

Tópico 04

Introdução à programação de computadores

Introdução a python

1. Introdução

Aluna(o), é importante que você entenda a necessidade do Python no mundo computacional!

Cada linguagem de programação tem sua função específica. Linguagens podem ter foco na facilidade de programar, na velocidade do código, no acesso ao *hardware*, na resolução de problemas estatísticos, na análise de grande quantidade de dados etc.

Python é uma linguagem muito cara ao corpo científico por diversas razões. É uma linguagem que facilita uma diversidade enorme de operações e muito útil para análise de dados, aprendizado de máquina, *big data*, entre outras.



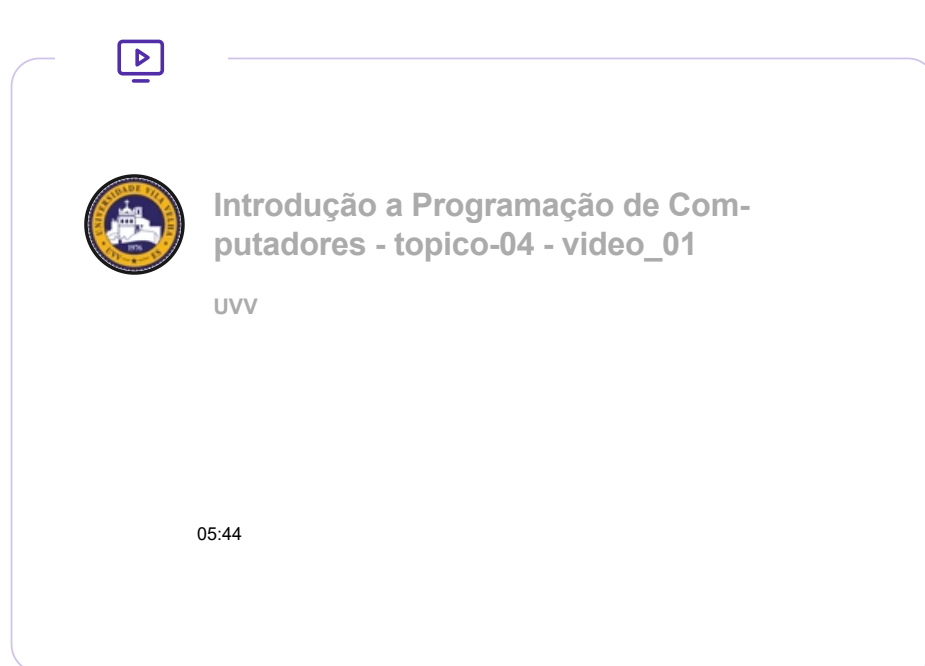
Vale notar que outras linguagens são úteis para as mesmas finalidades, no entanto, Python é uma linguagem amplamente difundida, especialmente para pesquisas, e é colaborativa e gratuita.

Assim, temos uma linguagem de altíssimo potencial bem em nossas mãos. No entanto, não se engane, Python tem suas limitações. Por ser uma linguagem que procura facilitar operações que são naturalmente complexas, ela é insuficiente em termos de velocidade, quando comparada a linguagens de baixo nível, como C. Para tal, é necessário utilizar um processador mais robusto e com mais memória, ou renunciar à performance (quando for possível).

2. Ambiente python

Python é uma linguagem de programação de computadores. É de fácil aprendizagem e muito poderosa. É uma programação de alto nível, e é classificada como orientada ao objeto.

A linguagem Python é de uso geral, o que significa que nós podemos desenvolver praticamente qualquer tipo de software utilizando Python. Quando trabalhamos com *data science*, precisamos de uma ferramenta para coletar os dados, limpar os dados, transformar, fazer o pré-processamento, criar modelos preditivos, realizar avaliações de modelos preditivos, e criar gráficos e dashboards. O Python permite realizar todas essas atividades, e essa é uma das razões pelas quais os cientistas de dados utilizam-no.



Além disso, o Python é uma linguagem de programação comum, ou seja, facilmente encontraremos equipes nas mais diversas empresas utilizando-a, e ela pode ser utilizada para diversas atividades, desde análises mais simples de dados, como a criação de uma aplicação web e *dashboards* ou para criar aplicações de inteligência artificial.

O Python é uma das linguagens mais poderosas para análise de dados, devido ao que chamamos de *PyData Stack*, que, resumidamente, é um conjunto de pacotes ou bibliotecas que foram criados por diversas pessoas, em linguagem Python, e disponibilizados gratuitamente.

Pacotes que compõem o PyData Stack.

