

INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 1 (Arrays)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

rafael.speroni@ifc.edu.br

Objetivo:



- Compreender o conceito de Array.

Programa sem Array

```
nro = 10

for i in range(0, nro):
    n1 = float(input("Informe a primeira nota do aluno: "))
    n2 = float(input("Informe a segunda nota do aluno: "))

    media = (n1+n2)/2
    print(f"A média do aluno é: {media}")
```

Mas e se eu precisasse mostrar todas as médias após sair desta estrutura de repetição? Isso seria possível?

Definição

Em programação de computadores, **um arranjo(array)** é uma estrutura de dados que armazena uma coleção de elementos de tal forma que cada um dos elementos possa ser identificado por, pelo menos, **um índice** ou **uma chave**. Essa estrutura de dados também é conhecida como **variável indexada, vetor e matriz**.

Arrays em Python são estruturas **homogêneas**, exclusivamente sobre **valores numéricos** (int, float, entre outros).

Como representar um vetor?

Vetor **média**

0	1	2	3	4	5	6	7	8	9

Requisitos



Para fazer uso da biblioteca **NumPy** é preciso:

Em todos os sistemas operacionais (Windows, macOS e Linux):

- Instale o [Anaconda](#) (instala todos os pacotes que você precisa e todas as várias outras ferramentas).
- Ou, utilize o comando **pip install numpy** no terminal.

Como declarar um vetor?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)
```

Como mostrar o conteúdo de um vetor?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)

# mostra os valores armazenados na estrutura
print(vetor)
```


Como preencher com valores do usuário?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)

#preenche o vetor com valores do tipo float
for i in range(N):
    vetor[i] = float(input(f'Informe um valor para V[{i}]):

# mostra os valores armazenados na estrutura
print(vetor)
```

Como preencher com valores do usuário?

```
import numpy as np
```

arrays possuem tamanhos determinados

```
# define o tamanho do array
```

```
N = 5
```

é preciso iniciar com zeros(0) neste caso

```
#preenche a estrutura com zeros
```

```
vetor = np.zeros(N)
```

precisamos de estrutura de repetição para percorrer cada posição do array.

```
#preenche o vetor com valores do tipo float
```

```
for i in range(N):
```

atribuindo valores float para o array

```
    vetor[i] = float(input(f'Informe um valor para V[{i}]):
```

```
# mostra os valores armazenados na estrutura
```

```
print(vetor)
```

exibindo o conteúdo do array;

Outra forma de mostrar os valores?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)

#preenche o vetor com valores do tipo float
for i in range(N):
    vetor[i] = float(input(f'Informe um valor para V[{i}]):

# outra forma de mostrar
for i in range(N):
    print(f'V[{i}] = {vetor[i]} ')
```

Outra forma de mostrar os valores?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)

#preenche o vetor com valores do tipo float
for i in range(N):
    vetor[i] = float(input(f'Informe um valor para V[{i}]):

# outra forma de mostrar
for i in range(N):
    print(f'V[{i}] = {vetor[i]} ')
```

para exibir o conteúdo do array
posição a posição é preciso fazer uso
de uma estrutura de repetição

Métodos especiais

```
# mostra o tipo da estrutura
print(type(vetor))

soma2 = vetor.sum() # somatório
media = vetor.mean() # média
desvio = vetor.std() # desvio padrão
max = vetor.max() # o maior valor
min = vetor.min() # o menor valor
argmax = vetor.argmax() # retorna a posição que contém o
maior valor da estrutura
argmin = vetor.argmin() # retorna a posição que contém o
menor valor da estrutura
```

Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de.. Algoritmos: lógica para desenvolvimento de programação de computadores.. 27. ed.. Érica. 2014

Referências Complementares

DOWNEY, Allen B. **Pense em Python**. 2ª Ed. Novatec. 2016

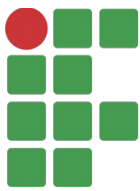
MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019

CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>



INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 1 (Arrays)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

rafael.speroni@ifc.edu.br

Objetivo:



- Compreender o conceito de Array.

Programa sem Array

```
nro = 10

for i in range(0, nro):
    n1 = float(input("Informe a primeira nota do aluno: "))
    n2 = float(input("Informe a segunda nota do aluno: "))

    media = (n1+n2)/2
    print(f"A média do aluno é: {media}")
```

Mas e se eu precisasse mostrar todas as médias após sair desta estrutura de repetição? Isso seria possível?

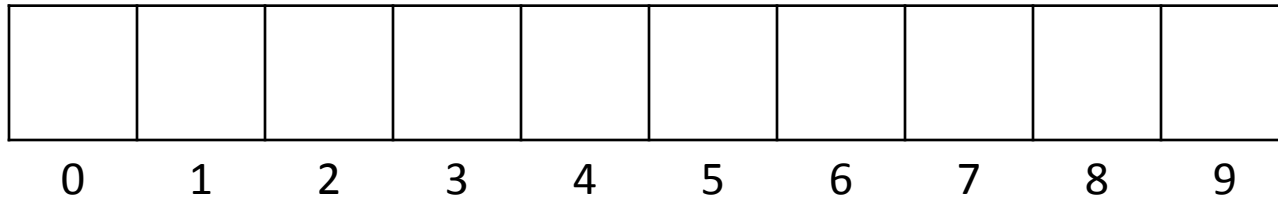
Definição

Em programação de computadores, **um arranjo(array)** é uma estrutura de dados que armazena uma coleção de elementos de tal forma que cada um dos elementos possa ser identificado por, pelo menos, **um índice** ou **uma chave**. Essa estrutura de dados também é conhecida como **variável indexada, vetor e matriz**.

Arrays em Python são estruturas **homogêneas**, exclusivamente sobre **valores numéricos** (int, float, entre outros).

Como representar um vetor?

Vetor **média**



Requisitos



Para fazer uso da biblioteca **NumPy** é preciso:

Em todos os sistemas operacionais (Windows, macOS e Linux):

- Instale o [Anaconda](#) (instala todos os pacotes que você precisa e todas as várias outras ferramentas).
- Ou, utilize o comando **pip install numpy** no terminal.

Como declarar um vetor?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)
```

Como mostrar o conteúdo de um vetor?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)

# mostra os valores armazenados na estrutura
print(vetor)
```

Como preencher com valores do usuário?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)

#preenche o vetor com valores do tipo float
for i in range(N):
    vetor[i] = float(input(f'Informe um valor para V[{i}]):

# mostra os valores armazenados na estrutura
print(vetor)
```

Como preencher com valores do usuário?

```
import numpy as np
```

arrays possuem tamanhos determinados

```
# define o tamanho do array
```

```
N = 5
```

é preciso iniciar com zeros(0) neste caso

```
#preenche a estrutura com zeros
```

```
vetor = np.zeros(N)
```

precisamos de estrutura de repetição para percorrer cada posição do array.

```
#preenche o vetor com valores do tipo float
```

```
for i in range(N):
```

atribuindo valores float para o array

```
    vetor[i] = float(input(f'Informe um valor para V[{i}]):
```

```
# mostra os valores armazenados na estrutura
```

```
print(vetor)
```

exibindo o conteúdo do array;

Outra forma de mostrar os valores?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)

#preenche o vetor com valores do tipo float
for i in range(N):
    vetor[i] = float(input(f'Informe um valor para V[{i}]):

# outra forma de mostrar
for i in range(N):
    print(f'V[{i}] = {vetor[i]} ')
```

Outra forma de mostrar os valores?

```
import numpy as np

# define o tamanho do array
N = 5

#preenche a estrutura com zeros
vetor = np.zeros(N)

#preenche o vetor com valores do tipo float
for i in range(N):
    vetor[i] = float(input(f'Informe um valor para V[{i}]):

# outra forma de mostrar
for i in range(N):
    print(f'V[{i}] = {vetor[i]} ')
```

para exibir o conteúdo do array
posição a posição é preciso fazer uso
de uma estrutura de repetição

Métodos especiais

