Resumo – Classes

Classes
    Cuidado com objetos mutáveis
Importando código externo
    Módulos *builtin*
F-strings

Classes

```
class Carro:
    def __init__(self, marca, ano):  # Construtor
        self.marca = marca
        self.ano = ano
        self.velocidade = 0
        self.posição = 0

    def acelerar(self, intensidade):  # Método de instância
        if intensidade == 1:
            self.velocidade += 5
        elif intensidade == 2:
            self.velocidade += 10
        self.posição = self.posição + self.velocidade

    def imprime_estado(self):  # Método de instância
        print("Dados do carro ", self.marca)
        print("O ano do carro é: ", self.ano)
        print("A posição do carro é: ", self.posição)
        print("A velocidade do carro é: ", self.velocidade)

    @classmethod
    def cria_toyota_2021(cls):  # Método de classe
        return cls(marca="Toyota", ano=2021)

    @staticmethod
    def calcula_posicao(pos_atual, velocidade):  # Método estático
        return pos_atual + velocidade

carro_a = carro.Carro("Volkswagen", 2019)
carro_b = carro.Carro("Toyota", 2020)
carro_a.acelerar(2)
carro_b.acelerar(1)
carro_a.imprime_estado()
carro_b.imprime_estado()
```

💡 Chamamos "instância" um objeto que é criado a partir de uma classe.

Cuidado com objetos mutáveis

```
carro_a = Carro("Volks", 2019)
carro_b = carro_a
# ambas variáveis estão referenciando o MESMO objeto
print(carro_a.ano)  # => 2019
print(carro_b.ano)  # => 2019
carro_b.ano = 2020  # Altera o ano do carro_b
print(carro_a.ano)  # => 2020 também!
# carro_a também é alterado!
```

💡 Isso ocorre com qualquer tipo de valor "mutável" (listas, conjuntos, dicionários..).

Importando código externo

É possível *importar* um trecho de código contendo outras funções ou variáveis no nosso programa. Chamamos um arquivo que contém código Python de `módulo` (*module*).

```
# modulo_a.py
def é_primo(n):
    divisores = 0
    i = 0
    while(i <= n):
        if n % i == 0:
            divisores += 1
        i += 1
    return divisores == 2
```

```
# modulo_b.py
import modulo_a  # Não se coloca o .py

num = int(input("Digite um número: "))

if modulo_a.é_primo(num):  #
    print("{} é primo".format(num))
else:
    print("{} não é primo".format(num))
```

Módulos *builtin*

A instalação padrão do Python já traz consigo diversos módulos que podemos utilizar. Por exemplo, o módulo `datetime` tem funções para lidar com datas e o `time` possui funções para facilitar trabalharmos com o tempo.

```
import time
import datetime
# import time, datetime => funcionaria do mesmo jeito importar ambos em uma linha

tempo = time.time()
print("Tempo em segundos desde Janeiro de 1970: ", tempo)

hoje = datetime.date.today()  # datetime.date => módulo dentro de outro
print("A data de hoje é: ", hoje)
```

💡 Módulos podem ser tanto arquivos com código quanto pastas contendo outros módulos. É o exemplo do `datetime.date`, onde `datetime` contém outro módulo dentro `date` que então define os métodos (como o `today()`).

F-strings

```
aula = "F-strings"

# Imprime a string e automaticamente o valor da variável aula (com um espaço antes)
print("Hoje vamos aprender sobre", aula)  # Sem usar f-strings

# Concatena a primeira string com o valor da variável aula e por fim com a string "!"
print("Hoje vamos aprender sobre " + aula + "!")

# Utilizando f-string – muito mais legível
print(f"Hoje vamos aprender sobre {aula}!")
```