Listas

Listas

É uma sequência de valores organizados entre colchetes [ ].

Esses dados podem ser de diferentes tipos:

inteiros, floats, strings e inclusive outras listas.

Exemplos:

dezenas = [10, 20, 30, 40]

pessoas= ['Alini', 'Pri', 'Lu', 'Jessica','Suzi']

lista_vazia = []

Listas

Os elementos de uma lista podem ser acessados pelo índice:

cores = ['amarelo', 'azul', 'branco', 'dourado']

```
>>> cores = ['amarelo', 'azul', 'branco', 'dourado']
>>> cores[0]
'amarelo'
>>> cores[2]
'branco'
>>>
```

Lembrando que a primeira posição de uma lista em Python é a posição 0:

Listas

Além de ser possível **acessar** cada elemento da lista pelo índice, é possível, **inserir**, **alterar** e **deletar**.

Listas são **mutáveis**, ou seja, é possível alterar seus elementos: **adicionando, removendo e/ou substituindo-os.**

Listas

Formato da função	Explicação	Exemplo
lista.append(exemplo)	Adiciona um novo elemento ao final da lista.	inteiros = [1, 2, 3] inteiros.append(10)
lista.insert(posição, elemento)	Adiciona um novo elemento numa determinada posição da lista. Todos os elementos daquela posição em diante são deslocados uma posição para frente.	disciplinas = [“SO”, “IA”] disciplinas.insert(0, ”CAP”)
lista.remove(elemento)	Remove o primeiro elemento especificado encontrado na lista.	nomes = [“Ana”, “Bia”] nomes.remove(“Ana”)
lista.count(elemento)	Conta quantas vezes o elemento especificado aparece na lista. Retorna zero se o elemento não pertencer à lista.	frutas = [“maçã”, “uva”] frutas.count(“uva”)

Listas

Formato da função	Explicação	Exemplo
lista.index(elemento)	Retorna a posição da primeira ocorrência do elemento especificado. Caso não seja encontrado, um erro será lançado.	tamanhos = [“M”, “GG”] tamanhos.index(“GG”)
lista.pop(posição	Remove e retorna o elemento da posição especificada. Caso não seja especificada uma posição, removerá o último elemento da lista.	Inteiros = [5, 6, 7, 8] inteiros.pop(2)
lista.sort()	Caso os elementos tenham implementação do operador menor (<) e sejam comparáveis (normalmente, de mesmo tipo), Python oferece uma função pronta para ordenar seu conteúdo. Caso os elementos não sejam comparáveis, um erro será lançado.	cores = [“lilás”, “azul”] cores.sort()

Listas

Formato da função	Explicação	Exemplo
lista.reverse()	Inverte as posições dos elementos de uma lista. O último passa a ser o primeiro, o penúltimo passa a ser o segundo e assim por diante.	coffee = [“bolo”, “refri”] coffee.reverse()
list(“objeto iterável”)	Cria uma lista a partir de um objeto iterável, por exemplo, uma string. Se não for objeto iterável, um erro será lançado	palavra = “PyLadies” list(palavra)
len(lista)	Retorna o tamanho de uma lista. Funciona para qualquer objeto iterável. Se não for um objeto iterável, um erro será lançado.	ladies = [“Nath”, “Karol”] len(ladies)
elemento in lista	Verifica se um elemento está presente em uma lista.	trilhas = [“IA”, “web”] “IA” in trilhas

Listas

Exemplo de como utilizar algumas das funções apresentadas:

```
coffee = ['refri', 'bolinha de queijo', 'bolo', 'coxinha']
coffee.append('café')
coffee.insert(0, 'água')
coffee.remove('refri')
```

Resultado esperado:

>>['água', 'bolinha de queijo', 'bolo', 'coxinha', 'café']

Iterando uma lista:

```
lista = [1, 2, 3, 4, 10]
for numero in lista:
    print(numero ** 2)
```

```
1
4
9
16
100
```

Se aplica a strings também:

```
palavra = "casa"
for letra in palavra:
    print(letra)
```

```
c
a
s
a
```

Exercícios em Python [Listas-1]

Possuímos uma lista de cinco desenvolvedoras que participam do programa Maria vai com as devs.

