



FACULDADE DE TECNOLOGIA SENAC PELOTAS

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

# Algoritmos e Estruturas de Dados I

Faculdade de Tecnologia Senac Pelotas

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Prof. Edécio Fernando Iepsen

## ESTRUTURA DE DADOS

Rafael Albuquerque Pinto  
Lucas Plautz Prates  
Matheus da Silva Serpa  
Jana Maria Colares Gomes  
Cícero Macêdo Branco  
Rafael César Monteiro Nunes



# Estrutura de dados

Rafael Albuquerque Pinto; Lucas Pla...



Continuar lendo

ÚLTIMOS 30 DIAS

9 LIVROS ABERTOS





## Estrutura de dados

Pinto, Rafael Albuquerque



[Expandir](#) | [Reduzir](#)

▼ Iniciais	2
Sumário	9
▲ Introdução aos tipos abstratos de dados	13
O que é estrutura de dados?	14
Tipos de dados	16
Recursos necessários para criar um projeto com o uso de TADs	19
▼ Listas sequenciais: estáticas e dinâmicas	23

## 1 O que é estrutura de dados?

Na computação, a estrutura de dados consiste no modo de armazenamento e organização de dados em um computador. Quando estamos nos referindo ao armazenamento de dados, existe uma infinidade de opções nas quais o dado poderá ser armazenado de acordo com o seu tipo e base de dados. Os dados podem ser armazenados em pastas, arquivos de texto, planilhas ou até mesmo em um banco de dados instalado localmente ou em nuvem.

Os dados armazenados não consistem somente no local no qual são armazenados, mas nas relações entre eles. A organização dos dados é tão importante quanto o modo no qual são armazenamos, pois a correta escolha de sua categoria irá acarretar na *performance* ou na velocidade que o computador, ou usuário, irá demorar para encontrá-los. Logo, a correta escolha de como devemos armazenar e organizar os dados deverá levar em consideração o modo no qual eles serão utilizados no futuro.

No exemplo a seguir, você poderá identificar como as nossas ações diárias estão ligadas diretamente aos conceitos de dados e às suas respectivas estruturas, bem como o seu efeito se não utilizadas de forma adequada.



### Exemplo

# Estruturas de Dados: Listas

Uma lista é uma variável que pode possuir vários elementos. Pode-se acessar individualmente cada elemento a partir de um índice que acompanha a variável. Em Python, uma lista é representada como uma sequência de objetos separados por vírgula e **dentro de colchetes []**.

```
idade = []           # cria uma lista vazia  
idade = [20, 15, 30] # cria uma lista com elementos
```

A lista inicia pelo índice 0.





## Estrutura de dados

Pinto, Rafael Albuquerque



[Expandir](#) | [Reduzir](#)

▼ Iniciais 2

Sumário 9

^ Introdução aos tipos abstratos de dados 13

O que é estrutura de dados? 14

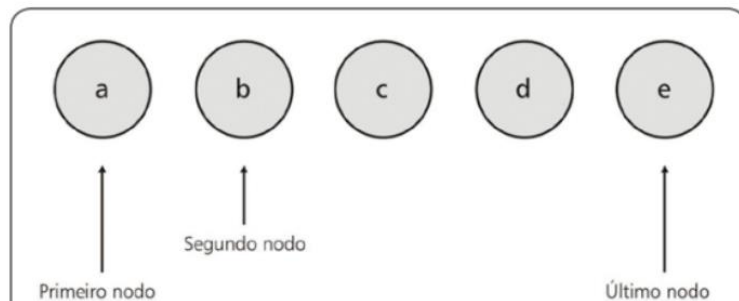
Tipos de dados 16

Recursos necessários para criar um 19

## 1 O que são listas?

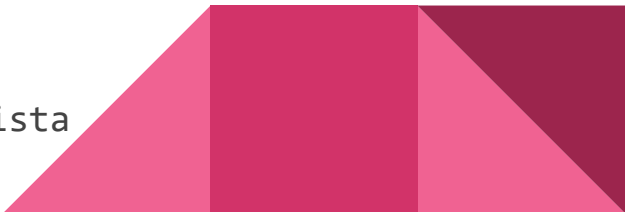
Na computação, uma lista pode ser definida como um conjunto de elementos do mesmo tipo, agrupados e identificados por um identificador único e separados entre si em “caixas”, chamados de nodos, que ocupam um endereço específico na memória. O relacionamento entre os nodos é definido por sua posição em relação aos demais nodos, assim como pessoas em uma fila, porém indexados na memória do computador.

