Live de Python #61

Programação orientada a objetos #1

Ajude a Live de Python

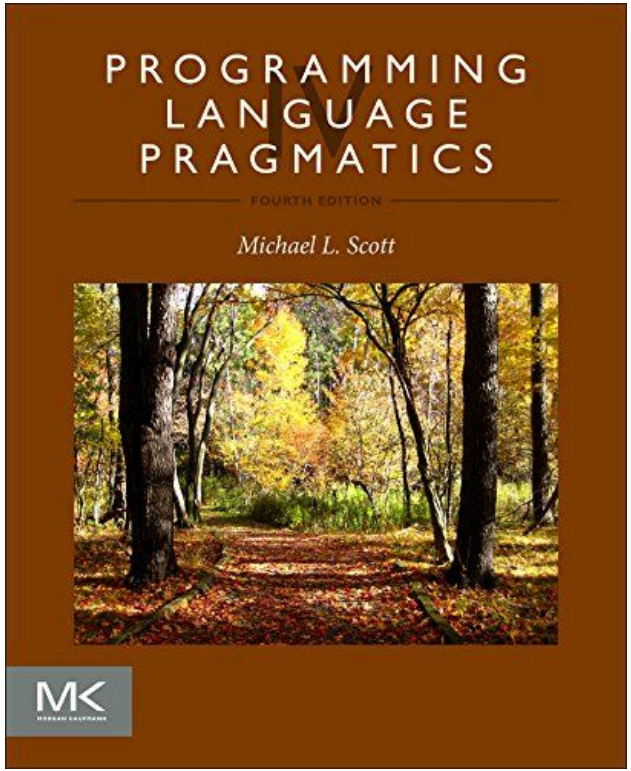
apoia.se/livedepython

picPay: @livedepython

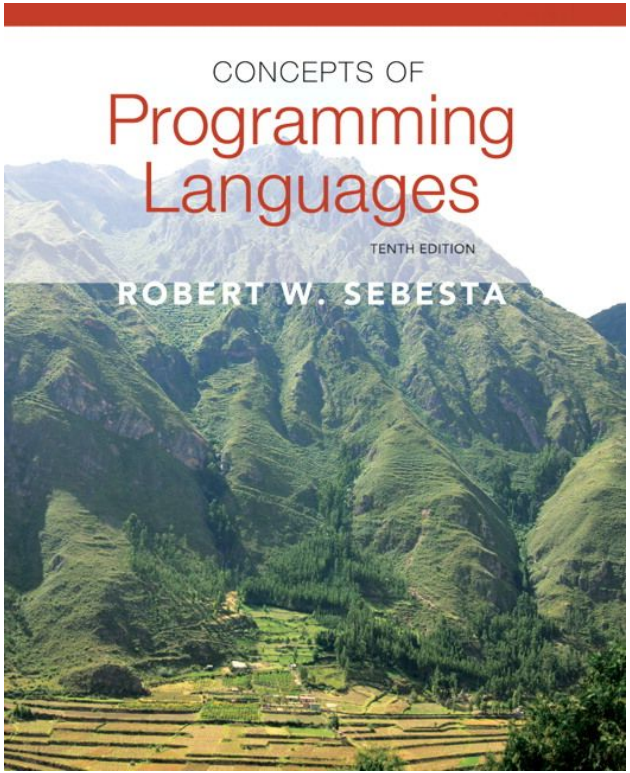
Roteiro

- Abstração
- Tipos definidos pelo usuário
- Nossa primeira abstração
- Colocando os pingos nos is

Referências



Caps: 7,9



Caps: 11, 12

Abstração

“Uma abstração é uma visão ou representação de uma **entidade** que inclui apenas os **atributos** mais significativos. Em um sentido geral, a abstração permite coletar **instâncias** de entidades em grupos nos quais seus atributos comuns não precisam ser considerados. Por exemplo, suponha que definimos pássaros como criaturas com os seguintes atributos: duas asas, duas pernas, uma cauda e penas.”

Sebesta - Concepts of programming languages 10th

Abstração

Pássaros

Abstração

Pássaros

Abstração

Pássaros

Abstração

Os pássaros têm características em comum. Como vimos antes, duas asas, duas patas. Quando formos descrever um pica-pau por exemplo, não há a necessidade de incluir as características de um pássaro. Apenas os que o diferem de outras espécies de pássaros.

Abstração

“No mundo das linguagens de programação, a abstração é uma arma contra a complexidade da programação. Seu objetivo é simplificar o processo de programação. É uma arma eficaz porque permite que os programadores se concentrem em atributos essenciais, ignorando os atributos subordinados.”

Sebesta - Concepts of programming languages 10th

Abstração

Na visão de Sebesta, temos dois tipos importantes de abstração na programação

- **Abstração de processos**
- **Abstração de dados**

Abstração de processos

Usando como base a função **sorted** do Python:

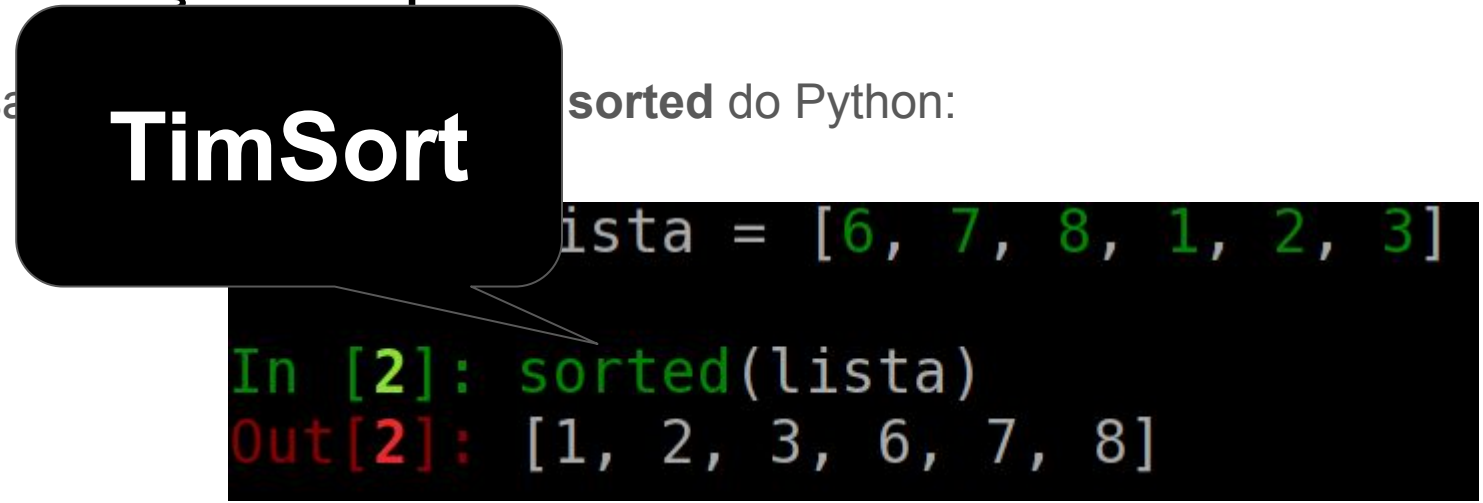
```
In [1]: lista = [6, 7, 8, 1, 2, 3]

In [2]: sorted(lista)
Out[2]: [1, 2, 3, 6, 7, 8]
```

A função sorted representa uma abstração de processos. Por exemplo, o usuário não precisa saber qual é a função de ordenação que está sendo usada. Poderia ser um merge sort, bubble sort, insertion sort.

Abstração de processos

Usa **sorted** do Python:



A função sorted representa uma abstração de processos. Por exemplo, o usuário não precisa saber qual é a função de ordenação que está sendo usada. Poderia ser um merge sort, bubble sort, insertion sort.

Abstração de dados

A abstração de dados pode ser definida como uma **representação de dados** de um **tipo de dados específico**.

(CALMAAAAAAAA)

Abstração de dados

Vamos imaginar um número.

1, 2, 3, 4 ...

O número é uma abstração de:

- **Uma soma de unidade de algo**
- **Quantidade determinada de algo**

2 pássaros voando

Abstração de dados

Um dado abstrato é chamado de **tipo**.

Exemplificando, em python temos vários tipos, ou várias abstrações:

- String -> Abstração de uma palavra
- int, float, complex... -> abstração de números
- Listas, tuplas, conjuntos -> abstrações de coleções de coisas abstratas

Tipos definidos pelo usuário

Embora existam tipos definidos em todas as linguagens existem momentos em que o usuário da linguagem precisa definir seus próprios tipos.

Os tipos criados pelo usuário tem que oferecer características associadas assim como os tipos definidos pela linguagem. Ou seja:

- Uma abstração que permita que o usuário a use sem necessariamente entender como aquilo é implementado (abstração de dados)
- Um conjunto de operações associadas a esse tipo (abstração de processos)

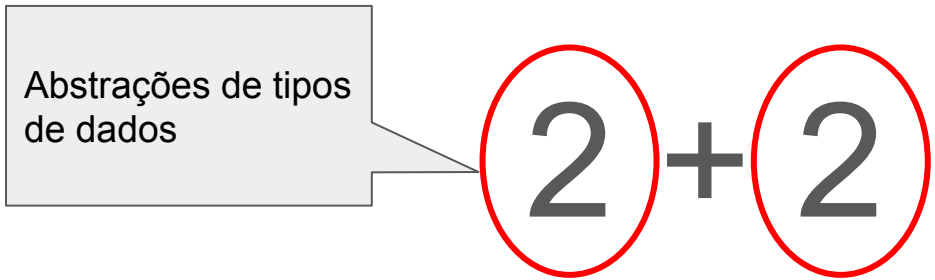
Tipos definidos pelo usuário

Vamos recapitular os números. Um tipo numérico é uma abstração de uma determinada quantia de algo (Abstração de tipo) e junto com ela são providos mecanismos que permitam que ele seja operável com outras entidades do mesmo tipo (Abstração de processos)

2 + 2

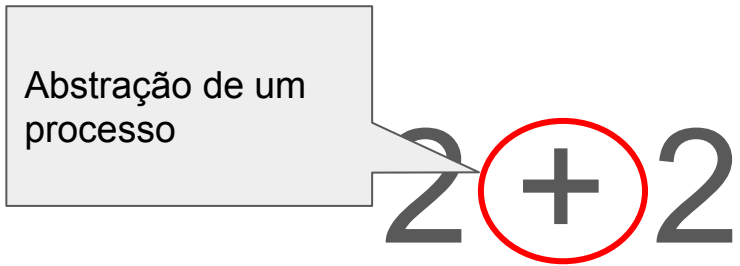
Tipos definidos pelo usuário

Vamos recapitular os números. Um tipo numérico é uma abstração de uma determinada quantia de algo (Abstração de tipo) e junto com ela são providos mecanismos que permitam que ele seja operável com outras entidades do mesmo tipo (Abstração de processos)



Tipos definidos pelo usuário

Vamos recapitular os números. Um tipo numérico é uma abstração de uma determinada quantia de algo (Abstração de tipo) e junto com ela são providos mecanismos que permitam que ele seja operável com outras entidades do mesmo tipo (Abstração de processos)



