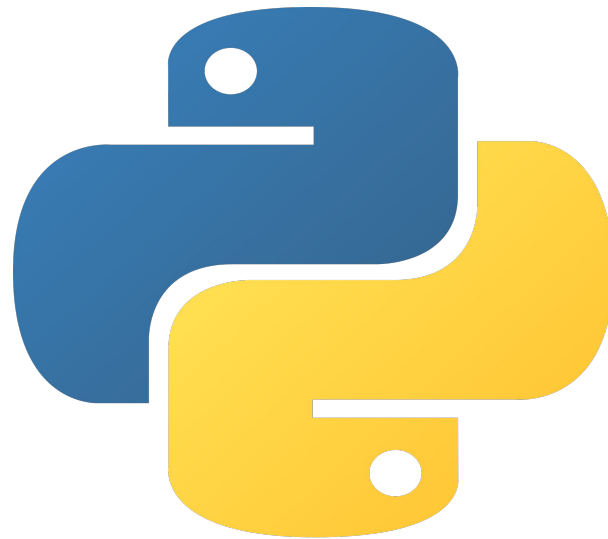


Python



Aula 3

Módulo 1 – Python e orientação a objetos

Professor: Márcio Barros



O que vamos aprender hoje?

- Estrutura de Dados



Estrutura de dados

Uma estrutura de dados é uma forma organizada de armazenar e gerenciar informações para que possamos acessá-las e manipulá-las de maneira eficiente. Pense nas estruturas de dados como recipientes que podem conter diferentes tipos de informações, como números, textos, listas de itens, etc.

Nessa aula trabalharemos com **Listas, Dicionários, Tuplas e Strings.**

Listas

Em Python, uma lista é uma estrutura de dados que permite armazenar múltiplos itens em uma única variável. Pense em uma lista como uma coleção ou uma sequência de elementos, que podem ser de qualquer tipo: números, strings, outros objetos e até mesmo outras listas.

Para criar uma lista, você coloca os itens entre colchetes [], separados por vírgulas. Por exemplo:

```
aula3.py > ...  
1  # Uma lista de números inteiros  
2  numeros = [1, 2, 3, 4, 5]  
3  
4  # Uma lista de strings  
5  frutas = ["maçã", "banana", "cereja"]  
6  
7  # Uma lista mista (números e strings)  
8  mista = [1, "maçã", 3.5, "banana"]
```

Para imprimir a lista inteira basta utilizar a função `print()`



Listas

Acessando apenas um elemento da lista

Os elementos de listas são **indexados**, ou seja, eles são acessados pelo índice (iniciando de zero).

frutas = ["maçã", "banana", "cereja"]

Índice 0

Índice 1

Índice 2



```
# Lista de frutas
frutas = ["maçã", "banana", "cereja"]

# Acessando o primeiro elemento (índice 0)
print(frutas[0]) # Saída: maçã

# Acessando o segundo elemento (índice 1)
print(frutas[1]) # Saída: banana

# Acessando o terceiro elemento (índice 2)
print(frutas[2]) # Saída: cereja
```

Listas

Modificando Elementos de uma Lista

Você pode alterar o valor de um elemento da lista, atribuindo um novo valor ao índice correspondente:

```
# Lista de frutas
frutas = ["maçã", "banana", "cereja"]

# Mudando o segundo elemento (índice 1)
frutas[1] = "laranja"

print(frutas) # Saída: ['maçã', 'laranja', 'cereja']
```



Listas

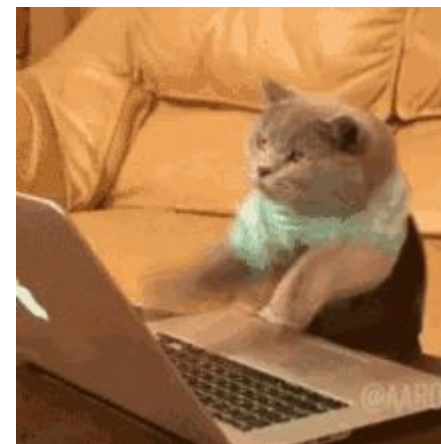
Listas aninhadas

Você pode criar uma lista dentro de outra lista:

```
# Lista de frutas
frutas = ["maçã", "banana", "cereja"]

# Mudando o segundo elemento (índice 1)
frutas[1] = "laranja"

print(frutas) # Saída: ['maçã', 'laranja', 'cereja']
```



Listas

Saber tamanho de uma lista

Você pode usar a função `print(len())` para saber o tamanho da lista

```
# Lista de números  
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
print(len(numeros)) #Saída= 10
```

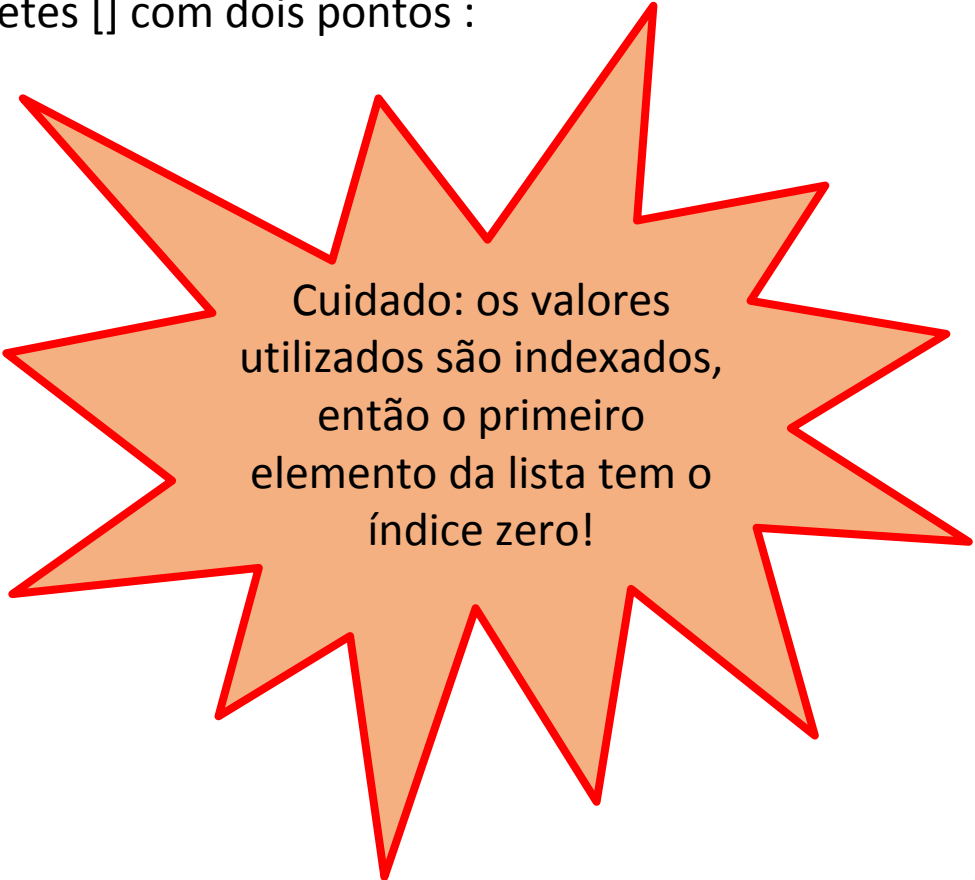
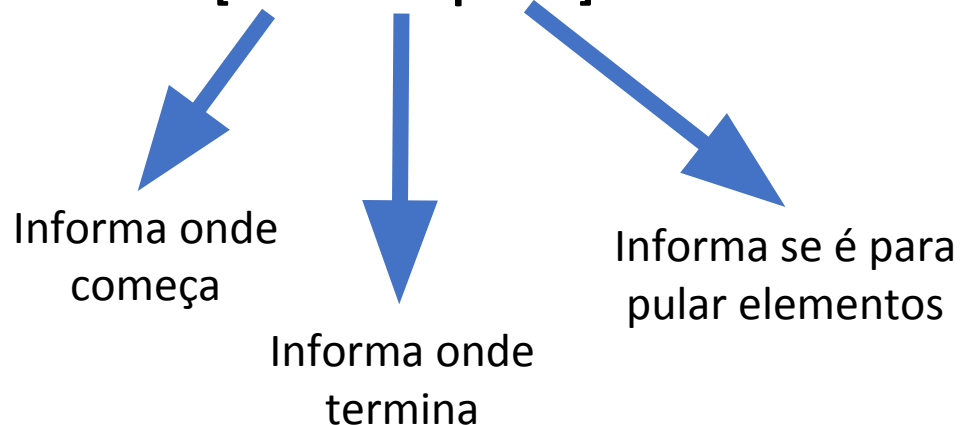


