Classes e Objetos

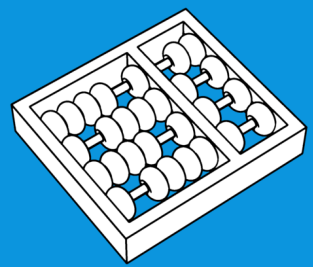
MC102 - Algoritmos e Programação de Computadores

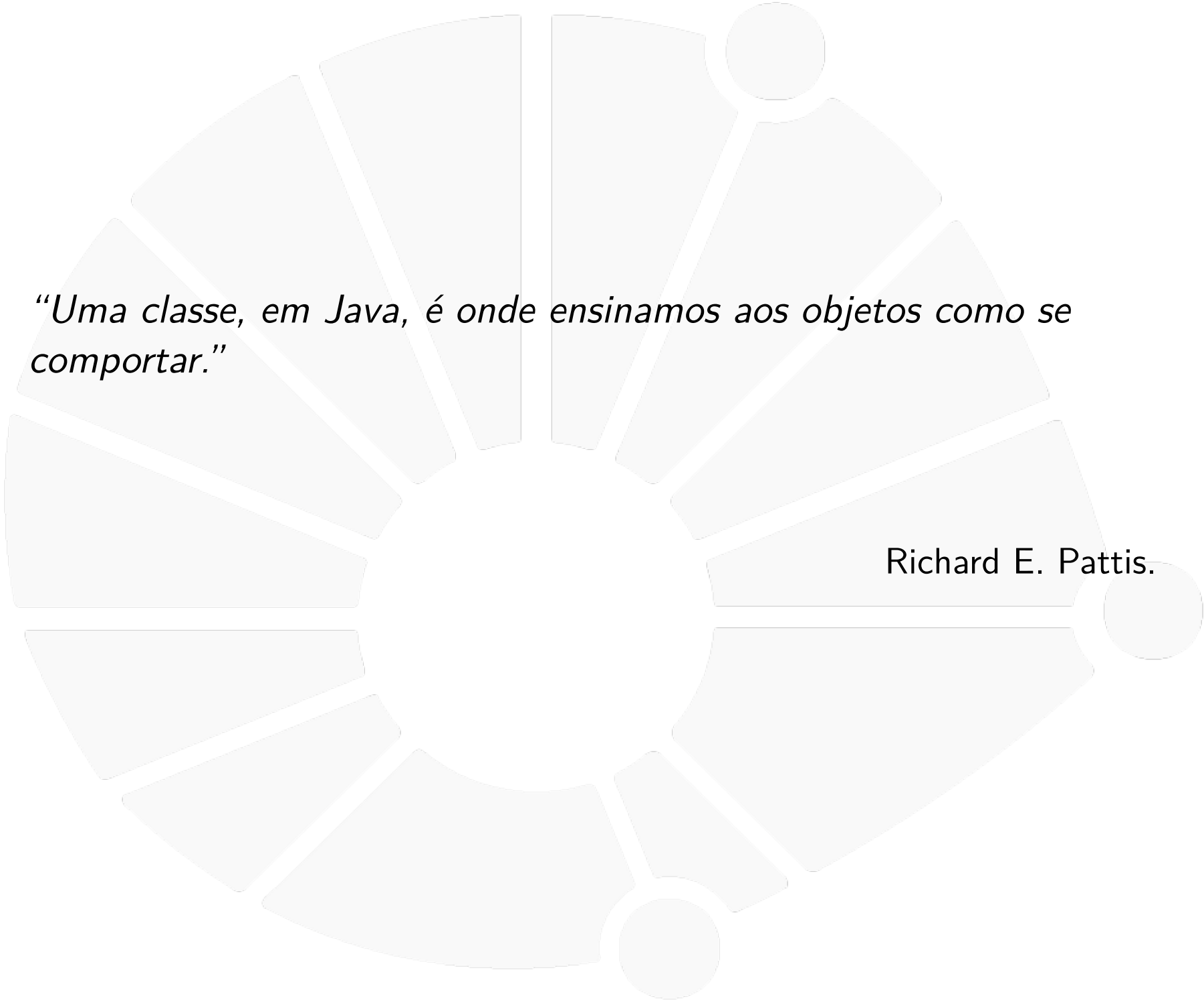
05/25

18

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

UNICAMP

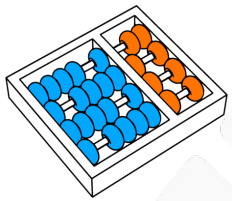




"Uma classe, em Java, é onde ensinamos aos objetos como se comportar."

