



PROGRAMA DE CAPACITAÇÃO

DATA SCIENCE + DATA INTELLIGENCE



Introdução ao Uso de Python para Análise de Dados.

Professor Marcelo Fernandes
Março/2022



Introdução ao Uso de Python para Análise de Dados.

Aula 02

Operadores de comparação

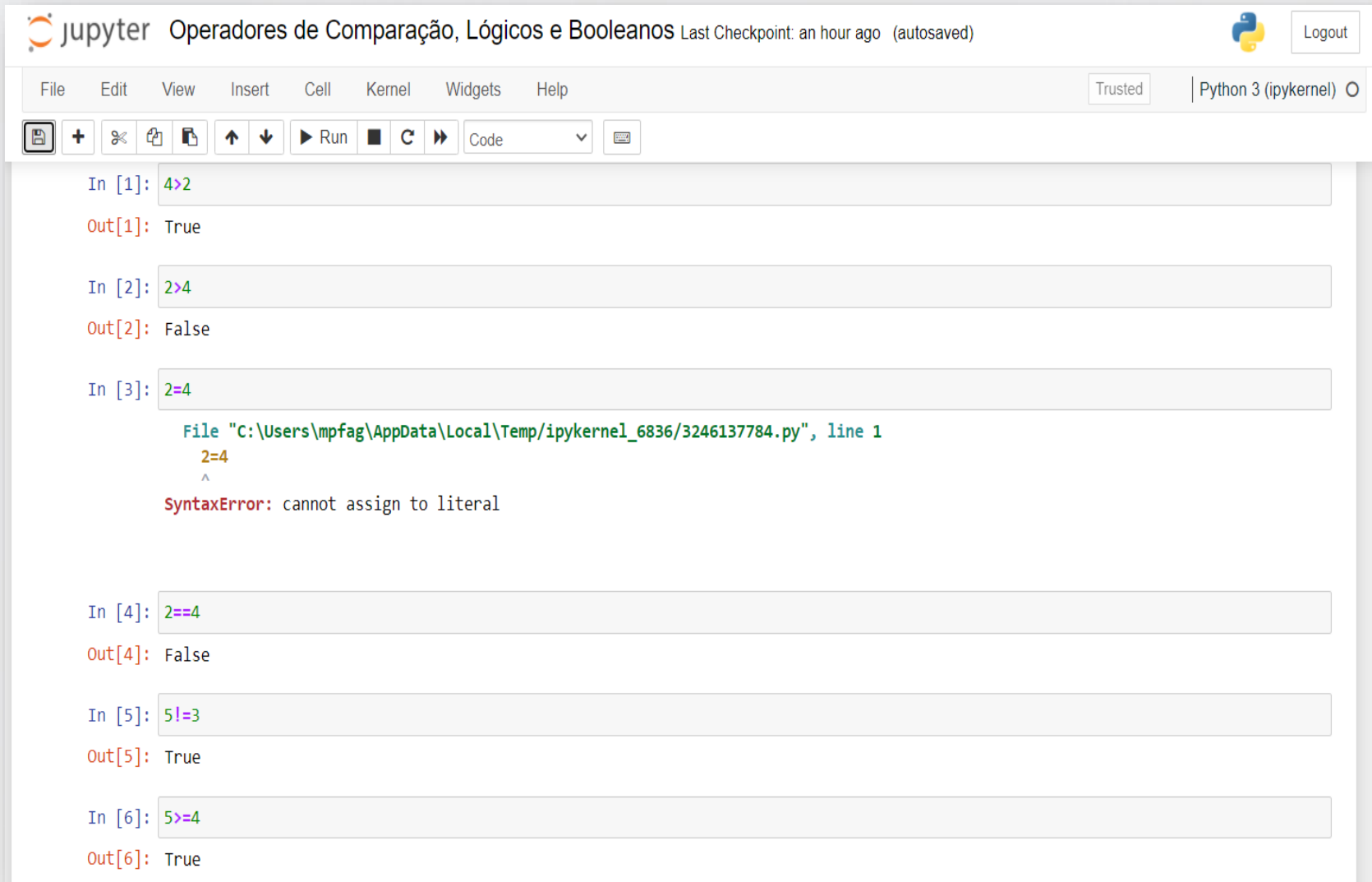
Operador	Nome	Função
<code>==</code>	Igual a	Verifica se um valor é igual ao outro
<code>!=</code>	Diferente de	Verifica se um valor é diferente ao outro
<code>></code>	Maior que	Verifica se um valor é maior que outro
<code>>=</code>	Maior ou igual	Verifica se um valor é maior ou igual ao outro
<code><</code>	Menor que	Verifica se um valor é menor que outro
<code><=</code>	Menor ou igual	Verifica se um valor é menor ou igual ao outro

Fonte: <https://pythonacademy.com.br/blog/operadores-aritmeticos-e-logicos-em-python#:~:text=Python%20nos%20disponibiliza%20tr%C3%AAs%20tipos,o%20or%20e%20o%20not%20>.

Operadores booleanos como **TRUE** e **FALSE** estão amplamente difundidos na lógica de programação da linguagem Python.

Além dos operadores de comparação listados acima, existem outros inúmeros operadores amplamente usados como “**is**”, “**is not**”, “**in**”, “**not in**”, “**and**”, “**or**”, “**%=**”, que veremos mais adiante.

Uso de Operadores Booleanos, Lógicos, de Comparação e de Atribuição



```
jupyter Operadores de Comparação, Lógicos e Booleanos Last Checkpoint: an hour ago (autosaved) Python 3 (ipykernel)

In [1]: 4>2
Out[1]: True

In [2]: 2>4
Out[2]: False

In [3]: 2=4
File "C:\Users\mpfag\AppData\Local\Temp\ipykernel_6836\3246137784.py", line 1
    2=4
    ^
SyntaxError: cannot assign to literal

In [4]: 2==4
Out[4]: False

In [5]: 5!=3
Out[5]: True

In [6]: 5>=4
Out[6]: True
```

❏ Note na célula de entrada 3 o erro ocorrido e verifique na célula 4, a sintaxe correta para comparação de igualdade entre 2 elementos. O correto é “==” e não “=”. Esse último é utilizado para atribuição de valores a um objeto ou variável (Ex.: a=2).

