PROGRAMAÇÃO ORIENTADA A OBJETOS EM PYTHON

prof. Guilherme Derenievicz

adaptado pelo prof. Jônata Tyska Carvalho

Departamento de Informática e Estatística - UFSC

---

1 - Objetos e Classes

Python

Python é uma linguagem que contempla tanto o paradigma **imperativo** quanto o paradigma **orientado a objetos**:

- **Imperativo:** o programa é uma sequência de comandos que modificam variáveis (estados). O foco é em *como resolver o problema*, utilizando comandos condicionais, de repetição, funções, atribuições, etc, para manipular dados (variáveis).
- **Orientado a Objetos:** O foco é na representação computacional do problema, identificando "objetos" importantes do problema e "organizando-os". Tais objetos mantêm, dentro de si, seus próprios dados e as operações que manipulam esses dados. O foco é em *quem está sendo afetado na resolução do problema*. Um importante conceito de OO é: **abstração**.

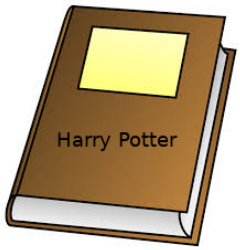
O que é um objeto?

> *Um objeto é uma representação computacional de algo, para executar uma determinada tarefa,...*

Neste caso, **algo** pode ser qualquer coisa? Considere os seguintes exemplos:



Neste caso, **livro**, **carro** e **pessoa** podem ser considerados objetos? Talvez! Depende do que se deseja representar... Se o objetivo é ensinar uma criança a ler, certamente essas figuras são boas representações. Mas elas também podem estar representando conceitos mais gerais que um único objeto: a primeira figura representa *qualquer* livro (ou, um conjunto de livros), e assim por diante.



Aqui temos uma representação mais específica, mas ainda podem ser representações de *conjuntos de objetos* (Qual livro do Harry Potter? Qual Gol? Qual João?) Se, adicionalmente, representarmos o código do livro, a placa do carro e o CPF do João, teremos representações altamente específicas que têm uma relação direta com objetos que existem no mundo real. O quão específico devemos ser depende de cada caso: para ensinar leitura na escola, a primeira representação é suficiente, para desenvolver um sistema do Detran, cada carro precisa ser bem especificado. Aqui, voltamos no conceito de abstração:

Abstrair: identificar aspectos importantes de um fenômeno e ignorar detalhes.

E com isso, podemos terminar a definição de objeto:

> *Um objeto é uma representação computacional de algo, para executar uma determinada tarefa, com limites bem definidos.*

O que é uma classe?

Considere o nível de abstração da segunda figura e que tenhamos diversas livros: Harry Potter, Dom Casmurro, O Senhor dos Anéis, Aprendendo Python, etc... Neste caso, o que significa a representação de livro da figura 1? Certamente não é um objeto, pois não é específico o suficiente de acordo com a nossa abstração, mas é um conceito mais geral, que engloba todos os objetos livros.

> *Uma classe é a definição de um grupo de objetos com propriedades** e **comportamentos** comuns. Assim, um objeto é uma **instância** de uma classe.*

Por exemplo, um carro tem as seguintes propriedades (além de outras):

- modelo
- cor
- placa
- velocidade

E os seguintes comportamentos (além de outros):

- buzinar()
- acelerar()
- frear()

Qualquer objeto carro terá as propriedades e comportamentos acima.

Podemos estender ainda mais a nossa definição de objeto:

> *Um objeto é uma representação computacional de algo, para executar uma determinada tarefa, com limites bem definidos. É uma coleção de dados (atributos) e operações (métodos) para manipular estes dados, representando uma entidade lógica no sistema.*

Exemplo:

Classe `Carro`

> **Atributos:**
>
> - modelo
> - cor
> - placa
> - velocidade
>
> **Métodos:**
>
> - buzinar()
> - acelerar()
> - frear()

Objeto `Carro 1` :

> - modelo = Gol
> - cor = Vermelho
> - placa = ABC-1234
> - velocidade = 0

Objeto `Carro 2`:

> - modelo = Fox
> - cor = Prata
> - placa = XYZ-1234
> - velocidade = 80

Aqui temos 2 objetos da classe `Carro`, cada um especificando os seus valores para cada atributo. Ambos os objetos podem "executar" qualquer método da classe. Nesse sentido, os métodos podem ser vistos como funções (do paradigma imperativo), mas que ficam "dentro" dos objetos, pois não fazem sentido existirem se não estiverem atrelados a estes.

