

# Fundamentos Matemáticos e Computacionais de Machine Learning

Especialização em Machine Learning e Big Data



---

Profa. Dra. Juliana Felix

jufelix16@uel.br



# Estruturas de Dados em Python

# Estruturas de Dados do Python

O Python possui 4 estruturas de dados básicas

- Lists - Listas
- Tuples - Tuplas
- Sets - Conjuntos
- Dictionaries - Dicionários



# Listas

*(Lists)*

# Listas

- Uma lista é nada mais do que uma lista de coisas.
- Esta lista pode conter números, strings, objetos, etc
- A lista pode conter elementos com características similares.
- Ex.
  - Lista de Convidados
    - João, Maria, Pedro, Carlos, Janete, ...
  - Lista de Compras
    - Roupa, Sapatos, Cinto, Meia, Bolsa, ...
  - Lista de números do último sorteio da Mega Sena



# Listas

Para criar listas no Python é necessário:

- Utilizar o símbolo `[]` (colchetes) para definir as listas;
- Armazenar a lista em uma VARIÁVEL;
- Separar itens da lista usando vírgula;

Exemplo:

```
>>> lista_compras = [ 'banana', 'laranja' , 'maçã' ]  
  
>>> print(lista_compras)  
  
['banana', 'laranja', 'maçã']
```

# Listas

Para acessarmos um item da lista vamos utilizar a estrutura:

```
>>> Nomedalista[ posição ]
```

- Buscando a posição [1]:

```
>>> print(lista_compras[1])
```

*laranja*

- Buscando a posição [0]:

```
>>> print(lista_compras[0])
```

*banana*

- Se tentarmos buscar a posição [3] teremos um erro.
- Isso ocorre, porque não existe uma posição 3, apenas as as posições [0], [1], [2].

# Listas

Outro ponto interessante é que conseguimos utilizar as posições negativas dentro da lista para buscar os dados na ordem inversa!

- O uso de números negativos permite acessar a lista na ordem inversa:

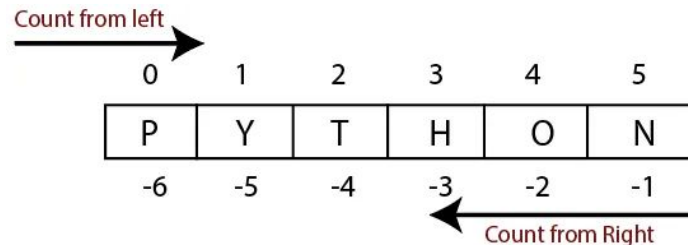
```
>>> print(lista_compras[-1])
```

*maçã*

```
[banana, laranja, maçã ]
```

```
[ 0, 1, 2 ] -> Posições números positivos
```

```
[ -3, -2, -1 ] -> Posições números negativos
```





# Listas

Para adicionar um item à lista:

- `Lista.append()`: adiciona o item ao final da lista;
- `Lista.insert()`: insere um item na lista na posição indicada

# Listas

Para deletar um item da lista:

- `Lista.del()`: remove um item da lista baseado na posição indicada;
- `Lista.remove()`: remove um item baseado no seu valor e não na sua posição;
- `Lista.pop()`: remove da lista\_compras o último item, mas não o exclui.

# Listas

No exemplo abaixo tentamos incluir o item 'carro' à nossa lista\_compras com o `.append()`:

```
>>> lista_compras.append('carro')  
>>> print(lista_compras)  
['banana', 'laranja', 'maçã', 'carro']
```

# Listas

Utilizando del para remover item com base na posição indicada:

```
>>> del lista_compras[3]
>>> print(lista_compras)
['banana', 'laranja', 'maçã']
```

# Listas

Já no código abaixo, adicionamos 'carro' à lista na posição indicada, com `.insert()`:

```
>>> lista_compras.insert(1, 'carro')  
>>> print(lista_compras)  
['banana', ' carro', 'laranja', 'maçã']
```

# Listas

Gerando uma lista com uma sequência de números:

```
>>> seq = list(range(5))
```

```
>>> print(seq)
```

```
[0, 1, 2, 3, 4]
```

# Listas

Para obter o comprimento da lista, você pode usar a função `len()`

```
>>> lista_compras = [ 'banana', 'laranja' , 'maçã' ]
```

```
>>> print(len( lista_compras ))
```

```
3
```



# Tuplas

*(Tuples)*



# Tuplas

As tuplas são como as listas em Python

- São uma sequência arbitrária de elementos
- Cada elemento pode ser acessado por um índice inteiro

```
Python 3.10.11 (main, Apr 20 2023, 13:58:42) [Clang 14.0.6 ] on darwin
>>> aluno = ('Ana Maria', 2022, 1)
>>> aluno[0]
'Ana Maria'
>>> aluno[1]
2022
>>> aluno[2]
1
```

# Tuplas

Todavia, as Tuplas diferem das listas por serem criadas/definidas utilizando parênteses e não colchetes.

```
Python 3.10.11 (main, Apr 20 2023, 13:58:42) [Clang 14.0.6 ] on darwin
>>> aluno = ('Ana Maria', 2022, 1)
>>> aluno[0]
'Ana Maria'
>>> aluno[1]
2022
>>> aluno[2]
1
```

# Tuplas

No entanto, diferente das listas, os elementos de uma tupla são imutáveis, ou seja, não podem ser modificados.

```
>>> aluno[1] = 2023
```

```
Traceback (most recent call last):
```

```
File "/Volumes/MacintoshSSD/My Applications/PyCharm.app/Contents/plugins/python/helpers/pydev/pydevconsole.py", line 364, in runcode
```

```
    coro = func()
```

```
File "<input>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item assignment
```

# Tuplas

- Podemos acessar os elementos de uma tupla, mas não podemos modificá-los
  - Todavia, podemos construir uma outra tupla e substituir a anterior
  - Além disso, podemos fazer concatenação e repetição de tuplas

```
>>> aluno1 = ('Bernardo', 2022, 1)
>>> aluno2 = ('Bianca', 2023, 1)
>>> aluno1 = (aluno1[0], 2023, aluno1[2])
>>> alunos = aluno1 + aluno2
>>> alunosR1 = aluno1 * 2
>>> alunosR2 = aluno2 * 4
>>> alunos
('Bernardo', 2023, 1, 'Bianca', 2023, 1)
>>> alunosR1
('Bernardo', 2023, 1, 'Bernardo', 2023, 1)
>>> alunosR2
('Bianca', 2023, 1, 'Bianca', 2023, 1, 'Bianca', 2023, 1, 'Bianca', 2023, 1)
```

# Tuplas

- Ao definir tuplas com zero ou um elemento, é preciso estar atento a pequenos detalhes
  - Tuplas vazias são definidas por um par de parênteses vazio
  - Tuplas com 1 elemento devem possuir uma vírgula depois do elemento para o Python saber que são tuplas e não um valor isolado

```
>>> tupla1 = ()      # Isto criará uma tupla vazia
>>> tupla2 = (1)     # Isto não criará uma tupla, e sim uma variável com 1 inteiro
>>> tupla3 = (1,)    # Isto criará uma tupla com um elemento
>>> tupla4 = (1, 2)  # Isto criará uma tupla de dois elementos, e equivale a tupla4 = (1, 2,)
```

# Tuplas

- Tuplas podem ser "desempacotadas" para variáveis distintas

```
>>> aluno1 = ('Bernardo', 2022, 1)
```

```
>>> (nome, ano, semestre) = aluno1
```

```
>>> nome  
'Bernardo'
```

```
>>> ano  
2022
```

```
>>> semestre  
1
```

# Tuplas

- O processo de criar e desempacotar tuplas podem ocorrer simultaneamente, isto permite truques interessantes, como de fazer SWAP entre variáveis

```
>>> a = 2
```

```
>>> b = 3
```

```
>>> a, b = b, a
```

```
>>> a
```

```
3
```

```
>>> b
```

```
2
```

# Tuplas

Por que usar tuplas?

- Apesar de listas e tuplas serem semelhantes, elas são normalmente utilizadas com propósitos diferentes
  - Tuplas são imutáveis e normalmente contém elementos de tipos diferentes que são acessados via unpacking ou indexação
  - Listas são mutáveis e normalmente contém elementos do mesmo tipo que são acessados via indexação ou iteração



# Tuplas

Por que usar tuplas?

- Tuplas geralmente são úteis para se unir um conjunto pequeno de dados em uma única variável, e indexá-lo ou desempacotá-lo quando necessário

## Exemplo:

- Passagem de parâmetros em funções – A função pode ter uma única variável como entrada, porém esta pode ser uma tupla, o que permite uma passagem de parâmetros bastante flexível



# Dicionários

*(Dictionaries)*

# Dicionários

- São estruturas de dados que implementam associações entre pares de valores
- O primeiro elemento do par é chamado de **chave** e o outro de **conteúdo**, ou **valor**
- Cada chave é associada a um (e um só) **conteúdo**

# Dicionários

- Exemplo: **Inventário**
  - Um inventário que associa produtos a quantidades disponíveis
    - Miojo: 10
    - Ovo: 12
    - Leite: 2
    - Pão: 5
- Esse problema poderia ser representado por uma lista de tuplas. Porém, dicionários permitem busca indexada pela chave
  - Não é necessário percorrer a lista procurando um item

# Dicionários

- Um dicionário é definido da seguinte forma

```
>>> nome_do_dicionario = { chave1:conteudo1, ..., chaveN:conteudoN }
```

- Exemplos

- Dicionário de palavras em português e inglês

- Note que as **chaves** do dicionário podem ser armazenadas em ordem variada

```
>>> dict = {'um': 'one', 'dois': 'two', 'tres': 'three'}  
>>> dict  
{ 'tres': 'three', 'um': 'one', 'dois': 'two' }
```

- Um dicionário pode ser criado vazio

```
>>> dict = {}  
>>> dict  
{ }
```

# Dicionários

- Acessando seus elementos
  - Cada elemento pode ser acessado por indexação usando a chave
  - Podemos alterar o conteúdo associado a uma chave
  - Novos itens podem ser adicionados a um dicionário
    - Basta fazer a atribuição a uma chave ainda não definida

```
>>> dict = {'um': 'one', 'dois': 'two', 'tres': 'three'}
>>> dict
{'um': 'one', 'dois': 'two', 'tres': 'three'}
>>> dict['quatro'] = 'four'
>>> dict
{'um': 'one', 'dois': 'two', 'tres': 'three', 'quatro': 'four'}
```

# Dicionários

- Um dicionário é uma classe e, portanto, possui diversos métodos já definidos
  - Um dos jeitos mais simples de manipular dicionários é utilizar os métodos que já fazem parte dele
- Forma geral de uso dos métodos
  - `nome_dicionario.nome_método()`

# Dicionários

- `clear()`: remove todos os elementos do dicionário

```
>>> dict = {'um': 'one', 'dois': 'two', 'tres': 'three'}  
>>> dict.clear()  
>>> dict  
{ }
```

- `copy()`: cria uma cópia do dicionário (atribuição não cria uma cópia)

```
>>> dict = {'um': 'one', 'dois': 'two', 'tres': 'three'}  
>>> dict2 = dict.copy()  
>>> id(dict)  
4364849088  
>>> id(dict2)  
4365092928
```

- Se você atribuir `dict` para `dict2` (`dict2 = dict`), ambos irão compartilhar a mesma memória.



# Dicionários

- Se você acessar um dado pela sua chave de forma direta (utilizando ['chave']) e a chave não existir, você terá um erro
- Assim, o recomendado é utilizar o método *get(chave,valor)* que obtém o conteúdo de chave.
  - Caso a chave não exista, retorna valor especificado após a vírgula

```
>>> dict[ 'cinco']
```

```
Traceback (most recent call last):
```

```
File "/Volumes/MacintoshSSD/My Applications/PyCharm.app/Contents  
  /plugins/python/helpers/pydev/pydevconsole.py", line 364, in runcode  
    coro = func()
```

```
File "<input>", line 1, in <module>
```

```
KeyError: 'cinco'
```

```
>>> dict.get('cinco', 'five')  
'five'
```

# Dicionários

## Métodos sobre dicionários

- *items()*: retorna uma lista com todos os pares chave/conteúdo do dicionário
- *keys()*: retorna uma lista com todas as chaves do dicionário
- *values()*: retorna uma lista com todos os valores do Dicionário

```
>>> dict.items()  
dict_items([('um', 'one'), ('dois', 'two'), ('tres', 'three'), ('quatro', '4')])
```

```
>>>> dict.keys()  
dict_keys(['um', 'dois', 'tres', 'quatro'])
```

```
>>> dict.values()  
dict_values(['one', 'two', 'three', '4'])
```

# Dicionários

## Métodos sobre dicionários

- `pop(chave)`: obtém o valor correspondente a chave e remove o par chave/valor do dicionário
- `popitem()`: retorna e remove um par chave/valor aleatório do dicionário. Pode ser usado para iterar sobre todos os elementos do dicionário

```
>>> dict.pop('um')  
'one'
```

```
>>>> dict.popitem()  
{'tres': 'three'}
```



# Conjuntos

(Sets)

# Conjuntos (Sets)

- Conjuntos são usados para armazenar múltiplos itens em uma única variável
- Em um conjunto (set) os itens
  - Não possuem ordenação específica
  - Não possuem valores duplicados
    - Duplicatas serão ignoradas na inserção ou em outras operações
  - Não são modificáveis
    - Porém podem ser removidos e novos podem ser adicionados
  - Não são indexáveis

```
>>> meu_conjunto = {'maçã', 'banana', 'cereja'}
```

# Conjuntos (Sets)

- Em conjuntos você pode adicionar elementos de tipos diferentes

```
>>> meu_conjunto = {'maçã', 'banana', 'cereja', True, 1, 2}
```

- Em conjuntos, True e 1, e False e 0 são considerados o mesmo valor para conjuntos

# Conjuntos (Sets)

- Conjuntos podem ser criados com o operador/construtor **set**

```
meu_conjunto = set(('maçã', 'banana', 'cereja')) # Observe os duplo-parênteses
```

- Você pode criar conjuntos vazios

```
meu_conjunto = set( )
```

- Valores podem adicionar ou remover itens com *add(item)* e *remove(item)*, ou *discard(item)*

```
meu_conjunto.add(1)
```

```
meu_conjunto.add(2)
```

```
meu_conjunto.remove(1)      # Se o 1 não existir, o método remove lança um erro
```

```
meu_conjunto.discard(3)    # Se o 3 não existir, o método discard não retorna nada
```

# Conjuntos (Sets)

- Para acessar os itens, você precisa iterar sobre eles

```
>>> meu_conjunto = set(('maçã', 'banana', 'cereja'))  
>>> for fruta in meu_conjunto:  
    print(fruta)
```



# Conjuntos (Sets)

- Você pode adicionar um conjunto em outro conjunto

```
meu_conjunto = set(('maçã', 'banana', 'cereja'))  
tropical = {"abacaxi", "manga", "mamão"}  
meu_conjunto.update(tropical)
```

- Você pode adicionar uma lista em um conjunto

```
meu_conjunto = set(('maçã', 'banana', 'cereja'))  
lista_frutas = ['kiwi', 'laranja']  
meu_conjunto.update(lista_frutas)
```

# Conjuntos (Sets)

- *pop()* remove um item aleatório
- *clear()* esvazia o conjunto
- *del()* deleta o conjunto completamente

# Conjuntos (Sets)

## Operações em conjuntos

- União - union ( ) / equivalente a update

```
set1 = {"a", "b", "c"}
```

```
set2 = {1, 2, 3}
```

```
set3 = set1.union(set2)
```

- Intersecção - intersection\_update( )

```
x = {"apple", "banana", "cherry"}
```

```
y = {"google", "microsoft", "apple"}
```

```
z = x.intersection(y)
```

- Diferença simétrica - symmetric\_difference\_update( )

```
x = {"apple", "banana", "cherry"}
```

```
y = {"google", "microsoft", "apple"}
```

```
x.symmetric_difference_update(y)
```