Richard E. Pattis.

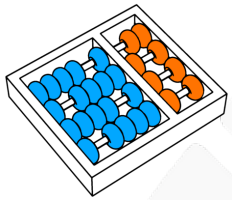
Dúvidas da aula anterior



Dúvidas selecionadas

- Qual seria a lógica pra resolver a do maior quadrado de 1?
- Existem outros estilos de dojo que vamos fazer esse semestre prof? (Por enquanto fizemos o mesmo, por isso pergunto)
- Como ficaria a solução completa do último problema do coding dojo, o de comparar duas palavras?
- Não entendi a diferença de .sort para sorted().
- Não entendi bem a resolução da questão de achar o maior quadrado de 1 na matriz binária.
- Ainda não entendi muito bem a função do tuples.
- Não entendi a resolução do problema do coding dono que era assim: achar o maior valor para a soma de uma matriz aplicando a primeira função (multiplicar 2 numeros consecutivos por -1)
- Você vai disponibilizar os códigos do dojo?
- Quando será o próximo coding dojo?
- Algum site específico além do beecrownd para treinar resolvendo os problemas como os de ontem?
- Tem alguma forma de eu fazer um for no Python igual nas outras linguagens(C, Java, Javascript, etc)? Tipo: for(var $= 0$, var $< x$, var $+= 1$).
- Aonde ficaram disponíveis as resoluções dos coding dojo?
- Quando você disse que tinha como resolver um dos problemas sem usar vários "for", do que você estava falando?

Orientação a objetos



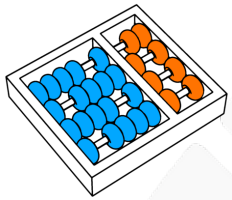
Relembrando: Tipo

O **tipo** de um dado define:

- As operações que podemos fazer com ele.
- Qual é o resultado.

Ex:

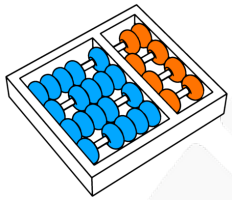
- O que acontece ao somar um **int** com um **int**?
- O que acontece ao somar um **int** com um **float**?
- O que acontece ao somar uma **string** com um **string**?
- O que acontece ao somar uma **string** com um **int**?



Orientação a Objetos — Conceito

Um paradigma de programação (entre muitos outros) onde:

- **Objetos** armazenam dados como seus **atributos**
  - "Variáveis" que pertencem ao objeto.
- Os objetos podem ser manipulados através de seus **métodos**:
  - Funções que acessam ou modificam os atributos.
- Objetos de uma **Classe** têm os mesmos atributos e métodos:
  - Mas os valores dos atributos podem ser diferentes.
  - A classe faz o papel do **tipo**.
  - Ex: `[1, 2, 3]` e `[]` são objetos da classe `list`:
    - Ambos respondem ao método `append`.
- A computação é feita pela interação entre os vários objetos.

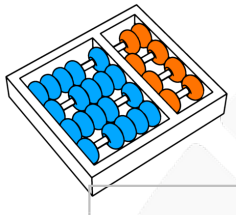


Criando uma classe bem simples

```python
class Estudante:  # Define a classe Estudante

    # Define um método chamado __init__ que nos diz como
    # inicializar o objeto. self é o próprio objeto.
    def __init__(self, nome, RA, curso, nota):
        self.nome = nome
        self.RA = RA
        self.curso = curso
        self.nota = nota

# poderia ter usado parâmetros posicionais também
ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
print(ana.nome, ana.RA, ana.curso, ana.nota)
```

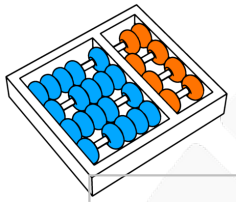


Criando uma classe bem simples

```
1  class Estudante:  # Define a classe Estudante
2
3      # Define um método chamado __init__ que nos diz como
4      # inicializar o objeto. self é o próprio objeto.
5      def __init__(self, nome, RA, curso, nota):
6          self.nome = nome
7          self.RA = RA
8          self.curso = curso
9          self.nota = nota
10
11 # poderia ter usado parâmetros posicionais também
12 ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
13 print(ana.nome, ana.RA, ana.curso, ana.nota)
```

Criamos um novo estudante escrevendo **Estudante(...)**:

- Recebe um parâmetro a menos (o **self**).



