# Aula 5 - Classes e módulos



## Agenda

- Apresentar o `for`
- Objetos e classes
- Módulos e `import`

## `for` (repetição)

```
minha_lista = [1, 4, 5, 9]
i = 0
elemento = minha_lista[i]
i += 1
elemento = minha_lista[i]

# Ou, usando for:
for elemento in minha_lista:
    print(elemento)
```

## Sintaxe do `for`

```
for <nome da variável> in <iterável>:
    <instrução 1>
    <instrução 2>
    ...
    <instrução n>
```

💡 Qualquer *iterável* pode ser utilizado, *lists, tuples, dicts* e até *strings*. ***sets* não são iteráveis.**

## Objetos

> Uma estrutura de dados para representar objetos com **estado** e **operações**.

💡 Listas, tuplas, dicionários e conjuntos são objetos. Eles possuem estado e operações

```
minha_lista = [1, 2, 3]  # Cria um objeto lista
minha_lista.append(4)  # Altera o estado para [1, 2, 3, 4]
# objeto.atributo ou objeto.operação(...)
```

```
###
# Representar um carro utilizando uma lista
###

# marca, ano, velocidade, posição
carro = ["Volkswagen", 2019, 0, 0]
```

```
def imprime_estado(carro):
    print("Posição: {}, velocidade atual: {} km/h".format(carro[3], carro[2]))

def acelerar(carro, intensidade):
    if intensidade == 1:
        carro[2] += 5
    elif intensidade == 2:
        carro[2] += 10
    carro[3] += carro[3] + carro[2]  # pos = pos + v

acelerar(carro, 1)
# Novo estado!
imprime_estado(carro)
```

# Classe

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

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

    def imprime_estado(self):
        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)

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.

## *Aliasing*

Cuidado ao associar duas variáveis à mesma **instância** de objeto. Quando o interpretador Python encontra uma linha de código `carro = Carro()`, é reservado um espaço de memória para essa instância - `Carro()` é uma função que retorna uma nova instância da classe `Carro` - e a variável `carro` (`c` minúsculo) é o identificador dessa instância/objeto.

Quando você associa uma outra variável à primeira, por exemplo, `carro_2 = carro`, não é criada uma nova instância (ou seja, não é reservado espaço na memória do seu computador pra armazenar um novo objeto), mas simplesmente um novo "apelido" (*alias*) é criado para a instância `carro`. Observe o exemplo abaixo o que acontece quando alteramos o estado de uma variável "apelido":

```
carro_a = Carro("Volks", 2019)
carro_b = carro_a  # carro_b é apenas um alias
# 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!
# carro_a também é alterado, pois referencia o mesmo objeto
```

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

# Importando código externo

Até o momento a gente sempre utilizou apenas um arquivo para escrever nossos programas. Mas 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()`).