Classes e Objetos (II)

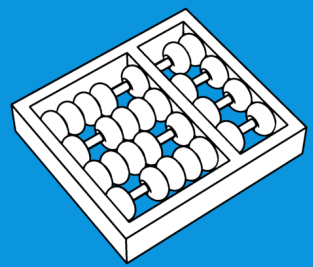
MC102 - Algoritmos e Programação de Computadores

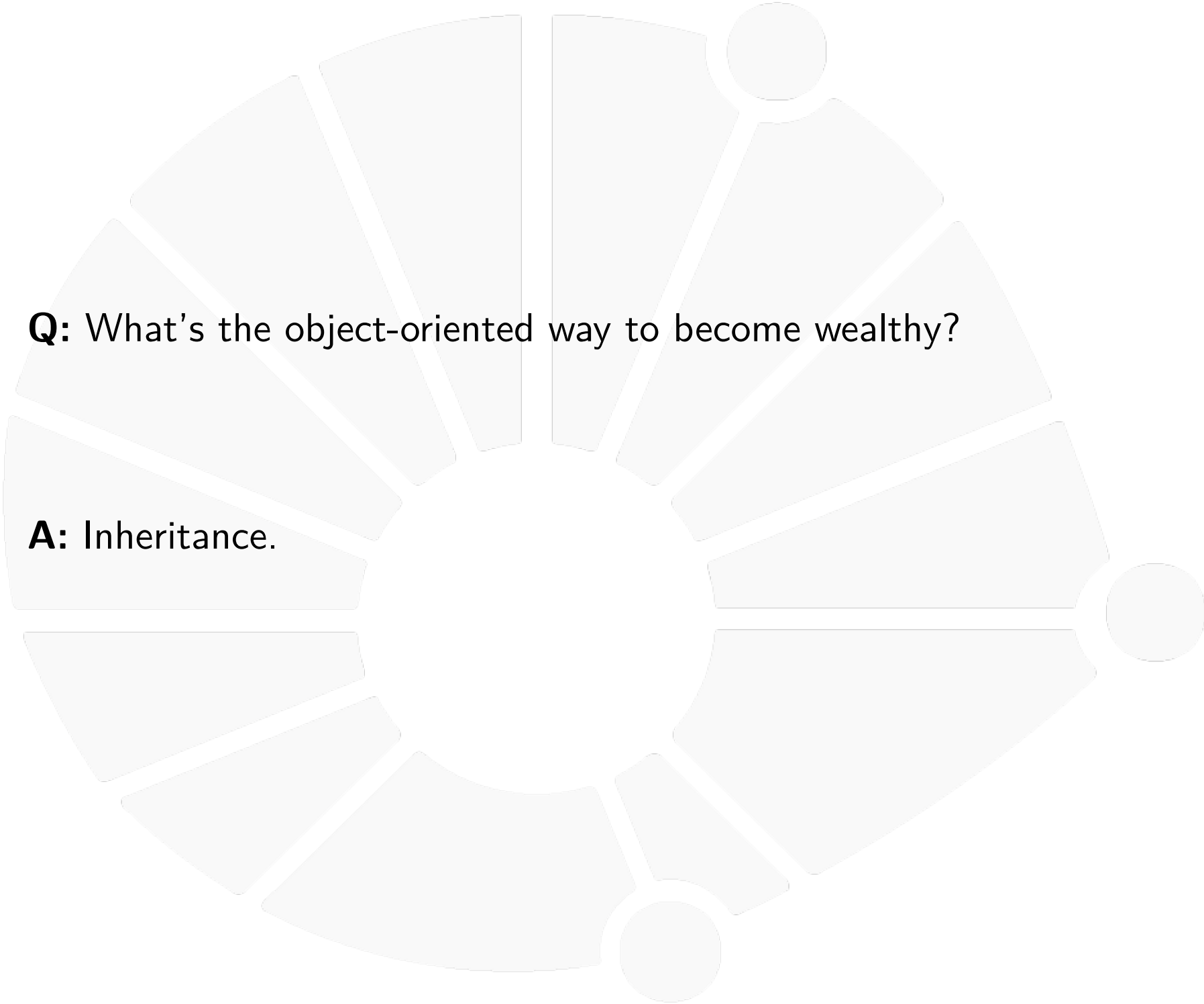
05/25

19

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP

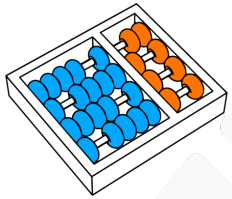




Q: What's the object-oriented way to become wealthy?

