Advanced Institute for Artificial Intelligence

# Programação em Python

https://advancedinstitute.ai

# Programação Python

## Manipulação de dados utilizando Pandas

# Referências

## Referências e Fontes das Imagens


- 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)
- Página Oficial Pandas

# Pandas Framework

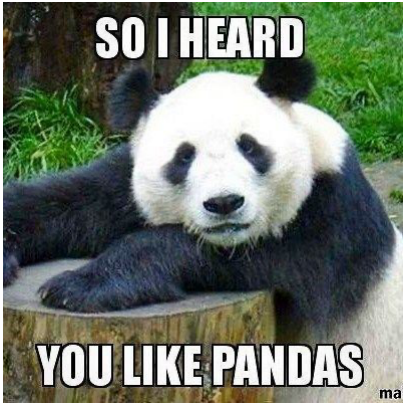
## Introdução

- 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***;

## Introdução

- Para trabalharmos com alguns projetos, precisamos utilizar bibliotecas específicas;
- Pandas está sempre atrelado a bibliotecas de processamentos numéricos como Numpy e SciPy;
- No caso de bibliotecas de análise como *scikit-learn* o Pandas também estará atuante no fornecimento de informações.

# Pandas Framework



## 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;

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

## Estruturas de Dados - `Series`

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

```
In [7]: a.index
Index(['a', 'b', 'c'], dtype='object')
# RangeIndex(start=0, stop=3, step=1) caso não seja explicito

In [8]: a.values
array([10, -5,  1])
```

- Indexação utilizando valores do índice:

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

## Estruturas de Dados - `Series`

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

```
In [25]: estados = {"SP": 46.2, "MG": 21.2, "RJ": 17.3, "BA": 14.9}
In [26]: a = pd.Series(estados)
SP    46.2
...
dtype: float64

In [29]: a.index = ["SP", "MG", "RJ", "RN"]
...
RN    14.9
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()`**

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

## Estruturas de Dados - `Series`

- Operações aritméticas e filtragem de dados em `Series`

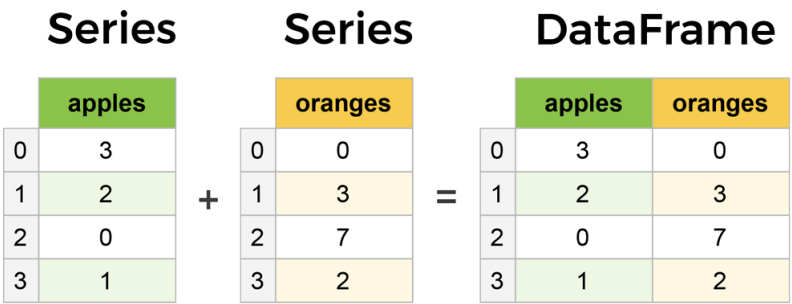
```
In [48]: a * np.arange(1,5)
MG    21.2
RJ    34.6
BA    44.7
RN     NaN
dtype: float64

In [49]: a[a>15]
MG    21.2
RJ    17.3
dtype: float64
```

## Estruturas de Dados - `Series`

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

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

## Estruturas de Dados - `DataFrame`



## Estruturas de Dados - `DataFrame`

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

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

## Alinhamento por Índices

- Durante Operações, **`DataFrames`** são alinhados pelo índice;

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

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

## Adicionado e Removendo Colunas e Linhas

- Linhas:
  - Para adicionar, utilizar a função **append()**
  - Para remover, utilizar a função **drop()**

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