Toda lista apresenta um nodo inicial, o primeiro elemento da lista. A partir deste seguirá uma sequência de nodos conforme uma ordem predefinida pelo programador, assim como uma fila de banco. Todos os nodos de uma lista têm, geralmente, o mesmo tipo de dado, podendo ser do tipo primitivo, como um inteiro ou abstrato, criado pelo programador. Na Figura 1 é possível verificar de forma ilustrada um exemplo de nodo com a sua respectiva posição.



# Funções para manipulação de listas

<code>idade.append(12)</code>	<code># acrescenta um elemento ao vetor idade</code>
<code>idade.pop()</code>	<code># retira um elemento do vetor. Sem parâmetros, retira o último. # Ou então, indica-se o número do elemento a ser removido</code>
<code>idade.insert(0, 5)</code>	<code># indica o local da inserção (índice, conteúdo)</code>
<code>idade.remove(12)</code>	<code># remove um elemento pelo conteúdo (se existir)</code>
<code>idade = range(5)</code>	<code># cria um vetor com os valores [0, 1, 2, 3, 4]</code>
<code>len(idade)</code>	<code># retorna o tamanho (número de elementos) do vetor</code>
<code>max(idade)</code>	<code># maior valor</code>
<code>min(idade)</code>	<code># menor valor</code>
<code>sum(idade)</code>	<code># soma os elementos do vetor</code>
<code>12 in idade</code>	<code># verifica se existe</code>
<code>idade.count(12)</code>	<code># conta o número de ocorrências</code>
<code>idade.sort()</code>	<code># classifica os elementos do vetor</code>
<code>idade.reverse()</code>	<code># inverte a ordem dos elementos da lista</code>



# Métodos `sort()` x `sorted()`, `reverse()` x `reversed()`

- `sort()` e `reverse()` modificam a lista original
- `sorted()` e `reversed()` retornam uma nova lista



# Funções da biblioteca *statistics*

## Averages and measures of central location

These functions calculate an average or typical value from a population or sample.

<code>mean()</code>	Arithmetic mean ("average") of data.
<code>fmean()</code>	Fast, floating point arithmetic mean, with optional weighting.
<code>geometric_mean()</code>	Geometric mean of data.
<code>harmonic_mean()</code>	Harmonic mean of data.
<code>median()</code>	Median (middle value) of data.
<code>median_low()</code>	Low median of data.
<code>median_high()</code>	High median of data.
<code>median_grouped()</code>	Median, or 50th percentile, of grouped data.
<code>mode()</code>	Single mode (most common value) of discrete or nominal data.
<code>multimode()</code>	List of modes (most common values) of discrete or nominal data.
<code>quantiles()</code>	Divide data into intervals with equal probability.

**Você deve importar a function da biblioteca:**

```
from statistics import median
```



`statistics.fmean(data, weights=None)`

Convert *data* to floats and compute the arithmetic mean.

This runs faster than the `mean()` function and it always returns a `float`. The *data* may be a sequence or iterable. If the input dataset is empty, raises a `StatisticsError`.

```
>>> fmean([3.5, 4.0, 5.25])  
4.25
```

Optional weighting is supported. For example, a professor assigns a grade for a course by weighting quizzes at 20%, homework at 20%, a midterm exam at 30%, and a final exam at 30%:

```
>>> grades = [85, 92, 83, 91]  
>>> weights = [0.20, 0.20, 0.30, 0.30]  
>>> fmean(grades, weights)  
87.6
```

If *weights* is supplied, it must be the same length as the *data* or a `ValueError` will be raised.

*New in version 3.8.*

*Changed in version 3.11:* Added support for *weights*.

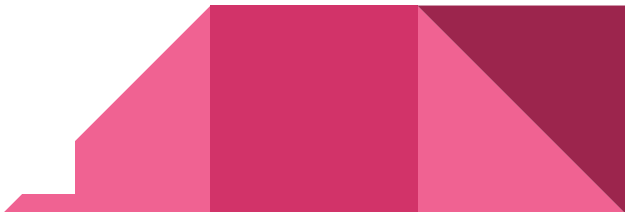


# Localizar item na lista: método .index()

A sintaxe do `index()` método fica assim:

```
my_list.index(item, start, end)
```

Vamos decompô-lo:

- `my_list` é o nome da lista que você está pesquisando.
  - `.index()` é o método de pesquisa que usa três parâmetros. Um parâmetro é obrigatório e os outros dois são opcionais.
  - `item` é o parâmetro necessário. É o elemento cujo índice você está procurando.
  - `start` é o primeiro parâmetro opcional. É o índice a partir do qual você iniciará sua pesquisa.
  - `end` o segundo parâmetro opcional. É o índice onde você terminará sua pesquisa.
- 

Se você tentar pesquisar um item, mas não houver correspondência na lista que está pesquisando, o Python lançará um erro como valor de retorno - especificamente, retornará um arquivo `ValueError`.