A: Inheritance.

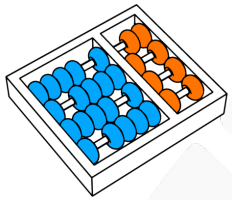
Dúvidas da aula anterior



Dúvidas selecionadas

- Não entendi propriedade.
- Posso definir operações para minha classe além da operação de igualdade?
- Não entendi direito o que significa self.
- O uso dos decorators é necessário para fazer os getters e setters da classe ou apenas serve para indicar que o método é propriedade da classe?
- Qual é exatamente a diferença entre classes e objetos? E objetos e instâncias?
- Na criação da classe, eu posso predefinir alguns valores de atributos de um objeto?
- Eu não entendi uma coisa: se a classes são como os tipos de python (str, interesses, float), quer dizer que esses tipos também são classes que foram pré programadas?
- Qual a diferença entre a classe criada em sala e um dicionário?
- Tem como eu usar equal para comparar dois objetos de classes diferentes?
- Não entendi o dataclasses.
- A programação orientada a objeto é um jeito mais eficiente de codar ou tem algo que só é possível fazer com ela?
- É possível instanciar uma classe dentro do atributo de outra classe?
- Posso criar classes de excepções para pegar um erro customizado em meu código?

Herança



Herança

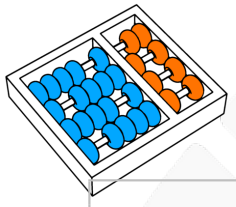
Outro conceito comum em orientação a objetos é **Herança**

Muitas vezes temos o conceito de **"é um"**:

- Estudante de Graduação **é um** Estudante.
- Estudante de Pós-Graduação **é um** Estudante.
- Quadrado **é um** Retângulo que **é um** Paralelogramo.
- Inteiro **é um** Racional que **é um** Real.

E isso nos permite reutilizar códigos:

- A classe filha **herda** métodos e atributos da classe mãe.

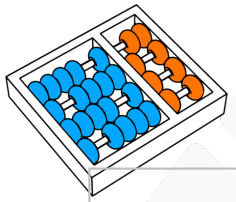


A classe Retângulo

```
1  class Retangulo:
2      def __init__(self, largura, altura):
3          self._largura = largura
4          self._altura = altura
5
6      def area(self):
7          return self._largura * self._altura
8
9      def perimetro(self):
10         return 2 * self._largura
11             + 2 * self._altura
12
13     def __str__(self):
14         return ("Retângulo "
15             + str(self._largura) + "x"
16             + str(self._altura))
17
18 r = Retangulo(10, 3)
19 print(r)
20 print(r.area(), r.perimetro())
```

Será impresso:

```
1 Retângulo 10x3
2 30 26
```



A classe Quadrado

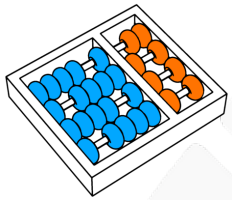
```
1  # Quadrado herda de Retangulo
2  class Quadrado(Retangulo):
3      def __init__(self, lado):
4          # super é o objeto mãe
5          super().__init__(lado, lado)
6
7      @property
8      def lado(self):
9          # lemos da classe mãe
10         return self._largura
11
12     @lado.setter
13     def lado(self, lado):
14         # alteramos na classe mãe
15         self._largura = lado
16         self._altura = lado
17
18      # Estamos sobrescrevendo __str__
19     def __str__(self):
20         return "Quadrado de lado "
21             + str(self.lado)
```

Ao fazer:

```
1  q = Quadrado(4)
2  print(q)
3  print(q.area(),
4      q.perimetro())
```

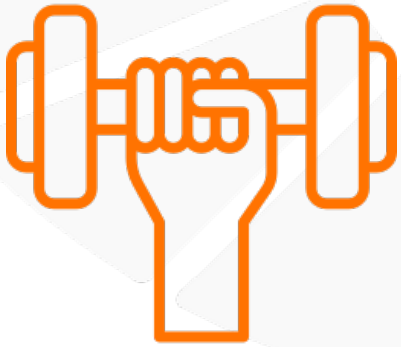
Será impresso:

