

© *AI2 – Advanced Institute for Artificial Intelligence*

O conteúdo do **Programa de Residência em IA** é de propriedade exclusiva do AI2 sendo cedido para uso, único e exclusivo, do(a) aluno(a), não podendo ser compartilhado, distribuído, comercializado e/ou gravado, seja da forma que for.

# Programação em Python

---

<https://advancedinstitute.ai>



# Programação Python

---

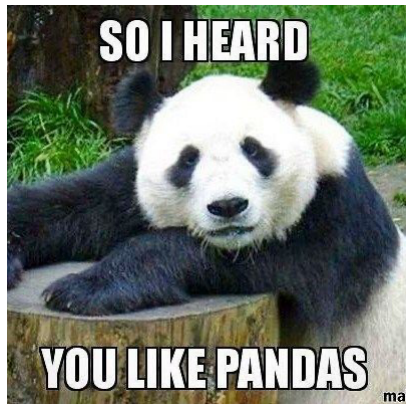
Manipulação de dados utilizando Pandas

## Referências e Fontes das Imagens

- ❑ Pandas Oficial
- ❑ Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython (Book)
- ❑ Learning the Pandas Library: Python Tools for Data Munging, Analysis, and Visualization (Book)

## Introdução

- ❑ Estruturas de dados de **alto nível**;
- ❑ Funções projetadas para tornar o trabalho com **dados estruturados** ou tabulares **rápido, fácil e expressivo**;
- ❑ Permite que o Python seja um ambiente de análise de dados poderoso e produtivo;
- ❑ Combina ideias de **operações de alto desempenho do NumPy** com recursos de manipulação de dados de planilhas e **bancos de dados relacionais**;
- ❑ Indexação sofisticada para facilitar *reshaping*, *splitting*, realizar agregações e *slicing/subsetting*;
- ❑ Estrutura de dados inspiradas no `data.frame` do **R**;



## Estruturas de Dados - Series

- `ndarray` unidimensional com rótulos no eixo
- Uma coluna em uma tabela ou planilha com o **mesmo tipo de dados**;
- Possui um único eixo;
- Array de *labels*, chamado de índice;
  - Podem ser **valores não numéricos** e até **valores repetidos**
  - Podem ser omitidos durante a construção do objeto do tipo `Series`

	apples
0	3
1	2
2	0
3	1

## Estruturas de Dados - Series

- Índice à esquerda e os valores à direita;
- Criação automática de índice (valores de 0 a  $N - 1$ ) caso não seja explicitado;

```
1 In [1]: import pandas as pd
2 In [2]: a = pd.Series([10, -5, 1], name="Dados", index=[a,b,c])
3 a      10
4 b     -5
5 c      1
6 Name: Dados, dtype: int64
```



## Estruturas de Dados - Series

- ❑ Acesso a valores e ao índice utilizando atributos `values` e `index`
- ❑ Indexação utilizando Índice

```
1 In [7]: a.index
2 Index(['a', 'b', 'c'], dtype='object')
3 # RangeIndex(start=0, stop=3, step=1) caso não seja explícito
4
5 In [8]: a.values
6 array([10, -5, 1])
```

- ❑ Indexação utilizando valores do índice:

```
1 In [13]: a['a'], a.a
2 (10, 10)
```

## Estruturas de Dados - Series

- ❑ Criação a partir de um dicionário;
- ❑ Atribuição do Índice *a posteriori*;

```
1 In [25]: estados = {"SP": 46.2, "MG": 21.2, "RJ": 17.3, "BA": 14.9}
2 In [26]: a = pd.Series(estados)
3 SP      46.2
4 ...
5 dtype: float64
6
7 In [29]: a.index = ["SP", "MG", "RJ", "RN"]
8 ...
9 RN      14.9
10 dtype: float64
```

## Estruturas de Dados - Series

- Valores do tipo NA (*not available*)
  - Convenção do R para especificar valores faltando ou erros de leitura;
    - NaN: *not a number*
  - Operações `pd.isnull()` e `pd.notnull()`

```
1 In [41]: index = ["MG", "RJ", "BA", "RN"]
2 In [42]: a = pd.Series(estados, index=index)
3 MG      21.2
4 ...
5 RN      NaN
6 dtype: float64
```

## Estruturas de Dados - Series

- Operações aritméticas e filtragem de dados em Series

```
1 In [48]: a * np.arange(1,5)
2 MG      21.2
3 RJ      34.6
4 BA      44.7
5 RN      NaN
6 dtype: float64
7
8 In [49]: a[a>15]
9 MG      21.2
10 RJ      17.3
11 dtype: float64
```