Listas

Fatiando listas

Fatiamento de listas é o processo de obter uma parte da lista original, criando uma nova lista contendo apenas os elementos desejados. Isso é feito usando a notação de colchetes [] com dois pontos :

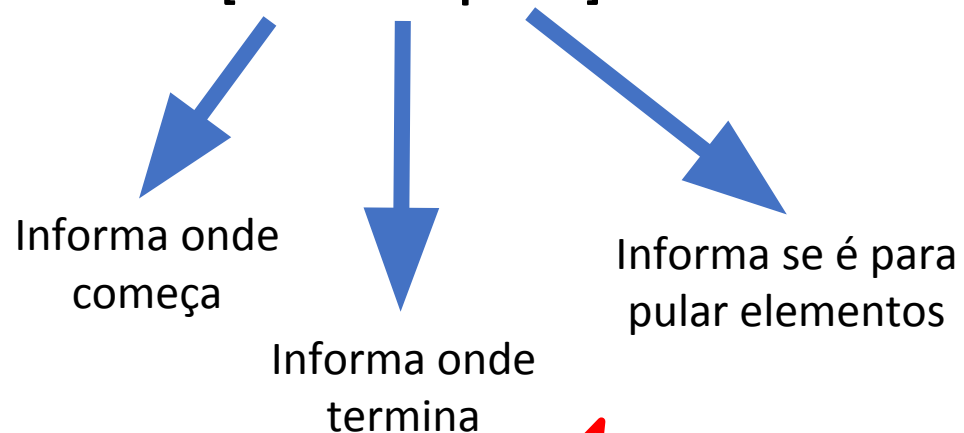
Sintaxe: **lista[início:fim:passo]**



Cuidado: os valores utilizados são indexados, então o primeiro elemento da lista tem o índice zero!

Listas

Sintaxe: **lista[início:fim:passo]**



Cuidado: os valores utilizados são indexados, então o primeiro elemento da lista tem o índice zero!

```
# Lista de números
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Fatiar do índice 2 ao 5 (elementos nos índices 2, 3 e 4)
sublista = numeros[2:5]
print(sublista) # Saída: [2, 3, 4]
```

Listas

Sintaxe: **lista[início:fim:passo]**

Informa onde
começa

Informa onde
termina

Informa se é para
pular elementos

Cuidado: os valores
utilizados são indexados,
então o primeiro elemento
da lista tem o índice zero!

```
# Fatiar do início ao índice 4 (elementos nos índices 0, 1, 2, 3)
sublista = numeros[:4]
print(sublista) # Saída: [0, 1, 2, 3]

# Fatiar do índice 5 até o final
sublista = numeros[5:]
print(sublista) # Saída: [5, 6, 7, 8, 9]

# Fatiar os últimos 3 elementos
sublista = numeros[-3:]
print(sublista) # Saída: [7, 8, 9]
```

Listas

Vamos treinar?

Passos:

- 1- Crie uma lista com 6 nomes diferentes (ex: animais = ["rato", "gato", "capivara", "mosca", "cobra", "baleia"]).
- 2- Imprima a lista inteira.
- 3- Imprima apenas o primeiro elemento da lista.
- 4- Imprima o último elemento da lista.
- 5- Imprima do primeiro ao terceiro elemento da lista.
- 6- Imprima a lista pulando um número (ex: rato, capivara, cobra).
- 7- Imprima a lista de trás pra frente.
- 8- Imprima a lista de trás pra frente pulando um número.
- 9- Imprima o tamanho da lista.
- 10- Altere o primeiro elemento da lista para um que não esteja nela (ex: trocar "rato" por "papagaio").

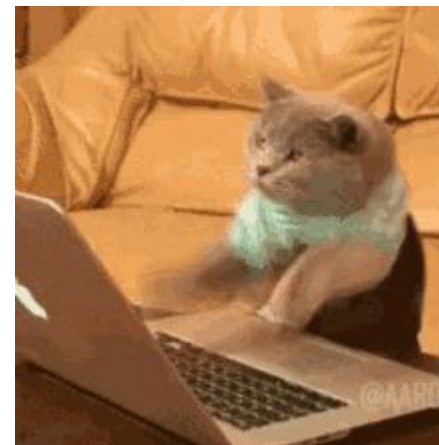


Listas

Você também pode criar uma lista vazia:

```
#Criando uma lista vazia  
lista= []
```

Mas como depois
adicionar elementos
nessa lista vazia???





Listas

Funções importantes em listas:

- **append()** - Adiciona um item ao final da lista.
- **insert()** - Insere um item em uma posição específica.
- **remove()** - Remove o primeiro item com o valor especificado.
- **pop()** - Remove o item na posição especificada (ou o último item se não for especificada a posição).
- **sort()** - Ordena a lista.
- **index()** – Pega o índice do elemento especificado.
- **reverse()** – Inverte a ordem da lista
- **count()** – Conta o número de ocorrências de um elemento.