Construa um programa que descubra quantas delas possuem o nome começando com a letra “A”.

As desenvolvedoras se chamam: Andorinha, Azaléia, Amélia, Margarida e Rosa.

Exercícios em Python [Listas-1]

```
devs = ['amelia', 'azaleia', 'andorinha', 'margarida', 'rosa']
nome_com_a = 0

for nome in devs:
  if nome[0] == 'a':
    nome_com_a += 1

print(f"Foram encontradas {nome_com_a} desenvolvedoras que começam com a letra A")
```

```
Foram encontradas 3 desenvolvedoras que começam com a letra A
```

Exercícios em Python [Listas-2]

Leia um número N e, em seguida, leia N números inteiros. Imprima o maior e o menor entre os N inteiros lidos.

Exemplo de entrada: 10

10 -5 1 2 3 100 4 5 -17 1

Saída esperada: Maior: 100

Menor: -17

Exercícios em Python [Listas-2]

```
n = int(input("Digíte a quantidade de números que deseja processar:"))
maior = -1000000
menor = 1000000
entrada = map(int, input(f"Digíte {n} números, separado por vírgula:").split(","))

for atual in entrada:
  menor = min(menor, atual)
  maior = max(maior, atual)

print(f"Maior: {maior}")
print(f"Menor: {menor}")
```

```
Digíte a quantidade de números que deseja processar:10
Digíte 10 números, separado por vírgula:10, -5, 1, 2, 3, 100, 4, 5, -17,1
Maior: 100
Menor: -17
```

Exercícios em Python [Listas-2-segunda solução <3 mais linda também]

```python
quantidade = int(input("Quantos números você quer escolher? "))
index = 0
lista = []
while (index < quantidade):
    proximo = int(input("Digite o proximo número: "))
    index += 1
    lista.append(proximo)

lista.sort()
print(lista)
print(f"Menor: {lista[0]} , Maior: {lista[-1]}")
```

```
exemplos
C:\Users\alini\Documents\repository\Virtualenvs\mvcad-cursos\Scripts\pyth
Quantos números você quer escolher? 5
Digite o proximo número: 10
Digite o proximo número: -10
Digite o proximo número: 100
Digite o proximo número: 8
Digite o proximo número: 5
[-10, 5, 8, 10, 100]
Menor: -10 , Maior: 100

Process finished with exit code 0
```

Exercícios em Python [Listas-3]

O RH da Serasa precisa controlar a presença dos membros nas atividades do MVCAD, isto é, precisa anotar na planilha de frequência todos os que participaram.

Como você está estudando Python, se voluntariou a ajudar a tornar essa tarefa um pouco mais fácil, você pretende entregar a lista de presentes já ordenada alfabeticamente para o RH!

Dado um número N, que representa quantas pessoas participaram, e N nomes, imprima os N nomes ordenados alfabeticamente.

Exemplo de entrada:

```
4
Suzi
Pri
Alini
Lu
```

Exemplo de entrada:

```
Alini
Lu
Pri
Suzi
```

Exercícios em Python [Listas-3]

```
presentes = int(input("Quantas pessoas estão presentes?"))
pessoas = []
for i in range(presentes):
  pessoas.append(input("Digite o nome do participante:"))
pessoas.sort()
print("Participantes por ordem alfabética:")
print("\n".join(pessoas))
```

```
Quantas pessoas estão presentes?4
Digite o nome do participante:Suzi
Digite o nome do participante:Alini
Digite o nome do participante:Lu
Digite o nome do participante:Pri
Participantes por ordem alfabética:
Alini
Lu
Pri
Suzi
```

Dicionários

Dicionários

Dicionário é um objeto que relaciona uma chave a um valor.

Um dicionário se parece com uma lista, mas é mais geral. Em uma lista, os índices têm que ser números inteiros; em um dicionário, eles podem ser de (quase) qualquer tipo.

