

CLASSES E OBJETOS (II)

MC102 - Algoritmos e
Programação de
Computadores

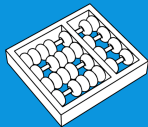
Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

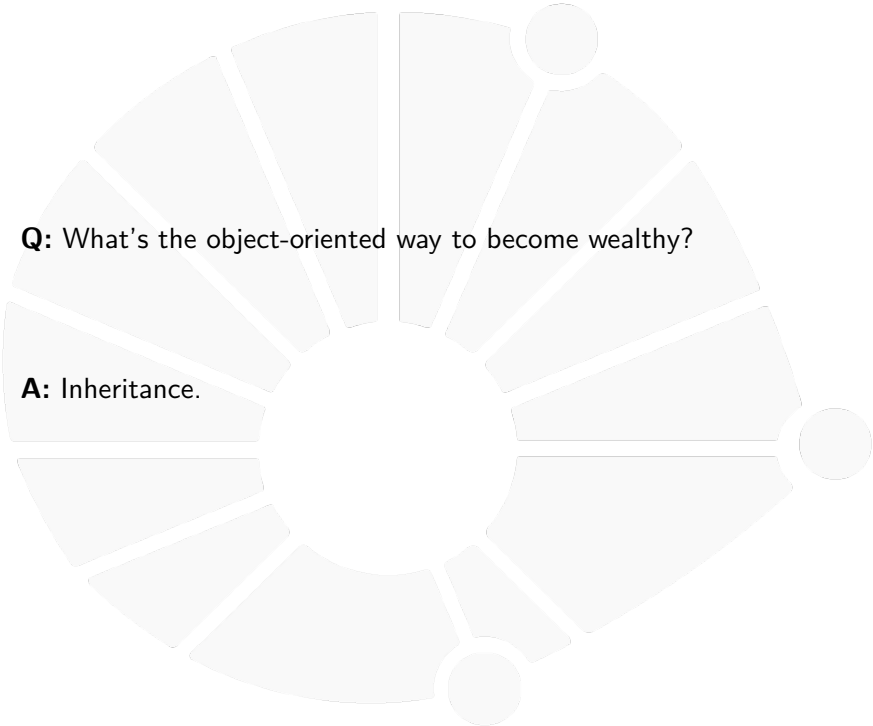
05/24

18



UNICAMP



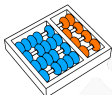
A circular diagram consisting of 12 light gray segments arranged in a ring around a central white circle. Three small light gray circles are positioned at the top, right, and bottom of the ring, each overlapping one of the segments.

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

A: Inheritance.



DÚVIDAS DA AULA ANTERIOR



Dúvidas selecionadas

- ▶ Não entendi muito bem o que são os decorators.
- ▶ Não entendi muito bem o uso de `__` no ini.
- ▶ Ao não informar o parâmetro `self` do método de uma função, o método se torna estático (acessível pelo nome da classe)?
- ▶ Ainda não entendi bem o conceito de paradigma e de programação orientada a objetos. Qual a diferença do paradigma funcional e orientado a objetos?
- ▶ Em quais ocasiões é legal criar uma classe?
- ▶ Existe um modificador de acesso "protected" em Python, como no Java?
- ▶ POO é recomendado para projetos maiores? nos exercícios do lab eles serão dispensáveis, ou ainda sim seria recomendado seu uso?
- ▶ Não sei se entendi muito bem sobre o `@property...` Então ele basicamente torna uma função de dentro da classe acessível?
- ▶ Ao criar uma classe, além de usar o `__str__`, é possível fazer a mesma coisa pra outras formas de dados, como o float?
- ▶ O hash de 2 objetos pode ser igual mesmo eles sendo de tipos diferentes?
- ▶ Eu consigo fazer métodos privados só colocando o `_` na frente?
- ▶ Mesmo usando o `@dataclass` eu posso subscrever o `__init__`?
- ▶ Ainda não entendi muito bem o funcionamento do `__underline` antes de uma função. Para que serve usar essa técnica?



HERANÇA



Herança

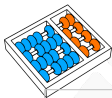
Outro conceito comum em orientação a objetos é **Herança**

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.

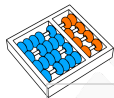


A classe Retângulo

```
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:

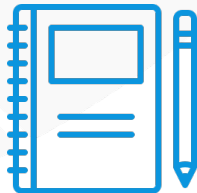
```
1 Quadrado de lado 4
2 16 16
```




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

```
1 class Estudante:
2     """Representa um estudante com suas informações e nota.
3
4     Capaz de armazenar o nome, RA, curso e a nota do estudante.
5     """
6
7     def __init__(self, nome, RA, curso, nota):
8         self.nome = nome
9         self.RA = RA
10        self.curso = curso
11        self.nota = nota
12
13    @property
14    def nota(self):
15        """Nota do estudante.
16
17        A nota deve estar entre 0 e 10.
18        """
19        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:

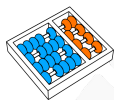
```
1 r1 = Retangulo(10, 3)
2 r2 = Retangulo(10, 3)
3 print(r1, r2)
4 print("Mesma id?", id(r1) == id(r2))
5 r1.largura = 7
6 print(r1, r2)
```

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!

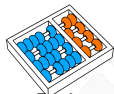
- ▶ E objetos em geral!

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

Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

05/24

18



UNICAMP