Listas

Funções importantes em listas:

```
# Criando uma lista de frutas
frutas = ["maçã", "banana", "cereja"]

# Adicionando uma fruta ao final da lista
frutas.append("laranja")
print(frutas) # Saída: ['maçã', 'banana', 'cereja', 'laranja']

# Inserindo uma fruta na posição 1
frutas.insert(1, "kiwi")
print(frutas) # Saída: ['maçã', 'kiwi', 'banana', 'cereja', 'laranja']

# Removendo a primeira ocorrência de 'banana'
frutas.remove("banana")
print(frutas) # Saída: ['maçã', 'kiwi', 'cereja', 'laranja']

# Removendo o último elemento da lista
frutas.pop()
print(frutas) # Saída: ['maçã', 'kiwi', 'cereja']

# Ordenando a lista
frutas.sort()
print(frutas) # Saída: ['cereja', 'kiwi', 'maçã']
```



Listas

```
# Lista de frutas
frutas = ["maçã", "banana", "cereja", "banana", "laranja", "maçã"]

# 1. Encontrando o índice da primeira ocorrência de 'maçã'
indice_maca = frutas.index("maçã")
print(f"O índice da primeira 'maçã' é: {indice_maca}") # Saída: O índice da primeira 'maçã' é: 0

# 2. Contando quantas vezes 'banana' aparece na lista
quantidade_banana = frutas.count("banana")
print(f"'banana' aparece {quantidade_banana} vezes na lista.") # Saída: 'banana' aparece 2 vezes na lista.

# 3. Invertendo a ordem da lista
frutas.reverse()
print(f"A lista invertida é: {frutas}") # Saída: A lista invertida é: ['maçã', 'laranja', 'banana', 'cereja', 'banana', 'maçã']
```



Listas

Copiando uma lista

```
#Será que é assim que se copia uma lista?  
numeros=[0,1,2,3,4,5]  
copia=numeros  
print(numeros) #Saída [0, 1, 2, 3, 4, 5]  
print(copia) #Saída [0, 1, 2, 3, 4, 5]  
  
#Testando se é uma copia:  
copia[0]=200  
print(numeros) #Saída [200, 1, 2, 3, 4, 5] (mudou também a original!)  
print(copia) #Saída [200, 1, 2, 3, 4, 5]
```



Você não fez uma
cópia, só fez uma
referência a lista
original

Listas

Copiando uma lista

Para copiar de fato você pode usar **`copia = lista[::]`** ou **`copia = list(lista)`**

```
#Fazendo do jeito correto
numeros=[0,1,2,3,4,5]
copia1=numeros[::]
copia2 = list(numeros)

print(numeros, copia1, copia2) #Saída [0, 1, 2, 3, 4, 5] [0, 1, 2, 3, 4, 5] [0, 1, 2, 3, 4, 5]

#Conferindo se cada uma é independente
numeros[0]= 20
copia1[0]=50
copia2[0]= 90

print(numeros) #Saída [20, 1, 2, 3, 4, 5]
print(copia1) #Saída [50, 1, 2, 3, 4, 5]
print(copia2) #Saída [90, 1, 2, 3, 4, 5]
```





Prática sobre listas

1) O que será impresso na seguinte sequência de comandos?

```
a = [1, 2, 3]
```

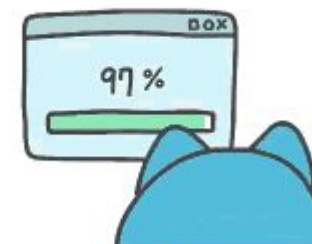
```
b = a[:]
```

```
b[0] = 5
```

```
print(a[3])
```

Prática sobre listas

```
Traceback (most recent call last):  
  File "c:\Users\marci\OneDrive\Área de Trabalho\FUCTURA\AULA 3\aula3.py", line 228, in <module>  
    print(a[3])  
        ~^^^  
IndexError: list index out of range
```



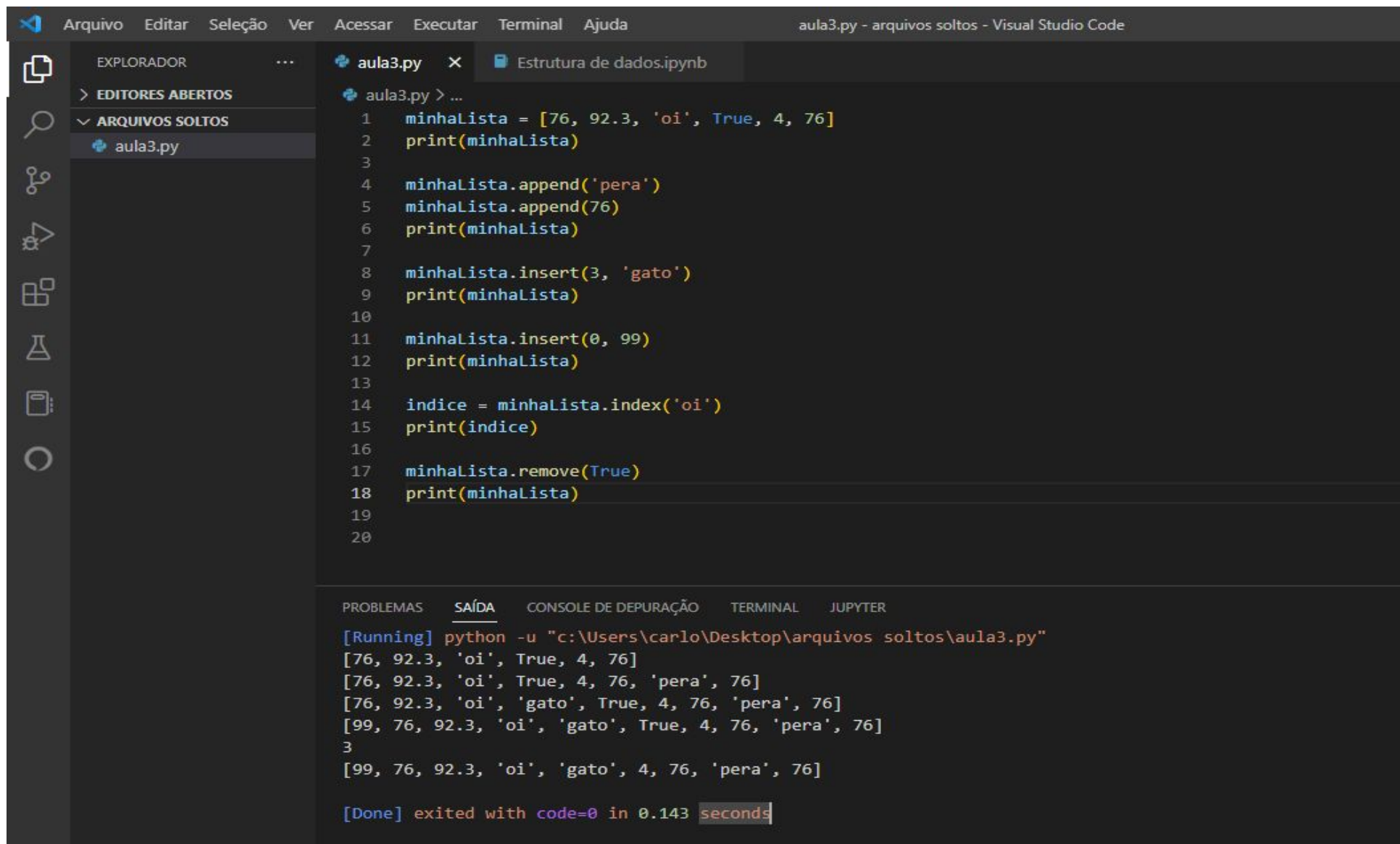
Prática sobre listas

2) Crie uma lista chamada '**minhaLista**', com os seguintes itens: **76, 92.3, "oi", True, 4, 76**.

Depois execute os seguintes comandos:

- a) Inserir "pera" e 76 no final da lista.
- b) Inserir o valor "gato" na posição de índice 3.
- c) Inserir o valor 99 no início da lista.
- d) Encontrar o índice de "oi".
- e) Copie a lista em outra lista chamada "copia"
- e) Remover True da lista copiada
- f) Imprimir as 2 listas.