Criando uma classe bem simples

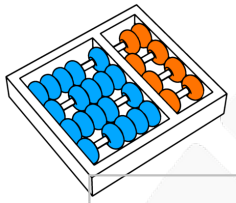
```python
class Estudante:  # Define a classe Estudante

    # Define um método chamado __init__ que nos diz como
    # inicializar o objeto. self é o próprio objeto.
    def __init__(self, nome, RA, curso, nota):
        self.nome = nome
        self.RA = RA
        self.curso = curso
        self.nota = nota

# poderia ter usado parâmetros posicionais também
ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
print(ana.nome, ana.RA, ana.curso, ana.nota)
```

Dizemos que:

- **ana** é um objeto da classe **Estudante**.
- **ana** é uma instância de **Estudante**.



Criando uma classe bem simples

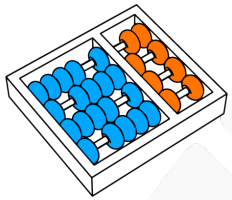
```python
class Estudante:  # Define a classe Estudante

    # Define um método chamado __init__ que nos diz como
    # inicializar o objeto. self é o próprio objeto.
    def __init__(self, nome, RA, curso, nota):
        self.nome = nome
        self.RA = RA
        self.curso = curso
        self.nota = nota

# poderia ter usado parâmetros posicionais também
ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
print(ana.nome, ana.RA, ana.curso, ana.nota)
```

Podemos acessar os atributos da instância usando o **.**

- i.e., **objeto.atributo**.
- Para a leitura ou escrita.

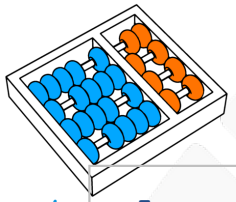


Um método para a classe **Estudante**

```
1  class Estudante:
2      def __init__(self, nome, RA, curso, nota):
3          self.nome = nome
4          self.RA = RA
5          self.curso = curso
6          self.nota = nota
7
8      def aprovado(self):              # define o método aprovado
9          return self.nota >= 5.0
10
11 ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
12
13 if ana.aprovado():                   # chama o método aprovado
14     print(ana.nome, "está aprovado")
15 else:
16     print(ana.nome, "está reprovado")
```

Note que não passamos parâmetro para **ana.aprovado()**:

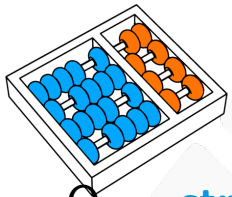
- O Python já sabe que **self** é **ana**.



Outro Método — Imprimindo o estudante

```
1  class Estudante:
2      def __init__(self, nome, RA, curso, nota):
3          self.nome = nome
4          self.RA = RA
5          self.curso = curso
6          self.nota = nota
7
8      def aprovado(self):
9          return self.nota >= 5.0
10
11     def imprime(self):
12         print("RA:", self.RA,
13               "Nome:", self.nome,
14               "Curso:", self.curso,
15               "Nota:", self.nota)
16
17 ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
18 ana.imprime()
19 # RA: 123456 Nome: Ana Curso: 42 Nota: 10.0
```

O Python nos deixa fazer algo ainda mais legal do que isso...



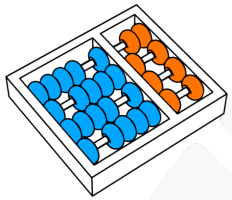
Método __str__

O __str__ é chamado quando precisa converter para str!

```
1  class Estudante:
2      def __init__(self, nome, RA, curso, nota):
3          self.nome = nome
4          self.RA = RA
5          self.curso = curso
6          self.nota = nota
7
8      def aprovado(self):
9          return self.nota >= 5.0
10
11     def __str__(self):
12         return (f"RA: {self.RA} Nome: {self.nome}" +
13                 f"Curso: {self.curso} Nota: {self.nota}")
14
15 ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
16 print(ana)
```

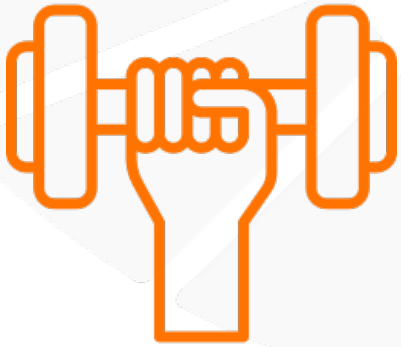
Ele é o que chamamos de **método mágico**:

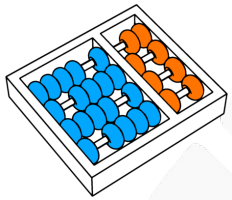
- E existem vários outros que podemos definir.