```
# mostra o tipo da estrutura
print(type(vetor))

soma2 = vetor.sum() # somatório
media = vetor.mean() # média
desvio = vetor.std() # desvio padrão
max = vetor.max() # o maior valor
min = vetor.min() # o menor valor
argmax = vetor.argmax() # retorna a posição que contém o
maior valor da estrutura
argmin = vetor.argmin() # retorna a posição que contém o
menor valor da estrutura
```

Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de.. Algoritmos: lógica para desenvolvimento de programação de computadores.. 27. ed.. Érica. 2014

Referências Complementares

DOWNEY, Allen B. **Pense em Python**. 2ª Ed. Novatec. 2016

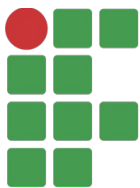
MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019

CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>



INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 2 (Listas)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

rafael.speroni@ifc.edu.br

Objetivo:



- Compreender o conceito de Listas.

Listas e Vetores

- Em **Python** existem três tipos principais de variáveis compostas: **Listas**, **Tuplas** e **Dicionários**, além de *ndarrays*.

Vamos aos conceitos:

- **Listas** são um tipo de variável que permite armazenar vários valores, acessados por um índice. Estes valores podem ser de um mesmo tipo ou de **tipos diversos**.
- **Vetores** são um tipo de variável que permite armazenar vários valores, acessados por um índice. Estes valores **devem ser do mesmo tipo**.

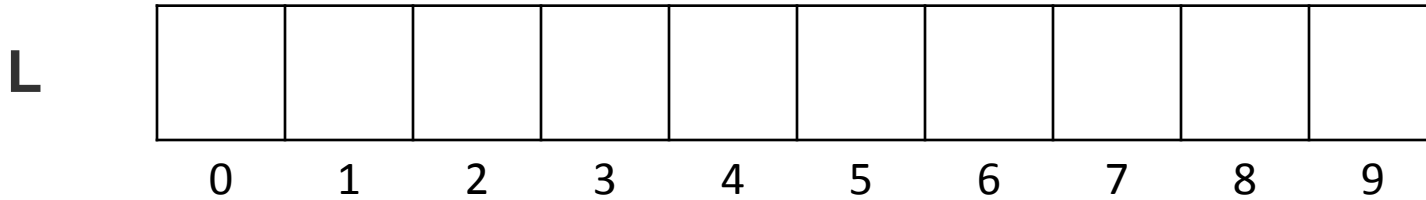
Ou seja, listas são mais flexíveis que vetores.

Como imaginar uma lista:

- Imagine um **edifício** que contém apartamentos, no térreo é o andar 0, o primeiro andar é o andar 1 e assim por diante;
- O índice é utilizado para especificarmos o “**apartamento**” onde guardamos os nossos dados;
- Em um prédio de **6 andares** teremos nosso índice variando de **0 até 5**;
- Se chamarmos nosso prédio de **P**, teremos **P[0], P[1], P[2], P[3], P[4], P[5]**;

P

Outra forma de representar um vetor ou lista?



Listas são mais **flexíveis** que prédios e podem crescer e ou diminuir com o tempo.

Como declarar uma lista?

```
L = []
```

- Este comando cria uma lista denominada de **L** e os colchetes (**[]**) após o símbolo de igualdade indicam que é uma lista vazia;

```
Z = [15, 8, 9]
```

- Este comando cria uma lista denominada de **Z** e esta lista contém 3 elementos: **15, 8 e 9**

Como acessar um elemento de uma lista?

```
Z = [15, 8, 9]
```

- Se acessarmos `Z[0]` o resultado que será exibido será 15, pois estamos acessando o primeiro elemento das listas, ou seja, o elemento que está armazenado no índice 0.
- `Z[1]` será igual a 8 e `Z[2]` será igual a 9.

Como alterar um elemento de uma lista?

```
z = [15, 8, 9]
```

- Se atribuirmos `z[0] = 10` o conteúdo de `z[0]` será atualizado. E ficaremos com:

```
z = [10, 8, 9]
```

Como ler e mostrar uma lista?

```
# cálculo das médias com notas digitadas
```

```
notas = [0, 0, 0, 0, 0]
```

```
soma = 0
```

```
x = 0
```

```
while x < 5:
```

```
    notas[x] = float(input(f"Nota {x}:"))
```

```
    soma += notas[x]
```

```
    x = x + 1
```

```
x = 0
```

```
while x < 5:
```

```
    print(f"Nota {x}: {notas[x]:6.2f}")
```

```
    x += 1
```

```
print(f"Média: {soma/ x:5.2f}")
```

é preciso iniciar com zeros(0) neste caso

precisamos de estrutura de repetição para percorrer cada posição da lista.

atribuindo valores float para a lista

exibindo o conteúdo de cada posição da lista;

Como verificar o tamanho da lista?

```
Z = [15, 8, 9]
```

- Podemos utilizar a função **len()**;

```
print(len(Z))
```

- Será exibido o valor 3, o que corresponde ao número de elementos da lista;

Adição de elementos em uma lista

```
Z = [15, 8, 9]
```

- Para adicionarmos um elemento ao final da lista podemos utilizar o método **append**;

```
Z.append(23)
```

- Outras formas de adicionar elementos a lista:

```
Z += [2] ou Z = Z + [2]
```

Removendo elementos de uma lista

```
Z = [15, 8, 9]
```

- Para removermos um elemento da lista podemos utilizar a instrução **del**;

```
del Z[1]
```

- irá ficar:

```
Z = [15, 9]
```

Importante observar que os elementos removidos da lista não ocuparão mais espaço e os índices serão reorganizados.

Podemos apagar também fatias da lista:

```
del Z[1:99] restaria Z = [0, 99, 100]
```


Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de.. Algoritmos: lógica para desenvolvimento de programação de computadores.. 27. ed.. Érica. 2014

Referências Complementares

DOWNEY, Allen B. **Pense em Python**. 2ª Ed. Novatec. 2016

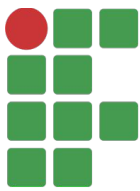
MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019

CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>



INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 3 (Tuplas e Conjuntos)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

rafael.speroni@ifc.edu.br

Objetivo:



- Compreender o conceito de Tuplas.
- Compreender o conceito de Conjuntos.

Tuplas

Vamos aos conceitos:

- **Tuplas** são uma estrutura de dados similar às listas, com a grande diferença de serem imutáveis.

```
tupla = (1, 2, 3, 4, 5)
```

Tupla vs Listas

```
lista = [1, 2, 3, 4, 5]  
tupla = (1, 2, 3, 4, 5)
```

Tuplas

Em Python, criamos tuplas utilizando parênteses ();

- Como ficaria no Python:

```
t = ("a", "b", "c")
```

- Os parênteses são opcionais

```
t = "a", "b", "c"
```

irá exibir o mesmo resultado;

Tuplas **suportam a maior parte das operações de listas**, como fatiamento e indexação;

Como declarar uma tupla?

```
t = ()
```

- Este comando cria uma tupla denominado de **t** e os parenteses () após o símbolo de igualdade indicam que é uma tupla vazia;

```
t = "a", "b", "c"
```

- Este comando cria uma tupla com 3 elementos (**ato de empacotar**);
Quando criamos tuplas com um único elemento é preciso ter cuidado:

```
t1 = (1)
```

```
t2 = (2,)
```

```
t3 = 1,
```

Neste exemplo somente t2 e t3 são tuplas.

Como alterar um elemento de uma tupla?

```
t = "a", "b", "c"
```

Tuplas **não** podem ter seus valores alterados, mas suporta cálculos:

```
print(t * 2)
```

Gera como resultado:

```
('a', 'b', 'c', 'a', 'b', 'c')
```

Encontrando um elemento de uma tupla?

- Além de consultar utilizando o índice;
- Podemos utilizar o **for**:

```
for elemento in t:  
    print(elemento)
```


Ato de desempacotar?

- Podemos usar o * para indicar vários valores a desempacotar:

```
*a, b = (1, 2, 3, 4, 5)
```

- No caso dizemos coloque o último valor em b e os demais valores em a.
- O conteúdo de a será igual à [1, 2, 3, 4], enquanto o de b será 5.
- Com o comando: `tuple(a)`, teremos a convertido para uma tupla e não mais uma lista.

Desempacota como lista.

Conjuntos(set)

Vamos aos conceitos:

- **Conjuntos** são uma estrutura de dados que implementa operações de união, intersecção, diferença, entre outros.
- A principal diferença é **não admitir a repetição de elementos.**
- **Não** mantém a ordem dos elementos.

Conjuntos

Em Python, criamos tuplas utilizando parênteses `set()`;

- Como ficaria no Python:

```
C = set()
```

- Podemos adicionar valores com o método `add()`:

```
C.add(1)
```

```
C.add(2)
```

```
C.add(8)
```

- Ao exibirmos o resultado, teremos:

{8, 1, 2}

Mesmo o 8 sendo o último elemento a ser inserido no conjunto ele aparece na primeira posição ;

Conjuntos - adicionando

- Adicionando mais valores:

```
C.add(-1)
```

```
C.add(0)
```

- Ao exibirmos o resultado, teremos:
{0, 1, 2, 8, -1}

Conjuntos

- Podemos utilizar o operador in para pesquisar em um conjunto:

```
print(1 in C) resultado será TRUE
```

- Um set(conjunto) pode ser criado a partir de listas, tuplas e qualquer outra estrutura que seja enumerável.

