



**Instituto de
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



MC102 – Aula 05

Listas e Tuplas

Algoritmos e Programação de Computadores

Zanoni Dias

2023

Instituto de Computação

Listas

Tuplas

Exercícios

Listas

- Podemos utilizar a estrutura de lista em Python para armazenar múltiplos dados.
- Listas podem ser criadas de forma implícita, listando os elementos entre colchetes.

```
1 frutas = ["Abacaxi", "Banana", "Caqui", "Damasco",  
2         "Embaúba", "Figo", "Graviola"]  
3 numeros = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]  
4 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]  
5 dados = ["Carlos", 19, True, "Pedro", "Ana", 1.78, 2001]
```

- Podemos também declarar uma lista de maneira explícita utilizando a função `list`.

```
1 a = list(range(10))  
2 # a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
1 a = list()  
2 # a = []
```

```
1 empresas = list(["Toyota", "Volkswagen", "Ford"])  
2 # empresas = ['Toyota', 'Volkswagen', 'Ford']
```

```
1 unicamp = list("Unicamp")  
2 # unicamp = ['U', 'n', 'i', 'c', 'a', 'm', 'p']  
3 # Strings são listas de caracteres
```

Selecionando um Elemento

- Podemos acessar o i -ésimo elemento da seguinte forma:

```
1 lista[i - 1]
```

- Essa operação retorna como resposta uma cópia do i -ésimo elemento da lista.
- O primeiro elemento de uma lista ocupa a posição 0.

Selecionando um Elemento

- Selecionando o primeiro elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[0])
3 # A
```

- Selecionando o segundo elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[1])
3 # B
```

- Selecionando o quinto elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[4])
3 # E
```

Selecionando um Elemento

- Podemos também acessar os elementos de uma lista, de trás para frente, da seguinte forma:

```
1 lista[-i]
```

- Como resposta, obtemos uma cópia do i -ésimo elemento da lista, de trás para frente.
- O último elemento de uma lista ocupa a posição -1.

Selecionando um Elemento

- Selecionando o último elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[-1])
3 # H
```

- Selecionando o penúltimo elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[-2])
3 # G
```

- Selecionando o antepenúltimo elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[-3])
3 # F
```

Selecionando um Elemento

- Caso seja informada uma posição inválida, será gerado um erro:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[8])
3 # IndexError: list index out of range
```

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[-9])
3 # IndexError: list index out of range
```

Determinando o Tamanho de uma Lista

- A função `len` recebe como parâmetro uma lista e retorna o seu tamanho (número de elementos).
- Exemplos:

```
1 frutas = ["Abacaxi", "Banana", "Caqui", "Damasco",  
2   "Embaúba", "Figo", "Graviola"]  
3 numeros = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]  
4 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]  
5 print(len(frutas))  
6 # 7  
7 print(len(numeros))  
8 # 11  
9 print(len(letras))  
10 # 8
```

Selecionando Intervalos

- Podemos selecionar um trecho de uma lista:

```
1 lista[start:stop:step]
```

- O trecho inicia na posição `start` (inclusive) e vai até a posição `stop` (exclusive), selecionando de `step` em `step` os elementos da lista.
- Esta operação retorna uma nova lista, à qual normalmente nos referimos como uma sublista.
- Caso os parâmetros `start`, `stop` ou `step` não sejam especificados, Python automaticamente assume que seus valores são a posição do primeiro elemento (0), o tamanho da lista (`len(lista)`) e um (1), respectivamente.

Selecionando Intervalos

- Selecionando do segundo até o quarto elemento de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]  
2 print(letras[1:4])  
3 # ["B", "C", "D"]
```

- Selecionando os três primeiros elementos de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]  
2 print(letras[:3])  
3 # ["A", "B", "C"]
```

- Selecionando os quatro últimos elementos de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]  
2 print(letras[-4:])  
3 # ["E", "F", "G", "H"]
```

Selecionando Intervalos

- Selecionando os elementos das posições pares de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[::2])
3 # ["A", "C", "E", "G"]
```

- Selecionando os elementos das posições ímpares de uma lista:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[1::2])
3 # ["B", "D", "F", "H"]
```

- Obtendo os elementos de uma lista, em ordem inversa:

```
1 letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
2 print(letras[::-1])
3 # ["H", "G", "F", "E", "D", "C", "B", "A"]
```

Alterando um Elemento

- Podemos alterar o i -ésimo elemento de uma lista da seguinte forma:

```
1 lista[i - 1] = valor
```

- Exemplo:

```
1 empresas = ["Apple", "Samsung", "LG", "Facebook"]
2 empresas[2] = "Google"
3 print(empresas)
4 # ['Apple', 'Samsung', 'Google', 'Facebook']
```

Alterando um Trecho da Lista

- Podemos alterar um trecho de uma lista associando uma nova lista para esse trecho, inclusive uma lista vazia.