Operadores de atribuição

Operador	Equivalente a
----------	---------------

=	$x = 1$
---	---------

+=	$x = x + 1$
----	-------------

-=	$x = x - 1$
----	-------------

*=	$x = x * 1$
----	-------------

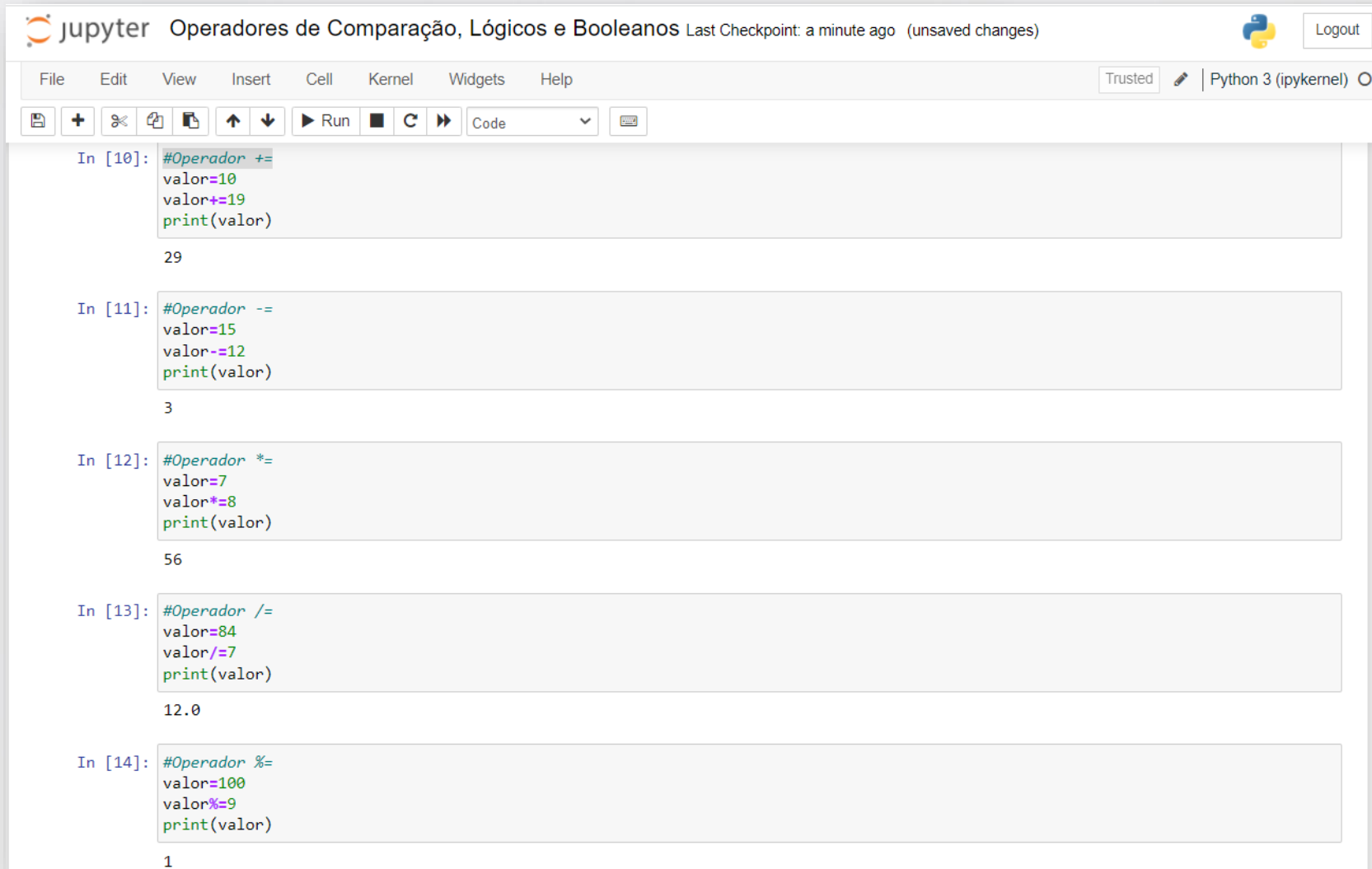
/=	$x = x / 1$
----	-------------

%=	$x = x \% 1$
----	--------------

Esses operadores são utilizados no momento de atribuição de valores às variáveis e controlam como a atribuição será estabelecida. Pode ser usado para definir um valor inicial ou sobrescrever um valor já existente.

Fonte: <https://pythonacademy.com.br/blog/operadores-aritmeticos-e-logicos-em-python#:~:text=Python%20nos%20disponibiliza%20tr%C3%AAs%20tipos,o%20or%20e%20o%20not%20>

Uso de Operadores Booleanos, Lógicos, de Comparação e de Atribuição



The screenshot shows a Jupyter Notebook titled "Operadores de Comparação, Lógicos e Booleanos". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar indicating "Trusted" and "Python 3 (ipykernel)". The notebook contains five code cells, each demonstrating a different assignment operator:

```
In [10]: #Operador +=
valor=10
valor+=19
print(valor)

29

In [11]: #Operador -=
valor=15
valor-=12
print(valor)

3

In [12]: #Operador *=
valor=7
valor*=8
print(valor)

56

In [13]: #Operador /=
valor=84
valor/=7
print(valor)

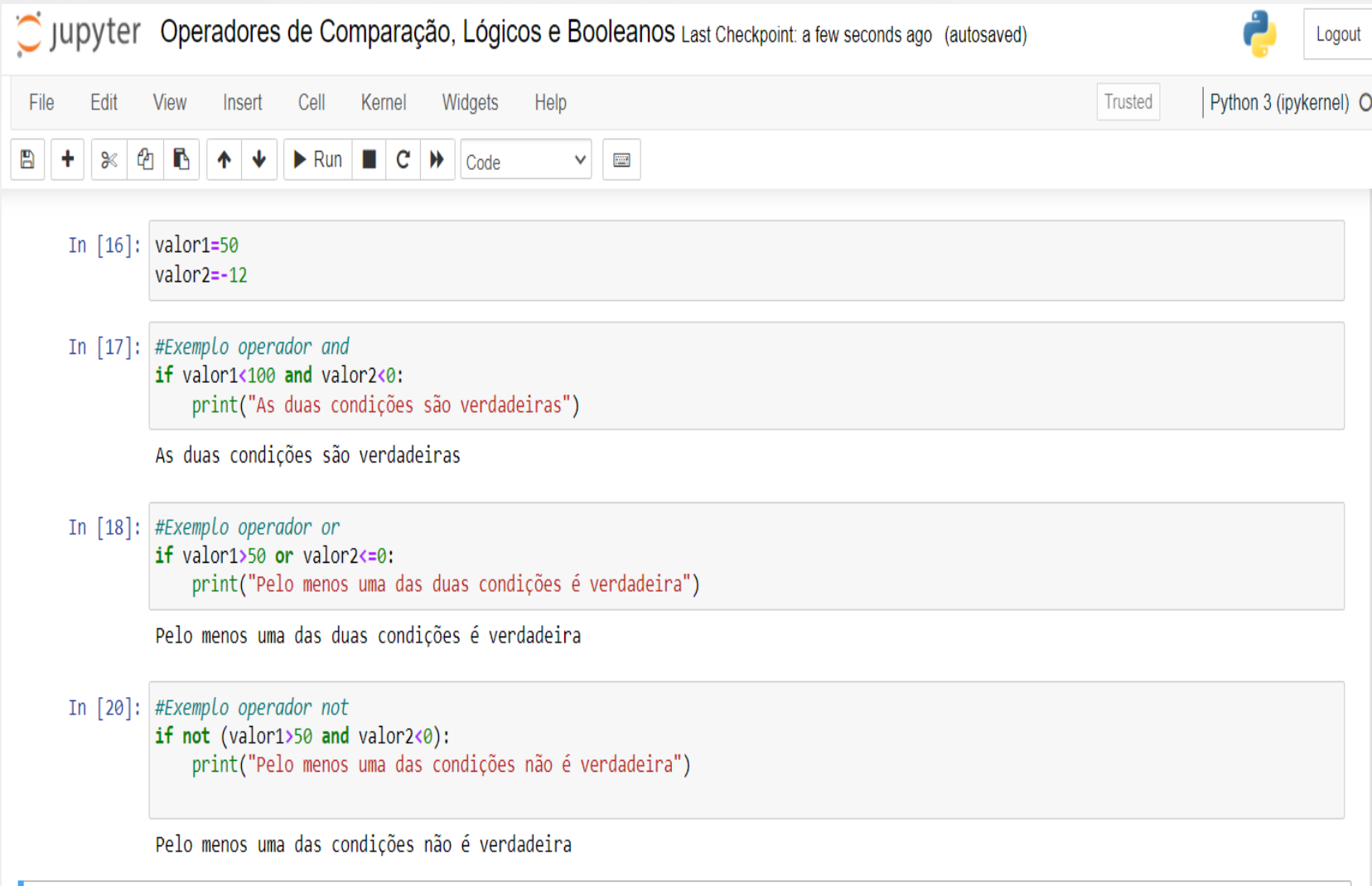
12.0

In [14]: #Operador %=
valor=100
valor%=9
print(valor)

1
```

