class5

December 11, 2021

1 Introdução a Python

1.1 Aula 5

2 Sumário

- Classes e objetos
- Erros e exceções
- Iteradores e geradores
- Exercícios

3 Classes e objetos

[1]:
```
class Person:

    def __init__(self, name, age, sex):
        self.name = name
        self.age = age
        self.sex = sex
```

[2]:
```
person = Person("John", 35, "male")

person.name, person.age, person.sex
```

[2]:
```
('John', 35, 'male')
```

[3]:
```
class Point:

    n = 0

    def __init__(self, x, y):
        self.x = x
        self.y = y
        Point.n += 1
```

[4]:
```
point1 = Point(1,2)

point1.x, point1.y, point1.n
```

[4]:
```
(1, 2, 1)
```

[5]:
```
point2 = Point(3,4)

point2.x, point2.y, point2.n
```

[5]:
```
(3, 4, 2)
```

[6]:
```
class Triangle:

    def __init__(self, base, height):
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height
```

[7]:
```
triangle = Triangle(14, 10)

triangle.area()
```

[7]:
```
70.0
```

[8]:
```
triangle.base = 12

triangle.base
```

[8]:
```
12
```

[9]:
```
class Vehicle:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def print_characteristics(self):
        print(f"Brand = {self.brand} | Model = {self.model}")
```

[10]:
```
class Car(Vehicle):
    def __init__(self, brand, model):
        super().__init__(brand, model)
```

[11]:
```
car = Car("Ford", "Mustang")
```

```
car.print_characteristics()
```

```
Brand = Ford | Model = Mustang
```

[12]:
```
class Motorcycle(Vehicle):
    def __init__(self, brand, model, year):
        super().__init__(brand, model)
        self.year = year
```

[13]:
```
motorcycle = Motorcycle("Ducati", "Multistrada V4", 2021)

motorcycle.brand, motorcycle.model, motorcycle.year
```

[13]:
```
('Ducati', 'Multistrada V4', 2021)
```

[14]:
```
class Boat(Vehicle):
    def __init__(self, brand, model, passengers):
        super().__init__(brand, model)
        self.passengers = passengers

    def capacity(self): #se o nome fosse igual a algum dos métodos da␣
↪superclasse, esse método seria sobrescrito
        print(f"This boat can take up to {self.passengers} passengers.")
```

[15]:
```
boat = Boat("Quicksilver", "605 Pilothouse", 6)

boat.capacity()
```

```
This boat can take up to 6 passengers.
```

[16]:
```
isinstance(boat,Vehicle)
```

[16]:
```
True
```

4 Erros e exceções

`try`: Permite testar se um bloco de código levanta erros

`except`: Permite lidar com o erro

`finally`: Permite executar código, independentemente do resultado dos blocos try/except

[17]:
```
try:
    print(a)
except:
    print("Something went wrong")
```

```
Something went wrong
```

[18]:
```
try:
    print(a)
except NameError:
    print("Undefined variable")
except:
    print("Something else went wrong")
```

```
Undefined variable
```

[19]:
```
try:
    print("This is class number 5")
except:
    print("Something went wrong")
else:
    print("Nothing went wrong")
```

```
This is class number 5
Nothing went wrong
```

[20]:
```
try:
    print(a)
except:
    print("Something went wrong")
finally:
    print("Reached the end of the try/except block")
```

```
Something went wrong
Reached the end of the try/except block
```

[21]:
```
"""x = "car"

if x is not type(int):
    raise Exception("Only integers allowed")""" #podíamos levantar uma exceção␣
↪pré-definida, como TypeError
```

[21]:
```
'x = "car"\n\nif x is not type(int):\n    raise Exception("Only integers
allowed")'
```

5 Iteradores e geradores

5.1 Iteradores

[22]:
```
phones = ["iPhone", "Samsung", "Xiaomi"] #poderia ser qualquer iterável

phones_iterator = iter(phones)

next(phones_iterator)
```

[22]:
```
'iPhone'
```

[23]:
```
class Numbers:

    def __init__(self):
        self.x = 1

    def __iter__(self):
        return self #tem necessariamente de retornar o próprio iterador

    def __next__(self):
        y = self.x
        self.x += 1
        return y #tem necessariamente de retornar o próximo item da sequência

numbers = Numbers()
numbers_iterator = iter(numbers)

print(next(numbers_iterator))
print(next(numbers_iterator))
print(next(numbers_iterator))
```

```
1
2
3
```

[24]:
```
class Even:

    def __init__(self):
        self.x = 2

    def __iter__(self):
        return self

    def __next__(self):
        if self.x > 6:
            raise StopIteration
```

```
        y = self.x
        self.x += 2
        return y

even = Even()
even_iterator = iter(even)

for i in even_iterator:
    print(i)
```

```
2
4
6
```

5.2 Geradores

[25]:
```
def odd():
    n = -1

    n += 2
    yield n

    n += 2
    yield n

    n += 2
    yield n

odd = odd()

print(next(odd))
print(next(odd))
print(next(odd))
```

```
1
3
5
```

[26]:
```
"""def powers_of_3(max):
    n = 3
    while n <= max:
        yield n
        n *= 3

powers_of_3 = powers_of_3(81)
```

```
print(next(powers_of_3))
print(next(powers_of_3))
print(next(powers_of_3))
print(next(powers_of_3))
print(next(powers_of_3))"""
```

[26]:
```
'def powers_of_3(max):\n    n = 3\n    while n <= max:\n        yield n\n
n *= 3\n    \npowers_of_3 = powers_of_3(81)\n\nprint(next(powers_of_3))\nprint(n
ext(powers_of_3))\nprint(next(powers_of_3))\nprint(next(powers_of_3))\nprint(nex
t(powers_of_3))'
```

6 Exercícios

1) Escreva uma função que calcula o inverso de um número recebido como argumento, sendo que levanta uma exceção caso esse número seja igual a 0.
2) Escreva uma função que tenta ler um ficheiro que não existe.
3) Dados os termos a, b e c de uma equação de segundo grau, escreva uma função que retorna o resultado da aplicação da fórmula resolvente. Para além disso, caso o binómio discriminante seja inferior a 0, deverá levantar uma exceção.
4) Escreva a classe **Esfera** que contém o atributo *raio* e os métodos area() e volume() que irão devolver a área e o volume de uma esfera, respetivamente.
5) Escreva a classe **Animal** que poderá ser reutilizada pelas classes **Cao** e **Ave**. A primeira deverá ter os seguintes atributos: *cor* e *peso*. Para além disso, nas subclasses não só deverá complementar com uma característica específica, como também deverá implementar o método som() que deverá simular o som feito por cada.
6) Escreva a classe **Compra** que poderá ser reutilizada pela classe **CompraOnline**. A primeira deverá ter os seguintes atributos: *preco* e *quantidade*. Para além disso deverá implementar o método total() que devolve o total gasto nessa compra. Já a subclasse deverá implementar o mesmo método, mas tenha em conta que a taxa de entrega corresponde a 5% do valor total.
7) Crie um iterador que emule o comportamento da função enumerate() (tenha em conta que esta função não só devolve os itens de um iterável, como também os respetivos índices).
8) Crie um iterador que emule o comportamento da função range().
9) Crie um gerador que retorne a sequência de Fibonacci.
10) Crie um gerador que retorna a sequência com os fatoriais dos números inteiros não negativos.