```
1 lista[start:stop] = [valor_1, ..., valor_n]
```

- Exemplo:

```
1 lista = [0, 1, 2, 3, 4, 5]
2 lista[2:4] = ["A", "B"]
3 print(lista)
4 # [0, 1, 'A', 'B', 4, 5]
5 lista[2:4] = [8, 8, 8]
6 print(lista)
7 # [0, 1, 8, 8, 8, 4, 5]
8 lista[4:6] = []
9 print(lista)
10 # [0, 1, 8, 8, 5]
```


Verificando a Inclusão de um Elemento

- Podemos verificar se um elemento está ou não em uma lista utilizando o operador de teste de inclusão `in`.

```
1 elemento in lista
```

- Esse operador retorna `True` ou `False` caso o elemento esteja ou não na lista, respectivamente.
- Exemplo:

```
1 top5 = ["Black Mirror", "Breaking Bad", "Friends",  
2       "Game of Thrones", "The Big Bang Theory"]  
3 print("House MD" in top5)  
4 # False  
5 print("Game of Thrones" in top5)  
6 # True  
7 print("friends" in top5)  
8 # False
```

Inserindo Elementos

- Podemos inserir novos elementos em uma lista utilizando o método `append`.
- O método `append` recebe como parâmetro um objeto que será inserido no final da lista.
- Exemplo:

```
1 animais = []
2 animais.append("Gato")
3 print(animais)
4 # ['Gato']
5 animais.append("Cachorro")
6 print(animais)
7 # ['Gato', 'Cachorro']
8 animais.append("Coelho")
9 print(animais)
10 # ['Gato', 'Cachorro', 'Coelho']
```

Inserindo Elementos

- Para inserir um novo elemento em uma posição específica de uma lista, utilizamos o método `insert`.
- O método `insert` recebe como parâmetros uma posição e um objeto que será inserido na posição especificada.
- Cada elemento da posição especificada até o fim da lista é realocado para a posição seguinte. Dessa forma, nenhum elemento é removido.
- Exemplo:

```
1 frutas = ["Abacaxi", "Banana", "Damasco"]
2 frutas.insert(2, "Caqui")
3 print(frutas)
4 # ['Abacaxi', 'Banana', 'Caqui', 'Damasco']
5 frutas.insert(len(frutas), "Embaúba")
6 print(frutas)
7 # ['Abacaxi', 'Banana', 'Caqui', 'Damasco', 'Embaúba']
```

Inserindo Elementos

- Para inserir um novo elemento em uma posição específica de uma lista, utilizamos o método `insert`.
- O método `insert` recebe como parâmetros uma posição e um objeto que será inserido na posição especificada.
- Cada elemento da posição especificada até o fim da lista é realocado para a posição seguinte. Dessa forma, nenhum elemento é removido.
- Exemplo:

```
1 frutas = ["Abacaxi", "Banana", "Damasco"]
2 frutas.insert(2, "Caqui")
3 print(frutas)
4 # ['Abacaxi', 'Banana', 'Caqui', 'Damasco']
5 frutas.insert(1000, "Embaúba")
6 print(frutas)
7 # ['Abacaxi', 'Banana', 'Caqui', 'Damasco', 'Embaúba']
```

Inserindo Elementos

- Para inserir um novo elemento em uma posição específica de uma lista, utilizamos o método `insert`.
- O método `insert` recebe como parâmetros uma posição e um objeto que será inserido na posição especificada.
- Cada elemento da posição especificada até o fim da lista é realocado para a posição seguinte. Dessa forma, nenhum elemento é removido.
- Exemplo:

```
1 frutas = ["Abacaxi", "Banana", "Damasco"]
2 frutas.insert(2, "Caqui")
3 print(frutas)
4 # ['Abacaxi', 'Banana', 'Caqui', 'Damasco']
5 frutas.append("Embaúba")
6 print(frutas)
7 # ['Abacaxi', 'Banana', 'Caqui', 'Damasco', 'Embaúba']
```

Exercício

- Escreva um programa que recebe como entrada um número inteiro positivo n . Em seguida, seu programa deve ler n números inteiros e adicioná-los em uma lista. Por fim, seu programa receberá um número inteiro x e deve verificar se x pertence ou não à lista.