Criando uma classe

Vamos fazer um exercício?



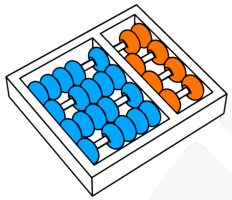


Exercício

Crie uma classe **Turma** que:

- Armazena estudantes.
- Permite adicionar estudantes.
- Permite imprimir os estudantes.
- Permite imprimir os estudantes aprovados.
- Permite imprimir os estudantes reprovados.

Encapsulamento



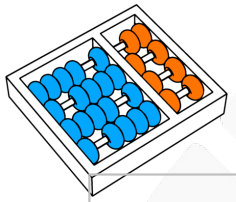
Encapsulamento

De maneira geral, é ruim escrevermos algo do tipo:

```
1 ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
2 ...
3 ana.nota = 9.3
```

Isso porque estamos acessando o atributo diretamente:

- **nota** pode ser apenas entre 0 e 10...
- Queremos alterar **nota** apenas através de um método!
- Chamamos isso de **encapsulamento**:
  - Deveríamos acessar o objeto apenas pelos seus métodos.
  - Já que os atributos são de sua **responsabilidade**.



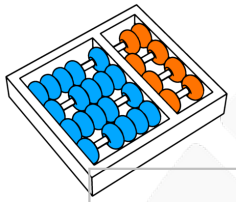
Primeira versão

```
1  class Estudante:
2      def __init__(self, nome, RA, curso, nota):
3          self.nome = nome
4          self.RA = RA
5          self.curso = curso
6          self.set_nota(nota)
7
8      def get_nota(self):
9          return self.nota
10
11     def set_nota(self, nota):
12         self.nota = nota
13
14 ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
15 ana.set_nota(9.3)
```

Regras:

- Leitura deve ser feita pelo método **get_nota**.
- Escrita deve ser feita pelo método **set_nota**.

Mas ainda podemos escrever **ana.nota = 9.3**...

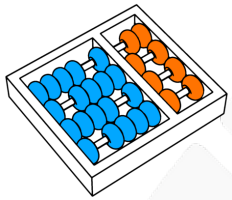


Segunda versão — Um passo atrás...

```
1  class Estudante:
2      def __init__(self, nome, RA, curso, nota):
3          self.nome = nome
4          self.RA = RA
5          self.curso = curso
6          self._nota = nota
7
8      @property  # isso é chamado de decorator em Python
9      def nota(self):
10         return self._nota
11
12     @nota.setter
13     def nota(self, nota):
14         self._nota = nota
15
16 ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
17 ana.nota = 9.3
```

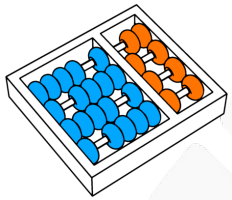
Voltamos a poder escrever **ana.nota = 9.3**:

- Porém, a função da linha 13 é sempre chamada!
- E, se formos ler, a função da linha 9 é sempre chamada!



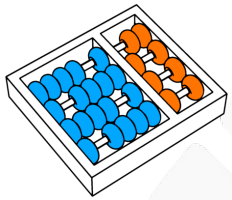
Terceira versão — Nota inválida

```
1  class Estudante:
2      def __init__(self, nome, RA, curso, nota):
3          self.nome = nome
4          self.RA = RA
5          self.curso = curso
6          self._nota = nota
7
8      @property
9      def nota(self):
10         return self._nota
11
12     @nota.setter
13     def nota(self, nota):
14         if nota < 0 or nota > 10:
15             raise ValueError("Nota inválida!")
16         self._nota = nota
17
18 ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
19 ana.nota = 10.3
```



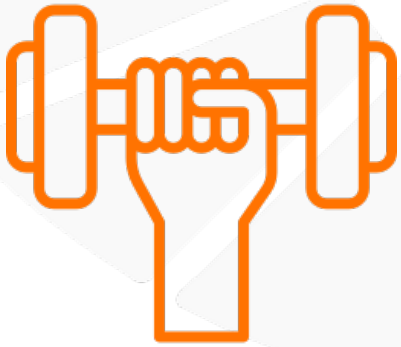
Observações

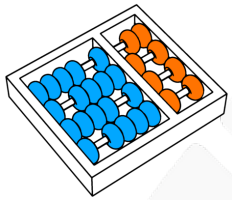
- Você poderia ter apenas a função de leitura (**@property**).
- Em geral, a boa pratica seria que todos os atributos fossem privados e acessados apenas por funções.
- Mas você não precisa se preocupar com isso nessa disciplina...
- O _ pode ser usado para métodos também:
  - Indica que o método não deve ser chamado de fora.
  - Pode ser um cálculo parcial, por exemplo.
  - Isto é, um método auxiliar.



Encapsulamento

Vamos fazer alguns exercícios

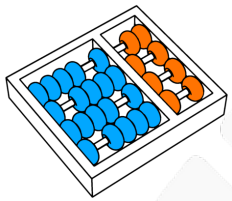




Exercícios

1. Refaça a classe Turma usando encapsulamento
2. Refaça a classe Estudante usando encapsulamento de forma que a nota do estudante é a média aritmética de três notas

Dataclasses

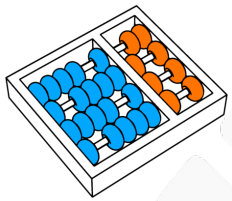


Dataclasses

É comum termos uma classe apenas para guardar dados. Ex:

```
1  class Estudante:
2      def __init__(self, nome, RA, curso, nota):
3          self.nome = nome
4          self.RA = RA
5          self.curso = curso
6          self.nota = nota
7
8      def aprovado(self):
9          return self.nota >= 5.0
10
11     def __str__(self):
12         return (f"RA: {self.RA} Nome: {self.nome}" +
13                 f"Curso: {self.curso} Nota: {self.nota}")
14
15 ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
16 print(ana)
```

Para simplificar a escrita, o Python adicionou as **dataclasses**.



Dataclasses

Código usando dataclass

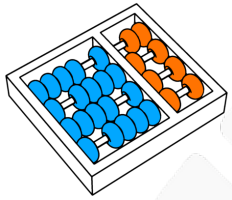
```python
from dataclasses import dataclass

@dataclass  # decorador para a classe estudante
class Estudante:
    nome: str
    RA: int
    curso: int
    nota: float

    def aprovado(self):
        return self.nota >= 5.0

ana = Estudante(nome="Ana", RA=123456, curso=42, nota=10.0)
print(ana)
```

É impresso **Estudante(nome='Ana', RA=123456, curso=42, nota=10.0)**.

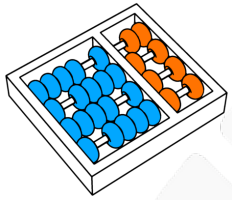


Dataclasses

Basicamente usando dataclasses você ganha várias coisas:

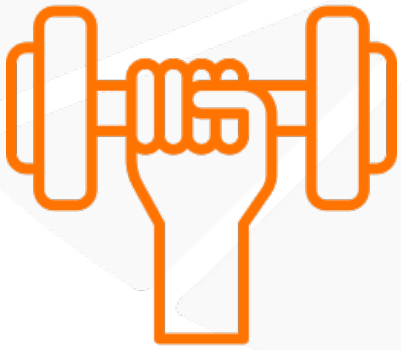
- Um método **__init__**.
- Um método **__repr__**.
- Um método **__eq__**.
- Outros métodos e comportamentos dependendo da opções passada para o decorador:
  - Ex: **@dataclass(order=True)**.
- Há também como ter valores padrão para atributos:
  - Porém precisa ter alguns cuidados.

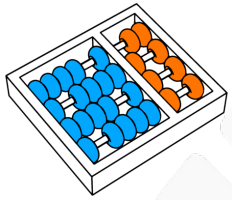
Sugestão de leitura:
`https://realpython.com/python-data-classes`



Dataclasses

Vamos fazer um exercício





Exercício

1. Refaça a classe Turma usando dataclass.

   Dica para não ter problemas:
   - **from dataclasses import dataclass, field**.
   - Defina a lista de estudantes na turma da seguinte forma: **estudantes: list = field(default_factory=list)**

Classes e Objetos

MC102 - Algoritmos e Programação de Computadores

05/25

18

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

UNICAMP