```
b = set([2, 3, 6, 12])
```

Conjuntos - operações

```
b = {2, 3, 6, 12}
```

```
a = {1, 2, 6, 28}
```

Considerando os dois conjuntos **a** e **b**

```
print(a - b) resulta em: {1, 28}
```

```
print(a | b) resulta em: {1, 2, 3, 6, 12, 28}
```

```
print(a & b) resulta em: {2, 6}
```

- representa a diferença entre conjuntos;
- | representa a união entre conjuntos; e
- & representa a intersecção entre conjuntos;

Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de.. Algoritmos: lógica para desenvolvimento de programação de computadores.. 27. ed.. Érica. 2014

Referências Complementares

DOWNEY, Allen B. **Pense em Python**. 2ª Ed. Novatec. 2016

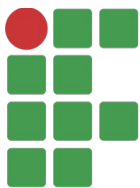
MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019

CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>



INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 4 (Dicionários)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

rafael.speroni@ifc.edu.br

Objetivo:



- Compreender o conceito de Dicionários.

Listas e Dicionários

- Em **Python** existem três tipos principais de variáveis compostas: Listas, Tuplas e **Dicionários**.

Listas e Dicionários

Vamos aos conceitos:

- **Dicionários** são uma estrutura de dados similar às listas, mas com propriedades de acesso diferentes. Cada dicionário é composto por um conjunto de chaves e valores.
- O **dicionário consiste** em relacionar **uma chave a um valor**.

Como imaginar um dicionário:

Preços de mercadorias

Produto	Preço
Alface	0,99
Batata	1,45
Tomate	4,99
Feijão	3,50

- A tabela exibida pode ser vista como um dicionário, em que **chave** seria o **produto** e o **valor** seu **preço**;

Dicionários

- Em Python, criamos dicionários utilizando chaves({ });
- Como ficaria no Python:

```
tabela = {"Alface": 0.99,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

- Um dicionário é acessado por suas chaves, exemplo:

```
print(tabela["Alface"])
```

irá exibir 0.99;

ele diferencia letras maiúsculas de minúsculas, ou seja, é case sensitive

Diferente das listas em que o índice é um número
Dicionários utilizam suas chaves como índice.

Como declarar um dicionário?

```
D = {}
```

- Este comando cria um dicionário denominado de **D** e as chaves ({ }) após o símbolo de igualdade indicam que é um dicionário vazio.

```
tabela = {"Alface": 0.99,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

- Este comando cria um dicionário denominada de **tabela** e este dicionário contém 4 elementos.

Como acessar um elemento de um dicionário?

```
tabela = {"Alface": 0.99,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

- Se acessarmos `tabela["Feijão"]` o resultado que será exibido será 3.50, pois estamos acessando a última chave do dicionário, ou seja, o valor que está relacionado a chave "Feijão".
- `tabela["Alface"]` será igual a 0.99 e `tabela["Batata"]` será igual a 1.45.

Como alterar um elemento de um dicionário?

```
tabela = {"Alface": 0.99,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

- Se atribuirmos `tabela["Alface"] = 1.50` o conteúdo de `tabela["Alface"]` será atualizado. E ficaremos com:

```
tabela = {"Alface": 1.50,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

O valor anterior será perdido

Como alterar um elemento de um dicionário?

```
tabela = {"Alface": 0.99,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

- Se atribuirmos `tabela["Cenoura"] = 2.10` o conteúdo de `tabela` será atualizado. E ficaremos com:

```
tabela = {"Alface": 1.50,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50,  
          "Cenoura": 2.10}
```

Encontrando um elemento de um dicionário?

```
tabela = {"Alface": 0.99,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

- Se buscarmos por uma chave que não existe no dicionário, isso nos retornará um erro, por exemplo:

```
print(tabela["Manga"])
```

Esse comando irá gerar um erro:

```
print(tabela["Manga"])  
KeyError: 'Manga'
```

Encontrando um elemento de um dicionário?

```
tabela = {"Alface": 0.99,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

- Para sabermos se a chave existe no dicionário podemos utilizar o comando *in*:

```
print("Manga" in tabela) resposta False;  
print("Batata" in tabela) resposta True;
```

Obtendo as chaves e os valores de um dicionário?

```
tabela = {"Alface": 0.99,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

- Podemos utilizar os métodos `keys()` e `values()`, para sabermos as chaves existentes no dicionário, ou os valores:

```
print(tabela.keys())
```

no terminal:

```
dict_keys(['Alface', 'Batata', 'Tomate', 'Feijão'])
```

```
print(tabela.values())
```

no terminal:

```
dict_values([0.99, 1.45, 4.99, 3.5])
```

Removendo elementos de um dicionário

```
tabela = {"Alface": 0.99,  
          "Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

- Para removermos um elemento do dicionário podemos utilizar a instrução **del**;

```
del tabela["Alface"]
```

- irá ficar:

```
tabela = {"Batata": 1.45,  
          "Tomate": 4.99,  
          "Feijão": 3.50}
```

Como ler e mostrar um dicionário?