- NumPy (*numerical Python*) – talvez a biblioteca mais famosa. O Pacote tem diversas instruções para operações matemáticas que são as bases de *machine learning*. Possui inúmeras operações matemáticas prontas para uso.
- Pandas – podemos dizer que é o Excel para o Python. trabalhamos com dados TABULARES facilmente.
- Scikit learn – é o pacote para *machine learning*. Com ele, já estão implementados diversos métodos, algoritmos e técnicas bem interessantes que simplificam a vida do desenvolvedor.
- Matplotlib – utilizado para trabalhar com os mais diversos gráficos e visualizações de dados.



Devido a popularidade do Python, podemos trabalhar de forma on-line, ou seja, conectados à internet, de forma que não precisamos realizar nenhuma instalação na nossa máquina local. Além disso, também podemos baixar e instalar gratuitamente todos os recursos necessários do Python, para poder operar em nossa própria máquina.

De forma simples, para utilizarmos o Python em nossa máquina local, precisamos de dois itens que são:

- O interpretador da linguagem.
- Local para escrevermos nossos códigos.

O interpretador é um programa que aceita comandos escritos em Python e os executa, linha a linha. É ele quem vai traduzir nossos programas em um formato que pode ser executado pelo computador (MENEZES, 2019).

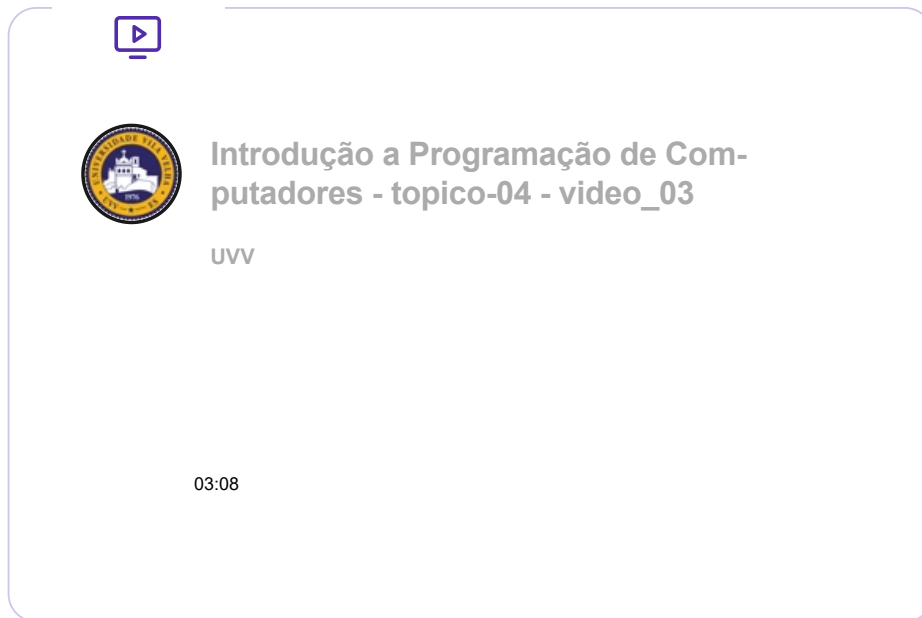
Após instalarmos o interpretador da linguagem, que pode ser obtido diretamente na página da linguagem Python (www.python.org), uma das opções para escrever nossos códigos é um editor de textos. Podemos escrever diretamente em um arquivo *notepad* (o editor do Windows, por exemplo). Porém, existem diversas ferramentas mais indicadas para tal.



Introdução a Programação de Computadores - topico-04 - video_02

UVV

05:33



Outra forma de escrevermos em Python, é utilizando o Jupyter Notebook. Ele pode ser obtido diretamente na página do Jupyter (www.jupyter.org), ou junto com a IDE Anaconda, que inclusive disponibiliza junto o interpretador Python e, durante sua instalação, já estrutura todo nosso ambiente de desenvolvimento. O Jupyter Notebook roda via *browser web*, porém, depende do interpretador Python que deve estar instalado em sua máquina local. É uma excelente forma de trabalhar com *data science*, inclusive porque documenta todo o processo. Temos também o PyCharm, que tem como objetivo o desenvolvimento mais avançado, para criar softwares mais robustos.



Outra forma de trabalharmos com Python é utilizando a ferramenta Colab. O Colab se assemelha ao Jupyter Notebook, rodando via web, porém, utiliza recursos totalmente on-line. O Colab é o ambiente de colaboração do Google, e o interpretador Python está instalado nos servidores, que disponibiliza para nós toda estrutura necessária para trabalharmos com a linguagem, de forma gratuita.



Introdução a Programação de Computadores - topico-04 - video_04

UVV

04:15



Introdução a Programação de Computadores - topico-04 - video_05

UVV

04:32







Linguagem Python e suas principais IDEs.



É importante destacar que nós utilizaremos, neste tópico, o arquivo com todos nossos códigos, cuja extensão é “.ipynb”, que é o formato específico de um notebook Python (Colab, Jupyter, entre outros). Ao escrevermos um *script* em Python, a extensão é “.py”.