❑ Ao lado, um exemplo para cada um dos principais tipos de operadores de atribuição presentes no Python.

Uso de Operadores Booleanos, Lógicos, de Comparação e de Atribuição



The image shows a Jupyter Notebook interface with the title "Operadores de Comparação, Lógicos e Booleanos". The interface includes a top bar with the Jupyter logo, the title, and a "Logout" button. Below the top bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar are buttons for "Trusted" and "Python 3 (ipykernel)". Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, running, and other functions. The main area of the notebook contains four code cells, each with input and output.

```
In [16]: valor1=50
         valor2=-12
```

```
In [17]: #Exemplo operador and
         if valor1<100 and valor2<0:
             print("As duas condições são verdadeiras")

         As duas condições são verdadeiras
```

```
In [18]: #Exemplo operador or
         if valor1>50 or valor2<=0:
             print("Pelo menos uma das duas condições é verdadeira")

         Pelo menos uma das duas condições é verdadeira
```

```
In [20]: #Exemplo operador not
         if not (valor1>50 and valor2<0):
             print("Pelo menos uma das condições não é verdadeira")

         Pelo menos uma das condições não é verdadeira
```

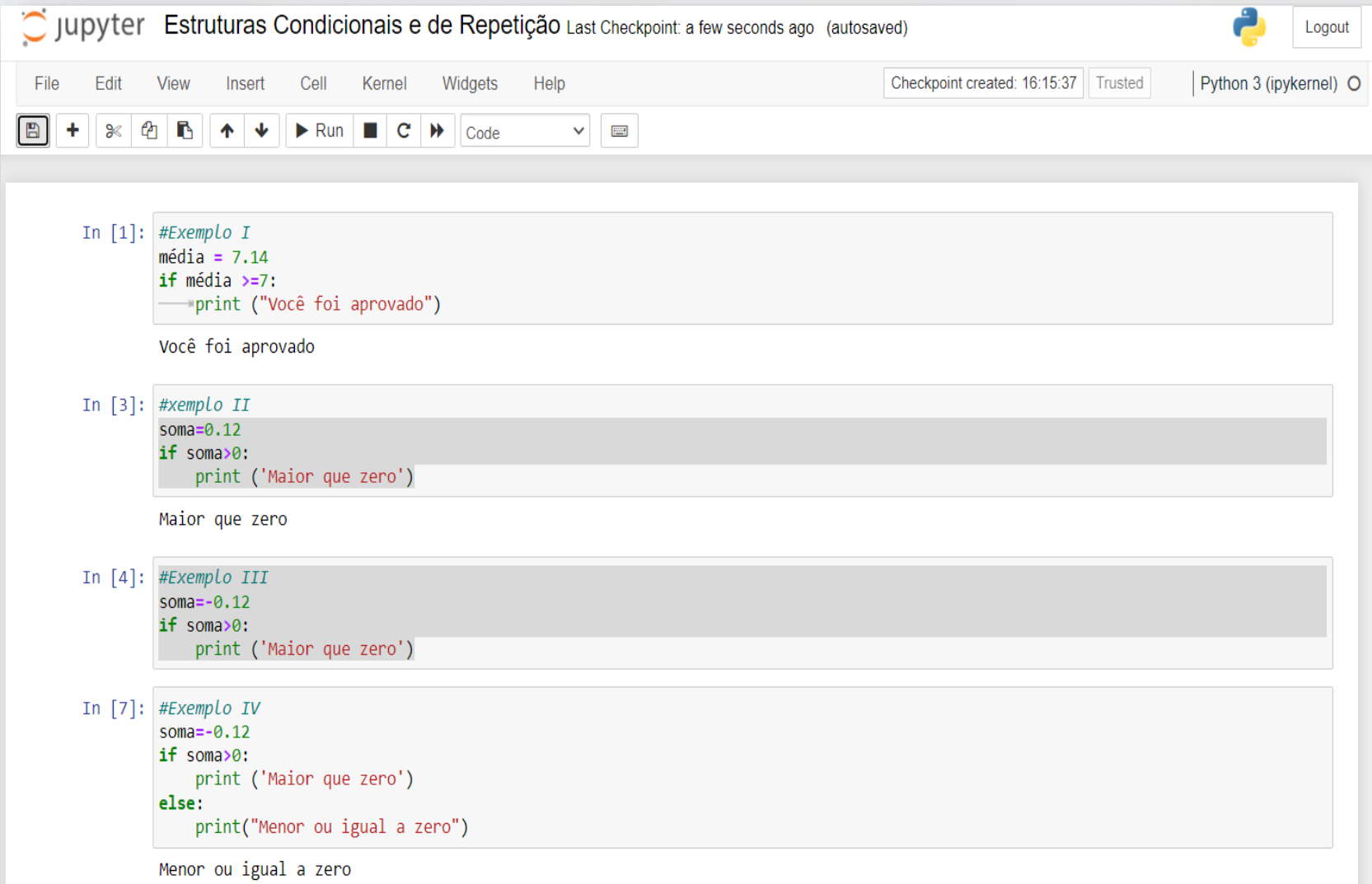
❑ Perceba que os operadores só funcionam em minúsculo (and, or, not). Experimente tentar os operadores escritos em maiúsculo.