```
1 n = int(input("Quantos números serão lidos? "))
2 lista = []
3 for i in range(n):
4     lista.append(int(input()))
5 x = int(input("Qual o número a procurar? "))
6 if x in lista:
7     print(x, "pertence à lista")
8 else:
9     print(x, "não pertence à lista")
```

Exercício

- Escreva um programa que leia números positivos e os armazene numa lista (até que um número não positivo seja fornecido). Por fim, seu programa receberá um número inteiro x e deve verificar se x pertence ou não à lista.

```
1 print("Entre com números positivos:")
2 lista = []
3 while True:
4     p = int(input())
5     if p <= 0:
6         break
7     lista.append(p)
8 x = int(input("Qual o número a procurar? "))
9 if x in lista:
10    print(x, "pertence à lista")
11 else:
12    print(x, "não pertence à lista")
```

Obtendo a Posição de um Elemento

- O método `index` é utilizado para obter a posição de um elemento em uma lista.
- Como parâmetro, o método `index` recebe um elemento a ser buscado na lista.
- A posição da primeira ocorrência do elemento especificado como parâmetro é retornada como resposta.
- Caso o elemento não esteja na lista, um erro será gerado.

Obtendo a Posição de um Elemento

- Exemplo:

```
1 cinema = ["Sony Pictures", "Walt Disney",  
2   "Universal Pictures", "Warner"]  
3 print(cinema.index("Warner"))  
4 # 3  
5 print(cinema.index("Disney"))  
6 # ValueError: 'Disney' is not in list
```

- Podemos antes de usar o método `index` verificar se o elemento está na lista, prevenindo assim o erro.

```
1 cinema = ["Sony Pictures", "Walt Disney",  
2   "Universal Pictures", "Warner"]  
3 if "Disney" in cinema:  
4     print(cinema.index("Disney"))  
5 else:  
6     print("Disney não está na lista.")  
7 # Disney não está na lista.
```

Removendo Elementos

- Podemos remover um elemento de uma lista utilizando o método `remove`.
- O método `remove` recebe como parâmetro o elemento a ser removido da lista.
- A primeira ocorrência do elemento especificado como parâmetro é removida da lista.
- Caso o elemento não esteja na lista, um erro será gerado.

Removendo Elementos

- Exemplo:

```
1 paises = ["Argentina", "Argentina", "Brasil", "Canadá"]
2 paises.remove("Argentina")
3 print(paises)
4 # ['Argentina', 'Brasil', 'Canadá']
5 paises.remove("Dinamarca")
6 # ValueError: list.remove(x): x not in list
```

Removendo Elementos

- Podemos, antes de usar o método `remove`, verificar se o elemento está na lista, prevenindo assim o erro.

```
1 paises = ["Argentina", "Argentina", "Brasil", "Canadá"]
2 if "Dinamarca" in paises:
3     paises.remove("Dinamarca")
4 else:
5     print("Dinamarca não está na lista.")
6 # Dinamarca não está na lista.
```

Removendo Todas as Ocorrências de um Elemento

- Podemos remover todas as ocorrências de um elemento, usando o método `remove` iterativamente.

```
1 n = int(input("Quantos números serão lidos? "))
2
3 lista = []
4 for i in range(n):
5     lista.append(int(input()))
6
7 x = int(input("Qual o número deve ser removido? "))
8
9 while x in lista:
10     lista.remove(x)
11
12 print(lista)
```

- Outra opção para remover um elemento de uma lista é utilizando o método `pop`.
- O método `pop` recebe como parâmetro a posição do elemento a ser removido da lista. Caso o parâmetro seja omitido, o último elemento da lista será removido.
- Como resposta, o método retorna o elemento removido.
- Cada elemento da posição especificada até o fim da lista é realocado para a posição anterior.

Removendo Elementos

- Exemplo:

```
1 paises = ["Argentina", "Dinamarca", "Brasil", "Canadá"]
2 print(paises.pop(1))
3 # Dinamarca
4 print(paises)
5 # ['Argentina', 'Brasil', 'Canadá']
6 print(paises.pop())
7 # Canadá
8 print(paises)
9 # ['Argentina', 'Brasil']
```

- O método `count` é utilizado para contar a quantidade de vezes que um elemento ocorre em uma lista.
- O método `count` recebe como parâmetro um elemento.
- Como resposta, o método retorna a quantidade de ocorrências do elemento na lista.

Contando Elementos

- Exemplo:

```
1 paises = ["Brasil", "brasil", "Brazil", "Brasil"]
2 print(paises.count("Brasil"))
3 # 2
4 print(paises.count("brasil"))
5 # 1
6 print(paises.count("Brazil"))
7 # 1
8 print(paises.count("brazil"))
9 # 0
```

Removendo Todas as Ocorrências de um Elemento

- Como vimos anteriormente, podemos remover todas as ocorrências de um elemento, usando o método `remove` iterativamente.

```
1 n = int(input("Quantos números serão lidos? "))
2
3 lista = []
4 for i in range(n):
5     lista.append(int(input()))
6
7 x = int(input("Qual número deve ser removido? "))
8 c = lista.count(x)
9
10 for i in range(c):
11     lista.remove(x)
12
13 print(lista)
```

Forma Compacta de Criar Listas

- Em Python, é possível criar uma lista com uma única linha de código, sem a necessidade de se criar uma lista vazia e depois adicionar os elementos um a um.

```
1 n = int(input("Quantos números serão lidos? "))
2
3 lista = []
4 for i in range(n):
5     lista.append(int(input()))
6
7 x = int(input("Qual o número deve ser removido? "))
8 c = lista.count(x)
9
10 for i in range(c):
11     lista.remove(x)
12
13 print(lista)
```

Forma Compacta de Criar Listas

- Em Python, é possível criar uma lista com uma única linha de código, sem a necessidade de se criar uma lista vazia e depois adicionar os elementos um a um.

```
1 n = int(input("Quantos números serão lidos? "))
2
3
4 lista = [int(input()) for i in range(n)]
5
6
7 x = int(input("Qual o número deve ser removido? "))
8
9
10 lista = [i for i in lista if i != x]
11
12
13 print(lista)
```

Invertendo a Ordem dos Elementos

- O método `reverse` inverte a ordem dos elementos de uma lista.
- O método `reverse` não recebe nenhum parâmetro e modifica automaticamente a lista.
- Exemplo:

```
1 semana = ["Domingo", "Segunda", "Terça", "Quarta",  
2   "Quinta", "Sexta", "Sábado"]  
3 print(semana)  
4 # ['Domingo', 'Segunda', 'Terça', 'Quarta', 'Quinta',  
5 #  'Sexta', 'Sábado']  
6 semana.reverse()  
7 print(semana)  
8 # ['Sábado', 'Sexta', 'Quinta', 'Quarta', 'Terça',  
9 #  'Segunda', 'Domingo']
```

Ordenando Listas

- Uma lista pode ser ordenada utilizando o método `sort`.
- O método `sort` possui o parâmetro opcional `reverse`, que indica se a lista deve ser ordenada de forma crescente (`False`) ou decrescente (`True`). Por padrão, o valor desse parâmetro é `False` (ordenação crescente).
- Exemplo:

```
1 a = [5, 3, 1, 4, 2, 6]
2 a.sort()
3 print(a)
4 # [1, 2, 3, 4, 5, 6]
5 a.reverse()
6 print(a)
7 # [6, 5, 4, 3, 2, 1]
```

Ordenando Listas

- Uma lista pode ser ordenada utilizando o método `sort`.
- O método `sort` possui o parâmetro opcional `reverse`, que indica se a lista deve ser ordenada de forma crescente (`False`) ou decrescente (`True`). Por padrão, o valor desse parâmetro é `False` (ordenação crescente).
- Exemplo:

```
1 a = [5, 3, 1, 4, 2, 6]
2
3
4
5 a.sort(reverse = True)
6 print(a)
7 # [6, 5, 4, 3, 2, 1]
```

Ordenando Listas

- Podemos usar a função `sorted` para obter uma cópia ordenada de uma lista, sem alterar a lista original.
- Exemplo:

```
1 a = [5, 3, 1, 4, 2, 6]
2 print(sorted(a))
3 # [1, 2, 3, 4, 5, 6]
4 print(a)
5 # [5, 3, 1, 4, 2, 6]
6 print(sorted(a)[::-1])
7 # [6, 5, 4, 3, 2, 1]
```


Copiando Listas

- Podemos atribuir uma lista para diferentes variáveis, mas as variáveis estarão relacionadas à mesma lista (objeto).
- Isso implica que qualquer modificação feita em uma variável afetará todas as outras.
- Exemplo:

```
1 a = [1]
2 b = a
3 b.append(2)
4 c = b
5 c.append(3)
6 print(a)
7 # [1, 2, 3]
8 print(b)
9 # [1, 2, 3]
10 print(c)
11 # [1, 2, 3]
```

Copiando Listas

- Se quisermos uma cópia independente de uma lista podemos utilizar o método `copy`.
- O método `copy` retorna uma cópia da lista.
- Esta cópia pode ser atribuída a uma variável.
- Exemplo:

```
1 a = [1]
2 b = a.copy()
3 b.append(2)
4 c = b.copy()
5 c.append(3)
6 print(a)
7 # [1]
8 print(b)
9 # [1, 2]
10 print(c)
11 # [1, 2, 3]
```

Clonando Listas

- Podemos clonar uma lista, para obter uma cópia independente.
- Uma lista pode ser clonada utilizando o operador de seleção de intervalos `[:]` ou com a função `list()`.
- Este clone pode ser atribuído a uma variável.
- Exemplo:

```
1 a = [1]
2 b = a[:]
3 b.append(2)
4 c = list(b)
5 c.append(3)
6 print(a)
7 # [1]
8 print(b)
9 # [1, 2]
10 print(c)
11 # [1, 2, 3]
```

Concatenando Listas

- O operador + pode ser utilizado com listas com o objetivo de concatená-las.
- Como resultado, uma nova lista é obtida seguindo a ordem da concatenação realizada.
- Exemplo:

```
1 a = [1, 2]
2 b = [3, 4]
3 c = [5, 6]
4 print(a + b + c)
5 # [1, 2, 3, 4, 5, 6]
6 print(c + b + a)
7 # [5, 6, 3, 4, 1, 2]
8 print(b + c + a)
9 # [3, 4, 5, 6, 1, 2]
```

Funções Úteis para Listas Numéricas

- A função `min` retorna o menor valor em uma lista:

```
1 numeros = [2.14, 5.32, 2.45, 1.43, 3.27]
2 print(min(numeros))
3 # 1.43
```

- A função `max` retorna o maior valor em uma lista:

```
1 numeros = [2.14, 5.32, 2.45, 1.43, 3.27]
2 print(max(numeros))
3 # 5.32
```

- A função `sum` retorna a soma de todos os elementos de uma lista:

```
1 numeros = [2.14, 5.32, 2.45, 1.43, 3.27]
2 print(sum(numeros))
3 # 14.61
```

Descrição

Crie uma lista com os nomes dos super-heróis que devem participar da *Iniciativa Vingadores* seguindo a ordem:

- Homem de Ferro
- Capitão América
- Thor
- Hulk
- Viúva Negra
- Gavião Arqueiro

```
1 vingadores = ["Homem de Ferro", "Capitão América",  
2 "Thor", "Hulk", "Viúva Negra", "Gavião Arqueiro"]
```

Exercício

Descrição

Agora, inclua o Homem-Aranha no final da lista e imprima em qual posição está o Thor.

```
1 vingadores = ["Homem de Ferro", "Capitão América",  
2   "Thor", "Hulk", "Viúva Negra", "Gavião Arqueiro"]  
3 vingadores.append("Homem-Aranha")  
4 if "Thor" in vingadores:  
5     print(vingadores.index("Thor")) # 2
```

Descrição

Infelizmente a Viúva Negra e o Homem de Ferro não fazem mais parte da *Iniciativa Vingadores*, então retire-os da lista.

```
1 vingadores = ["Homem de Ferro", "Capitão América",  
2   "Thor", "Hulk", "Viúva Negra", "Gavião Arqueiro",  
3   "Homem-Aranha"]  
4 if "Viúva Negra" in vingadores:  
5     vingadores.remove("Viúva Negra")  
6 if "Homem de Ferro" in vingadores:  
7     vingadores.remove("Homem de Ferro")  
8 print(vingadores)  
9 # ['Capitão América', 'Thor', 'Hulk',  
10 #  'Gavião Arqueiro', 'Homem-Aranha']
```


Tuplas

- Vimos que é possível adicionar, remover ou alterar elementos de uma lista.
- Já tuplas, uma vez criadas, não permitem modificações.
- Ou seja, tuplas são listas imutáveis.
- Listas e tuplas podem armazenar:
 - Dados homogêneos (Exemplos: listas/tuplas de emails, salários ou notas).
 - Dados heterogêneos (Exemplo: cadastro de uma pessoa em uma academia com as informações de nome, idade e peso).

- Podemos declarar uma tupla utilizando ().

```
1 variavel = (elemento_1, elemento_2, ..., elemento_n)
```

- Também podemos declarar uma tupla de maneira explícita utilizando a função tuple.

```
1 variavel = tuple([elemento_1, elemento_2, ..., elemento_n])
```

- Declaração implícita:

```
1 got = ("Game of Thrones", 2011, 2019, 9.4)
2 print(got)
3 # ('Game of Thrones', 2011, 2019, 9.4)
4 type(got)
5 # <class 'tuple'>
```

- Declaração explícita:

```
1 got = tuple(["Game of Thrones", 2011, 2019, 9.4])
2 print(got)
3 # ('Game of Thrones', 2011, 2019, 9.4)
4 type(got)
5 # <class 'tuple'>
```

- Declaração implícita:

```
1 t1 = ("a")
2 t2 = ("a",)
3 print(type(t1), type(t2))
4 # <class 'str'> <class 'tuple'>
5 t3 = ("a", "b", "c")
6 print(t3)
7 # ('a', 'b', 'c')
```

- Declaração explícita:

```
1 t1 = tuple([2019])
2 print(t1)
3 # (2019,)
4 t2 = tuple("MC102")
5 print(t2)
6 # ('M', 'C', '1', '0', '2')
7 # Strings também podem ser tuplas de caracteres
```

- Vimos que tuplas são imutáveis, mas se tentarmos modificá-las, o que acontece?

```
1 empresas = ("Google", "Facebook", "Amazon")
2 empresas[1] = "Samsung"
3 # TypeError: 'tuple' object does not support item
   assignment
```

- Um erro é gerado informando que o tipo tupla não permite modificações.

Tuplas

- Tudo o que vimos para listas também podemos aplicar para tuplas, exceto operações, métodos ou funções que adicionem, removam ou modifiquem elementos.
- Selecionando o primeiro elemento de uma tupla:

```
1 letras = ("A", "B", "C", "D", "E", "F", "G", "H")
2 print(letras[0])
3 # A
```

- Selecionando o último elemento de uma tupla:

```
1 letras = ("A", "B", "C", "D", "E", "F", "G", "H")
2 print(letras[-1])
3 # H
```

- Selecionando do segundo até o quarto elemento de uma tupla:

```
1 letras = ("A", "B", "C", "D", "E", "F", "G", "H")
2 print(letras[1:4])
3 # ('B', 'C', 'D')
```

- Selecionando os três primeiros elementos de uma tupla:

```
1 letras = ("A", "B", "C", "D", "E", "F", "G", "H")
2 print(letras[:3])
3 # ('A', 'B', 'C')
```

- Selecionando os quatro últimos elementos de uma tupla:

```
1 letras = ("A", "B", "C", "D", "E", "F", "G", "H")
2 print(letras[-4:])
3 # ('E', 'F', 'G', 'H')
```


- Verificando se um elemento está na tupla:

```
1 top5 = ("Black Mirror", "Breaking Bad", "Friends",  
2       "Game of Thrones", "The Big Bang Theory")  
3 print("House MD" in top5)  
4 # False  
5 print("Game of Thrones" in top5)  
6 # True  
7 print("friends" in top5)  
8 # False
```

- Concatenando tuplas:

```
1 a = (1, 2)
2 b = (3, 4)
3 c = (5, 6)
4 print(a + b + c)
5 # (1, 2, 3, 4, 5, 6)
6 print(c + b + a)
7 # (5, 6, 3, 4, 1, 2)
8 print(b + c + a)
9 # (3, 4, 5, 6, 1, 2)
```

- Obtendo a posição de um elemento em uma tupla:

```
1 cinema = ("Sony Pictures", "Walt Disney",  
2         "Universal Pictures", "Warner")  
3 print(cinema.index("Warner"))  
4 # 3  
5 print(cinema.index("Disney"))  
6 # ValueError: tuple.index(x): x not in tuple
```

- Evitando o erro caso o elemento não esteja na tupla:

```
1 cinema = ("Sony Pictures", "Walt Disney",  
2         "Universal Pictures", "Warner")  
3 if "Disney" in cinema:  
4     print(cinema.index("Disney"))  
5 else:  
6     print("Disney não está na tupla.")  
7 # Disney não está na tupla.
```

- Imprimindo todos os elementos de uma tupla (um elemento por linha).

```
1 cinema = ("Sony Pictures", "Walt Disney",  
2          "Universal Pictures", "Warner")  
3 for estudio in cinema:  
4     print(estudio)  
5 # Sony Pictures  
6 # Walt Disney  
7 # Universal Pictures  
8 # Warner
```

- Criando uma tupla de forma iterativa.

```
1 n = int(input("Quantos números serão lidos? "))
2 tupla = ()
3
4 for i in range(n):
5     x = int(input("Entre com um número: "))
6     tupla = tupla + tuple([x])
7
8 print(tupla)
```

- Criando uma tupla de forma iterativa.

```
1 n = int(input("Quantos números serão lidos? "))
2 tupla = ()
3
4 for i in range(n):
5     x = int(input("Entre com um número: "))
6     tupla = tupla + (x,)
7
8 print(tupla)
```

Exercícios

- Escreva um programa que, dado um vetor de números inteiros e um número inteiro x , construa um vetor com os índices de todas as posições de x no vetor original.
- Escreva um programa que, dado um vetor de números inteiros, calcule a média dos elementos do vetor.
- Escreva um programa que, dado um vetor de números inteiros, calcule a mediana dos elementos do vetor.
- Escreva um programa que, dado um vetor de números inteiros, construa um vetor com os elementos que são a moda do vetor.

Todas as posições de um elemento

Escreva um programa que, dado um vetor de números inteiros e um número inteiro x , construa um vetor com os índices de todas as posições de x no vetor original.

- Possível Resposta:

```
1 v = [1, 2, 3, 2, 1, 1, 2, 2, 3, 1]
2 x = int(input("Entre com um número a ser buscado: "))
3 pos = []
4 for i in range(len(v)):
5     if v[i] == x:
6         pos.append(i)
7 print(pos)
```

Exercício

Média dos valores de um vetor

Escreva um programa que, dado um vetor de números inteiros, calcule a média dos elementos do vetor.

- Possível Resposta:

```
1 v = [1, 2, 3, 2, 1, 1, 2, 2, 3, 1]
2 soma = 0
3 n = 0
4 for i in v:
5     soma = soma + i
6     n = n + 1
7 print(soma/n)
```

Exercício

Média dos valores de um vetor

Escreva um programa que, dado um vetor de números inteiros, calcule a média dos elementos do vetor.

■ Possível Resposta:

```
1 v = [1, 2, 3, 2, 1, 1, 2, 2, 3, 1]
2 soma = 0
3
4 for i in v:
5     soma = soma + i
6
7 print(soma/len(v))
```

Exercício

Média dos valores de um vetor

Escreva um programa que, dado um vetor de números inteiros, calcule a média dos elementos do vetor.

■ Possível Resposta:

```
1 v = [1, 2, 3, 2, 1, 1, 2, 2, 3, 1]
2
3
4
5
6
7 print(sum(v)/len(v))
```

Mediana dos valores de um vetor

Escreva um programa que, dado um vetor de números inteiros, calcule a mediana dos elementos do vetor.

■ Possível Resposta:

```
1 v = [1, 2, 3, 2, 1, 1, 2, 2, 3, 1]
2 v.sort()
3 n = len(v)
4 if n % 2 == 1:
5     mediana = v[(n - 1) // 2]
6 else:
7     mediana = (v[(n // 2) - 1] + v[(n // 2)]) / 2
8 print(mediana)
```

Moda dos valores de um vetor

Escreva um programa que, dado um vetor de números inteiros, construa um vetor com os elementos que são a moda do vetor.

■ Possível Resposta:

```
1 v = [1, 2, 3, 2, 1, 1, 2, 2, 3, 1]
2 moda = []
3 max = 0
4 for i in v:
5     if v.count(i) > max:
6         moda = [i]
7         max = v.count(i)
8     elif v.count(i) == max:
9         if not(i in moda):
10             moda.append(i)
11 print(moda)
```

- Escreva um programa que, dados dois vetores ordenados de números inteiros, produza um vetor ordenado com todos os números dos dois vetores originais.
- Escreva um programa que, dado um vetor de números inteiros, construa um vetor com os elementos que possuem apenas uma cópia no vetor original.
- Escreva um programa que, dado um vetor de números inteiros, construa um vetor formado por exatamente uma cópia de cada um dos elementos do vetor original.

Intercalação de vetores ordenados

Escreva um programa que, dados dois vetores ordenados de números inteiros, produza um vetor ordenado com todos os números dos dois vetores originais.

■ Possível Resposta:

```
1 a = [1, 1, 2, 4, 5, 6, 8, 9, 9, 9]
2 b = [0, 1, 3, 3, 6, 7, 7, 8, 9]
3 c = []
4 while a and b:
5     if a[0] < b[0]:
6         c.append(a.pop(0))
7     else:
8         c.append(b.pop(0))
9 c = c + a + b
10 print(c)
```


Vetor com os elemento únicos da lista original

Escreva um programa que, dado um vetor de números inteiros, construa um vetor com os elementos que possuem apenas uma cópia no vetor original.

- Possível Resposta:

```
1 v = [4, 2, 3, 2, 1, 1, 5, 2, 3, 1]
2 u = []
3 for i in v:
4     if v.count(i) == 1:
5         u.append(i)
6 print(u)
```

Exercício

Vetor com uma cópia de cada elemento original

Escreva um programa que, dado um vetor de números inteiros, construa um vetor formado por exatamente uma cópia de cada um dos elementos do vetor original.

- Possível Resposta:

```
1 v = [4, 2, 3, 2, 1, 1, 5, 2, 3, 1]
2 u = []
3 for i in v:
4     if not(i in u):
5         u.append(i)
6 print(u)
```

- Escreva um programa que, dados dois vetores que representam dois conjuntos A e B , construa um vetor que represente a união dos dois conjuntos ($A \cup B$).
- Escreva um programa que, dados dois vetores que representam dois conjuntos (A e B), construa um vetor que represente a interseção dos dois conjuntos ($A \cap B$).
- Escreva um programa que, dados dois vetores que representam dois conjuntos (A e B), construa um vetor que represente a diferença dos dois conjuntos ($A \setminus B$).

União de conjuntos

Escreva um programa que, dados dois vetores que representam dois conjuntos A e B , construa um vetor que represente a união dos dois conjuntos ($A \cup B$).

- Possível Resposta:

```
1 A = [1, 2, 3, 4, 5]
2 B = [3, 4, 5, 6, 7]
3 C = list(A)
4
5 for i in B:
6     if not(i in C):
7         C.append(i)
8 print(C)
```

Interseção de conjuntos

Escreva um programa que, dados dois vetores que representam dois conjuntos (A e B), construa um vetor que represente a interseção dos dois conjuntos ($A \cap B$).

- Possível Resposta:

```
1 A = [1, 2, 3, 4, 5]
2 B = [3, 4, 5, 6, 7]
3 C = []
4
5 for i in A:
6     if i in B:
7         C.append(i)
8 print(C)
```

Diferença de conjuntos

Escreva um programa que, dados dois vetores que representam dois conjuntos (A e B), construa um vetor que represente a diferença dos dois conjuntos ($A \setminus B$).

- Possível Resposta:

```
1 A = [1, 2, 3, 4, 5]
2 B = [3, 4, 5, 6, 7]
3 C = []
4
5 for i in A:
6     if not(i in B):
7         C.append(i)
8 print(C)
```