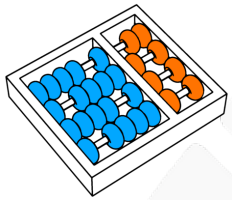
```
1  Quadrado de lado 4
2  16 16
```



Herança

Vamos fazer um exercício

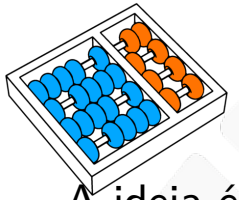




Exercício

Faça uma classe **EstudanteDePos**, considerando que esses estudantes recebem conceito **A**, **B**, **C**, ou **D**, dependendo de sua nota final

Comentários

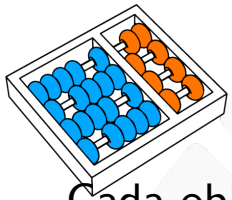


Comentando classes

A ideia é a mesma de comentar funções

```python
class Estudante:
    """Representa um estudante com suas informações e nota.

    Capaz de armazenar o nome, RA, curso e a nota do estudante.
    """

    def __init__(self, nome, RA, curso, nota):
        self.nome = nome
        self.RA = RA
        self.curso = curso
        self.nota = nota

    @property
    def nota(self):
        """Nota do estudante.

        A nota deve estar entre 0 e 10.
        """
        return self._nota
```

Identificador de objetos



id de Objetos

Cada objeto no Python tem um único identificador:

- Pode ser acessado pela função **id**.

Ex:

```
1 r1 = Retangulo(10, 3)
2 r2 = Retangulo(3, 3)
3 print(r1, r2)
4 print("Mesma id?", id(r1) == id(r2))
```

Será impresso:

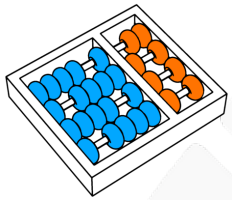
```
1 Retângulo 10x3 Retângulo 3x3
2 Mesma id? False
```

Os retângulos são objetos diferentes... e se executarmos:

```
1 r1.largura = 7
2 print(r1, r2)
```

Será impresso:

```
1 Retângulo 7x3 Retângulo 3x3
```



id de Objetos

Outro exemplo:

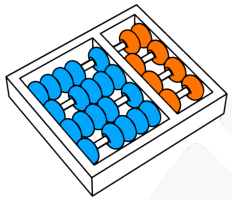
```
1 r1 = Retangulo(10, 3)
2 r2 = Retangulo(10, 3)
3 print(r1, r2)
4 print("Mesma id?", id(r1) == id(r2))
5 r1.largura = 7
6 print(r1, r2)
```

Será impresso:

```
1 Retângulo 10x3 Retângulo 10x3
2 Mesma id? False
3 Retângulo 7x3 Retângulo 10x3
```

Isto é, os retângulos continuam sendo objetos diferentes...

- Cada chamada de **Retangulo(10, 3)** criou um novo objeto.



id de Objetos

Mais um exemplo:

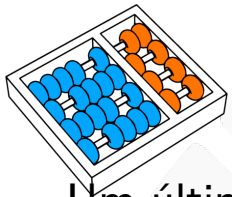
```
1 r1 = Retangulo(10, 3)
2 r2 = r1
3 print(r1, r2)
4 print("Mesma id?", id(r1) == id(r2))
5 r1.largura = 7
6 print(r1, r2)
```

Será impresso:

```
1 Retângulo 10x3 Retângulo 10x3
2 Mesma id? True
3 Retângulo 7x3 Retângulo 7x3
```

Isto é, **r1** e **r2** são o mesmo objeto!

- Dizemos que eles referenciam o mesmo objeto.



id de Objetos

Um último exemplo:

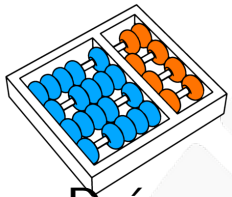
```
1 r1 = 10
2 r2 = 10
3 print(r1, r2)
4 print("Mesma id?", id(r1) == id(r2))
5 r1 = 15
6 print(r1, r2)
```

Será impresso:

```
1 10 10
2 Mesma id? True
3 15 10
```

A constante **10** existe apenas uma vez:

- Por isso **id(r1) == id(r2)**.
- Mas **r1** passou a referenciar outro objeto:
  - Note que atribuímos para **r1**.
  - Não usamos um método do objeto.
- O **10** que o **r2** referencia continua o mesmo.