3ª Sessão “Mão na Massa” (tempo: 10 minutos)

Qual o resultado das seguintes expressões (teste cada uma delas em Python), avaliadas em um notebook chamado Mão na Massa – Sessão III:

- a) Crie um notebook do zero, renomeando-o para Mão na Massa – Sessão III
- b) `valor=100, valor+=120`
- c) `valor=50, valor/=12`
- d) `Valor=100, valor*=30`
- d) Dado que `valor1=50, valor2=150`, como vc avalia `valor1<=50 and valor2>30`?
- e) Dado que `valor1=-20, valor2=1`, como vc avalia `valor1>0 or valor1=0`?
- f) Dado que `valor1=50, valor2=0`, como vc avalia `valor1-valor2>1 and valor1+valor2>0`?
- g) Se a condição `(valor1+valor2)>0` for verdadeira, exiba a mensagem "Verdade!"
- h) Se a condição `not (valor2>0)` for verdadeira, exiba a mensagem "Verdade!"
- i) Teste as seguintes condições no Python: T and F, F and F, F or F, T or F, T or T. O que você observou?

Estruturas condicionais no Python (if, if else, elif, etc.)



The screenshot shows a Jupyter Notebook titled "Estruturas Condicionais e de Repetição". The interface includes a top bar with the Jupyter logo, the title, and a "Logout" button. Below the title bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. To the right of the menu bar, it says "Checkpoint created: 16:15:37", "Trusted", and "Python 3 (ipykernel)". Below the menu bar is a toolbar with icons for saving, adding, undo, redo, and running code. The notebook contains four code cells, each with a prompt "In [X]:" and a code block. The first cell, labeled "In [1]:", contains a comment "#Exemplo I", a variable assignment "média = 7.14", and an if statement "if média >= 7:" followed by an indented "print('Você foi aprovado')". The output of this cell is "Você foi aprovado". The second cell, labeled "In [3]:", contains a comment "#Exemplo II", a variable assignment "soma=0.12", and an if statement "if soma>0:" followed by an indented "print('Maior que zero')". The output is "Maior que zero". The third cell, labeled "In [4]:", contains a comment "#Exemplo III", a variable assignment "soma=-0.12", and an if statement "if soma>0:" followed by an indented "print('Maior que zero')". The output is "Maior que zero". The fourth cell, labeled "In [7]:", contains a comment "#Exemplo IV", a variable assignment "soma=-0.12", and an if-else statement: "if soma>0:" followed by an indented "print('Maior que zero')", and "else:" followed by an indented "print('Menor ou igual a zero')". The output is "Menor ou igual a zero".

```
In [1]: #Exemplo I
média = 7.14
if média >= 7:
    print("Você foi aprovado")

Você foi aprovado

In [3]: #Exemplo II
soma=0.12
if soma>0:
    print('Maior que zero')

Maior que zero

In [4]: #Exemplo III
soma=-0.12
if soma>0:
    print('Maior que zero')

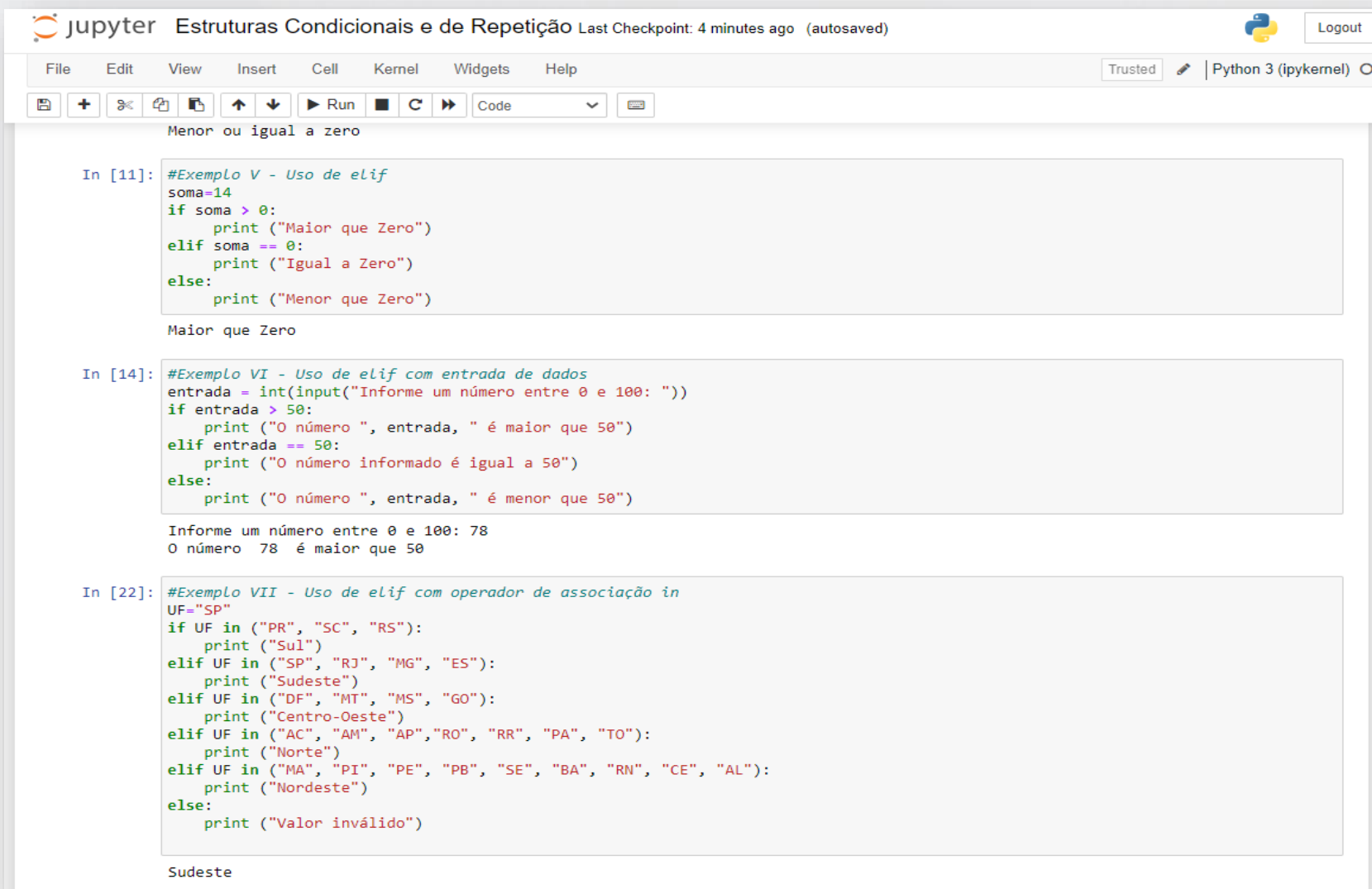
Maior que zero

In [7]: #Exemplo IV
soma=-0.12
if soma>0:
    print('Maior que zero')
else:
    print("Menor ou igual a zero")

Menor ou igual a zero
```

❑ A estrutura condicional if e if...else é bastante comum para gerar uma resposta baseada no teste de uma condição específica.

Estruturas condicionais no Python (if, if else, elif, etc.)



The screenshot shows a Jupyter Notebook titled "Estruturas Condicionais e de Repetição" with a last checkpoint of 4 minutes ago. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook contains three code cells:

```
In [11]: #Exemplo V - Uso de elif
soma=14
if soma > 0:
    print ("Maior que Zero")
elif soma == 0:
    print ("Igual a Zero")
else:
    print ("Menor que Zero")

Maior que Zero
```

```
In [14]: #Exemplo VI - Uso de elif com entrada de dados
entrada = int(input("Informe um número entre 0 e 100: "))
if entrada > 50:
    print ("O número ", entrada, " é maior que 50")
elif entrada == 50:
    print ("O número informado é igual a 50")
else:
    print ("O número ", entrada, " é menor que 50")

Informe um número entre 0 e 100: 78
O número 78 é maior que 50
```

```
In [22]: #Exemplo VII - Uso de elif com operador de associação in
UF="SP"
if UF in ("PR", "SC", "RS"):
    print ("Sul")
elif UF in ("SP", "RJ", "MG", "ES"):
    print ("Sudeste")
elif UF in ("DF", "MT", "MS", "GO"):
    print ("Centro-Oeste")
elif UF in ("AC", "AM", "AP", "RO", "RR", "PA", "TO"):
    print ("Norte")
elif UF in ("MA", "PI", "PE", "PB", "SE", "BA", "RN", "CE", "AL"):
    print ("Nordeste")
else:
    print ("Valor inválido")

Sudeste
```

❑ Percebam que o **elif** permite múltiplos “if” sem a necessidade de indentação (elemento importante no Python). Outro ponto interessante é a presença do **operador “in”**, que permite fazer associações com listas de elementos.

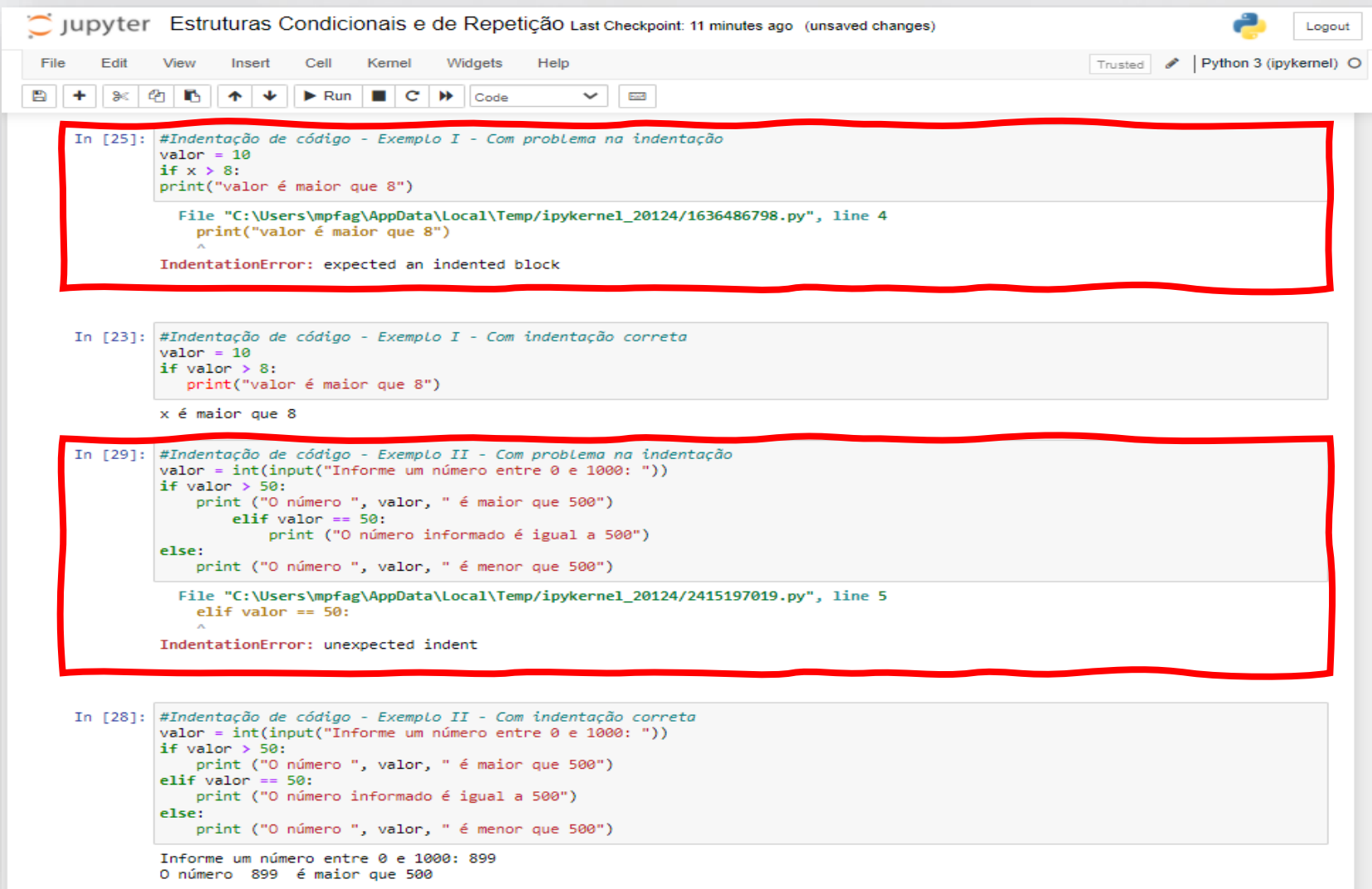
indentar

Recuar o texto em relação à margem da folha; inserir espaços entre a margem e o começo da linha de um parágrafo: antes de finalizar este trabalho, é (...)