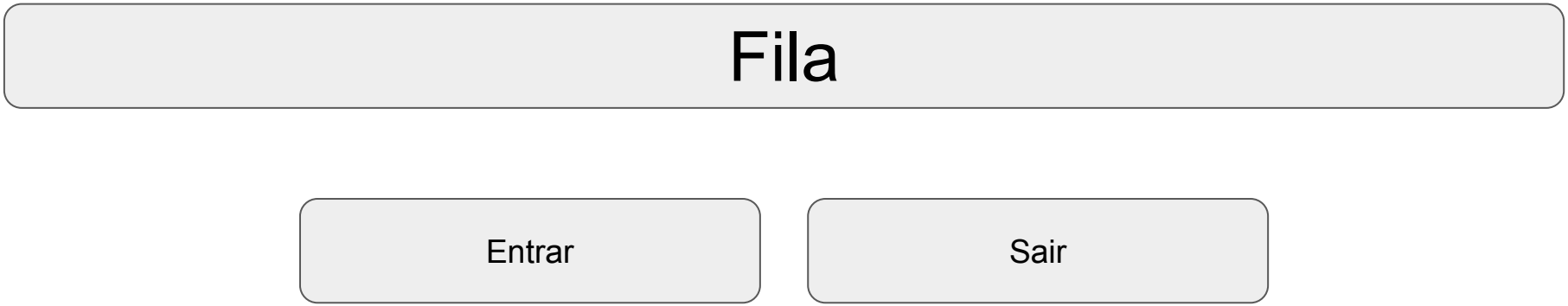
Tipos definidos pelo usuário

Existem algumas vantagens em usar tipos abstratos:

- Confiabilidade: os objetos só podem ser usados com suas determinadas operações
- Redução de código evidente (ele está lá, mas tá escondido)
- Redução de conflito de nomes
- ...

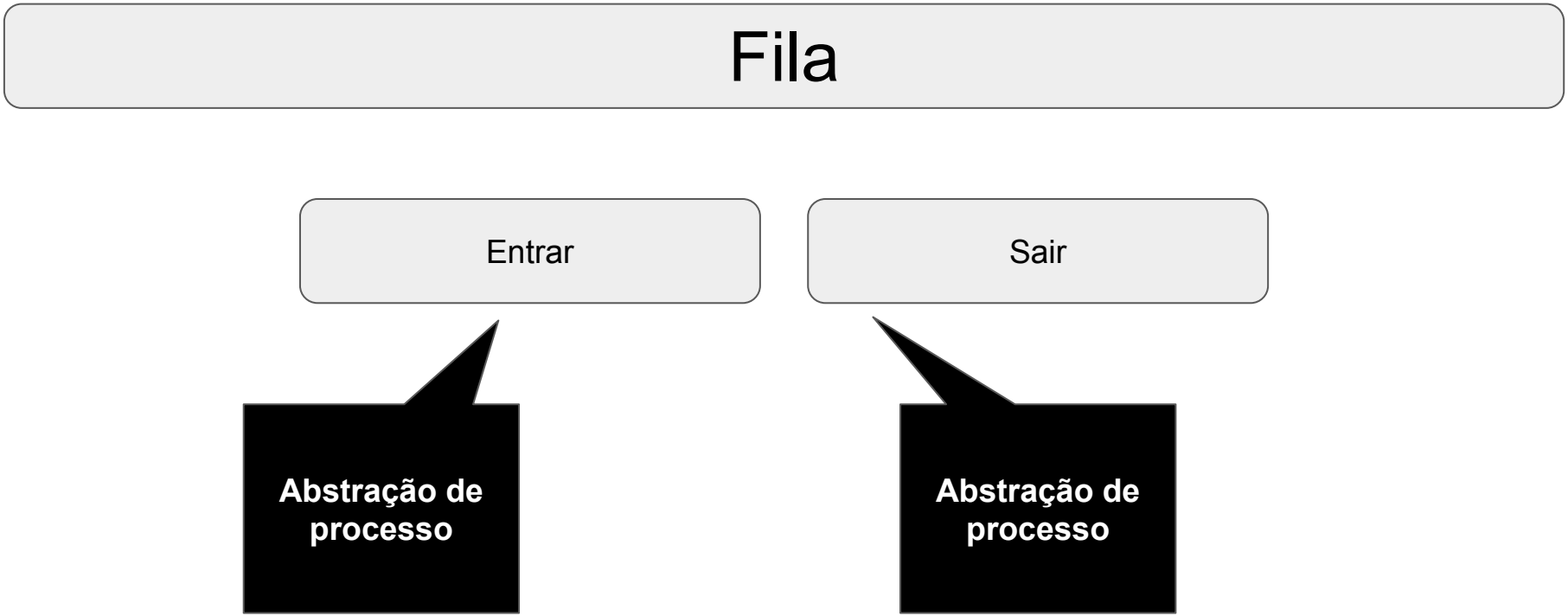
Nossa primeira abstração

Nossa primeira abstração



Nossa primeira abstração

Nossa primeira abstração



Agora estamos nivelados

Ou quase, Glossário

Classes (Dicionário)

- “Conjunto de pessoas que têm a mesma função, os mesmos interesses ou a mesma condição numa sociedade: a classe operária.”
- “Entidade lógica que satisfaz a certos axiomas, e que se pode representar intuitivamente como uma coleção de objetos.”
- “Conjunto dos alunos colocados sob a direção de um professor: uma classe turbulenta.”

https://www.dicio.com.br/classe/

Classes (Dicionário)

- “Conjunto de pessoas que têm a mesma função, os mesmos interesses ou a mesma condição numa sociedade: a classe operária.”
- “Entidade lógica que satisfaz a certos axiomas, e que se pode representar intuitivamente como uma coleção de objetos.”
- “Conjunto dos alunos colocados sob a direção de um professor: uma classe turbulenta.”

https://www.dicio.com.br/classe/

Pássaros

Classes

Na programação uma classe pode ser uma “estrutura” para **abstração de dados**. Uma maneira de dizer a linguagem que você quer construir seu próprio tipo.

```
In [1]: class Número:
   ...:     pass
   ...:

In [2]: class Pássaro:
   ...:     pass
   ...:

In [3]: class Pedra:
   ...:     pass
   ...:
```

Atributos

Atributos são **“Adjetivos”**. A aquilo que é atribuído a algo. Ou seja...

“O pássaro **preto**” -> preto é um adjetivo referente a cor do pássaro.

Nesse sentido “cor” é um **atributo**

Atributos

Ou seja....

Falando razamente “Atributos são variáveis internas de uma abstração de dados”

```
In [1]: class Pássaro:
   ...:     asas = 2
   ...:     bico = 1
   ...:

In [2]: Pássaro.asas
Out[2]: 2

In [3]: Pássaro.bico
Out[3]: 1
```

Métodos

Métodos são a **abstração de processos** de uma **abstração de dados**.

Ou seja, vamos relembrar dos números. Os números executam operações com outro números. Soma, subtração, divisão ....

São abstrações de processos **embutidas nos dados**

Métodos

Métodos no sentido mais cru de todos: “São funções internas da classe”.

```
In [5]: class Pássaro:
   ...:     def voar():
   ...:         print('voando')
   ...:     def pousar():
   ...:         print('pousando')
   ...:

In [6]: Pássaro.voar()
voando

In [7]: Pássaro.pousar()
pousando
```

Métodos

Métodos no sentido mais cru de todos: “São funções internas da classe”.

```
In [8]: class Calculadora:
   ...:     def soma(x, y):
   ...:         return x + 7
   ...:     def subtração(x, y):
   ...:         return x - y
   ...:

In [9]: Calculadora.soma(2, 2)
Out[9]: 9

In [10]: Calculadora.subtração(2, 2)
Out[10]: 0
```

Métodos

Métodos também pode manipular o **estado** dos atributos. Ou seja, as coisas podem ser mais dinâmicas. O uso disso acontece usando duas palavras específicas **self** e **cls.**

Por um momento vamos pensar em self e cls como quando o dado quer conversar com ele mesmo. Ou seja **myself**, do inglês “Eu mesmo”

Métodos

```
In [3]: class Pássaro:
   ...:     estado = 'indefinido'
   ...:
   ...:     def voar(self):
   ...:         self.estado = 'Voando'
   ...:         print(self.estado)
   ...:
   ...:     def pousar(self):
   ...:         self.estado = 'Parado'
   ...:         print(self.estado)
   ...:

In [4]: Pássaro.estado
Out[4]: 'indefinido'
```

Métodos

```
In [4]: Pássaro.voar()
---------------------------------------------------------
TypeError                 Traceback (most recent call last)
<ipython-input-4-901ac0c055ab> in <module>()
----> 1 Pássaro.voar()

TypeError: voar() missing 1 required positional argument: 'self'
```

Instância

Um aglicismo, ou seja, uma palavra aportuguesada de *instance*. Que nada mais quer dizer que “Exemplo”.

Ou seja, a classe representa a “classe” ou seja. No exemplo dos pássaros, a classe Pássaros é uma abstração do pássaro. Ela é **genérica**. Já a instância é O pássaro.

Ou seja, a classe como tipo é uma generalização de todos os pássaros. O exemplo é um representante único daquele grupo de entidades

instância

Tudo junto, agora

```
In [1]: class Pássaro:
   ...:     estado = 'indefinido'
   ...:
   ...:     def voar(self):
   ...:         self.estado = 'Voando'
   ...:         print(self.estado)
   ...:
   ...:     def pousar(self):
   ...:         self.estado = 'Parado'
   ...:         print(self.estado)
   ...:

In [2]: p1 = Pássaro()

In [3]: p2 = Pássaro()
```

Tudo junto, agora

```
In [1]: class Pássaro:
   ...:     estado = 'indefinido'
   ...:
   ...:     def voar(self):
   ...:         self.estado = 'Voando'
   ...:         print(self.estado)
   ...:
   ...:     def pousar(self):
   ...:         self.estado = 'Parado'
   ...:         print(self.estado)
   ...:

In [2]: p1 = Pássaro()

In [3]: p2 = Pássaro()
```

Atributo

Tudo junto, agora

```
In [1]: class Pássaro:
   ...:     estado = 'indefinido'
   ...:
   ...:     def voar(self):
   ...:         self.estado = 'Voando'
   ...:         print(self.estado)
   ...:
   ...:     def pousar(self):
   ...:         self.estado = 'Parado'
   ...:         print(self.estado)
   ...:

In [2]: p1 = Pássaro()

In [3]: p2 = Pássaro()
```

Método

Método

Tudo junto, agora

```
In [1]: class Pássaro:
   ...:     estado = 'indefinido'
   ...:
   ...:     def voar(self):
   ...:         self.estado = 'Voando'
   ...:         print(self.estado)
   ...:
   ...:     def pousar(self):
   ...:         self.estado = 'Parado'
   ...:         print(self.estado)
   ...:

In [2]: p1 = Pássaro()

In [3]: p2 = Pássaro()
```

Tudo junto, agora

```
In [1]: class Pássaro:
   ...:     estado = 'indefinido'
   ...:
   ...:     def voar(self):
   ...:         self.estado = 'Voando'
   ...:         print(self.estado)
   ...:
   ...:     def pousar(self):
   ...:         self.estado = 'Parado'
   ...:         print(self.estado)
   ...:

In [2]: p1 = Pássaro()

In [3]: p2 = Pássaro()
```

```
In [4]: p1.estado
Out[4]: 'indefinido'

In [5]: p1.voar()
Voando

In [6]: p1.estado
Out[6]: 'Voando'

In [7]: p2.estado
Out[7]: 'indefinido'
```

Montando a nossa fila