Prática sobre listas



The screenshot shows the Visual Studio Code interface with a Python file named `aula3.py` open. The file contains a list `minhaLista` and several operations performed on it. The output of the script is displayed in the terminal window at the bottom.

```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  aula3.py - arquivos soltos - Visual Studio Code
```

EXPLORADOR

EDITORES ABERTOS

ARQUIVOS SOLTOS

aula3.py

```
aula3.py > ...
1  minhaLista = [76, 92.3, 'oi', True, 4, 76]
2  print(minhaLista)
3
4  minhaLista.append('pera')
5  minhaLista.append(76)
6  print(minhaLista)
7
8  minhaLista.insert(3, 'gato')
9  print(minhaLista)
10
11 minhaLista.insert(0, 99)
12 print(minhaLista)
13
14 indice = minhaLista.index('oi')
15 print(indice)
16
17 minhaLista.remove(True)
18 print(minhaLista)
19
20
```

PROBLEMAS SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL JUPYTER

```
[Running] python -u "c:\Users\carlo\Desktop\arquivos_soltos\aula3.py"
[76, 92.3, 'oi', True, 4, 76]
[76, 92.3, 'oi', True, 4, 76, 'pera', 76]
[76, 92.3, 'oi', 'gato', True, 4, 76, 'pera', 76]
[99, 76, 92.3, 'oi', 'gato', True, 4, 76, 'pera', 76]
3
[99, 76, 92.3, 'oi', 'gato', 4, 76, 'pera', 76]

[Done] exited with code=0 in 0.143 seconds
```

Prática sobre listas

3) O que é impresso pelo trecho de código a seguir?

```
uma_lista = [4, 2, 8, 6, 5]  
uma_lista = uma_lista + ['gato', 'bode', 'bola']  
print(uma_lista)
```

Prática sobre listas

Concatenação de listas.

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  JUPYTER
[Running] python -u "c:\Users\carlo\Desktop\arquivos soltos\aula3.py"
[4, 2, 8, 6, 5, 'gato', 'bode', 'bola']

[Done] exited with code=0 in 0.14 seconds
|
```


Atividade.



Exercícios

[Listas: Clique aqui](#)





Tuplas

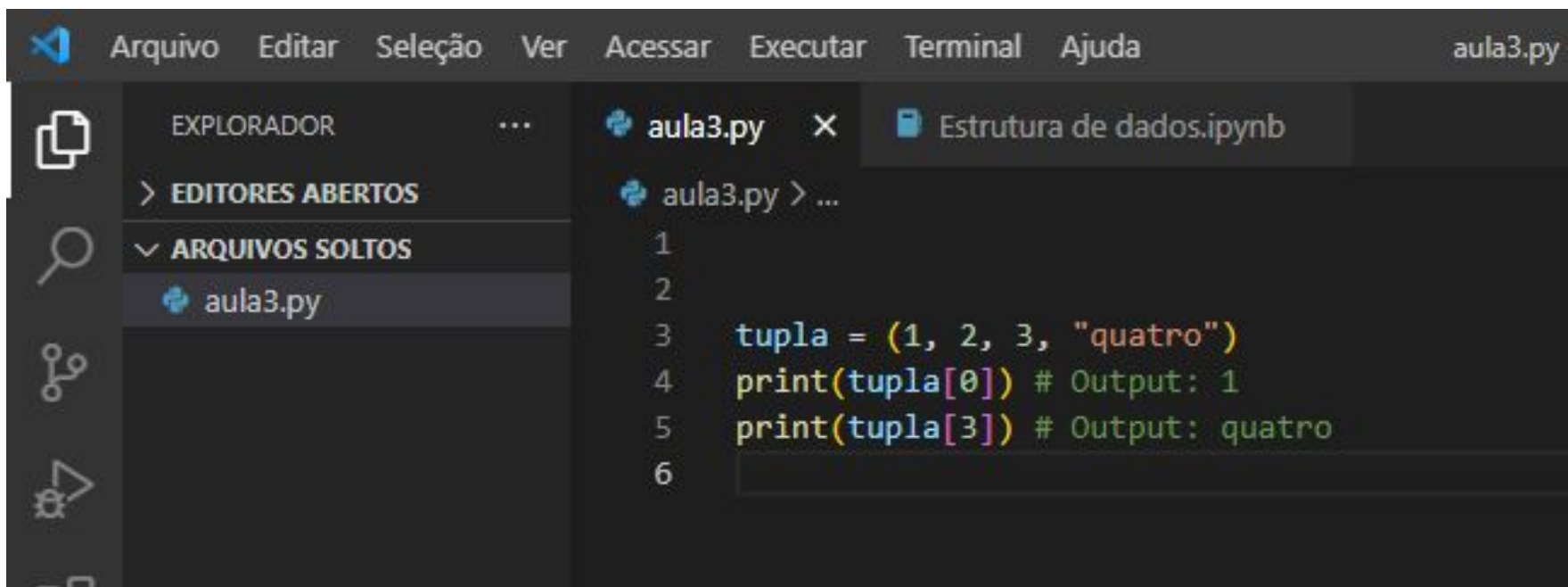
Uma tupla é uma coleção ordenada e imutável de elementos. Isso significa que, uma vez criada, uma tupla não pode ser alterada – você não pode adicionar, remover ou modificar seus elementos. As tuplas são definidas usando parênteses ().

Embora as tuplas sejam imutáveis, podemos manipulá-las de várias maneiras.

Tuplas

1) Acessando elementos de uma tupla:

Podemos acessar os elementos de uma tupla usando o índice de cada elemento. Elas também são indexadas da mesma forma que as listas.



```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  aula3.py -
```

EXPLORADOR ...

> EDITORES ABERTOS

✓ ARQUIVOS SOLTOS

aula3.py

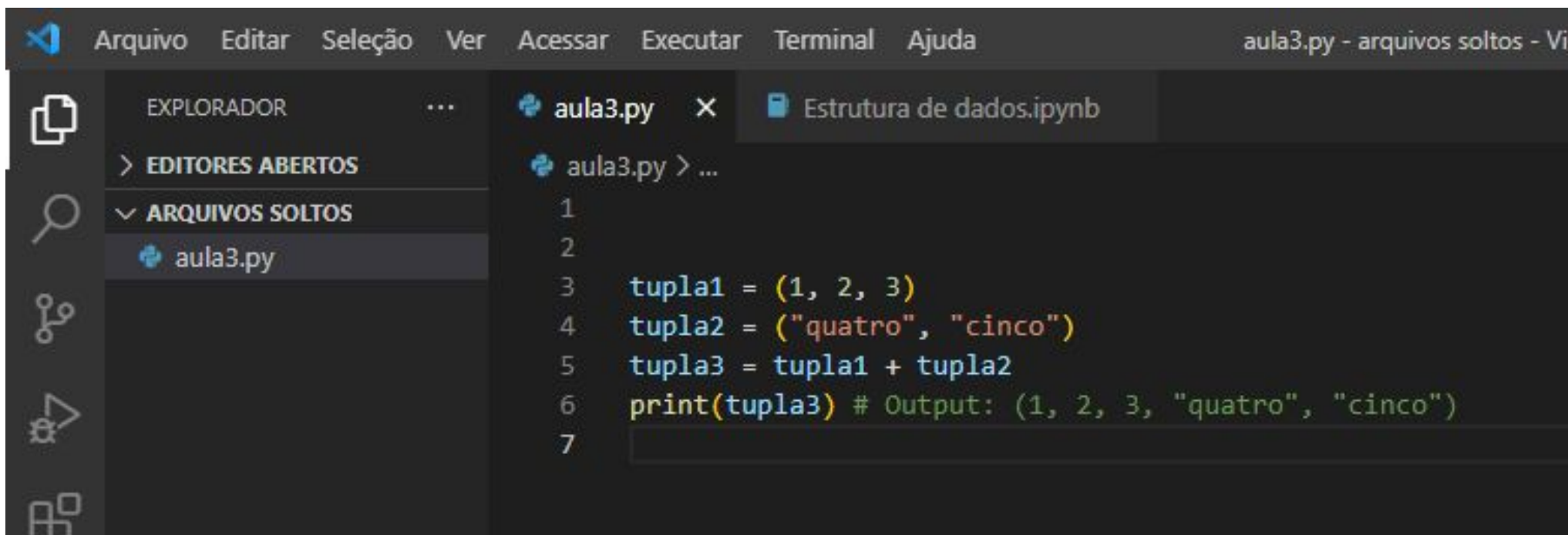
aula3.py > ...

```
1
2
3  tupla = (1, 2, 3, "quatro")
4  print(tupla[0]) # Output: 1
5  print(tupla[3]) # Output: quatro
6
```

Tuplas

2) Concatenando tuplas:

Podemos criar uma nova tupla concatenando duas ou mais tuplas usando o operador "+":



```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  aula3.py - arquivos soltos - Vi
```

EXPLORADOR ...

> EDITORES ABERTOS

ARQUIVOS SOLTOS

aula3.py

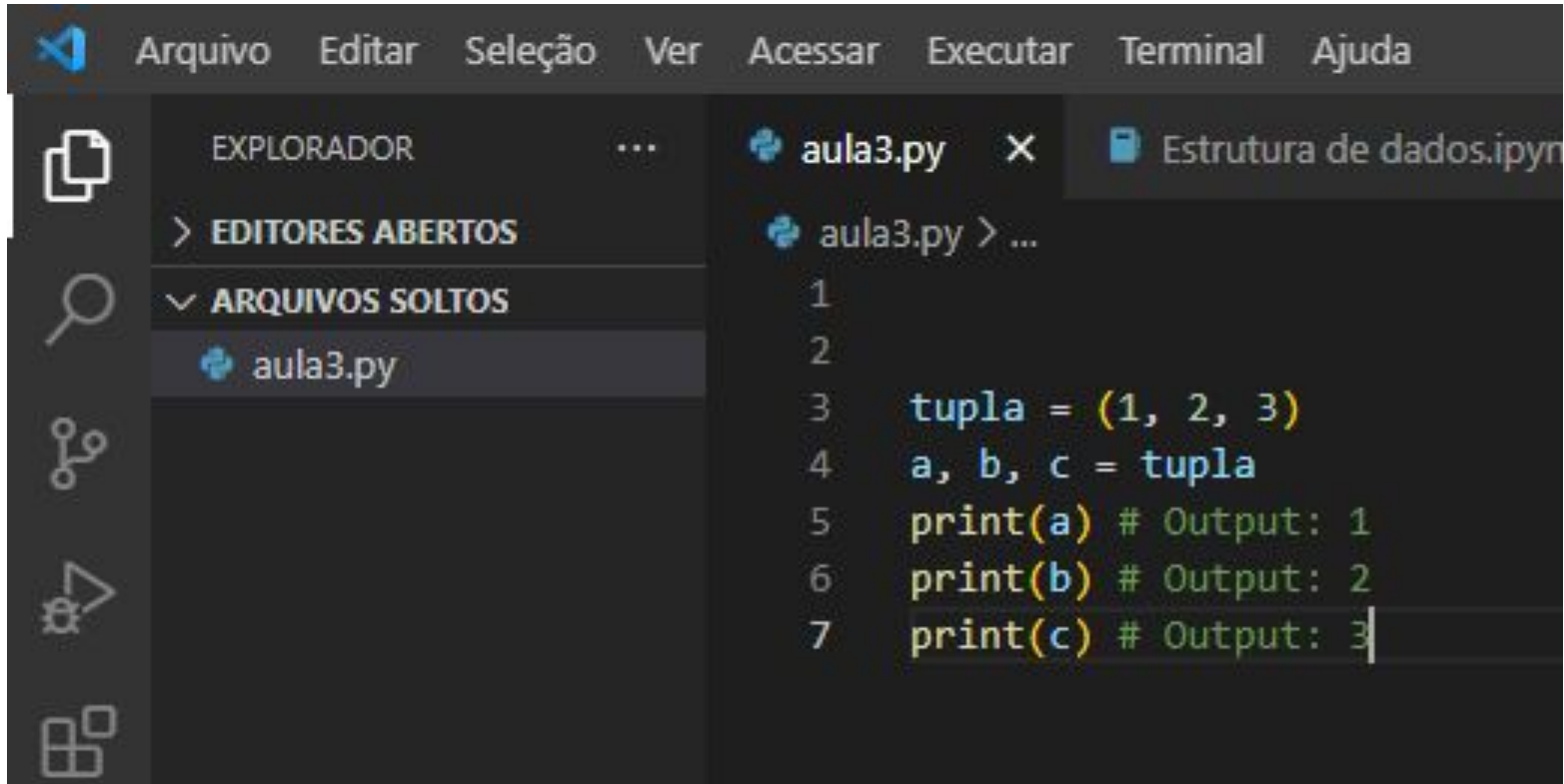
aula3.py > ...

```
1
2
3  tupla1 = (1, 2, 3)
4  tupla2 = ("quatro", "cinco")
5  tupla3 = tupla1 + tupla2
6  print(tupla3) # Output: (1, 2, 3, "quatro", "cinco")
7
```

Tuplas

3) Desempacotando tuplas:

Podemos desempacotar os elementos de uma tupla em variáveis separadas usando a sintaxe de atribuição múltipla. Por exemplo:



```
Arquivo  Editor  Seleção  Ver  Acessar  Executar  Terminal  Ajuda

EXPLORADOR
> EDITORES ABERTOS
v ARQUIVOS SOLTOS
  aula3.py

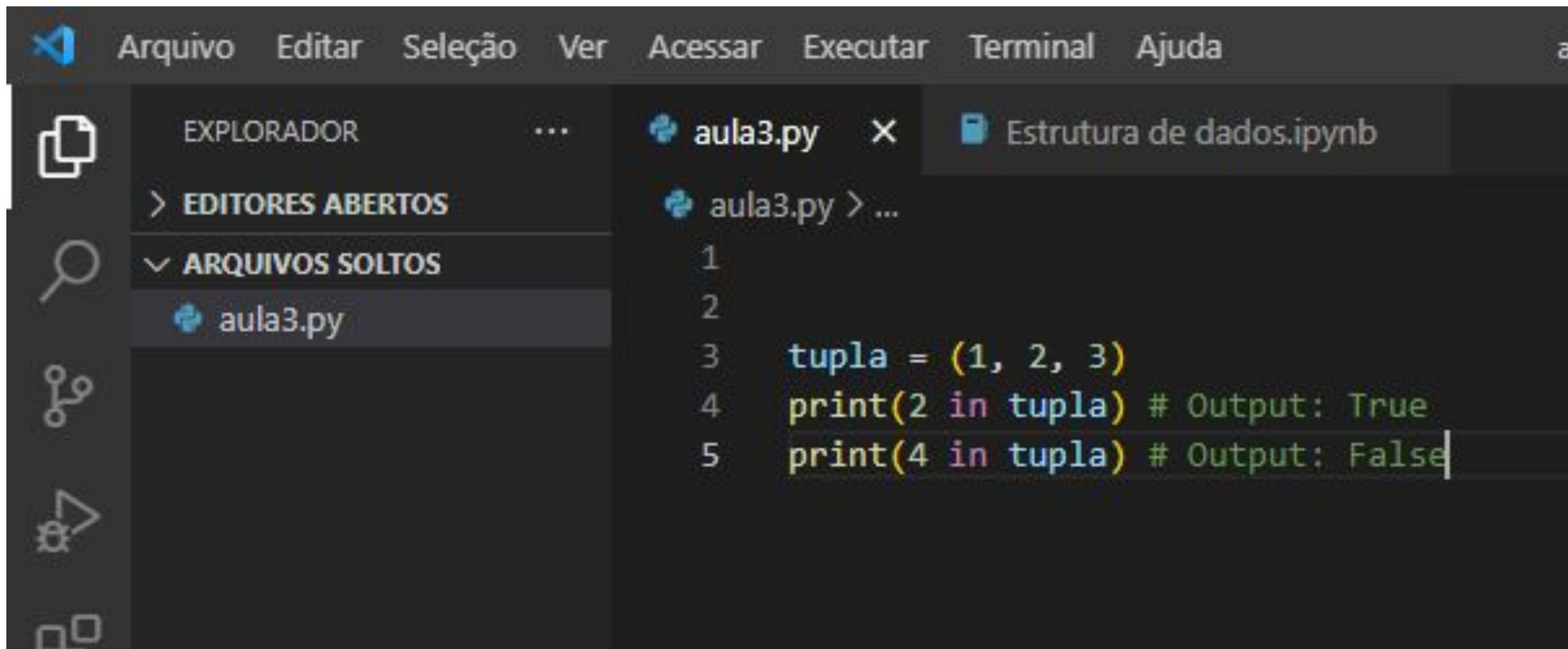
aula3.py > ...
1
2
3  tupla = (1, 2, 3)
4  a, b, c = tupla
5  print(a) # Output: 1
6  print(b) # Output: 2
7  print(c) # Output: 3
```

Tuplas

4) Verificando se um elemento está em uma tupla:

Podemos verificar se um elemento está presente em uma tupla usando o operador "in".

Por exemplo:



```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda

EXPLORADOR  ...
> EDITORES ABERTOS
  ARQUIVOS SOLTOS
    aula3.py

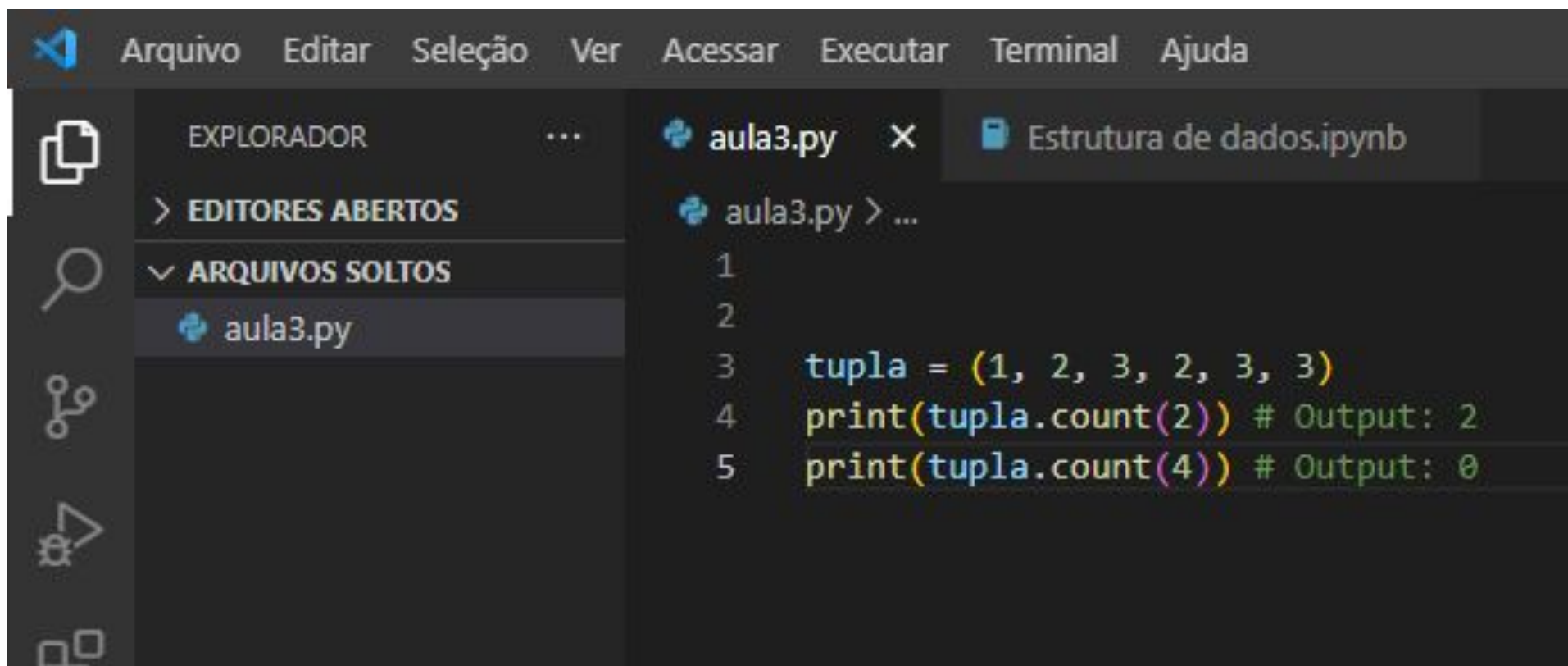
aula3.py  x  Estrutura de dados.ipynb

aula3.py > ...
1
2
3  tupla = (1, 2, 3)
4  print(2 in tupla) # Output: True
5  print(4 in tupla) # Output: False
```

Tuplas

5) Contando o número de ocorrências de um elemento em uma tupla: Podemos contar o número de ocorrências de um elemento em uma tupla usando o método `"count()"`.

Por exemplo:



```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda

EXPLORADOR  ...
> EDITORES ABERTOS
v ARQUIVOS SOLTOS
  aula3.py

aula3.py  x  Estrutura de dados.ipynb

aula3.py > ...
1
2
3  tupla = (1, 2, 3, 2, 3, 3)
4  print(tupla.count(2)) # Output: 2
5  print(tupla.count(4)) # Output: 0
```



Tuplas

5) Fatiando tuplas:

Você pode fatiar tuplas da mesma forma que fizemos com as listas.

Por exemplo:

```
# Tupla de números
numeros = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

# Fatiar do índice 2 ao 5
subtupla = numeros[2:5]
print(subtupla) # Saída: (2, 3, 4)

# Fatiar do início até o índice 4
subtupla = numeros[:4]
print(subtupla) # Saída: (0, 1, 2, 3)

# Fatiar do índice 5 até o final
subtupla = numeros[5:]
print(subtupla) # Saída: (5, 6, 7, 8, 9)

# Fatiar com passo
subtupla = numeros[::2]
print(subtupla) # Saída: (0, 2, 4, 6, 8)
```




Dicionário

Um dicionário é uma estrutura de dados que permite armazenar e manipular pares de **chave : valor**. É semelhante a uma lista, mas em vez de acessar seus elementos por um índice numérico, **você acessa seus valores através de uma chave**.

Para criar um dicionário em Python, você pode usar **chaves { }** e separar as chaves e os valores com dois pontos :

Dicionário

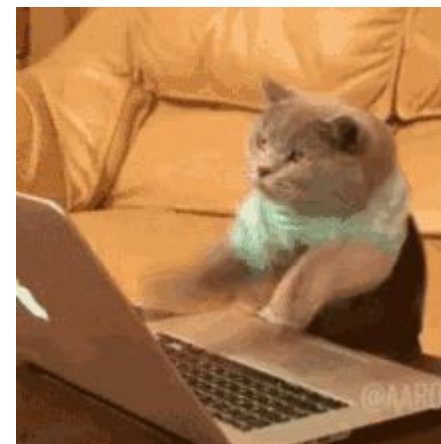
Criando um dicionário

Você pode criar um dicionário de várias maneiras.

```
# Criando dicionário vazio
dicionario_vazio = {}

# Criando dicionário com alguns pares chave-valor
dados_pessoais = {
    "nome": "Márcio",
    "idade": 28,
    "cidade": "Recife"
}

# Criando dicionário com a função dict()
outro_dicionario = dict(nome="Anderson", idade=25, cidade="Recife")
```



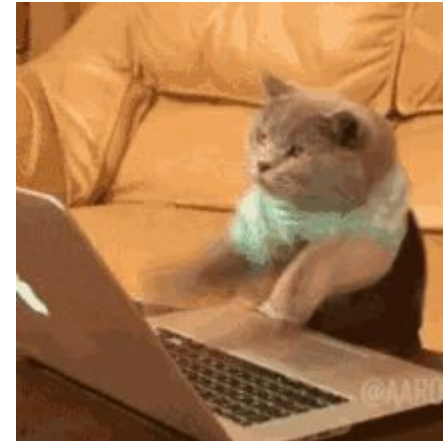
Dicionário

Acessando valores em um dicionário

```
# Dicionário de dados pessoais
dados_pessoais = {
    "nome": "Márcio",
    "idade": 28,
    "cidade": "Recife"
}

# Acessando valores
nome = dados_pessoais["nome"]
idade = dados_pessoais["idade"]
cidade = dados_pessoais["cidade"]

print(nome)    # Saída: Márcio
print(idade)   # Saída: 28
print(cidade)  # Saída: Recife
```



Dicionário

Adicionando e modificando valores em um dicionário

Você pode adicionar novos pares chave-valor ou modificar os existentes simplesmente atribuindo um valor a uma chave.

```
# Dicionário de dados pessoais
dados_pessoais = {
    "nome": "Márcio",
    "idade": 28,
    "cidade": "Recife"
}

# Adicionando um novo par chave-valor
dados_pessoais["profissão"] = "Professor"

# Modificando um valor existente
dados_pessoais["idade"] = 31

print(dados_pessoais)
# Saída: {'nome': 'Márcio', 'idade': 31, 'cidade': 'Recife', 'profissão': 'Professor'}
```



Dicionário

Removendo elemento em um dicionário

Você pode remover itens de um dicionário usando as funções **del()** ou **pop()**.

Deleta

Salva em nova variável e deleta

```
# Dicionário de dados pessoais
dados_pessoais = {
    "nome": "Márcio",
    "idade": 28,
    "cidade": "Recife",
    "profissão": "Professor"
}

# Removendo um item com del
del dados_pessoais["cidade"]

# Removendo um item com pop()
idade = dados_pessoais.pop("idade")

print(dados_pessoais) # Saída: {'nome': 'Márcio'}

print(idade) # Saída: 28
```



Dicionário

Conferindo elemento em um dicionário

Você pode verificar se um item está presente no dicionário usando **in**.

```
#Criando dicionário
lista_de_compras= {
    "arroz": 3,
    "feijão": 5,
    "cebola": 10
}

#Verificando se uma chave existe no dicionário
print("cebola" in lista_de_compras) #Saída: True
print("batata" in lista_de_compras) #Saída: False
```





Dicionário

Funções importantes em dicionários:

- `keys()` - Retorna uma visualização das chaves no dicionário.
- `values()` - Retorna uma visualização dos valores no dicionário.
- `items()` - Retorna uma visualização dos pares chave-valor no dicionário.
- `get()` - Retorna o valor para a chave especificada.
- `update()` - Atualiza o dicionário com pares chave-valor de outro dicionário ou iterável.
- `popitem()` - Remove e retorna o último par chave-valor inserido.
- `clear()` - Remove todos os itens do dicionário.
- `copy()` - Retorna uma cópia superficial do dicionário.

Dicionário

```
#Criando dicionário
dados_pessoais = {"nome": "Márcio", "idade": 28, "cidade": "Recife"}

#Retornando as chaves do dicionário
chaves = dados_pessoais.keys()
print(chaves) # Saída: dict_keys(['nome', 'idade', 'cidade'])

#Retornando valores do dicionário
valores = dados_pessoais.values()
print(valores) # Saída: dict_values(['Márcio', 28, 'Recife'])

# Retornando os pares de chave:valor do dicionário
pares = dados_pessoais.items()
print(pares) # Saída: dict_items([('nome', 'Márcio'), ('idade', 28), ('cidade', 'Recife')])

# Usando get() para obter o valor da chave "nome"
valor_nome = dados_pessoais.get("nome")
print(valor_nome) # Saída: Márcio
```



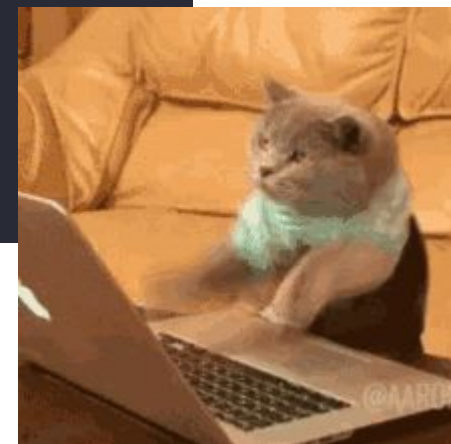
Dicionário

```
#Criando dicionário
dados_pessoais = {"nome": "Márcio", "idade": 28, "cidade": "Recife"}

# Usando update() para mesclar com outro dicionário
endereco = {"rua": "Rua dos médicos", "número": 30}
dados_pessoais.update(endereco)
print(dados_pessoais) # Saída: {'nome': 'Márcio', 'idade': 28, 'cidade': 'Recife', 'rua': 'Rua dos médicos', 'número': 30}

# Usando popitem() para remover o último item
ultimo_item = dados_pessoais.popitem()
print(ultimo_item) # Saída: ('número', 30)

# Usando clear() para apagar todo o dicionário
dados_pessoais.clear()
print(dados_pessoais) # Saída: {}
```



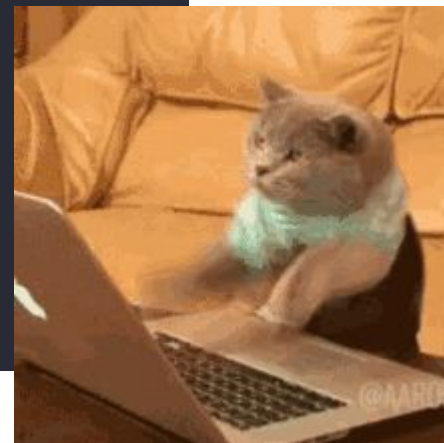
Dicionário

Transformando dicionário em lista

```
#Criando dicionário
dados_pessoais = {"nome": "Márcio", "idade": 28, "cidade": "Recife"}

#Convertendo dicionário em lista
chave = list(dados_pessoais)
valor= list(dados_pessoais.values())

print(chave)
print(valor)
```





Prática sobre dicionário

1) Crie um dicionário chamado 'meu_dicionario' , com os seguintes itens: "nome"="nome", "idade"= 0, "cidade"= "cidade". Depois execute os seguintes comandos:

- a) Modifique o valor das chaves: nome, idade e cidade.
- b) Insira uma chave chamada "animal_favorito" e coloque um valor.
- c) Imprima o valor da chave idade após alteração.
- d) Confira se o animal formiga está no dicionário.
- e) Imprima as chaves do dicionário.
- e) Converta os valores do dicionário em uma lista.
- f) Remova e mostre na tela o último valor do dicionário.
- g) Imprima o dicionário completo



Prática Dicionário

O que acontece em cada sequencia de comandos a seguir?

```
compras = {'cebola': 15, 'batata': 35, 'tomate': 12}
```

a) `print(compras['batatas'])`

e) `print(compras.get('tomate '))`

b) `compras ['cebola'] = 20`
`print(len(compras))`

f) `comprei = compras.keys()`
`comprei.sort()`
`print(comprei)`

c) `print('tomate' in compras)`

g) `del compras ['batata']`
`print('batata' in compras)`

d) `print(compras['cenoura'])`



Dicionário

```
Traceback (most recent call last):
  File "c:\Users\marci\OneDrive\Área de Trabalho\FUCTURA\AULA 3\aula3.py", line 271, in <module>
    print(compras['batatas'])
    ~~~~~^~~~~~
KeyError: 'batatas'
```

```
ocal/Programs/Python/Pyt
CTURA/AULA 3/aula3.py"
12
```

```
PS C:\Users\marci\OneDrive\Área de Trabalho\FUCTURA\AULA 3>
3
```

```
PS C:\Users\marci\OneDrive\Área de Trabalho\FUCTURA\AULA 3>
ocal/Programs/Python/Pyt
CTURA/AULA 3/aula3.py"
True
```

```
Traceback (most recent call last):
  File "c:\Users\marci\OneDrive\Área de Trabalho\FUCTURA\AULA 3\aula3.py", line 271, in <module>
    comprei.sort()
    ~~~~~^~~~~~
AttributeError: 'dict_keys' object has no attribute 'sort'
```

```
Traceback (most recent call last):
  File "c:\Users\marci\OneDrive\Área de Trabalho\FUCTURA\AULA 3\aula3.py", line 271, in <module>
    print (compras['cenoura'])
    ~~~~~^~~~~~
KeyError: 'cenoura'
```

```
PS C:\Users\marci\OneDrive\Área de Trabalho\FUCTURA\AULA 3>
ocal/Programs/Python/Pyt
CTURA/AULA 3/aula3.py"
False
```

Atividade



Listas: [clique aqui](#).

Dicionários: [clique aqui](#)



Strings



Strings



Strings

Strings são **sequências de caracteres** que podem ser usadas para representar texto em Python. Elas são declaradas entre **aspas simples (' ')** ou **duplas (" ")**.

Strings

```
# Declarando uma string:

mensagem = "Hello, world!"

# Concatenando strings:

primeiro_nome = "João"
sobrenome = "Silva"
nome_completo = primeiro_nome + " " + sobrenome

# Acessando caracteres individuais em uma string:

mensagem = "Hello, world!"
primeiro_caractere = mensagem[0]
ultimo_caractere = mensagem[-1]

# Encontrando o comprimento de uma string:

mensagem = "Hello, world!"
comprimento = len(mensagem)
```



Strings

```
mensagem = "Hello, world!"
maiuscula = mensagem.upper()
minuscula = mensagem.lower()

# Dividindo uma string em substrings com base em um delimitador:

mensagem = "Hello, world!"
palavras = mensagem.split(",")

# Verificando se uma substring está presente em uma string:

mensagem = "Hello, world!"
if "world" in mensagem:
    print("A substring 'world' está presente na mensagem.")

# Substituindo caracteres em uma string:

mensagem = "Hello, world!"
nova_mensagem = mensagem.replace("world", "Python")
```



Strings

```
# Formatando uma string:

nome = "João"
idade = 30
mensagem = "Meu nome é {} e eu tenho {} anos.".format(nome, idade)
mensagem2 = f"Meu nome é {nome} e eu tenho {idade} anos."

# Utilizando caracteres de escape em uma string:

mensagem = "Eu gosto de programar em Python!\nEle é uma linguagem muito poderosa." # lembram dele?
print(mensagem)

# Convertendo uma string em um número:

numero_string = "123"
numero_inteiro = int(numero_string)
numero_float = float(numero_string)

#Acessando substrings de uma string:

mensagem = "Hello, world!"
substring = mensagem[0:5] # Retorna os primeiros cinco caracteres
```



Strings



```
# Obtendo um caractere específico:

frase = "Hello, world!"
primeiro_caractere = frase[0] # Retorna "H"
ultimo_caractere = frase[-1] # Retorna "!"

# Obtendo uma substring:

frase = "Hello, world!"
primeira_palavra = frase[0:5] # Retorna "Hello"
ultimas_duas_palavras = frase[-6:] # Retorna "world!"

# Saltando caracteres durante o fatiamento:

frase = "Hello, world!"
apenas_letras_impares = frase[1::2] # Retorna "el, ol!"

# Usando fatiamento para reverter uma string:

frase = "Hello, world!"
reverso = frase[::-1] # Retorna "!dlrow ,olleH"
```



Strings

```
# Obtendo um conjunto específico de caracteres:

frase = "Hello, world!"
letras_pares = frase[::2] # Retorna "Hlo ol!"

# Removendo espaços em branco de uma string:

frase = "  Hello, world!  "
sem_espacos = frase.strip() # Retorna "Hello, world!"
```



Atividades

Strings: [clique aqui](#).

Listas: [clique aqui](#).

Dicionários: [clique aqui](#).



Não desista!



“ 90% DO SUCESSO
SE BASEIA
EM INSISTIR ”

WOODY ALLEN
CINEASTA