[] Dicio.com.br

Indentação é um elemento fundamental e muito importante da linguagem Python, pois além de promover a legibilidade e a identificação de um bloco de códigos, é essencial para o bom funcionamento do código. Resumidamente, se a indentação não estiver correta, o programa pode se comportar de maneira inadequada ou nem executar.

Indentação na linguagem Python



Jupyter Estruturas Condicionais e de Repetição Last Checkpoint: 11 minutes ago (unsaved changes) Python 3 (ipykernel) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [25]: #Indentação de código - Exemplo I - Com problema na indentação
valor = 10
if x > 8:
print("valor é maior que 8")

File "C:\Users\mpfag\AppData\Local\Temp\ipykernel_20124\1636486798.py", line 4
print("valor é maior que 8")
^
IndentationError: expected an indented block
```

```
In [23]: #Indentação de código - Exemplo I - Com indentação correta
valor = 10
if valor > 8:
    print("valor é maior que 8")

x é maior que 8
```

```
In [29]: #Indentação de código - Exemplo II - Com problema na indentação
valor = int(input("Informe um número entre 0 e 1000: "))
if valor > 50:
    print ("O número ", valor, " é maior que 500")
    elif valor == 50:
        print ("O número informado é igual a 500")
else:
    print ("O número ", valor, " é menor que 500")

File "C:\Users\mpfag\AppData\Local\Temp\ipykernel_20124\2415197019.py", line 5
    elif valor == 50:
    ^
IndentationError: unexpected indent
```

```
In [28]: #Indentação de código - Exemplo II - Com indentação correta
valor = int(input("Informe um número entre 0 e 1000: "))
if valor > 50:
    print ("O número ", valor, " é maior que 500")
elif valor == 50:
    print ("O número informado é igual a 500")
else:
    print ("O número ", valor, " é menor que 500")

Informe um número entre 0 e 1000: 899
O número 899 é maior que 500
```

Percebam a diferença no recuo dos blocos que geraram erro vs os que rodaram normalmente. Notem que nos cenários com indentação correta, o comando "print()" fica fora da estrutura condicional.

4ª Sessão “Mão na Massa” (tempo: 10 minutos)

Implemente esse racional no Python:

- a) Atribua 500 para a variável “valor”. Se valor for maior que 500, imprima “Grande” na tela. Se o valor estiver entre 300 e 500, imprima “Médio” na tela. Se o valor for menor que 300, imprima “Pequeno” na tela. Altere o conteúdo da variável valor, para ver como o resultado se comporta.
- b) Como ficaria esse código se você tivesse que incluir uma 4ª condição, $\text{valor} < 0$ e mostrar o valor “Negativo” na tela?
- c) Crie um código usando estruturas condicionais para classificar essa lista de veículos em carro, moto ou inválido: “Fusca”, “Maverick”, “Toro”, “CG 160 Titan”, “Abacaxi”, “Biz 125”, “Cerato”, “Kawasaki Z 250”.

Estruturas de dados (listas, tuplas, dicionários, etc.)



Estruturas de dados representam uma área da computação que trabalha com formas eficientes para organização e manipulação de dados.

Listas, tuplas, conjuntos e dicionários são 3 das estruturas de dados mais importantes da linguagem Python.

Estrutura	Ordenação?	Mutável?	Construtor	Exemplo
Lista	Sim	Sim	[...] ou list()	[18,30,'A', 'B']
Tupla	Sim	Não	(...) ou tuple()	(18,30,'A', 'B')
Conjunto	Não	Sim	{...} ou set()	{'SP', 'RJ', 12,21}
Dicionário	Sim	Sim	{...} ou dict()	{'SP':12, "RJ":21}

Na sequência, vamos trabalhar com vários exemplos usando cada uma das estruturas e as operações mais comuns.



Uma lista é uma estrutura de dados que armazena os dados em sequência, em que cada elemento possui sua posição, denominada de índice. **O primeiro elemento é sempre o índice zero** e a cada elemento inserido na lista esse valor é incrementado. Veja a seguir alguns exemplos de listas:

```
In [13]: #Exemplo 1 - Lista numérica
lista1=[1,3,4,5]
print(lista1)
print(*lista1, sep = ',')
type(lista1)
```

```
[1, 3, 4, 5]
1,3,4,5
```

Out[13]: list

```
In [14]: #Exemplo2 - Lista com strings
lista2=["Bruno", 'Caio', 'Marina', "Paula"]
print(lista2)
print(*lista2, sep = ',')
type(lista2)
```

```
['Bruno', 'Caio', 'Marina', 'Paula']
Bruno,Caio,Marina,Paula
```

Out[14]: list

```
In [19]: #Exemplo3 - Lista com números e strings
lista3=[25,44,"João", "Paulo"]
print(lista3)
print(*lista3, sep = ' - ')
type(lista3)
```

```
[25, 44, 'João', 'Paulo']
25 - 44 - João - Paulo
```

Out[19]: list

❑ Soma dos elementos de uma lista

```
In [23]: #Operações básicas com listas - SOMA
print(lista1)
sum(lista1)
```

```
[1, 3, 4, 5]
```

```
Out[23]: 13
```

❑ Tamanho de uma lista

```
In [24]: #Operações básicas com listas - TAMANHO DA LISTA
print(lista2)
len(lista2)
```

```
['Bruno', 'Caio', 'Marina', 'Paula']
```

```
Out[24]: 4
```

❑ Soma dos elementos de uma lista

```
In [25]: #Operações básicas com listas - MÉDIA DOS VALORES DE UMA LISTA
print(lista1)
sum(lista1)/len(lista1)
```

```
[1, 3, 4, 5]
```

```
Out[25]: 3.25
```

❑ Selecionando elementos específicos

```
In [34]: #Selecionando elementos específicos de uma lista
lista5=["Brasil", "Chile", "Uruguai", "Colômbia"]
print(lista5[0])
print(lista5[1])
print(lista5[2])
print(lista5[3])
```

```
Brasil
Chile
Uruguai
Colômbia
```

```
In [36]: #Selecionando elementos específicos de uma lista - De trás pra frente
lista5=["Brasil", "Chile", "Uruguai", "Colômbia"]
print(lista5[-1])
print(lista5[-2])
print(lista5[-3])
print(lista5[-4])
```

```
Colômbia
Uruguai
Chile
Brasil
```

IMPORTANTE: Notem que o primeiro elemento de uma lista é sempre indexado com 0 e não com 1. Esse índice vai crescendo incrementalmente de 1 em 1.

IMPORTANTE: Percebam que também é um possível selecionar elementos de uma lista na ordem reversa, de trás pra frente, começando pelo índice “-1”.

❑ Removendo elementos de uma lista