```
# cálculo de ocorrências de nomes
```

```
alunos = ['Joao', 'Pedro', 'Lucas', 'Pedro', 'Ana']
```

```
dic = {}
```

```
for p in alunos:
```

```
    if p in dic:
```

```
        dic[p] += 1
```

```
    else:
```

```
        dic[p] = 1
```

```
print(dic)
```

lista com o nome de
estudantes

dicionário vazio

irá percorrer um a um dos
elementos da lista

verifica se cada nome já está no
dicionário.

Resultado: {'Joao': 1, 'Pedro': 2, 'Lucas': 1, 'Ana': 1}

Resumo

	Listas	Tuplas	Dicionários	Conjuntos
Ordem dos elementos	Fixa	Fixa	Mantida a partir do Python 3.7	Indeterminada
Tamanho	Variável	Fixa	Variável	Variável
Elementos repetidos	Sim	Sim	Pode repetir valores, mas as chaves devem ser únicas	Não
Pesquisa	Sequencial, índice numérico	Sequencial, índice numérico	Direta por chave	Direta por valor
Alterações	Sim	Não	Sim	Sim
Uso primário	Sequências	Sequências Constantes	Dados indexados por chave	Verificação de unicidade, operações com conjuntos

Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de.. Algoritmos: lógica para desenvolvimento de programação de computadores.. 27. ed.. Érica. 2014

Referências Complementares

DOWNEY, Allen B. **Pense em Python**. 2ª Ed. Novatec. 2016

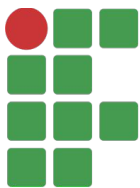
MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019

CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>



INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 5 (Matrizes)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

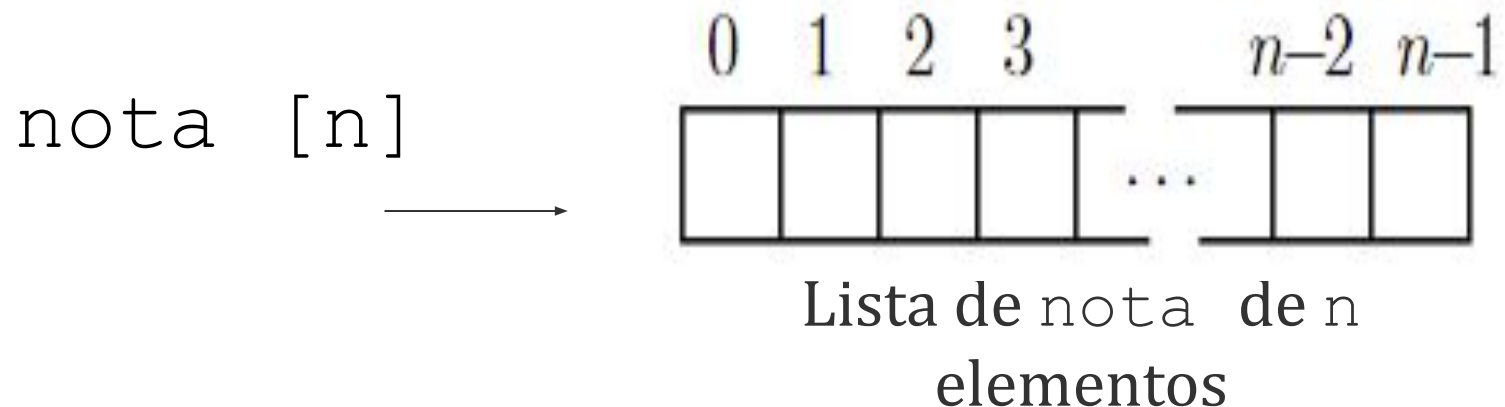
rafael.speroni@ifc.edu.br

Objetivo

- Compreender o conceito da **estrutura de dados** conhecida como **matrizes** (em Python).
 - declaração
 - inicialização
 - atribuição
 - acesso ao conteúdo

Como representar uma matriz?

- Array **unidimensional**
 - Uma **listas** identificada por **nota** de **n** posições:



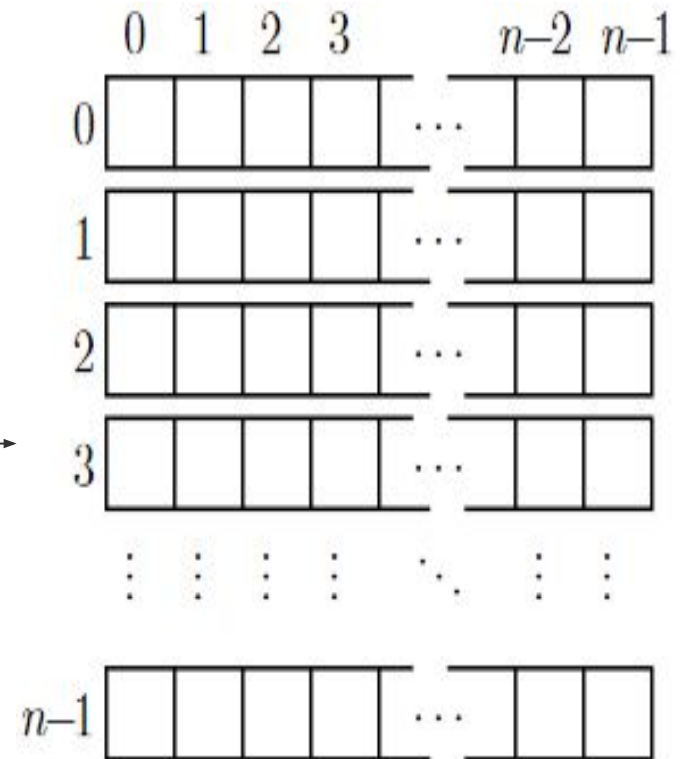
Como representar uma matriz?

- **Listas de Listas** é uma estrutura **bidimensional**, ou seja **matriz**
 - Uma lista identificada por **nota** de **$n \times n$** posições:

nota [n] [n]

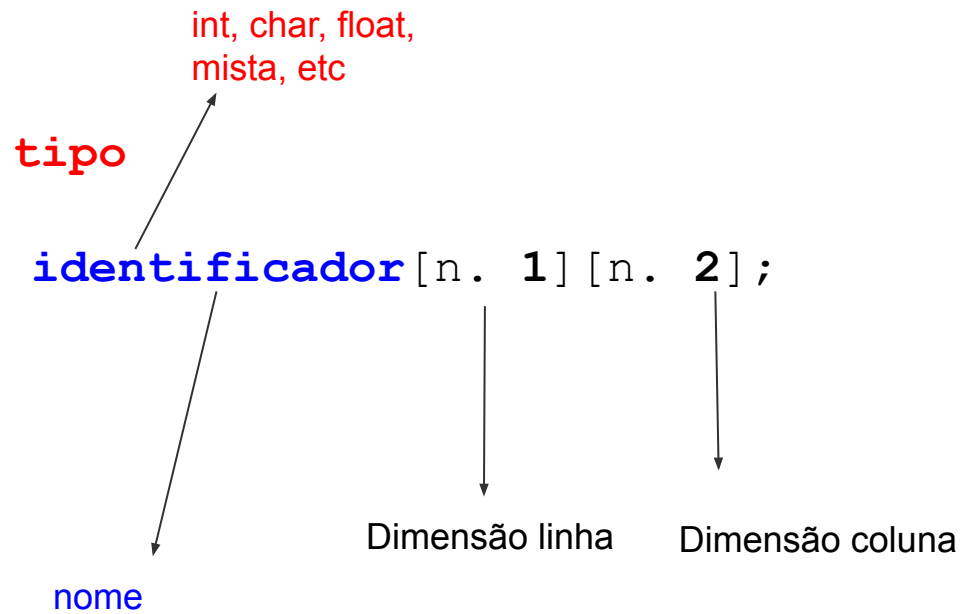


Lista de listas
nota de $n \times n$
elementos



Matrizes

- Em Python:



Exemplos

Matriz

	0	1	2	3	4	5
X 0						
1					5	



Acessando elementos
em Python

`x[1][4]`

Declarando listas em
Python

	0	1	2
0			
1			
2			
3			



`MAT = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]`

Exemplos

Matriz

		1	2	3	4	5
X	1	12	9	3	7	-23
	2	15	4	2	34	-4
	3	3	45	3	0	-3

→

```
X = [[12, 9, 3, 7, -23],  
     [15, 4, 2, 34, -4],  
     [3, 45, 3, 0, -3]]
```

		0	1	2
MAT	0			
	1			
	2			
	3			D

→

Acessando elementos
em Python

```
MAT[3][2]
```

Exemplos

Acessando elementos em Python

```
X[3][4] = 0;  
X[2][1] = -5;  
X[4][4] = 1;
```



	0	1	2	3	4
0					
1					
2		-5			
3					0
4					1

X

Matriz

```
uma_lista = [3, 67, "gato",  
             [56, 57, "cachorro"], [ ],  
             3.14, False]
```

```
uma_lista[5] = 3.14  
uma_lista[3][0] = 56  
uma_lista[3][2][0] = c  
uma_lista[2][0] = g
```


Como declarar uma matriz (lista de listas)?

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
frutas = ["maca", "pera", "jaca", "caju"]
```

	frutas			
	0	1	2	3
0	m	a	c	a
1	p	e	r	a
2	j	a	c	a
3	c	a	j	u

Podemos utilizar numpy?

```
import numpy as np
```

```
M1 = np.array([[1,2],[3,4]])
```

```
print(M1)
```

```
[[1 2]
 [3 4]]
```

Sim! Desde que os dados a serem armazenados sejam apenas numéricos.

Como preencher uma matriz

```
import numpy as np

# define o tamanho do array
N = 3

#preenche a estrutura com zeros
matriz = np.zeros((N,N))

#mostra a matriz
print(matriz)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
```

Preencher uma matriz usando listas

```
matriz = [[0, 0, 0],[0, 0, 0], [0, 0, 0]]

# Inserindo dados na matriz usando for.
for l in range(len(matriz)):
    for c in range(len(matriz[l])):
        matriz[l][c] = int(input("Informe valores
        para a matriz"))

print("Matriz 3x3: ", matriz)
```

Preencher uma matriz

```
l = 0
# Inserindo dados na matriz usando while.
matriz = [[0, 0, 0],[0, 0, 0], [0, 0, 0]]
while l < len(matriz):
    c = 0
    while c < len(matriz[l]):
        matriz[l][c]= int(input("Informe valores para a
matriz"))
        c +=1
    print()
    l += 1

print("Matriz 3x3: ", matriz)
```

Preencher uma matriz

```
matriz = []

# Inserindo dados na matriz usando for através da
linha.
for l in range(3):
    lista = []
    for c in range(3):
        lista.append(int(input("Informe valores
        para a matriz")))
    matriz.append(lista)

print("Matriz 3x3: ", matriz)
```

Exibir uma matriz

```
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
print("Matriz 3x3: ", matriz)
```

```
Matriz 3x3: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
# Exibindo a matriz com for.
```

```
print("Matriz impressa de outra forma:")
```

```
for lista in matriz:
    for elemento in lista:
        print(elemento, end=' ')
    print()
```

```
1 2 3
4 5 6
7 8 9
```

```
l = 0
```

```
# Exibindo a matriz com while.
```

```
print("Matriz impressa de outra forma:")
```

```
while l < len(matriz):
    c = 0
    while c < len(matriz[l]):
        print(matriz[l][c], end=' ')
        c += 1
    print()
    l += 1
```

```
1 2 3
4 5 6
7 8 9
```

Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de.. Algoritmos: lógica para desenvolvimento de programação de computadores.. 27. ed.. Érica. 2014

Referências Complementares

DOWNEY, Allen B. **Pense em Python**. 2ª Ed. Novatec. 2016

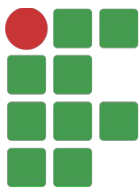
MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019

CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>



INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 6

(Subalgoritmos - parte I)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

rafael.speroni@ifc.edu.br

O que são subalgoritmos

- Sempre é possível dividir problemas grandes em problemas menores, e de menor complexidade de resolução.
- É um algoritmo que resolve parte de um problema.
- Nesse caso, o algoritmo completo é dividido num algoritmo principal e em diversos subalgoritmo (tantos quantos forem necessários).
- Geralmente está subordinado a um outro algoritmo.