Um dicionário contém uma coleção de índices, que se chamam chaves e uma coleção de valores. Cada chave é associada com um único valor. A associação de uma chave e um valor chama-se par chave-valor ou item.

Em linguagem matemática, um dicionário representa um mapeamento de chaves a valores, para que você possa dizer que cada chave “mostra o mapa a” um valor.

Dicionários

A função dict cria um novo dicionário sem itens.
Como dict é o nome de uma função integrada, você deve evitar usá-lo como nome de variável.

```
>>> marias = dict()
>>> marias
{}
```

As chaves {} representam um dicionário vazio. Para acrescentar itens ao dicionário, você pode usar colchetes (a linha abaixo cria um item que mapeia da chave nome ao valor 'Maria vai com as devs'.):

```
>>> marias['nome'] = 'Maria vai com as devs'
>>> marias
{'nome': 'Maria vai com as devs'}
```

Dicionários

```
dicionario_vazio = {}
camiseta = {"M":12, "G": 10, "GG":10}

dados = {
 "grupo": "PyLadies São Carlos" ,
 "fundação":2014,
 "linguagem": "Python",
 "membras": 37
}

alunas_curso = {"Ana": "BCC", "Amanda": "EnC"}
alunas_curso["Lúcia"] = "BCC"
```

Dicionários

Para verificar se uma chave já existe, utilize o operador in:

```
>>> dic = {'chave': 'valor'}
>>> 'chave'in dic
True
>>> 'chave_inexistente' in dic
False
>>>
```

Para deletar uma chave e seu valor de um dicionário, utilize o operador del:

```
>>> del(dic['chave'])
>>> dic
{}
>>>
```

Dicionários

Podemos obter todas as chaves de um dicionário com o método keys():

```
dicionario.keys()
```

E também podemos obter todos os valores de um dicionário com o método values():

```
dicionario.values()
```

Dicionários - Funções

Formato da função	Explicação	Exemplo
dicionario.clear()	Deleta todo o conteúdo do dicionário e retorna um dicionário vazio.	mvcad= {"nome": "Ana"} mvcad.clear()
dicionario.copy()	Retorna um novo dicionário, com uma cópia real do objeto. Se você fizer dict1 = dict2, ele criará apenas uma referência.	mvcad= {"nome": "Ana"} mvcad.copy()
dicionario.fromkeys(objeto_iterável, valor_default)	Cria um novo dicionário com as chaves contidas no objeto_iterável e valores baseadas no valor_default.	dict().fromkeys([1, 2, 3, 4], None)

Dicionários - Funções

Formato da função	Explicação	Exemplo
dicionario.get(chave)	Retorna o valor de uma dada chave, ou None caso ela não exista. É aconselhável o uso em vez de acesso direto em casos onde uma chave possa não existir, pois no acesso direto causaria erro.	mvcad= {"nome": "Alini", "estado": "SC"} mvcad.get('nome')
dicionario.items()	Retorna um objeto dict_items, que é composto por uma tupla que contém as chaves e outra que contém os valores. Útil em iterações para pegar chave e valor.	mvcad= {"nome": "Alini", "estado": "SC"} mvcad.items()
dicionario.keys()	Retorna um objeto dict_keys, que é composto por uma lista com as chaves.	mvcad= {"nome": "Alini", "estado": "SC"} mvcad.keys()

Dicionários - Funções

Formato da função	Explicação	Exemplo
dicionario.pop(chave)	**Retorna o valor de uma dada chave e exclui todo o elemento do dicionário.**	mvcad= {"nome": "Ana"} mvcad.pop('nome')
dicionario.popitem()	**Retorna um elemento do dicionário e exclui ele do dicionário. Não é possível escolher o elemento que será retornado.**	mvcad= {"nome": "Ana"} mvcad.popitem()
dicionario.update(dicionario)	**Atualiza elementos existentes ou adiciona elementos caso estes não existam num dicionário base.**	mvcad= {"nome": "Ana"} mvcad.update()

Dicionários

Para iterar sobre dicionários usando a chave como acesso:

```
for chave in dicionario:
    dicionario[chave]
```

Maria vai com as Devs

Intervalo:
Volta ás 11:20

Maria vai com as Devs

Recapitulando:

- Conhecemos o projeto que vamos construir
- Manipulação de Strings
- Listas
- Dicionários

E agora?