Atributos e Métodos

> *Atributos são variáveis que descrevem as características um objeto.*

> *Métodos são funções que definem o comportamento de um objeto.*

Declarar uma Classe em Python

In [2]:
```
class Carro:
    def __init__(self, modelo: str, cor: str, placa: str):
        self.modelo = modelo
        self.cor = cor
        self.placa = placa
        self.velocidade = 0

    def buzinar(self):
        print('carro', self.modelo, 'buzinou')

    def acelerar(self):
        self.velocidade += 10

    def frear(self):
        self.velocidade -= 10
        if self.velocidade < 0:
            self.velocidade = 0
```

- `self:` é uma referência ao objeto desta classe que será instanciado (é obrigatório e deve ser sempre o primeiro parâmetro dos métodos definidos na classe).
- `__init__():` é um método especial chamado de **construtor** que é executado automaticamente sempre que a classe é instanciada. É responsável por inicializar

atributos. Pode chamar quaisquer métodos da classe para isso.

- Observe que os atributos têm tipos específicos, que podemos descrever com a notação `atributo: tipo`. Neste exemplo, os atributos `modelo`, `cor` e `placa` da classe `Carro` são strings, enquanto o atributo `velocidade` é um inteiro.

Instanciar uma classe em Python:

In [4]:
```
carro1 = Carro('Gol', 'Vermelho', 'ABC-1234')
carro2 = Carro('Fox', 'Prata', 'XYZ-1234')

carro1.buzinar()
carro2.acelerar()
print('velocidade do', carro2.modelo, ':', carro2.velocidade, 'km/h')
```

```
carro Gol buzinou
velocidade do Fox : 10 km/h
```

Note como o parâmetro `self` **não é** utilizado na chamada do método. Note também a diferença entre acessar métodos e atributos do objeto `carro2`: para acessar os métodos utiliza-se parênteses, diferentemente do acesso aos atributos.

Métodos também podem ter argumentos e retorno. Exemplo:

In [5]:
```
class Carro:
    def __init__(self, modelo: str, cor: str, placa: str):
        self.modelo = modelo
        self.cor = cor
        self.placa = placa
        self.velocidade = 0

    def buzinar(self):
        print('carro', self.modelo, 'buzinou')

    def acelerar(self, valor: int):
        self.velocidade += valor
        if self.velocidade > 100:
            return 'cuidado!'
        else:
            return 'velocidade aceitável.'

    def frear(self):
        self.velocidade -= 10
        if self.velocidade < 0:
            self.velocidade = 0


def main():
    carro1 = Carro('Gol', 'Vermelho', 'ABC-1234')
    carro2 = Carro('Fox', 'Prata', 'XYZ-1234')

    carro1.buzinar()
    print(carro2.acelerar(80))
    print(carro2.velocidade, 'km/h')

main()
```

```
carro Gol buzinou
velocidade aceitável.
80 km/h
```

Exercício

Faça um programa que leia uma string `s` indicando um tipo de forma geométrica (considere que `s` pode ser `'quadrado'`, `'retangulo'` ou `'circulo'`). Conforme a forma geométrica lida, seu programa deve ler também a(s) sua(s) dimensão(ões), sendo `lado` para o quadrado, `base` e `altura` para o retângulo e `raio` para o circulo. Seu programa deve mostrar qual a área da forma geométrica lida. Para resolver este problema, pense quais são os objetos envolvidos e qual o nível de abstração adequado. O que seria um objeto nesse caso? E o que seria uma classe? Defina classes adequadas, com os respectivos atributos e métodos, e instancie seus objetos no programa principal.

In [ ]:
```
#escreva seu código aqui
```

Pilares da programação orientada a objetos

Como estudamos em aula este são os pilares que definem orientação a objetos. Estudaremos cada um deles em nossa disciplina.

- Abstração
- Encapsulamento
- Herança
- Polimorfismo

Conclusão

Orientação a Objetos **organiza** o código de maneira mais lógica e mais próxima do "mundo real", facilitando a reusabilidade, diminuindo a rigidez e a fragilidade do código.

Referências

Este material foi desenvolvido com base nas Notas de Aulas da disciplina *Desenvolvimento de Sistemas Orientados a Objetos I* do curso de Sistemas de Informação do INE-UFSC, de autoria do prof. Jean Carlo Rossa Hauck, e no material *Aulas de Introdução à Computação em Python*, do Departamento de Ciência da Computação do IME-USP, disponível em https://panda.ime.usp.br/aulasPython/static/aulasPython/index.html

In [ ]: