Teste de Software para Web

Revisão de Lógica de Programação

MSc. Jonathas Silva dos Santos

Todos os direitos reservados.

Roteiro

Listas

Tuplas

Exercícios

Listas

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]
```

Listas

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

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

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

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

```
unicamp = list("Unicamp")
# unicamp = ['U', 'n', 'i', 'c', 'a', 'm', 'p']
# 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:

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

- Selecionando o segundo elemento de uma lista:

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

- Selecionando o quinto elemento de uma lista:

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

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

- Selecionando o penúltimo elemento de uma lista:

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

- Selecionando o antepenúltimo elemento de uma lista:

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

Selecionando um Elemento

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

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

```
letras = ["A", "B", "C", "D", "E", "F", "G", "H"]
print(letras[-9])
# 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:

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

Selecionando Intervalos

- Podemos selecionar um trecho de uma lista:

```
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:

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

- Selecionando os três primeiros elementos de uma lista:

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

- Selecionando os quatro últimos elementos de uma lista:

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

Selecionando Intervalos

- Selecionando os elementos das posições pares de uma lista:

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

- Selecionando os elementos das posições ímpares de uma lista:

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

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

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

Alterando um Elemento

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

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

- Exemplo:

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

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

- Exemplo:

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

```
elemento in lista
```

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

```
top5 = ["Black Mirror", "Breaking Bad", "Friends",
  "Game of Thrones", "The Big Bang Theory"]
print("House MD" in top5)
# False
print("Game of Thrones" in top5)
# True
print("friends" in top5)
# 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:

```
animais = []
animais.append("Gato")
print(animais)
# ['Gato']
animais.append("Cachorro")
print(animais)
# ['Gato', 'Cachorro']
animais.append("Coelho")
print(animais)
# ['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.

```
print("Entre com números positivos:")
lista = []
while True:
  p = int(input())
  if p <= 0:
    break
  lista.append(p)
x = int(input("Qual o número a procurar? "))
if x in lista:
  print(x, "pertence à lista")
else:
  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:

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

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

```
cinema = ["Sony Pictures", "Walt Disney",
  "Universal Pictures", "Warner"]
if "Disney" in cinema:
  print(cinema.index("Disney"))
else:
  print("Disney não está na lista.")
# 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.

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

```
n = int(input("Quantos números serão lidos? "))

lista = []
for i in range(n):
  lista.append(int(input()))

x = int(input("Qual o número deve ser removido? "))

while x in lista:
  lista.remove(x)

print(lista)
```

Removendo Elementos

- 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:

```
paises = ["Argentina", "Dinamarca", "Brasil", "Canadá"]
print(paises.pop(1))
# Dinamarca
print(paises)
# ['Argentina', 'Brasil', 'Canadá']
print(paises.pop())
# Canadá
print(paises)
# ['Argentina', 'Brasil']
```

Contando Elementos

- 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:

```
paises = ["Brasil", "brasil", "Brazil", "Brasil"]
print(paises.count("Brasil"))
# 2
print(paises.count("brasil"))
# 1
print(paises.count("Brazil"))
# 1
print(paises.count("brazil"))
# 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 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  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:

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

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

Copiando Listas

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

```
a = [1]
b = a
b.append(2)
c = b
c.append(3)
print(a)
# [1, 2, 3]
print(b)
# [1, 2, 3]
print(c)
# [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:

```
a = [1]
b = a.copy()
b.append(2)
c = b.copy()
c.append(3)
print(a)
# [1]
print(b)
# [1, 2]
print(c)
# [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:

```
a = [1]
b = a[:]
b.append(2)
c = list(b)
c.append(3)
print(a)
# [1]
print(b)
# [1, 2]
print(c)
# [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:

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

Funções Úteis para Listas Numéricas

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

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

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

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

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

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

Exercício

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

```
vingadores = ["Homem de Ferro", "Capitão América",
  "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.

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

Exercício

Descrição

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

```
vingadores = ["Homem de Ferro", "Capitão América",
  "Thor", "Hulk", "Viúva Negra", "Gavião Arqueiro",
  "Homem-Aranha"]
if "Viúva Negra" in vingadores:
  vingadores.remove("Viúva Negra")
if "Homem de Ferro" in vingadores:
  vingadores.remove("Homem de Ferro")
print(vingadores)
# ['Capitão América', 'Thor', 'Hulk',
#  'Gavião Arqueiro', 'Homem-Aranha']
```

Tuplas

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).

Tuplas

- Podemos declarar uma tupla utilizando `( )`.

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

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

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

Tuplas

- Declaração implícita:

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

- Declaração explícita:

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

Tuplas

- Declaração implícita:

```
t1 = ("a")
t2 = ("a",)
print(type(t1), type(t2))
# <class 'str'> <class 'tuple'>
t3 = ("a", "b", "c")
print(t3)
# ('a', 'b', 'c')
```

- Declaração explícita:

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

Tuplas

- 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
```

Tuplas

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

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

- Selecionando os três primeiros elementos de uma tupla:

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

- Selecionando os quatro últimos elementos de uma tupla:

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

Tuplas

- Verificando se um elemento está na tupla:

```
top5 = ("Black Mirror", "Breaking Bad", "Friends",
  "Game of Thrones", "The Big Bang Theory")
print("House MD" in top5)
# False
print("Game of Thrones" in top5)
# True
print("friends" in top5)
# False
```

Tuplas

- Concatenando tuplas:

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

Tuplas

- Obtendo a posição de um elemento em uma tupla:

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

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

```
cinema = ("Sony Pictures", "Walt Disney",
  "Universal Pictures", "Warner")
if "Disney" in cinema:
  print(cinema.index("Disney"))
else:
  print("Disney não está na tupla.")
# Disney não está na tupla.
```

Tuplas

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

```
cinema = ("Sony Pictures", "Walt Disney",
  "Universal Pictures", "Warner")
for estudio in cinema:
  print(estudio)
# Sony Pictures
# Walt Disney
# Universal Pictures
# Warner
```

Tuplas

- 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)
```

Tuplas

- 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

Exercício 1

Descrição

Dada uma lista L de n valores inteiros, escreva um programa que remova todos os números pares da lista.

Exemplo:
Tamanho da lista L: 10
L: 1 2 3 4 5 6 7 8 9 10

Resposta:
1 3 5 7 9

Exercício 2

Descrição

Dadas duas listas $P1$ e $P2$, ambas com n valores reais que representam as notas de uma turma na prova 1 e na prova 2, respectivamente, escreva um programa que calcule a média da turma nas provas 1 e 2, imprimindo em qual das provas a turma obteve a melhor média.

Exemplo:
Tamanho da turma: 5
$P1$: 7.0 8.3 10.0 6.5 9.3
$P2$: 8.5 6.9 5.0 7.5 9.8

Resposta:
Média da turma na prova 1: 8.22
Média da turma na prova 2: 7.54
A turma obteve a melhor média na prova 1.

Exercício 3

Descrição

Dadas duas listas $L1$ e $L2$, com n e m valores inteiros, respectivamente, escreva um programa que concatene as listas $L1$ e $L2$ em uma nova lista $L3$. Em seguida, imprima a lista $L3$ ordenada de maneira crescente e decrescente.

Exemplo:
Tamanho da lista $L1$: 3
Tamanho da lista $L2$: 4
$L1$: 7 2 9
$L2$: 2 5 1 3

Resposta:
1 2 2 3 5 7 9
9 7 5 3 2 2 1