Mas, e daí?

Daí que isso explica porque funções alteram listas!

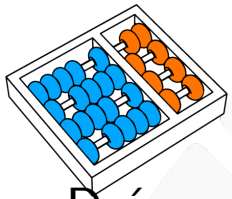
- E objetos em geral!

Exemplo:

```
1  def altera(lista):  # lista referencia [1, 2, 3]
2      lista.clear()  # método que remove os elementos da lista
3      print(id(lista))
4
5  lista = [1, 2, 3]
6  print(id(lista))
7  altera(lista)
8  print(lista)
```

Será impresso:

```
1 4547883488
2 4547883488
3 []
```



Mas, e daí?

Daí que isso explica porque funções alteram listas!

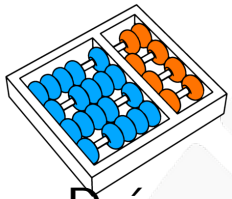
- E objetos em geral!

Exemplo:

```
1  def nao_altera(lista):  # lista referencia [1, 2, 3]
2      lista = []  # lista passa a referenciar []
3      print(id(lista))
4
5  lista = [1, 2, 3]
6  print(id(lista))
7  nao_altera(lista)
8  print(lista)
```

Será impresso:

```
1 4550516976
2 4547883488
3 [1, 2, 3]
```



Mas, e daí?

Daí que isso explica porque funções alteram listas!

- E objetos em geral!

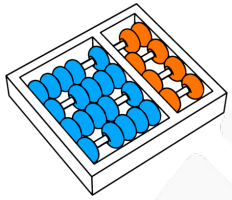
Exemplo:

```
1  def nao_altera2(x):
2      x = 0
3      print(id(x))
4
5  x = 10
6  print(id(x))
7  nao_altera2(x)
8  print(x)
```

Será impresso:

```
1 4546739392
2 4546739072
3 10
```

Mutável ou imutável?



Objetos Mutáveis e Imutáveis

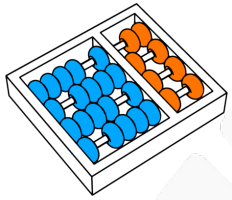
Objetos mutáveis são aqueles que podem ser alterados:

- Através de mudanças em seus atributos.
- Através de chamadas de método.
- Ex: **list**, **dict**, **set**.

Estes objetos podem ser alterados chamadas de funções:

Objetos imutáveis não podem ser alterados:

- Não há como acessar os atributos.
- Não há métodos que alteram o objeto.
- Ex: **int**, **float**, **bool**, **None**, **tuple**.

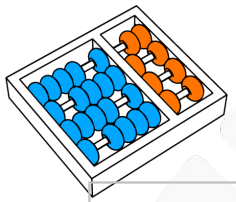


Métodos de Classe e Métodos Estáticos

Uma classe pode ter métodos que podem ser acessados sem termos um objeto.

Esses métodos são de dois tipos:

- Métodos de Classe: sabem quem é a classe.
  - Muitas vezes são formas de construir um objeto.
  - Ex: **datetime.date.today()**.
  - Ex: **datetime.date.fromisoformat('2022-12-25')**.
- Métodos Estáticos: não sabem quem é a classe.
  - Em geral são métodos utilitários.
  - Não são tão usados.



Métodos de Classe e Métodos Estáticos

```
1  from dataclasses import dataclass
2  @dataclass
3  class Retangulo:
4      largura: int
5      altura: int
6      @classmethod  # decorador de método de classe
7      def da_lista(cls, lista):  # cls é a classe
8          return cls(lista[0], lista[1])
9
10     @staticmethod  # decorador de método estático
11     def formula_da_area():
12         return "base x altura"
13
14     def area(self):
15         return self._largura * self._altura
16
17 lista = [2, 3]
18 r = Retangulo.da_lista(lista)
19 s = Retangulo.formula_da_area()
```

r é um **Retangulo** 2×3 e **s** é a string **"base x altura"**.

Classes e Objetos (II)

MC102 - Algoritmos e Programação de Computadores

05/25

19

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP