



- Listas
- Tuplas
- Sets (Conjuntos)
- Dictionaries

Python

Lógica de Programação

Estrutura de Dados com Python

01 Introdução

- Estruturas de dados são basicamente um conjunto de dados armazenados na memória de modo a fazer sentido.
- Para que possamos entender melhor, imagine que, eu queira criar uma lista com todos os itens que vou comprar no supermercado. Para isto, o Python tem a estrutura de dados chamada de lista, que é apenas uma das estruturas existentes.

02 Básico sobre Listas

- Imagine que você queira criar uma lista de todos os itens de uma feira. A primeira vista, você criaria uma variável para cada uma dos itens, não é mesmo? Mas, para isso você tem as listas, que aprenderemos agora. No exemplo a seguir, temos uma lista de *strings*, contendo 3 elementos.

```
1 listaFeira = ['maca', 'banana', 'uva']
```

- A lista acima, é uma lista com 3 elementos, na qual a primeira posição temos uma maçã, na segunda uma banana e na terceira uma uva. Entretanto, em Python, a primeira posição é zero, de modo que, a maçã está na posição 0, banana na posição 1 e uva na posição 2. A analogia é apresentada na imagem a seguir.



- A lista acima, é uma lista com 3 elementos, na qual a primeira posição temos uma maçã, na segunda uma banana e na terceira uma uva. Entretanto, em Python, a primeira posição é zero, de modo que, a maçã está na posição 0, banana na posição 1 e uva na posição 2. A analogia é apresentada na imagem a seguir.

```
1 listaFeira = ['maca', 'banana', 'uva']
2
3 print(listaFeira[0])
4 print(listaFeira[1])
5 print(listaFeira[2])
6
7 primeiroItemLista = listaFeira[0]
8 print('o primeiro item
   é', primeiroItemLista)
```

Saída

```
maca
banana
uva

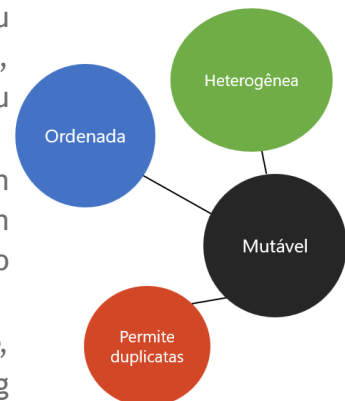
o primeiro item é maca
```

- Podemos verificar no código anterior que, `listaFeira[0]` obtém o primeiro elemento da lista ('maçã'), descrito em detalhes na imagem a seguir.
- Pode se tratar de uma lista, podemos criar uma variável para receber uma cópia dos elementos armazenados em alguma das posições da lista, algo como aconteceu na linha 7 do código anterior, na qual criamos a variável `primeiroItemLista`, que agora salva a 'maçã', que está na primeira posição (posição 0).

```
listaFeira = ['maca', 'banana', 'uva']
               ↑       ↑       ↑
             listaFeira[0] listaFeira[1] listaFeira[2]
```

03 Propriedades de Listas

- Os atributos que definem uma lista são apresentados a seguir.
- **Mutável** significa "mudável", ou seja, depois de criada a lista, podemos adicionar, remover ou modificar os elementos
- **Ordenada** significa que, cada item tem um índice (index) e seguem uma ordem (os novos são adicionados no final)
- **Permite duplicatas** significa que, se eu adicionar uma String 'Daniel', eu posso adicionar outra String 'Daniel' na mesma lista.
- **Heterogênea** significa que, uma lista pode ter ao mesmo tempo, elementos String, int e float, como o exemplo a seguir.
 - A primeira posição (0) é uma String ('maca')
 - A segunda posição (1) é um int
 - A terceira posição (1) é um float



```
1 lista = ['maca', 1, 1.0]
2
3 print(lista[0])
4 print(lista[1])
5 print(lista[2])
```

04 Criando Listas

- Anteriormente, nos antecipamos um pouco e criamos as nossas primeiras listas, não é mesmo? Agora, retomamos aos estágios preliminares de lista, que é a criação, que pode ser feita de duas formas, sendo elas:

- Usando o construtor chamado `list()`
- Utilizando colchetes `[]`, como fizemos anteriormente

```
1 lista1 = []
2 lista2 = list()
3
4 lista3 = ['A', 'B']
5 lista4 = list( ('A', 'B') )
```

- No exemplo acima, criamos as `lista1` e `lista2` vazias, isto é, sem elementos. Complementarmente, as `lista3` e `lista4` possuem duas Strings ('A' na posição 0 e 'B' na posição 1).
 - Atenção ao modo em que a `lista4` é criada, mais especificamente aos parênteses. Os parênteses indicados pela cor azul são relacionados ao construtor `list()`, enquanto que, os parênteses indicados pela cor vermelha estão associadas aos elementos, que por sua vez estão separados por vírgulas.

05 Acessando Elementos por Índice

- Os elementos da lista podem ser acessados por índice (em inglês, *index*). Como vimos anteriormente, os elementos em uma lista com Python, iniciam no índice 0 até tamanho - 1. Ou seja, em uma lista 3 elementos, vai de 0 a 2.

```
1 listaFeira = ['maca', 'banana', 'uva']
```

0	1	2
		

- Entretanto, o que não falamos é que, podemos ter índices negativos. Loucura, não?

```
1 lista1 = ['A', 'B', 'E', 'L', 'L', 'A']
2
3 print(lista1[1])
4 print(lista1[-5])
```

- No exemplo acima, nas linhas 3 e 4 será apresentado a String 'B'. O entendimento dos índices negativos é apresentado na imagem a seguir.

A	B	E	L	L	A
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

- Os elementos da lista podem ser acessados da direita para a esquerda usando índices negativos. O valor negativo começa de -1 a - tamanho da lista. Ou seja, indica que em Python podemos trabalhar de trás para frente.

06 Tamanho da Lista

- Em inglês, comprimento significa *length*. E vai ser com o método `len()` que vamos descobrir o tamanho de uma lista ou String.
- No exemplo a seguir, apresentamos como usar o método `len`.
 - Na linha 3 exibimos diretamente o tamanho
 - Nas linhas 5 e 6 também exibimos o tamanho, mas antes atribuímos o valor a uma variável

```
1 lista = ['A', 'B', 'E', 'L', 'L', 'A']
2
3 print(len(lista))
4
5 tamanhoLista = len(lista)
6 print(tamanhoLista)
```

- E agora, vamos mostrar o uso do método `len()` com Strings, que basicamente conta a quantidade de caracteres da String.

```
1 nome = 'Daniel'
2
3 print(len(nome))
4
5 tamanhoNome = len(nome)
6 print(tamanhoNome)
```

- Sim, mas quando vou usar esse método? No exemplo a seguir, usamos os métodos:

- `Sum()` que soma todos os elementos da lista
- `Len()` que retorna o tamanho (comprimento) da lista

```
1 notas = [10, 8, 9]
2
3 somaNotas = sum(notas)
4 media = somaNotas / len(notas)
5 print(media)
```

07 Imprimir Elementos da Lista

- Para apresentar os elementos de uma lista, chamamos comumente de iterar. Na sequência, apresentaremos duas formas de iterar uma lista.

```
1 lista = ['maçã', 'uva', 'banana']
2
3 ▼ for item in lista:
4     print(item)
```

```
1 lista = ['maçã', 'uva', 'banana']
2
3 ▼ for i in range(len(lista)):
4     print(lista[i])
```

- No exemplo à esquerda da página anterior, utilizamos o `for` sem o uso de índices, mas com uso de variável temporária (`item`). O funcionamento é o seguinte:

- Ao executar o `for` (da linha 3), a variável `item` recebe o valor que está na posição 0 da lista (isto é, 'maçã') e executa as linhas que possuem recuo (que estão atreladas ao `for`), isto é, a linha 4
- Ao terminar a execução das linhas recuadas (neste exemplo, apenas a linha 4), regressamos a linha 3 e a variável `item` recebe o valor que está na posição 1 da lista (isto é, 'uva') e executa as linhas que possuem recuo
- Ao terminar a execução das linhas recuadas, regressamos a linha 3 e a variável `item` recebe o valor que está na posição 2 da lista (isto é, 'banana') e executa as linhas que possuem recuo
- Como terminamos todos os elementos da lista, o `for` é encerrado e seguiremos a execução das linhas após o recuo (neste exemplo, linha 5 em diante)

- No exemplo à direita da página anterior, utilizamos índices. Notem que, o `for` é diferente do exemplo à esquerda, pois utilizamos o método `range()` associado ao método `len()`. A chamada `len(lista)` retorna o valor 3, de modo que, `range(3)` permite que o `for` execute de 0 até menor que 3 (isto é, 2). O funcionamento é o seguinte:

- Na primeira vez que o `for` é executado, a variável `i` recebe o valor 0 e na sequência ele exibe o trecho recuado (neste caso, linha 4) que é o `lista[i]`, que neste caso (que `i = 0`), exibiremos `lista[0]`, que 'maçã'. Quando terminar o trecho recuado, retomamos à linha 3
- Na segunda vez, exibimos o `lista[1]`
- Na terceira vez, exibimos o `lista[2]`
- Lembra que é até menor que 3 (no caso, 2)? Então neste caso, terminamos.

08 “Fatiar” uma lista

- Imagine que, você tenha uma lista com 100 elementos e precise apenas os dois primeiros elementos? Você precisa fatiar a lista. Fatiar em inglês significa *slice*.

```
1 listaFeira = ['maçã', 'uva', 'banana', 'pêra']
2
3 subLista = listaFeira[0:3]
4 print(subLista)
5
6 print(listaFeira[0:3])
```

- No exemplo acima, apresentamos duas maneiras de criar uma lista dos 2 primeiros elementos (`subLista`) da `listaFeira`.

- Neste exemplo, colocamos `listaFeira[0:3]`, que significa que a `subLista` é criada desde o índice 0 até menor que 3, ou seja, até 2
- Agora, a seguir, vamos apresentar outros exemplos que você pode se deparar.
 - Na linha 2, são exibidos os 4 primeiros elementos
 - Na linha 3, imprime o primeiro elemento (índice 0) e depois pula de 2 em 2
 - Na linha 4, invertemos a lista, ou seja, o último vira primeiro e o primeiro vira último (e assim sucessivamente)
 - Na linha 5, exibe a partir do elemento de índice 3 e imprime até o fim da lista

```
1 lista = [2, 4, 8, 10, 12, 14]
2 print(lista[:4])
3 print(lista[::2])
4 print(lista[::-1])
5 print(lista[3:])
```

09 Adicionar Elementos à Lista

- Para adicionar elementos à lista, temos 3 opções: `append()`, `insert()` e `extend()`.
- Vamos de `append()`? O método `append()` adiciona no fim da lista, como no exemplo a seguir, na qual na linha 2 adicionamos a String 'Nathaly' à lista.

```
1 lista = ['Daniel', 'Arthur']
2 lista.append('Nathaly')
3
4 ▼ for item in lista:
5     print(item)
```

- O método `insert()` possibilita a inserção de um novo elemento em uma dada posição da lista.

```
1 lista = ['Daniel', 'Arthur']
2 lista.insert(2, 'Nathaly')
3 lista.insert(0, 'Joselita')
```

- No exemplo acima, na linha 2, inserimos 'Nathaly' na posição 2, que não estava ocupada. Veja a ilustração a seguir.

- Antes da execução da linha 2:

Valores →	'Daniel'	'Arthur'
Índices →	0	1

- Depois da execução da linha 2:

Valores →	'Daniel'	'Arthur'	'Nathaly'
Índices →	0	1	2

```

1 lista = ['Daniel', 'Arthur']
2 lista.insert(2, 'Nathaly')
3 lista.insert(0, 'Joselita')

```

- Agora, ao executar a linha 3, em que vamos tentar adicionar 'Joselita' (minha sogra) temos uma curiosidade: esta posição está ocupada por 'Daniel', o que acontecerá? Veja a ilustração a seguir.

- Antes da execução da linha 3:

Valores →	'Daniel'	'Arthur'	'Nathaly'
Índices →	0	1	2

- Depois da execução da linha 3:

Valores →	'Joselita'	'Daniel'	'Arthur'	'Nathaly'
Índices →	0	1	2	3

- Note que, inserimos 'Joselita' exatamente na posição desejada (isto é, 0) e os itens à direita são deslocados uma posição para direita, ou seja, 'Daniel' que estava antes na posição 0, passa a assumir a posição 1 e assim sucessivamente.
- Por fim, vamos aprender sobre o método `extend()`, que adiciona uma lista de elementos ao fim da lista. Veja o exemplo a seguir.

```

1 lista = ['Daniel', 'Arthur']
2 lista.extend(['Nathaly'])

```

- No exemplo acima, adicionamos na linha 2, uma lista com um elemento (['Nathaly']) à lista anterior (com 2 elementos), de modo que, esta ficou agora com 3 elementos. Sim, mas se tiver mais de um elemento? Mesma coisa, veja no exemplo a seguir.

```

1 lista = ['Daniel', 'Arthur']
2 lista.extend(['Nathaly', 'Elisa'])

```

10 Tuplas (Tuples)

- As tuplas, seguem a mesma filosofia das listas, porém são imutáveis de modo que não podemos modificar os itens.
- Ou seja, quando uma tupla é criada, NÃO é possível adicionar, remover ou alterar os elementos.
- Vamos entender como funciona a sua criação, que pode ser feitas de três maneiras:
 - Na linha 1, criamos a tupla com uso de parênteses
 - Na linha 2, sem uso de parênteses, que são ocacionais
 - Na linha 3, usando o construtor `tuple()`.
 - Todas as 3 tuplas tem Strings, mas poderiam ter tipos variados

```

1 tupla1 = ('Daniel', 'Nathaly', 'Arthur')
2 tupla2 = 'Daniel', 'Nathaly', 'Arthur'
3 tupla3 = tuple(('Daniel', 'Nathaly', 'Arthur'))

```



Se você quiser criar uma tupla de um único elemento, você precisa colocar uma vírgula no fim. Por que? Veja o exemplo a seguir.

```

1 tupla = 'Daniel',
2 print(type(tupla))
3
4 nome = 'Daniel'
5 print(type(nome))

```

No exemplo acima, o que diferencia as linhas 1 e 4 é basicamente a vírgula, correto? Mas é isso que faz a variável da linha 1 ser uma tupla e a variável da linha 4 ser uma String (str).

- Ou seja, é bem parecido com listas. Inclusive, as operações a seguir são IGUAIS em listas e tuplas:
 - Tamanho com `len()`, Imprimir elementos, Acessar elementos por índice, Índices negativos e "Fatiar" uma lista. Então não vou repetir 😊
- Tuplas são usadas para adicionar diferentes tipos de dados com quantidade de elementos previamente definidos

11 Tuplas (Tuples) são imutáveis?

- Imutáveis significa que você não pode adicionar, remover ou alterar os valores. No exemplo a seguir, a execução da linha 1 funciona perfeitamente, pois uma lista podemos modificar os elementos.
 - Em contrapartida, a linha 2 não é possível, pois não é possível alterar o valor de um elemento de uma tupla.

```

1 tupla1 = ('Daniel', 'Nathaly', 'Arthur')
2 tupla1[0] = 'Daniel Abella'
3
4 lista1 = ['Daniel', 'Nathaly', 'Arthur']
5 lista1[0] = 'Daniel Abella'

```



12 Exclusão de Tuplas e Listas

- Podemos basicamente excluir uma *tupla*, ou seja, apagar ela do programa, de modo que, após a sua exclusão, a variável não pode ser utilizada. Para isto, usamos a operação `del`, cujo exemplo é apresentado a seguir.

```

1 numbers=4,5,6
2 del numbers
3 print(numbers)
4

```

```

Traceback (most recent call last):
  File "main.py", line 3, in <module>
    print(numbers)
NameError: name 'numbers' is not defined

```

- Note que, na linha 2 realizamos a exclusão (em "americanizado" chamamos de deleção). E, na linha 3 em diante não podemos mais usar a variável `numbers`, que dá erro.

13 Exclusão de um Elemento da Lista

- Como vimos anteriormente, podemos excluir uma lista ou tupla INTEIRA. Entretanto, podemos excluir um elemento da lista ou um intervalo de elementos de uma lista. Este tipo de exclusão não funciona com tuplas, como veremos a seguir.

```
1 frutas = ['banana', 'uva', 'maçã']
2 del frutas[0]
3 print(frutas)
4
5 nomes = 'Daniel', 'Arthur', 'Nathaly'
6 del nomes[0]
7 print(nomes)
```



- No exemplo acima podemos verificar a exclusão de um elemento. Na linha 2, eliminamos o primeiro elemento (posição 0) de uma lista. Entretanto, a operação da linha 6 não é possível, vai dar erro, pois **tuplas são imutáveis**.
- Complementarmente, no exemplo abaixo, usamos a operação `del frutas[:2]`, na qual o dois pontos seguido do número 2 nos informa que, exclua da primeira posição, até a posição 2 (ou seja, não inclui a posição 2). Desta maneira, o resultado da linha 3 será `['maçã', 'melancia', 'kiwi']`

```
1 frutas = ['banana', 'uva', 'maçã', 'melancia', 'kiwi']
2 del frutas[:2] #exclui ATÉ a posição 2
3 print(frutas)
```

14 Descobrindo Índice de um Elemento

- Em qual posição de uma lista ou tupla o meu elemento está? Usaremos o método `index`, cujo exemplo está a seguir. Note que, em ambos os casos, nas linhas 2 e 6, buscamos identificar em qual índice está o elemento 20.

```
1 lista1 = [10, 20, 30, 40, 50]
2 posicao = lista1.index(20)
3 print(posicao)
4
5 tupla1 = (10, 20, 30, 40, 50)
6 posicao = tupla1.index(20)
7 print(posicao)
```

1
1
Saída

- Acima é realiza uma busca completa, concorda? Eu posso delimitar o intervalo de busca usando o método `index`, mas da seguinte forma:

- `index(item, indiceInicioBusca, indiceFimBusca)`
- No exemplo abaixo, iniciamos uma busca pelo elemento 60, iniciando pelo índice 2 até o índice 4 (isto é, até 5).

```
1 tuple1 = (20, 30, 40, 50, 60)
2 posicao = tuple1.index(60, 2, 5)
3 print(posicao)
```



Se você fizer uma busca usando `index` por algum elemento que não existe, uma exceção será lançada. Abaixo vemos um exemplo do problema e a respectiva *exception* que é lançada.

```
1 tuple1 = (20, 30, 40, 50, 60)
2 posicao = tuple1.index(61, 2, 5)
3 print(posicao)
```

ValueError: tuple.index(x): x not in tuple

Não sabe o que é *exception*, né? Mais a frente falamos, mas caso esteja com pressa, *Google It!*

- Ou seja, é bem parecido com listas. Inclusive, as operações a seguir são IGUAIS em listas e tuplas:
 - Tamanho com `len()`, Imprimir elementos, Acessar elementos por índice, Índices negativos e “Fatiar” uma lista. Então não vou repetir 😊
- Tuplas são usadas para adicionar diferentes tipos de dados com quantidade de elementos previamente definidos

15 Como saber se um element está na lista ou tupla?

- Usamos basicamente o operador `in`, apresentado no exemplo a seguir, que pode ser usado em tuplas ou listas. Note que, podemos perguntar diretamente se um elemento está na lista/tuple, como nas linhas 3 e 4, bem como podemos colocar a condição em um `if`, a exemplo das linhas 6 e 11.

```
1 tuple1 = (10, 20, 30, 40, 50)
2
3 print(20 in tuple1)
4 print(21 in tuple1)
5
6 ▼ if 20 in tuple1:
7     print('esta na tupla')
8
9 variavel = 20
10
11 ▼ if variavel in tuple1:
12     print('esta na tupla')
```

- Como funciona com listas? Só mudar na linha 1 de parênteses (característica de uma dupla) para colchetes (característica de uma lista). Vai funcionar sem, mas eu ainda faria uma pequena modificaçãozinha: mudaria o nome da variável de `tuple1` para `list1` para ficar redondo! Apesar que vai exigir mudança no código inteiro.

```
1 list1 = [10, 20, 30, 40, 50]
```

16 Gambiarra: Adicionar Elementos em uma Tupla

- Como dissemos anteriormente, não podemos mudar uma tupla, pois ela é imutável. Mas, adoramos uma gambiarra, digo, workaround (sua tradução em inglês), que deixa muito + elegante.



- No workaround a seguir, para adicionar em uma tupla, a convertemos em uma lista, fazemos nossas mudanças (adicionar, remover, alterar um elemento) e por fim, transformamos de volta a lista em uma tupla. Liiiindo, não? PN.

```
1 tuple1 = (0, 1, 2, 3, 4, 5)
2 listaGambiarra = list(tuple1)
3 listaGambiarra.append(6)
4 tuple1 = tuple(listaGambiarra)
5 print(tuple1)
```

17 Métodos Filé de Listas

- Vamos aprender métodos bem úteis que podem ser aplicados a listas, sendo eles: `append()`, `insert()`, `remove()`, `pop()`, `clear()`, `sort()` e `reverse()`. São métodos *built-in*.
- A seguir temos um exemplo completo. Experimente por um `print` entre cada uma das linhas para entender melhor.
 - Na linha 2, adicionar um elemento ao fim da lista
 - Na linha 3, inserimos o valor 10 na posição 1
 - Na linha 4, removemos o elemento 10
 - Na linha 5, removemos o elemento da posição 2
 - Na linha 7, ordenamos os elementos de ordem crescente (do menor ao maior)
 - Na linha 9, usamos o mesmo método, porém ordenamos de maneira decrescente (do maior ao menor)

```
1 lista1 = [20, 30, 40]
2 lista1.append(50) #insere no final
3 lista1.insert(1,10) #insere 10 na posição 1
4 lista1.remove(10) #remove o elemento 10
5 lista1.pop(2) #remove o elemento da posição 2
6 lista1.insert(0,70) #insere 70 na posição 0
7 lista1.sort() #ordena de modo crescente
8 #vai ficar [20,30,50,70]
9 lista1.sort(reverse=True) #ordena de modo decrescente
10 #vai ficar [70,50,30,20]
11 print(len(lista1))
```

18 Métodos Filé de Listas e Tuplas

- Existem mais métodos bacanas que podem ser usados em listas e tuplas, sendo eles `min`, `max` e `sum`.

```
1 lista1 = [20, 30, 40]
2
3 menorElemento = min(lista1)
4 maiorElemento = max(lista1)
5 somaDosElementos = sum(lista1)
6 mediaDosElementos = sum(lista1) / len(lista1)
7
8 tupla1 = (20, 30, 40)
9
10 menorElemento = min(tupla1)
11 maiorElemento = max(tupla1)
12 somaDosElementos = sum(tupla1)
13 mediaDosElementos = sum(tupla1) / len(tupla1)
```

- No exemplo acima, usamos o método `min` para obter o menor elemento de uma lista/tupla, `max` para obter o maior elemento de uma lista/tupla. Por fim, usamos `sum` para obter a soma de todos os elementos de uma lista.

19 Básico de Conjuntos

- Conjuntos (em Inglês, *Set*) é uma estrutura de dados que não é ordenada e não possui duplicadas, ou seja, se eu adicionei o elemento 'Abella', não pode ter outro elemento com o mesmo valor. Além disso, é heterogêneo, ou seja, a
 - Possivelmente no ensino médio tenhamos discutido conjuntos, mas hoje vamos entender na prática o funcionamento.
- Para criar um conjuntos, temos duas maneiras. A primeira deles é usando o método `set()`.

```
1 livrosAbella = set(('Gestão A3', 'Scrum Arretado'))
2 print(type(livrosAbella))
```

- A segunda é usar chaves, na qual os valores são separados por vírgula.

```
1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}
2 print(type(livrosAbella))
```

- No exemplo a seguir, verificamos que o conjunto é bem parecido com as outras estruturas que vimos. Atenção ao método `add` que adiciona um elemento e ao `update` que adiciona mais de um elemento ao mesmo tempo.

```
1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}
2
3 ▼ for livro in livrosAbella:
4     print(livro)
5
6 ▼ if 'Gestão A3' in livrosAbella:
7     print('existe')
8
9 ▼ if not 'Use a Cabeça PMP' in livrosAbella:
10     print('não está na lista')
11
12 print('Gestão A3' in livrosAbella)
13
14 livrosAbella.add('Novo Livro')
15 livrosAbella.update(['Novo Livro 2', 'Novo Livro 3'])
16
17 print(len(livrosAbella))
```

20 Remoção em Conjuntos

- Existem 3 métodos principais usados para remoção.
- O primeiro deles é o **remove**, que remove o elemento informado como parâmetro e caso este elemento não exista no conjunto, lança uma exceção.

- No exemplo a seguir, a linha 4 lança uma exception, pois o 'Livro 3' foi excluído na linha 3

```
1 livrosAbella = {'Gestão A3',  
2               'Scrum Arretado', 'Livro3'}  
3 livrosAbella.remove('Livro3')  
4 livrosAbella.remove('Livro3')
```

- O segundo deles, o **discard**, possui a mesma função do remove, porém não lança uma exceção.

- No exemplo a seguir, a linha 4 não lança nenhuma exception, diferente do exemplo anterior.

```
1 livrosAbella = {'Gestão A3',  
2               'Scrum Arretado', 'Livro3'}  
3 livrosAbella.discard('Livro3')  
4 livrosAbella.discard('Livro3')
```

- O terceiro e último método é o **pop**, que exclui um elemento de maneira aleatória.

- Ou seja, se você não sabe quem vai ser excluído, você não informa nada no parâmetro.

```
1 livrosAbella = {'Gestão A3',  
2               'Scrum Arretado', 'Livro3'}  
3 livrosAbella.pop()
```

- E, se eu quiser limpar um conjunto? Ou seja, apagar todos os elementos existentes “de uma lapada só”. Usamos o **clear**.

- No exemplo a seguir, a linha 3 imprimirá 0

```
1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}  
2 livrosAbella.clear()  
3 print(len(livrosAbella))
```

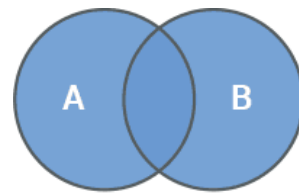
- E, se eu quiser excluir um conjunto, ou seja, torna-lo inutilizável, basta usar o comando **del**, como no exemplo a seguir.

- A linha 3 causará um erro, pois esta foi inutilizada na linha 2

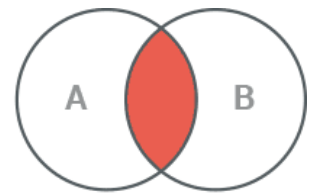
```
1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}  
2 del livrosAbella  
3 print(len(livrosAbella))
```

21 Operações em Conjuntos

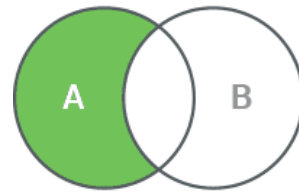
- Vamos voltar ao ensino médio e entender as operações em conjuntos, sendo elas: união (**|**), interseção (**&**), diferença (**-**) e diferença simétrica (**^**).



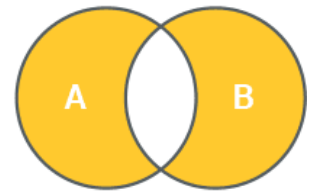
Union



Intersection



Difference



Symmetric Difference

Fonte: learnbyexample.org/python-set/

- Acima, relacionamos as principais operações em conjuntos.

União (Union) de Conjuntos

- A primeira operação que vamos aprender com conjuntos é a União. Relembrando com nostalgia o ensino médio, a união retoma todos os itens dos 2 conjuntos relacionados. Como conjunto, se um elemento estiver presente nos 2 conjuntos relacionados, apenas 1 é incluído no conjunto resultante.

- Existem 2 formas de fazer união com conjuntos. A primeira delas é usando o método **union()** e a segunda é com o operador **|**. O exemplo de uso é apresentado a seguir, cujo resultado é {'banana', 'maçã', 'uva'}

```
1 conjuntoA = {'banana', 'maçã'}  
2 conjuntoB = {'uva', 'banana'}  
3  
4 print(conjuntoA | conjuntoB)  
5 print(conjuntoA.union(conjuntoB))
```

Interseção (Intersection) de Conjuntos

- A interseção retoma os elementos contidos em ambos os conjuntos. Para isto, usa o método **intersection()** ou o operador **&**, como podemos ver no exemplo a seguir, que retorna {'banana'}.

```
1 conjuntoA = {'banana', 'maçã'}  
2 conjuntoB = {'uva', 'banana'}  
3  
4 print(conjuntoA & conjuntoB)  
5 print(conjuntoA.intersection(conjuntoB))
```

Diferença (Difference) de Conjuntos

- A diferença retornará os itens que estão apenas no primeiro conjunto (conjuntoA). Para isto, usamos o operador **-** ou o método **difference()**. No exemplo a seguir, a saída é {'maçã'}.

```

1 conjuntoA = {'banana', 'maçã'}
2 conjuntoB = {'uva', 'banana'}
3
4 print(conjuntoA - conjuntoB)
5 print(conjuntoA.difference(conjuntoB))

```

Diferença Simétrica (*Symmetric Difference*) de Conjuntos

- A diferença simétrica retornará os itens que estão em ambos conjuntos, mas desconsiderando os que estiverem nos 2 conjuntos. Para isto, usamos o operador `^` ou o método `symmetric_difference()`. No exemplo a seguir, a saída é `{'maçã', 'uva'}`.

```

1 conjuntoA = {'banana', 'maçã'}
2 conjuntoB = {'uva', 'banana'}
3
4 print(conjuntoA ^ conjuntoB)
5 print(conjuntoA.symmetric_difference(conjuntoB))

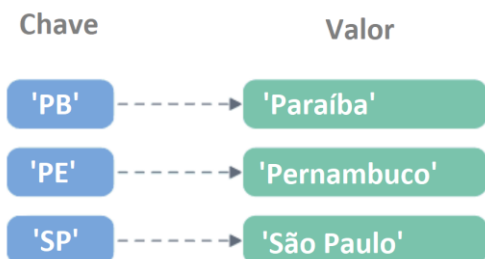
```

Resumo das Operações com Conjuntos

Operação	Método	Operador
União	<code>union()</code>	<code> </code>
Inserseção	<code>Intersection()</code>	<code>&</code>
Diferença	<code>difference()</code>	<code>-</code>
Diferença Simétrica	<code>symmetric_difference()</code>	<code>^</code>

22 Dicionários

- Também conhecido por *Hash Maps*, Dicionários é basicamente um mapeamento entre conjunto de índices (também conhecidos por chaves) a um conjunto de valores, tal como é apresentado na imagem a seguir. Note que, para cada chave (por exemplo, 'PB') é associada um valor (por exemplo, 'Paraíba').



- A associação de uma chave a um valor é conhecido por **chave:valor** ou **item**. O dicionário apresentado na imagem anterior, caso fosse em código Python, seria o seguinte.

```

1 estados = {
2     'PB': 'Paraíba',
3     'PE': 'Pernambuco',
4     'SP': 'São Paulo',
5 }

```

- Alternativamente podemos criar dicionários com uso do método `dict()`.

```

1 estados = dict({
2     'PB': 'Paraíba',
3     'PE': 'Pernambuco',
4     'SP': 'São Paulo'})

```

- Uma coisa muito importante é que as chaves devem ser únicas, de modo que, se você colocar 2 chaves iguais, a segunda sobrescreverá a primeira, tal como no exemplo a seguir.

```

1 estados = dict({
2     'PB': 'Paraíba',
3     'PE': 'Pernambuco',
4     'PB': 'Parahyba'})
5 print(estados)

```

```
{'PB': 'Parahyba', 'PE': 'Pernambuco'}
```

Adição de Elementos em Dicionários

- No exemplo a seguir, adicionamos o elemento 'Paraíba' com chave 'PB'. Além disso, adicionamos à chave 'SP' uma tupla ('São Paulo', 'Sampa').

```

1 estados = {} #Dicionário vazio
2 estados['PB'] = 'Paraíba' #Adiciona chave PB e valor Paraíba
3 estados['SP'] = 'São Paulo', 'Sampa' #Adiciona chave SP e valor que é uma tupla com {'PB': 'Parahyba', 'SP': ('São Paulo', 'Sampa')}

```

Listagem em um Dicionário

- No exemplo a seguir, apresentamos a chave e o valor associado à chave.

```

1 estados = { 'pb': 'paraiba', 'sp': 'sampa' }
2
3 for i in estados:
4     print(i)
5     print(estados[i])

```

Obter o Elemento de uma Chave

- Se eu quiser saber qual o valor da chave 'PB'? Mostramos a seguir duas formas. A primeira delas usando o colchete com o valor da chave (linha 4) e a segunda usando o método `get` (linha 5).

```

1 estados = {}
2 estados['PB'] = 'Parahyba'
3
4 print(estados['PB']) #Obtém valor da chave 'PB'
5 print(estados.get('PB')) #Obtém valor da chave 'PB'

```

Exclusão de Elementos em Dicionários

- Existem 3 métodos associados à exclusão de elementos, sendo eles `popitem`, `pop` e `clear`.

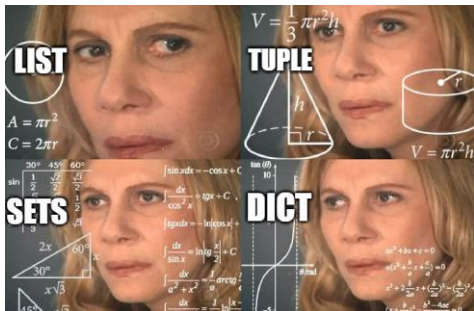
Exclusão de Elementos em Dicionários

- O método `popitem` remove um item (chave/valor) de maneira aleatória e retorna como tupla
- O método `pop`, recebe uma chave, e exclui e retorna o item (chave/valor) associado à chave informada.
- O método `clear()` faz um “limpa”, ou seja, apaga todos os itens

```
1 estados = {}
2 estados['PB'] = 'Paraíba'
3 estados['PE'] = 'Pernambuco'
4 estados['SP'] = 'São Paulo', 'Sampa'
5
6 del estados['PB'] #Apaga o item de chave 'PB'
7
8 elementoEliminado = estados.popitem()
9 print(elementoEliminado)
10
11 elementoEliminado = estados.pop('PE')
12 print(elementoEliminado)
13
14 estados.clear() #limpa o dicionário
15 print(estados)
```

23Resumo

- Ficou assim, né? Vamos resumir.



- A seguir apresentamos um resumo de cada estrutura de dados.

Estrutura	Ordenado?	Construtor	Exemplo
list	Sim	[] ou list()	['A','E']
tuple	Sim	() ou tuple()	('A','E')
set	Não	{ } ou set()	{ 'A','E' }
dictionary	Sim	{ } ou dict()	{ 'PB':20, 'PE':22 }

Resumo de Listas

```
listaFeira = ['maca', 'banana', 'uva']
               ↑       ↑       ↑
listaFeira[0] listaFeira[1] listaFeira[2]
```

Método	Descrição	Exemplo
<code>append(elemento)</code>	Adiciona ao fim da lista	<code>lista.append(12)</code>
<code>clear()</code>	Remove todos elementos	<code>lista.clear()</code>
<code>count(x)</code>	Quantas 'x' tem na lista?	<code>lista.count('x')</code>
<code>index(x)</code>	Posição de X na lista?	<code>lista.index('x')</code>
<code>insert(i,elemento)</code>	Add 'X' na posição i	<code>lista.insert(0,'X')</code>
<code>pop()</code>	Remove o último elemento	<code>lista.pop()</code>
<code>remove(x)</code>	Remove o elemento 'x'	<code>lista.remove('x')</code>
<code>reverse()</code>	Inverte os elementos	<code>lista.reverse()</code>
<code>sort()</code>	Ordena de modo crescente	<code>lista.sort()</code>
<code>sort()</code>	Ordena de modo decrescente	<code>lista.sort(reverse=True)</code>

Resumo de Tuplas

```
1 tupla1 = ('Daniel', 'Nathaly', 'Arthur')
2 tupla2 = 'Daniel', 'Nathaly', 'Arthur'
3 tupla3 = tuple(('Daniel', 'Nathaly', 'Arthur'))
```

Método	Descrição	Exemplo
<code>del tupla</code>	Exclui a tupla	<code>del tupla</code>
<code>del tupla[0]</code>	Remove um elemento da tupla	<code>del tupla[0]</code>
<code>count(x)</code>	Quantas 'x' tem na tupla?	<code>tupla.count('x')</code>
<code>index(x)</code>	Posição de X na tupla?	<code>tupla.index('x')</code>
<code>in</code>	Se um elemento tá na tupla	<code>If 20 in tupla</code>
<code>max</code>	Maior valor da tupla	<code>max(tupla)</code>
<code>min</code>	Menor valor da tupla	<code>min(tupla)</code>
<code>len</code>	Tamanho da tupla	<code>len(tupla)</code>

Resumo de Conjuntos

```
1 livrosAbella = set(('Gestão A3', 'Scrum Arretado'))
2 print(type(livrosAbella))
1 livrosAbella = {'Gestão A3', 'Scrum Arretado'}
2 print(type(livrosAbella))
```

Método	Descrição	Exemplo
<code>add</code>	Adiciona um elemento	<code>livros.add('A')</code>
<code>update</code>	Adiciona mais de 1 elemento	<code>livros.update('A','B')</code>
<code>count(x)</code>	Quantas 'x' tem no set?	<code>livros.count('x')</code>
<code>index(x)</code>	Posição de X no set?	<code>livros.index('x')</code>
<code>in</code>	Se um elemento tá no set	<code>If 20 in livros</code>
<code>max</code>	Maior valor do set	<code>max(livros)</code>
<code>min</code>	Menor valor do set	<code>min(livros)</code>
<code>len</code>	Tamanho do set	<code>len(livros)</code>
<code>remove</code>	Remover do conjunto (set)	<code>livros.remove('A')</code>
<code>discard</code>	Igual remove, mas não lança exceção	<code>livros.discard('A')</code>
<code>pop</code>	Exclui um elemento de modo aleatório	<code>livros.pop()</code>

Resumo das Operações com Conjuntos

Operação	Método	Operador
União	<code>union()</code>	
Inserseção	<code>Intersection()</code>	&
Diferença	<code>difference()</code>	-
Diferença Simétrica	<code>symmetric_difference()</code>	^

Resumo de Dicionários

Método	Descrição	Exemplo
<code>clear()</code>	Limpa o dicionário	<code>dict.clear()</code>
<code>get()</code>	Obtém o valor de uma chave	<code>dict.get(chave))</code>
<code>items()</code>	Retorna a listas contendo uma tupla para cada chave/valor	<code>print(dict.items())</code> #Resposta: (('Física', 88), ('Matemática', 65))
<code>keys()</code>	Retorna a lista de chaves	<code>dict.keys()</code> #Resultado ['name', 'salary', 'age']
<code>pop()</code>	Remove um elemento de uma chave	<code>removido = dict.pop('Chave')</code>
<code>popitem()</code>	Remove o último item (chave/valor)	<code>dict.popitem()</code>
<code>setdefault()</code>	Retorna o valor de uma chave.	<code>valor = dict.setdefault(chave)</code>
<code>update()</code>	Atualiza um item	<code>dict.update({'Chave':48})</code>
<code>values()</code>	Retorna a lista de todos os valores	<code>marks.values()</code> # Resultado: dict_values([67, 87])