Analisar o arquivo CSV que contém os dados de matrícula e responder:

- Os dados encontrados na planilha são referentes a que? Como é o formato do arquivo?

  Resposta: Planilha contém dados dos alunos inscritos no curso, possui um cabeçalho e muitas linhas com os registros dos alunos

- Para armazenar estes dados, precisamos criar quantas tabelas?

  Resposta: Pessoa, Endereço e Curso

- O que vamos precisar conhecer para conseguir colocar esse arquivo pra dentro da nossa aplicação python?

  Resposta: ~~**Manipulação Strings, Listas, Dicionários**~~**,**

  Manipulação de arquivos, Conectar python com banco de dados, API

Manipulando arquivos de texto

Podemos abrir um arquivo de duas maneiras:

- Para somente leitura ('r')
- Com permissão de escrita ('w')

```
#
# leitura
#
f = open('nome-do-arquivo', 'r')

#
# escrita
#
f = open('nome-do-arquivo', 'w')
```

Manipulando arquivos de texto

Se não especificarmos o segundo parâmetro, a forma padrão leitura (‘r’) será utilizada:

```
>>> arquivo = open('nome-do-arquivo')
>>> arquivo
<_io.TextIOWrapper name='nome-do-arquivo.text' mode='r' encoding='UTF-8'>
```

O terceiro parâmetro é opcional e nele especificamos a codificação do arquivo:

```
arquivo = open(nome-do-arquivo, 'r', encoding="utf8")
```

Manipulando arquivos de texto

Se tentarmos abrir um arquivo para leitura que não existe, um erro será lançado:

```
>>> f = open('nome-errado.text', 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'nome-errado.text'
```

Se tentarmos abrir um arquivo para escrita que não existe, então ele será criado, porém, se ele já existir, todo seu conteúdo será apagado no momento em que abrimos o arquivo.

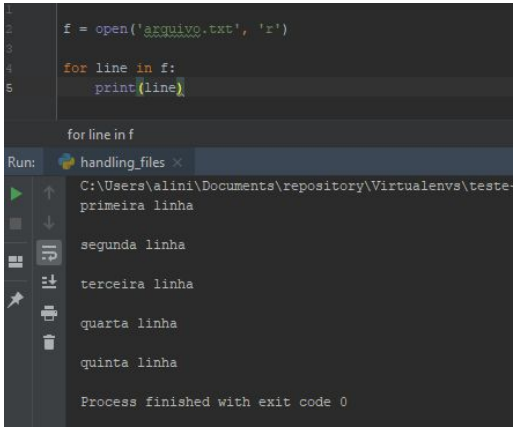
Devemos sempre fechar o arquivo aberto: **arquivo.close()**

Exemplos

Como exemplo utilizaremos o arquivo de texto **seu-arquivo.text** que possui o seguinte conteúdo:

primeira linha
segunda linha
terceira linha
quarta linha
quinta linha

Podemos abrir um arquivo e iterar por cada linha conforme exemplo abaixo:



Exemplos

Se quisermos ler todo o conteúdo do arquivo em uma única string podemos utilizar a função read():

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.read()
'primeira linha\nsegunda linha\nterceira linha\nquarta linha\nquinta linha'
```

A função retornará uma lista vazia [] quando encontrar o final do arquivo (após a última linha ter sido lida).

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.readline()
'primeira linha\n'
>>> f.readline()
'segunda linha\n'
>>> f.readline()
'terceira linha\n'
>>> f.readline()
'quarta linha\n'
>>> f.readline()
'quinta linha'
>>> f.readline()
''
```

Exemplos

Se quisermos ler todas linhas restantes em uma lista podemos utilizar a função readlines (estamos no plural).

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.readlines()
['primeira linha\n', 'segunda linha\n', 'terceira linha\n', 'quarta linha\n', 'quinta linha']
>>> f.readlines()
[]
```

Exemplos

Repare que ao chamarmos pela segunda vez a função retornar uma lista vazia pois ela, na verdade, retorna as linhas restantes. Como, ao abrir o arquivo, restavam todas as linhas então ela retornou todas as linhas.

Confundiu? Veja se este exemplo clareia as coisas.

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.readline()
'primeira linha\n'
>>> f.readline()
'segunda linha\n'
>>> f.readlines()
['terceira linha\n', 'quarta linha\n', 'quinta linha']
```

Exemplos

Para escrever em um arquivo sem apagar seu conteúdo, ou seja, adicionando (incluído) novo conteúdo seguimos 3 passos:

- Ler todo o conteúdo do arquivo (Cursor vai para o final do arquivo)
- efetuar a adição do conteúdo
- escrever as linhas

```
# Abra o arquivo (leitura)
arquivo = open('musica.txt', 'r')
conteudo = arquivo.readlines()

# insira seu conteúdo
# obs: o método append() é proveniente de uma lista
conteudo.append('Nova linha')

# Abre novamente o arquivo (escrita)
# e escreva o conteúdo criado anteriormente nele.
arquivo = open('musica.txt', 'w')
arquivo.writelines(conteudo)
arquivo.close()
```

Leitura e escrita de arquivos CSV

Arquivos CSV

CSV (Comma Separated Values, ou valores separados por vírgulas) correspondem ao idioma comum de troca de informações entre planilhas de cálculo ou base de dados.

Embora cada aplicativo tenha o seu método específico de **codificar seus dados**, todos possuem uma forma de **exportar e importar** dados em formato csv.

- Cada linha em um arquivo csv é um registro de dados.
- Cada registro consiste de um ou mais campos, separados por vírgulas.

Biblioteca CSV

A linguagem Python possui um módulo, não por acaso denominado csv, que facilita em muito o trabalho com este tipo de arquivo em Python.

Este módulo implementa funções que realizam a leitura e escrita sobre arquivos csv.
https://cadernodelaboratorio.com.br/arquivos-csv-em-python/

Biblioteca CSV

As funções que iremos conhecer são:

- **csv.reader()**: Retorna um objeto “reader” que permite a iteração sobre as linhas do arquivo csv

- **csv.writer()**: Retorna um objeto “writer” que converte o dado do usuário numa linha do arquivo csv.

- **csv.DictReader()**: Com esta classe trabalhamos com um dicionário no qual as chaves são dadas pelos nomes das colunas. Estes nomes podes ser dados através do parâmetro “fieldnames” ou obtidos diretamente da primeira linha do arquivo csv.
- **csv.DictWriter()**: Cria um objeto que permite a gravação de linhas no arquivo cvs a partir de um dicionário passado como parâmetro.

Exemplo CSV.reader()

```
with open('teste.csv') as csfile:
    reader = csv.reader(csfile)
```

Exemplo CSV.writer()

```
with open('teste.csv', 'w') as csfile:
    writer = csv.writer(csfile)
    writer.writerows("nova linha")
```

Exemplo CSV.DictWriter()

```
# w -> write, a-> append
with open('meu_arquivo.csv', 'a',  newline='') as csvfile:
    fieldnames = ['nome_coluna1', 'nome_coluna2']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames,  delimiter=";")

    writer.writeheader()
    writer.writerow({'nome_coluna1': 'boing 737', 'nome_coluna2': 'Ferrari'})
    writer.writerow({'nome_coluna1': 'airbus4', 'nome_coluna2': 'Volks6'})
```

Exemplo
CSV.DictReader()

```
with open('meu_arquivo.csv') as csvfile:
    reader = csv.DictReader(csvfile,  delimiter=";")
    for row in reader:
        print(row['nome_coluna1'], row['nome_coluna2'])
```

Maria vai com as Devs

Recapitulando:

- Conhecemos como funciona a manipulação de arquivos de texto no Python
- Conhecemos leitura e escrita de CSV no Python

Importando arquivo .csv para nosso projeto

E agora?

- Vamos voltar para nosso projeto
- Importar o arquivo CSV
- Separar os dados dos alunos em uma lista