```
In [28]: #Removendo elementos de uma lista
print(lista2)
lista2.remove('Bruno')
print(lista2)

['Bruno', 'Caio', 'Marina', 'Paula']
['Caio', 'Marina', 'Paula']
```

IMPORTANTE: O método **remove()** só remove o primeiro elemento que foi encontrado. Veja o exemplo a seguir:

```
In [30]: #Removendo elementos de uma lista - Elementos repetidos
lista4=["Rio", 'Rio', "São Paulo", "Belô"]
lista4.remove('Rio')
print(lista4)

['Rio', 'São Paulo', 'Belô']
```

Mais adiante, quando estudarmos estruturas de repetição (loops, for,...), veremos uma opção para remover todos os elementos selecionados.

❑ Removendo elementos de uma lista

```
In [51]: #Removendo elementos de uma lista - Método DEL
del lista4[2]
print(lista4)

['Rio', 'São Paulo']
```

❑ Removendo elementos de uma lista

```
In [57]: #Removendo elementos de uma lista - Método DEL
lista4=["Banana", "Banana", "Melão", "Cupuaçu", "Goiaba"]
print(lista4)
del lista4[0:2]
print(lista4)

['Banana', 'Banana', 'Melão', 'Cupuaçu', 'Goiaba']
['Melão', 'Cupuaçu', 'Goiaba']
```

Vejam que a seleção [0:2] inclui o elemento da posição 0 e da posição 1, mas não da posição 2. Ele é excludente. Assim, se quisermos os elementos 2,3,4, devemos colocar [2:5] e não [2:4]

❑ Alterando elementos de uma lista

```
In [37]: #Alterando elementos de uma lista
lista6=["EUA", "MÉXICO", "CANADÁ", "PARAGUAI"]
lista6[3]="CUBA"
print(lista6)

['EUA', 'MÉXICO', 'CANADÁ', 'CUBA']
```

```
In [38]: #Alterando elementos de uma lista - Exemplo que retorna erro
lista6=["EUA", "MÉXICO", "CANADÁ", "PARAGUAI"]
lista6[4]="CUBA"
print(lista6)

-----
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_22684\4193002934.py in <module>
      1 #Alterando elementos de uma lista - Exemplo que retorna erro
      2 lista6=["EUA", "MÉXICO", "CANADÁ", "PARAGUAI"]
----> 3 lista6[4]="CUBA"
      4 print(lista6)

IndexError: list assignment index out of range
```

Perceberam o erro? Pedimos para colocar “CUBA” na posição 4. Contudo, a lista só vai até a posição 3 (lembrem que o Python começa a indexação pela posição 0?).

❑ Adicionando um elemento a uma lista

```
In [47]: #Adicionando um elemento a uma lista
lista7=["Python", "R", "Scala"]
lista7.append("Julia")
print(lista7)

['Python', 'R', 'Scala', 'Julia']
```

O método **append()** é útil para adicionar um único elemento a uma lista.

❑ Adicionando vários elementos a uma lista

```
In [49]: #Adicionando varios elementos a uma lista
lista7.extend(["SQL", "Java", "JavaScript"])
print(lista7)

['Python', 'R', 'Scala', 'Julia', 'SQL', 'Java', 'JavaScript']
```

O método **extend()** é útil para adicionar mais de um elemento a uma lista.

❑ Substituindo elementos em uma lista

```
In [58]: #Substituindo elementos em uma lista
lista8=["Marina", "Paula", "Joana", "Maria"]
lista8[1]="Claudia"
print(lista8)

['Marina', 'Claudia', 'Joana', 'Maria']
```

```
In [60]: #Substituindo mais de um elemento em uma lista
lista8=["Marina", "Paula", "Joana", "Maria"]
lista8[1:3]=["Lúcia", "Mariana"]
print(lista8)

['Marina', 'Lúcia', 'Mariana', 'Maria']
```

Vejam que os nomes “Paula” e “Joana” foram substituídos por “Lúcia” e “Mariana”, nas posições 1 e 2. Observe que na indexação, colocamos 1:3, pois o último elemento não é considerado.

❑ Listas de listas

Uma lista de lista é nada mais do que uma ou mais listas dentro de outra lista. Mesmo nessa configuração, diversas manipulações são possíveis como mostra esse exemplo a seguir:

```
In [14]: #Listas de listas
lista9=[ [45,37,55,41,34], ["Paulo","Miguel","Anderson","Joaquim","Inácio"] ]
print(type(lista9))
print(len(lista9))
print(lista9[0])
print(lista9[1])
print(lista9[1][0])

<class 'list'>
2
[45, 37, 55, 41, 34]
['Paulo', 'Miguel', 'Anderson', 'Joaquim', 'Inácio']
Paulo
```



Uma **tupla** é muito parecida com uma lista, com uma diferença fundamental: Ela é geralmente imutável, ou seja, ao contrário de uma lista, normalmente seu conteúdo não pode ser alterado, seja adicionado, excluindo ou alterando dados. Isso é para garantir que a informação fique sempre a mesma, por exemplo, dados de clientes ou consumidores.

```
In [1]: #Exemplo inicial de tupla
        tupla1=(1,2,3,4,5)
        print(tupla1)
        type(tupla1)
```

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

```
Out[1]: tuple
```

```
In [2]: #Métodos disponíveis para trabalhar com tuplas
        dir(tuple)
```

```
Out[2]: ['__add__',
         '__class__',
         '__class_getitem__',
         '__contains__',
         '__delattr__',
         '__dir__',
         '__doc__',
         '__eq__',
         '__format__',
         '__ge__',
         '__getattribute__',
         '__getitem__',
         '__getnewargs__',
         '__gt__',
         '__hash__',
         '__init__',
         '__init_subclass__',
         ...]
```



É perfeitamente possível termos uma lista de tuplas, que, nesse caso, nada mais do que conjuntos de informações imutáveis de 2 clientes, conforme exemplo abaixo:

```
In [10]: #Mais um exemplo de tupla
cliente=("Marcelo Fernando", "166.765.876-99", "38.767.876-1")
type(cliente)
clientes=[]
clientes.append(cliente)
print(clientes)
type(clientes)
len(clientes)
```

```
[('Marcelo Fernando', '166.765.876-99', '38.767.876-1')]
```

Out[10]: 1

```
In [11]: #Incluindo mais dados de clientes
cliente1=("Robson Custela Assada", "676.313.098-00", "54.314.443-1")
clientes.append(cliente1)
print(clientes)
len(clientes)
```

```
[('Marcelo Fernando', '166.765.876-99', '38.767.876-1'), ('Robson Custela Assada', '676.313.098-00', '54.314.443-1')]
```

Out[11]: 2



Conjuntos (ou sets) nada mais são que uma coleção de itens distintos (únicos).

