 |  Methods defined here:
 |
 |  __init__(self, dono: __main__.Pessoa, numero, saldo=0)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  deposita(self, valor)
 |
 |  descreve(self)
 |
```

```
 |  ----------------------------------------------------------------------
----
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

type

O comando `type` diz a qual classe pertence um objeto

Extra - exemplo do uso do throw

Neste código fizemos a validação do argumento `numero` passando-o à função `int()`.

Caso o argumento não possa ser convertido em inteiro, disparamos uma exceção.

```
class Conta(object):
    def __init__(self, dono,  numero, saldo = 0):
        self.titular = dono
        self.id = int(numero)

        try:
            self.saldo = int(saldo)
        except:
            raise ValueError

    def deposita(self, valor):
        if valor > 0:
            self.saldo += valor
```

Caso tentássemos passar um argumento que não pode ser convertido em int:

```
ce = Conta("Ian", "7623", "chkocol")
```

O programa terminaria em erro com a seguinte mensagem:

```
---------------------------------------------------------------------
        -----
ValueError                                Traceback (most recent call
        last)
Input In [18], in Conta.__init__(self, dono, numero, saldo)
```

```
      6 try:
----> 7     self.saldo = int(saldo)
      8 except:

ValueError: invalid literal for int() with base 10: 'chkocol'

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call
        last)
Input In [19], in <cell line: 1>()
----> 1 ce = Conta("Ian", "7623", "chkocol")

Input In [18], in Conta.__init__(self, dono, numero, saldo)
      7     self.saldo = int(saldo)
      8 except:
----> 9     raise ValueError

ValueError:
```