Isso significa que o item que você está procurando não existe na lista.

Uma maneira de evitar que isso aconteça é agrupar a chamada do `index()` método em um `try/except` bloco.

Se o valor não existir, haverá uma mensagem para o console dizendo que ele não está armazenado na lista e, portanto, não existe.

```
programming_languages = ["JavaScript","Python","Java","Python","C++","Python"]

try:
    print(programming_languages.index("React"))
except ValueError:
    print("That item does not exist")
```

5. Estruturas de dados — docs.python.org/pt-br/3/tutorial/datastructures.html

Python » Brazilian Portuguese » 3.10.2 » 3.10.2 Documentation » O tutorial de Python » 5. Estruturas de dados

anterior | próximo | módulos | índice

Busca rápida  Ir

## Tabela de Conteúdo

- 5. Estruturas de dados
  - 5.1. Mais sobre listas
    - 5.1.1. Usando listas como pilhas
    - 5.1.2. Usando listas como filas
    - 5.1.3. Compreensões de lista
    - 5.1.4. Compreensões de lista aninhadas
  - 5.2. A instrução `del`
  - 5.3. Tuplas e Sequências
  - 5.4. Conjuntos
  - 5.5. Dicionários
  - 5.6. Técnicas de iteração
  - 5.7. Mais sobre condições
  - 5.8. Comparando sequências e outros tipos

# 5. Estruturas de dados

Esse capítulo descreve algumas coisas que você já aprendeu em detalhes e adiciona algumas coisas novas também.

## 5.1. Mais sobre listas

O tipo de dado lista tem ainda mais métodos. Aqui estão apresentados todos os métodos de objetos do tipo lista:

**`list.append(x)`**  
Adiciona um item ao fim da lista. Equivalente a `a[len(a):] = [x]`.

**`list.extend(iterable)`**  
Prolonga a lista, adicionando no fim todos os elementos do argumento *iterable* passado como parâmetro. Equivalente a `a[len(a):] = iterable`.

**`list.insert(i, x)`**  
Insere um item em uma dada posição. O primeiro argumento é o índice do elemento antes do qual será feita a inserção, assim `a.insert(0, x)` insere um elemento na frente da lista e `a.insert(len(a), x)` e equivale a `a.append(x)`.

**`list.remove(x)`**  
Remove o primeiro item encontrado na lista cujo valor é igual a `x`. Se não existir valor

Tópico anterior


Documentação: <https://docs.python.org/pt-br/3/tutorial/datastructures.html>

# Módulos (funções definidas pelo usuário)

Permite dividir um programa em pequenos trechos de código, onde cada um tem uma função bem definida.

Além da facilidade em lidar com trechos menores, pode-se também fazer uso da reutilização de código, já que estes trechos devem ser bem independentes.

As funções são um dispositivo de estruturação de programas quase universal. Em termos simples, uma função é um dispositivo que agrupa um conjunto de instruções, de modo que elas possam ser executadas mais de uma vez em um programa.



# Funções em Python

```
def cad_usuario():          # cria uma função  
    print('...')
```

```
cad_usuario()              # chama a função
```



# Funções com passagem de parâmetros

```
def ver_numero(num):      # num é passado por parâmetro
    if num % 2 == 0:
        print(f'{num} é par')
    else:
        print(f'{num} é ímpar')
```

```
ver_numero(5)             # chama a função passando 5
```



# Funções com retorno de valor

```
def ver_numero(num):  
    if num % 2 == 0:  
        return f'{num} é par'           # retorno da função  
    else:  
        return f'{num} é ímpar'        # retorno da função  
  
resp = ver_numero(5)                    # retorno é atribuído a resp
```





# Funções com retorno de valor

```
def ver_numero(num):  
    if num % 2 == 0:  
        tipo = f'{num} é par'  
    else:  
        tipo = f'{num} é ímpar'  
    return tipo                                # retorno da função  
  
resp = ver_numero(5)                          # retorno é atribuído a resp
```



# Funções com vários parâmetros

```
def titulo(texto, traco):  
    print()  
    print(texto)  
    print(traco*40)
```

```
titulo("Algoritmos e Estrutura de Dados", "=")
```



# Funções com parâmetros com valores *default*

```
def titulo(texto, traco="-"):  
    print()  
    print(texto)  
    print(traco*40)
```

```
titulo("Algoritmos e Estrutura de Dados", "=")  
titulo("Programação Web")
```



# Passagem de parâmetros nomeados

```
def titulo(texto, traco="-", num=40):  
    print()  
    print(texto)  
    print(traco*num)
```

```
titulo("Aula 2")
```

# parâmetros posicionais

```
titulo(num=20, texto="Aula 2")
```

# parâmetros nomeados