Vantagens de subalgoritmos

- Diminui a complexidade do problema.
- Permite focalizar a atenção em um problema pequeno de cada vez, o que ao final produzirá uma melhor compreensão do todo.
- Reusabilidade

Definição de subalgoritmos

- Cabeçalho, onde estão definidos o *nome e o tipo do subalgoritmo*, bem como os seus *parâmetros*;
- Corpo do subalgoritmo, onde se encontram as instruções, que serão executadas cada vez que ele é chamado.

Tipos de Subalgoritmos (Conceitos):

- As **funções** retornam um, e somente um valor ao algoritmo chamador;
- Os **procedimentos**, que retornam vários valores, ou nenhum, ao algoritmo chamador.

Forma geral de um procedimento

def NOME(PARÂMETROS):
 COMANDOS

```
def soma(a,b):  
    print(a+b)
```

Forma geral de uma função

```
def NOME( PARÂMETROS ):
    COMANDOS
    return ●
```

```
def soma(a,b):
    return a+b
```

Chamada de um subalgoritmo

```
def soma(a,b):  
    print(a+b)
```

```
soma(2,9)
```

```
soma(7,8)
```

```
soma(10,15)
```

11

15

25

Chamada de um subalgoritmo

```
def soma(a,b):  
    return a+b  
  
print(soma(2,9))  
  
print(soma(7,8))  
  
print(soma(10,15))
```

11
15
25

Chamada de um subalgoritmo

```
def e_impar(x):  
    return x % 2 == 1  
  
def par_ou_impar(x):  
    if e_impar(x):  
        return "Impar"  
    else:  
        return "Par"  
  
print(par_ou_impar(4))  
print(par_ou_impar(5))
```

```
Par  
Impar
```

Exercícios para praticar

1. Escreva uma função que receba dois números e retorne o maior.
Maximo(5, 6) ☐ retorno 6
Maximo(2, 1) ☐ retorno 2
2. Escreva uma função que receba um número e retorne **TRUE** se o número for par.
epar(6) ☐ retorno True
epar(1) ☐ retorno False
3. Escreva uma função que receba o lado de um quadrado e retorne a sua área ($A = \text{lado}^2$).
Area_quadrado(4) ☐ retorno 16
Area_quadrado(9) ☐ retorno 81
4. Escreva uma função que receba a base e a altura de um triângulo e retorne a sua área ($A = (\text{base} \times \text{altura})/2$).
Area_triangulo(6, 9) ☐ retorno 27
Area_triangulo(5, 8) ☐ retorno 20

Próximos Assuntos

- Variáveis Globais e Locais
- Passagem de parâmetros
- Recursividade

Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação: a construção de algoritmos e estruturas de dados**. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de. **Algoritmos: lógica para desenvolvimento de programação de computadores**. 27. ed.. Érica. 2014

Referências Complementares

MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019.

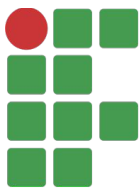
CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>

<https://panda.ime.usp.br/pensepy/static/pensepy/05-Funcoes/funcoes.html>



INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 7 (Subalgoritmos - parte II)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

rafael.speroni@ifc.edu.br

Para fixar:

Duas regras:

- Uma função deverá resolver apenas 1(um) problema;
- Quanto mais genérica for a sua solução, melhor ela será a longo prazo.

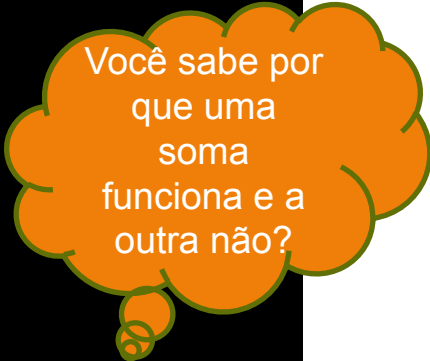
Dica: Se a função faz isso “e” aquilo já é um indicativo que talvez tenha que ser desmembrada.

Atenção as funções

```
# como não escrever uma função
```

```
def soma(L):  
    total = 0  
    x = 0  
    while x < 5:  
        total += L[x]  
        x += 1  
    return total
```

```
L = [1, 7, 2, 9, 15]  
print(soma(L))  
print(soma([7, 9, 12, 3, 100, 20, 4]))
```



Você sabe por
que uma
soma
funciona e a
outra não?

Atenção as funções

```
# soma dos valores de uma lista
```

```
def soma(L):
```

```
    total = 0
```

```
    x = 0
```

```
    while x < len(L):
```

```
        total += L[x]
```

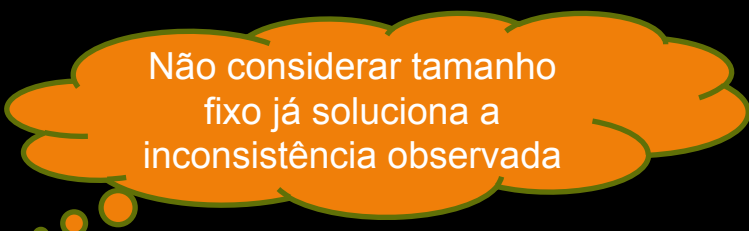
```
        x += 1
```

```
    return total
```

```
L = [1, 7, 2, 9, 15]
```

```
print(soma(L))
```

```
print(soma([7, 9, 12, 3, 100, 20, 4]))
```



Não considerar tamanho
fixo já soluciona a
inconsistência observada

Atenção às funções

Um problema clássico da programação é o cálculo do fatorial de um número:

- **Fatorial de 3** = $3 \times 2 \times 1$
= 6
- **Fatorial de 4** = $4 \times 3 \times 2$
 $\times 1$ = 24
- **Fatorial de 0** = 1

```
#calcula o fatorial
```

```
def fatorial(n):  
    fat = 1  
    while n > 1:  
        fat *= n  
        n -= 1  
    return fat
```

```
print(fatorial(0))  
print(fatorial(4))
```

Atenção às funções

Um problema clássico da programação é o cálculo do fatorial de um número:

- **Fatorial de 3** = $3 \times 2 \times 1$
= 6
- **Fatorial de 4** = $4 \times 3 \times 2$
 $\times 1$ = 24
- **Fatorial de 0** = 1

```
# calculo do fatorial  
#(outra forma)
```

```
def fatorial(n):  
    fat = 1  
    x = 1  
    while x <= n:  
        fat *= x  
        x += 1  
    return fat
```

```
print(fatorial(0))  
print(fatorial(4))
```

Variáveis locais

São variáveis **internas** a função, ou seja esta variável somente existirá enquanto a função está executando.

Não podemos acessar o seu valor fora da função que a criou e por isso passamos por parâmetros que retornam valores através das funções, de forma a viabilizar a troca de dados.

```
# calculo do fatorial  
#(outra forma)
```

Variável
local

```
def fatorial(n):
```

```
    fat = 1
```

```
    x = 1
```

```
    while x <= n:
```

```
        fat *= x
```

```
        x += 1
```

```
    return fat
```

```
print(fatorial(0))
```

```
print(fatorial(4))
```

Variáveis Globais

São variáveis **externas as funções**, que todos podem ter acesso.

```
#exemplo variável global
```

Variável Global

```
EMPRESA = 'Unidos Venceremos LTDA'
```

```
def imprime_cabecalho():
```

```
    print(EMPRESA)
```

```
    print("-" * len(EMPRESA))
```

```
imprime_cabecalho()
```

Unidos Venceremos LTDA

Variáveis Globais e Locais

Exemplos:

```
a = 5
def muda_e_imprime():
    a = 7
    print(f"A de dentro da função:{a}")
print(f"A antes de mudar:{a}")
muda_e_imprime()
print(f"A depois de mudar:{a}")
```

Variável Local

Resultado:

```
A antes de mudar:5
A de dentro da função:7
A depois de mudar:5
```

Variáveis Globais e Locais

Se quisermos modificar uma variável global dentro de uma função, devemos informar que estamos utilizando a variável global.

```
a = 5
def muda_e_imprime():
    global a
    a = 7
    print(f"A de dentro da função:{a}")
print(f"A antes de mudar:{a}")
muda_e_imprime()
print(f"A depois de mudar:{a}")
```

Variável global

Resultado:

```
A antes de mudar:5
A de dentro da função:7
A depois de mudar:7
```

Recursividade

- Uma função que chama a si mesma. Quando isso ocorre, temos uma função recursiva. Para isso utilizaremos o exemplo do fatorial;

$$\text{Fatorial}(n) = \begin{cases} 1 & 0 \leq n \leq 1 \\ n \times \text{fatorial}(n-1) & \end{cases}$$

Recursividade

```
#Função recursiva do fatorial
```

```
def fatorial(n):  
    if n == 0 or n == 1:  
        return 1  
    else:  
        return n * fatorial(n-1)
```

```
a = int(input('informe um valor para obter o fatorial: '))  
print("O resultado do fatorial é:",fatorial(a))
```

Recursividade - Exemplo

- Fatorial de 0
- Fatorial de 3

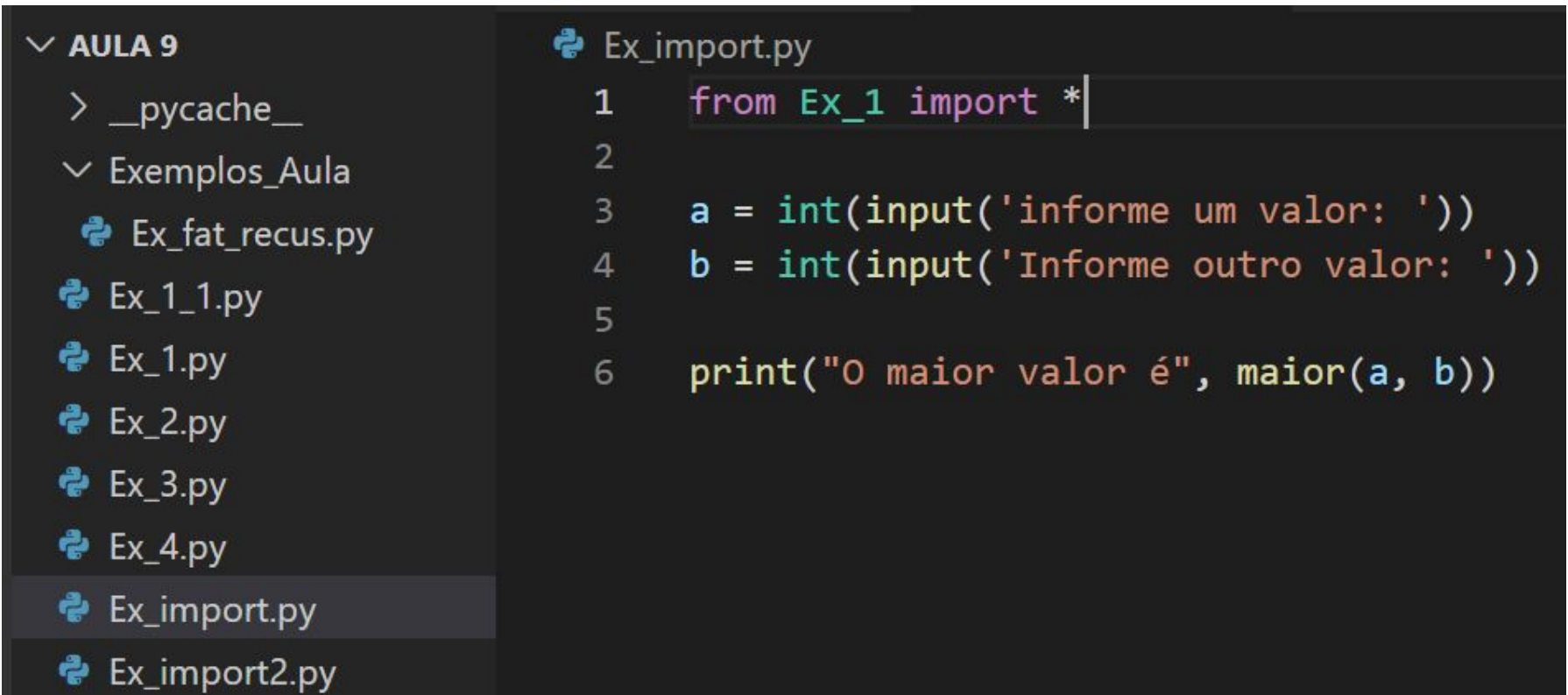
```
#Função recursiva do fatorial

def fatorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * fatorial(n-1)

a = int(input('informe um valor para
obter o fatorial: '))
print("O resultado do fatorial
é:",fatorial(a))
```

Reutilização de funções

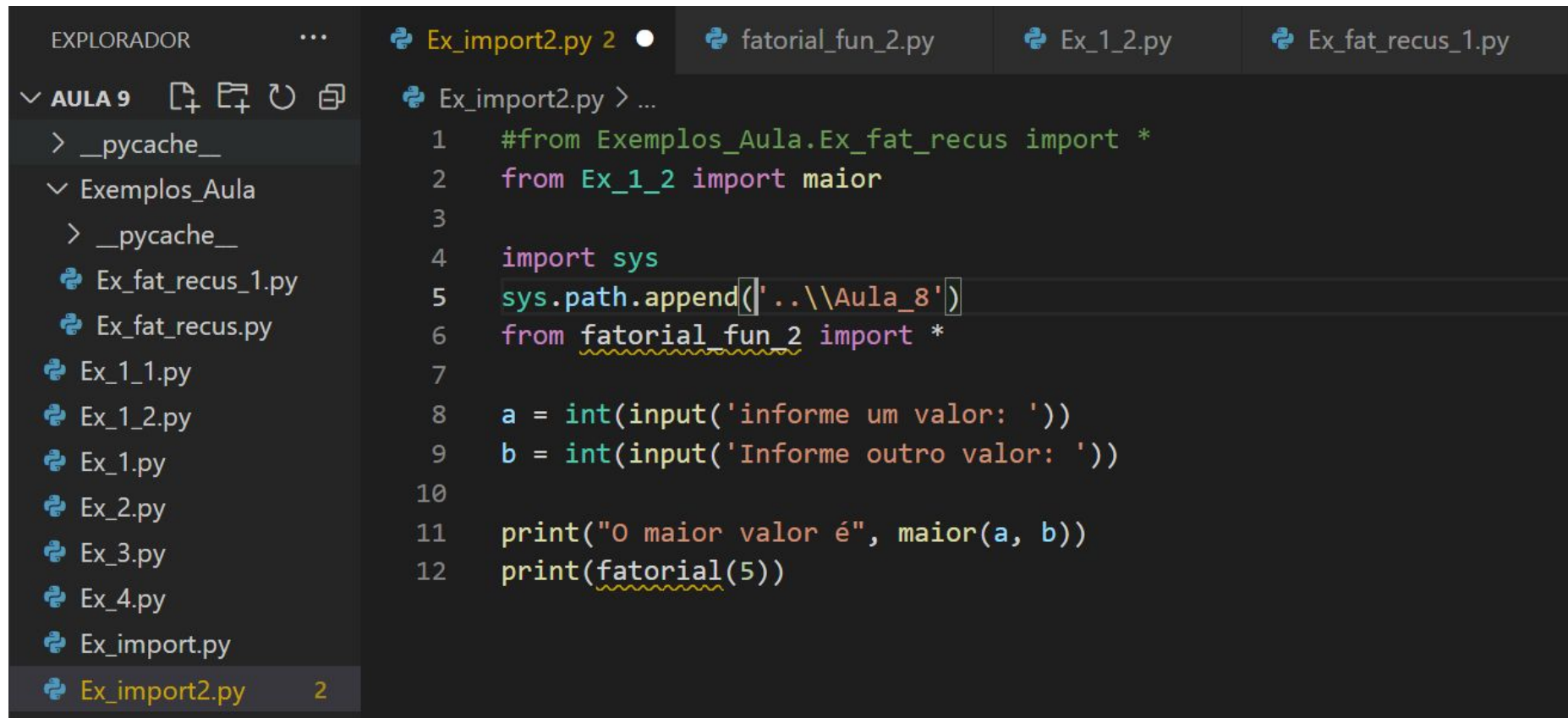
- Na mesma pasta:



```
Ex_import.py
1  from Ex_1 import *
2
3  a = int(input('informe um valor: '))
4  b = int(input('Informe outro valor: '))
5
6  print("O maior valor é", maior(a, b))
```