## Estruturas de Dados - Series

- ❑ Operações aritméticas - alinhamento pelo índice
- ❑ `add()`, `sub()`, `div()`, `floordiv()`, `mul()`, `pow()`

```
1 In [68]: estados1 = {"SP": 46.2, "MG": 21.2, "RJ": 17.3}
2 In [69]: estados2 = {"SP": 46.2, "MG": 10, "RJ": 17.3, "BA": 14.9}
3 In [70]: a+b # a.add(b)
4 BA      NaN
5 MG      31.2
6 RJ      34.6
7 SP      92.4
8 dtype: float64
```

## Estruturas de Dados - DataFrame

Series			Series			DataFrame		
	apples			oranges			apples	oranges
0	3	+	0	0	=	0	3	0
1	2		1	3		1	2	3
2	0		2	7		2	0	7
3	1		3	2		3	1	2

## Estruturas de Dados - DataFrame

- Abstração para Tabela
  - Cada coluna pode armazenar um tipo de dados
- Índices para linhas e colunas

```
1 In [75]: data = {"Estado": ["SP", "MG", "RJ", "BA"],
2               ...:      "Populacao": [46.2, 21.2, 17.3, 14.9],
3               ...:      "Ano": [2020, 2019, 2019, 2018]}
4 In [76]: a = pd.DataFrame(data)
5 In [77]: a
6      Estado  Populacao  Ano
7  0      SP      46.2  2020
8  1      MG      21.2  2019
9  2      RJ      17.3  2019
10 3      BA      14.9  2018
```

## Alinhamento por Índices

- Durante Operações, DataFrames são alinhados pelo índice;

```
1 In [119]: a = pd.DataFrame({"A": [1,2], "B": [3,4]})
2          A  B
3 0  1  3
4 1  2  4
5 In [121]: b = pd.DataFrame({"B": [4,5], "C": [6,7]})
6          B  C
7 0  4  6
8 1  5  7
9 In [123]: a.add(b)
10          A  B  C
11 0 NaN  7 NaN
12 1 NaN  9 NaN
```



## Adicionado e Removendo Colunas e Linhas

### □ Colunas:

- Para adicionar, indexar por uma coluna que não existe;
- Para remover, utilizar função `del()`

```
1 In [140]: b["X"] = 0
2          B  C  X
3  0  4  6  0
4  1  5  7  0
5 In [142]: del(b['X'])
6          B  C
7  0  4  6
8  1  5  7
```

## Adicionado e Removendo Colunas e Linhas

### □ Linhas:

- Para adicionar, utilizar a função `append()`
- Para remover, utilizar a função `drop()`

```
1 In [145]: b.append({"B": 50, "C" : 100}, ignore_index=True)
2         B      C
3  0      4      6
4  1      5      7
5  2     50     100
6 In [147]: b.drop(2)
7         B      C
8  0      4      6
9  1      5      7
```

## Principais funcionalidades

- ❑ **Reindexação:** utilizar a função `reindex()` para reordenar valores do `DataFrame`
- ❑ **Seleção de Elementos:**
  - Em um objeto do tipo `Series`, seleciona linhas;
  - Em um objeto do tipo `DataFrame` seleciona colunas;
  - Pode-se utilizar o valor do índice (`b['a']`), o número que representa o índice (`b[1:3]`) ou uma condição (`b[b['a'] < 1]`)
  - Seleção em `Dataframe` se restringe a colunas
    - Por conveniência a seleção de linhas pode ser feita, passando um *range* no `DataFrame` (`b[0:2]`)

## Principais funcionalidades - Cont.

- ❑ **Seleção de elementos** com atributos `loc` e `iloc`:
  - Seleção de dados utilizando o mesmo estilo do *Numpy*
  - Seleção baseada em rótulos dos índices (`loc`) e de valores inteiros (`iloc`)
- ❑ **Leitura de Fonte de Dados:**
  - Compatibilidade com múltiplos formatos;
  - Principais Formatos: CSV (`read_csv()`), JSON (`read_json()`), MS Excel (`read_excel()`), SQL (`read_sql()`)
    - Suporte a definição de índices, i.e., separação de coluna como índice.
    - Definição de tipos de dados

## Principais funcionalidades - Cont.

### □ Inspeção de dados:

- Inspecionando o começo ou fim do dataset com `head()` e `tail()`
- Buscando informações sobre as colunas: `info()`
- Informações sobre formato e índices: `.shape`, `.index`, `.columns`
- Verificando valores *NA*: `isnull().sum()`
- Sumário estatístico: `describe()` para variáveis categóricas e numéricas
- Contagem de ocorrências de valores em uma coluna: `value_counts()`

Dúvidas?