```
In [10]: #Exemplo de sets
numeros = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5]
numeros_distintos = set(numeros)
print("Números: " , numeros)
print("Números distintos: ", numeros_distintos)
print(type(numeros))
print(type(numeros_distintos))

Números: [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5]
Números distintos: {1, 2, 3, 4, 5}
<class 'list'>
<class 'set'>
```

Observem que quando criamos um conjunto, ele exibe os elementos únicos de uma lista.

Removendo itens de um conjunto

```
In [15]: #Removendo itens de um conjunto
numeros1=set([1,2,2,3,4])
numeros1.remove(2)
print(numeros1)
numeros2=set(["MG", "SC", "RJ", "SP","ES"])
numeros2.remove("SC")
print(numeros2)
```

```
{1, 3, 4}
{'MG', 'ES', 'SP', 'RJ'}
```

```
In [18]: #Removendo itens de um conjunto - ERRO NA SINTAXE
conjunto3=set([10,20,20,30,30,40])
conjunto3.remove(50)
print(conjunto3)
```

```
-----
KeyError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_22484\4149327364.py in <module>
      1 #Removendo itens de um conjunto - ERRO NA SINTAXE
      2 conjunto3=set([10,20,20,30,30,40])
----> 3 conjunto3.remove(50)
      4 print(conjunto3)

KeyError: 50
```

Observem que, no exemplo da direita, estamos tentando remover um item que não está no conjunto (no caso, o 50). Por conta disso, ele aponta uma mensagem de erro na tela.

Operações matemáticas com conjuntos

Símbolo matemático	Operador Python	Descrição
$e \in S$	<code>in</code>	elemento e é membro de S
$A \subseteq B$	<code><=</code>	A é um subconjunto de B
$A \subset B$	<code><</code>	A é um subconjunto próprio de B
$A \cup B$	<code> </code>	A união com B
$A \cap B$	<code>&</code>	A interseção com B
$A \setminus B$	<code>-</code>	diferença entre A e B

Algumas operações matemáticas, derivadas da teoria dos conjuntos, podem ser aplicadas , a partir da formação de estruturas de dados do tipo “**conjuntos**”.

Fonte: <https://algoritmoempython.com.br/cursos/programacao-python/conjuntos/>

Operações matemáticas com conjuntos

```
In [21]: #União de 2 conjuntos
A = {0, 1, 3, 5, 7, 9, 11, 22}
B = {0, 2, 4, 6, 8, 19}
C = A | B
C1 = A.union(B)
print(C)
print(C1)

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 19, 22}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 19, 22}
```

```
In [25]: #Intersecção de 2 conjuntos
A = {0, 1, 3, 5, 7, 9, 11, 22}
B = {0, 2, 4, 6, 8, 19}
C = A & B
C1 = A.intersection(B)
print(C)
print(C1)

{0}
{0}
```

Observem que a união entre A e B junta todos os elementos de A e B, enquanto a intersecção entre A e B exibe os elementos comuns entre esses 2 conjuntos.



Os dicionários são coleções de itens e seus elementos são armazenados de forma não ordenada. Os elementos de um dicionário são **chave** e **valor**, em que a chave servirá para indexar ou posicionar determinado elemento no dicionário, enquanto o valor poderá receber elementos diversos como listas, números, strings, dicionários, etc.

Sua sintaxe fundamental é **{'chave': 'valor'}**

```
In [2]: #Exemplo de um dicionário
dic1={"Marcelo":45, "Paulinha":38, "Ângela":34}
type(dic1)
```

```
Out[2]: dict
```

```
In [4]: #Exemplo adicional de um dicionário
dic2={"255.313.142-78": 5500, "671.414.414-88": 9876, "166.765.877-89": 4100}
type(dic2)
```

```
Out[4]: dict
```

Acessando as chaves de um dicionário:

```
In [5]: #Acessando as chaves de um dicionário
print(dic2.keys())

dict_keys(['255.313.142-78', '671.414.414-88', '166.765.877-89'])
```

Acessando os valores de um dicionário:

```
In [6]: #Acessando os valores de um dicionário
print(dic2.values())

dict_values([5500, 9876, 4100])
```

Acessando um valor específico associado a uma chave:

```
In [7]: #Acessando um dado específico do dicionário
print(dic2.get("671.414.414-88"))

9876
```

Interessante mencionar que esse racional de chave-valor é o princípio básico dos bancos não-relacionais (NoSQL), em que a modelagem do banco indexa os dados a uma chave.

Adicionando dados a um dicionário:

```
In [9]: #Adicionando dados a um dicionário
dic2.update({"111.222.333-44": 10000})
print(dic2)
```

```
{'255.313.142-78': 5500, '671.414.414-88': 9876, '166.765.877-89': 4100, '111.222.333-44': 10000}
```

Atualizando dados de um dicionário:

```
In [10]: #Atualizando dados de um dicionário
dic2.update({"255.313.142-78": 7700})
print(dic2)
```

```
{'255.313.142-78': 7700, '671.414.414-88': 9876, '166.765.877-89': 4100, '111.222.333-44': 10000}
```

Excluindo elemento de um dicionário:

```
In [23]: #Excluindo dados de um dicionário  
dic2.pop("166.765.877-89")  
print(dic2)
```

```
{'255.313.142-78': 5500, '671.414.414-88': 9876, '111.222.333-44': 10000}
```

Avaliando o tamanho de um dicionário:

```
In [33]: #Avaliando o tamanho de um dicionário  
print("O tamanho do dicionário é: ", len(dic2))
```

```
O tamanho do dicionário é: 3
```


5ª Sessão “Mão na Massa” (tempo: 15 minutos)

Realize as seguintes atividades usando um notebook Jupyter

- a) Crie uma lista com os números 1,3,5,7,9,11,13,15,17,19 e 21. a.1) Qual o tamanho da lista? a.2) Qual o máximo valor da lista? a.3) Qual a média dos elementos dessa lista? a.4) Ordene a lista em ordem decrescente. a.5) Adicione o elemento 10 a essa lista.
- b) Considere os conjuntos $A = \{100, 200, 300, 500, 600, 800\}$, $B = \{150, 200, 250, 350, 400, 500\}$ e $C = \{300, 200\}$. b.1) Qual a soma dos valores do conjunto D que é a intersecção de A, B e C? b.2) Qual a média da união dos elementos desses 3 conjuntos?
- c) Crie um dicionário com chaves & valores associados à população dos estados do Sul do Brasil. Qual a média dos valores desse dicionário?