Reutilização de funções

- Importando apenas uma função

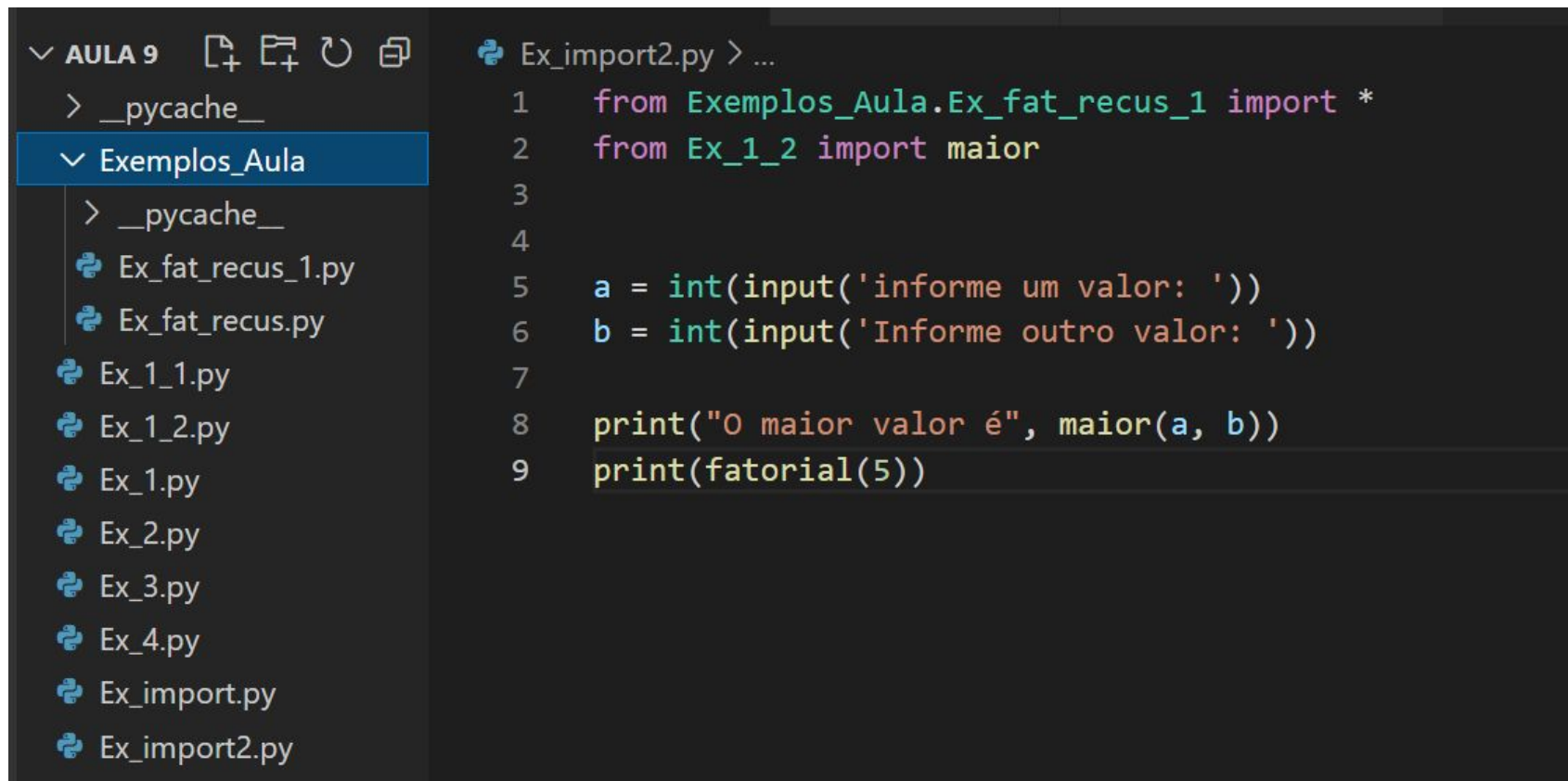


The screenshot shows a code editor with a sidebar on the left labeled 'EXPLORADOR' (Explorer) and a main editor area on the right. The sidebar shows a file tree for 'AULA 9' with subfolders '__pycache__' and 'Exemplos_Aula'. Under 'Exemplos_Aula', there are several Python files: '__pycache__', 'Ex_fat_recus_1.py', 'Ex_fat_recus.py', 'Ex_1_1.py', 'Ex_1_2.py', 'Ex_1.py', 'Ex_2.py', 'Ex_3.py', 'Ex_4.py', 'Ex_import.py', and 'Ex_import2.py' (which is selected and highlighted). The main editor area shows the code for 'Ex_import2.py' with the following content:

```
1  #from Exemplos_Aula.Ex_fat_recus import *
2  from Ex_1_2 import maior
3
4  import sys
5  sys.path.append(['..\Aula_8'])
6  from fatorial_fun_2 import *
7
8  a = int(input('informe um valor: '))
9  b = int(input('Informe outro valor: '))
10
11  print("O maior valor é", maior(a, b))
12  print(fatorial(5))
```

Reutilização de funções

- Em outra pasta, mas no mesmo projeto



```

AULA 9
├── __pycache__
└── Exemplos_Aula
    ├── __pycache__
    ├── Ex_fat_recus_1.py
    ├── Ex_fat_recus.py
    ├── Ex_1_1.py
    ├── Ex_1_2.py
    ├── Ex_1.py
    ├── Ex_2.py
    ├── Ex_3.py
    ├── Ex_4.py
    ├── Ex_import.py
    └── Ex_import2.py

Ex_import2.py > ...
1  from Exemplos_Aula.Ex_fat_recus_1 import *
2  from Ex_1_2 import maior
3
4
5  a = int(input('informe um valor: '))
6  b = int(input('Informe outro valor: '))
7
8  print("O maior valor é", maior(a, b))
9  print(fatorial(5))

```

Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação: a construção de algoritmos e estruturas de dados**. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de. **Algoritmos: lógica para desenvolvimento de programação de computadores**. 27. ed.. Érica. 2014

Referências Complementares

MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019.

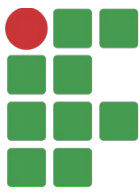
CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>

<https://panda.ime.usp.br/pensepy/static/pensepy/05-Funcoes/funcoes.html>



INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 8

(Subalgoritmos - parte III)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

rafael.speroni@ifc.edu.br

Passagem de Parâmetros:

O tipo de passagem de parâmetro define se as modificações realizadas nos parâmetros **dentro** da função irão ou não se refletir **fora** da função.

- ❑ Por valor (cópia);

Não afeta o algoritmo “que fez a chamada”.

- ❑ Por referência(endereço);

Afeta o dado do algoritmo principal.

Passagem de Parâmetro por valor

```
Exemplo_1.py > ...  
1  def teste(v1):  
2      #comandos na função  
3      print(v1)  
4      v1 = v1 + 2  
5      print(v1)  
6  
7  v = 10  
8  teste(v)  
9  
10 print(v)
```

10
12
10

Passagem de Parâmetro por referência

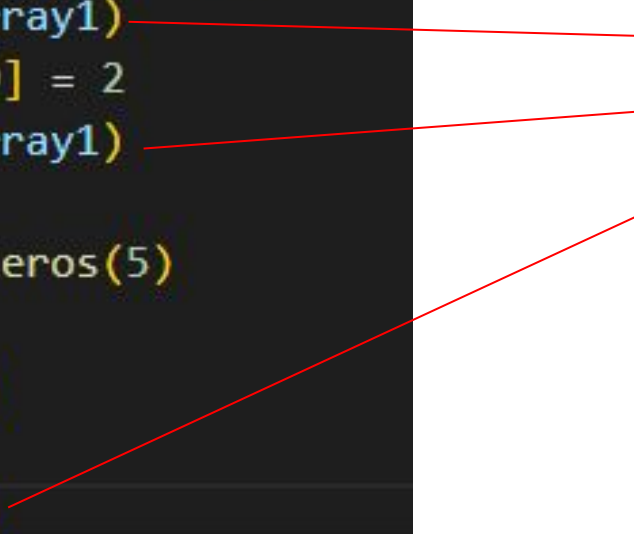
```
Exemplo_2.py > ...  
1  def teste(lista1):  
2      #comandos na função  
3      print(lista1)  
4      lista1.append(2)  
5      print(lista1)  
6  
7  lista = [10,11,12]  
8  |  
9  teste(lista)  
10  
11 print(lista)
```

[10, 11, 12]
[10, 11, 12, 2]
[10, 11, 12, 2]

Passagem de Parâmetro por referência

- O mesmo ocorre com arrays;

```
Exemplo3.py > ...  
1  import numpy as np  
2  
3  def teste(array1):  
4      #comandos na função  
5      print(array1)  
6      array1[0] = 2  
7      print(array1)  
8  
9  array = np.zeros(5)  
10  
11  teste(array)  
12  
13  print(array)
```



[0. 0. 0. 0. 0.]
[2. 0. 0. 0. 0.]
[2. 0. 0. 0. 0.]

Exercícios

1. Faça uma sub-rotina que recebe duas listas A e B com dez elementos inteiros como parâmetro. A sub-rotina deverá determinar e mostrar o vetor C que contém os elementos que estão em A, mas que não estão em B. A lista C deverá ser mostrada no programa principal.
2. Faça uma sub-rotina que receba como parâmetro uma matriz (lista de lista) A[5][5] e retorne a soma de todos os seus elementos.
3. Crie uma sub-rotina que receba como parâmetro um vetor V[25] de números inteiros e substitua todos os valores negativos de A por 0. O vetor deverá ser mostrado no programa principal.

Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação: a construção de algoritmos e estruturas de dados**. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de. **Algoritmos: lógica para desenvolvimento de programação de computadores**. 27. ed.. Érica. 2014

Referências Complementares

MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019.

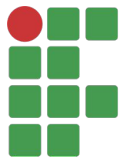
CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>

<https://panda.ime.usp.br/pensepy/static/pensepy/05-Funcoes/funcoes.html>



INSTITUTO FEDERAL

Catarinense

Campus Camboriú

AULA 9 (Arquivos)

Professora: Lidiane Visintin

lidiane.visintin@ifc.edu.br

Professor: Rafael de Moura Speroni

rafael.speroni@ifc.edu.br

Objetivo

- Compreender o conceito de **arquivos**.
 - Declaração
 - Abrindo arquivos
 - Funções para escrita e leitura

Arquivos em Python

- Arquivos podem representar diversas coisas.
 - Será trabalhado apenas **arquivos em disco**.
- Manipulação de arquivos.

Binário vs texto

Tipo	Vantagens	Desvantagens
Texto	<i>- Facilidade de leitura</i>	<i>- Maior gasto de memória</i> <i>- Maior gasto de tempo em buscas</i>
Binário	<i>- Menor gasto de memória</i> <i>- Menor gasto de tempo em buscas</i>	<i>Dificuldade de leitura</i>

Manipulando arquivos

Função	Descrição
<code>open()</code>	- Abrir um arquivo.
<code>close()</code>	- Fechar um arquivo.
	- Declara uma variável para um arquivo. Ao chamar a função <code>open()</code> , ela cria uma instância da estrutura <code>FILE</code> e retorna um objeto para ela.

Como abrir arquivos?

`File = open(<nome arquivo>, <modo>);`

modo de abertura do arquivo (texto, binário, para leitura, escrita ou ambos).

nome do arquivo a ser aberto (com ou sem o caminho):

Windows:

```
char *arquivo = "c:\\dados\\lidiane\\info.txt";
```

UNIX:

```
char *arquivo = "c:/dados/\\/lidiane/info.txt";
```

<modo>

<i>Modo</i>	<i>Significado</i>	<i>Se o arquivo NÃO existe...</i>	<i>Se o arquivo existe...</i>
r	Abre o arquivo para leitura.	O open() retorna Erro.	
w	Abre o arquivo para escrita.	O arquivo é criado.	O arquivo é <u>apagado</u> sem qualquer aviso e um novo arquivo vazio é criado em seu lugar.
a	Abre o arquivo para adicionar novos caracteres.	O arquivo é criado.	O arquivo é aberto para adição de caracteres no fim.

<modo>

<i>Modo</i>	<i>Significado</i>	<i>Se o arquivo NÃO existe...</i>	<i>Se o arquivo existe...</i>
r+	Abre o arquivo para leitura e escrita.	O open() retorna Erro.	O arquivo é aberto para adição de caracteres no início, sobrescrevendo caracteres já existentes.
w+	Abre o arquivo para leitura e escrita.	O arquivo é criado.	O arquivo é aberto para adição de caracteres no início, sobrescrevendo o arquivo inteiro.
a+	Abre o arquivo para leitura e para adicionar novos caracteres.	O arquivo é criado.	O arquivo é aberto para adição de caracteres no fim.

Abrindo um arquivo e escrevendo:

```
arquivo = open ("aula.txt", "w")  
  
for linha in range(1,101):  
    arquivo.write(f"{linha}\n")  
  
arquivo.close
```

Abrindo um arquivo e lendo o seu conteúdo:

```
arquivo = open ("aula.txt", "r")  
  
for linha in arquivo.readlines():  
    print(linha)  
  
arquivo.close
```

Usando with:

```
with open ("aula.txt", "r") as arquivo:  
    for linha in arquivo.readlines():  
        print(linha)
```


Gerando arquivos:

```
#gravando números pares e impares em arquivos
with open ("impares.txt", "w") as impares:
    with open ("pares.txt", "w") as pares:
        for n in range(0, 1000):
            if n % 2 == 0:
                pares.write(f"{n}\n")
            else:
                impares.write(f"{n}\n")
```

Gerando arquivos:

```
#gravando numeros pares e impares em arquivos
with open ("impares.txt", "w") as impares:
    with open ("pares.txt", "w") as pares:
        for n in range(0, 1000):
            if n % 2 == 0:
                pares.write(f"{n}\n")
            else:
                impares.write(f"{n}\n")
```

Gerando arquivos:

```
#gravando numeros pares e impares em arquivos
with open ("impares.txt", "w") as impares, open
("pares.txt", "w") as pares:
    for n in range(0, 1000):
        if n % 2 == 0:
            pares.write(f"{n}\n")
        else:
            impares.write(f"{n}\n")
```

Leitura e escrita

```
#lendo de um arquivo e gravando em outro arquivo
with open ("multiplos_de_4.txt", "w") as multiplos4:
    with open ("pares.txt", "r") as pares:
        for l in pares.readlines():
            if int(l) % 4 == 0:
                multiplos4.write(f"{l}\n")
```

Processamento de um arquivo

Arquivo de entrada “entrada.txt”

;Esta linha não deve ser impressa

>Esta linha deve ser impressa a direita

*Esta linha deve ser centralizada

Uma linha normal

Outra linha normal

Processamento de um arquivo

```
#processamento de um arquivo
```

```
LARGURA = 100
```

```
with open("entrada.txt") as entrada:
    for linha in entrada.readlines():
        if linha[0] == ";":
            continue
        elif linha[0] == ">":
            print(linha[1:].rjust(LARGURA))
        elif linha[0] == "*":
            print(linha[1:].center(LARGURA))
        else:
            print(linha)
```

Processamento de um arquivo

Arquivo de entrada “entrada.txt”

;Esta linha não deve ser impressa

>Esta linha deve ser impressa a direita

*Esta linha deve ser centralizada

Uma linha norma

Outra linha normal

```
Esta linha deve ser impressa a direita
```

```
Esta linha deve ser centralizada
```

```
Uma linha norma
```

```
Outra linha normal
```

Referências



Referências Básicas

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação: a construção de algoritmos e estruturas de dados**. 3. ed. Pearson Prentice Hall. 2005

MANZANO, José Augusto N. G; OLIVEIRA, Jayr Figueiredo de. **Algoritmos: lógica para desenvolvimento de programação de computadores**. 27. ed.. Érica. 2014

Referências Complementares

MENEZES, Nilo Ney de Coutinho. **Introdução a programação com Python**. 3ª Ed. Novatec. 2019.

CORMEN, Thomas H et al. **Algoritmos: teoria e prática**. 2. ed. Elsevier, Campus,. 2002

Referências na Internet

<https://docs.python.org/3/>

<https://www.w3schools.com/python/default.asp>

<https://panda.ime.usp.br/pensepy/static/pensepy/10-Arquivos/files.html>