Introdução a Programação de Computadores - topico-04 - video_06

UUV

04:33



Nome	Tipo	Tamanho
hello.py	Arquivo PY	1 KB
TOPICO_IV.ipynb	Arquivo IPYNB	9 KB

script python

Notebook (Colab, Jupyter, entre outros)

Modelo de extensões de arquivo.

O IPython (cuja extensão é o “ipynb”) pode fazer tudo que o Python faz. Ele oferece recursos extras, como conclusão de tabulação, testes, depuração, entre outros, diferentemente de um script Python (“.py”), com o qual apenas escrevemos códigos. Podemos pensar no IPython como sendo uma interface gráfica para a linguagem Python.

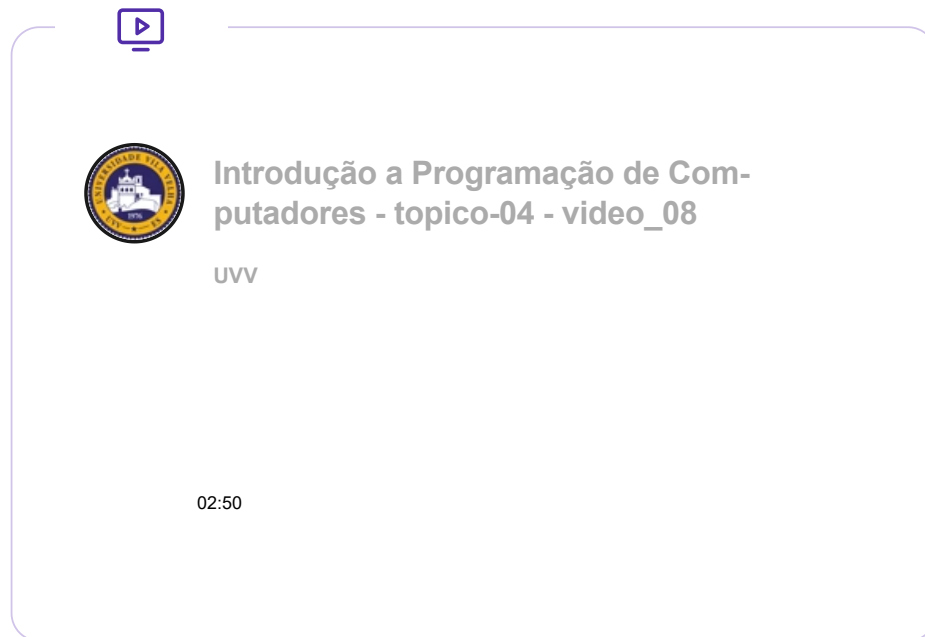




Introdução a Programação de Computadores - topico-04 - video_07

UUV

04:08

3. Tipos de dados e variáveis



Em Python, existem os tipos básicos e os tipos de alto nível de dados e variáveis.



E o que são tipos? São categorias de valores que são processados de forma semelhante. Exemplos:

- **Tipos primitivos:** são aqueles já embutidos no núcleo da linguagem.
- **Tipos simples:** números (int, long, float, complex) e cadeias de caracteres.
- **Tipos compostos:** listas, dicionários, tuplas e conjuntos.
- **Tipos definidos pelo usuário:** são correspondentes a classes (orientação objeto).

O que são **variáveis**? São nomes dados a áreas de memória.

- Nomes podem ser compostos de algarismos, letras ou `_`.
- O primeiro caractere não pode ser um algarismo.

- Não pode usar palavras reservadas (if, while etc.).
- Servem para guardar valores intermediários.
- Servem para construir estruturas de dados.

Entre os tipos básicos podemos citar alguns. O primeiro deles os **números**.

Há vários tipos numéricos que se pode usar em Python:

- **Int**: números inteiros..
- 2 , 3, -10 , 100
- **Floats**: números racionais de precisão variável.
- 1.0 , 10.5 , -19.00005 , 15e-5
- **Complex**: números complexos.
- 1+1j , 20j , 1000+100J



Um tipo não numérico é a **string**:

- São cadeias de caracteres.
- Constituem outro tipo fundamental do Python.
- Constantes string são escritas usando aspas simples ou duplas.
- “a” ou ‘a’
- Python usa a tabela *default* do sistema operacional.
- ASCII, UTF-8
- Caracteres não imprimíveis podem ser expressos usando notação “barra invertida” (\)
- \n : nova linha

- `\r` : *carriage return*
- `\t` : *tab*
- `\b` : *backspace*
- `\\` : `\`
- `\x41` : caractere cujo código hexadecimal é 41 (“A”)

Outro tipo básico não numérico são as **expressões booleanas**:

- Resultam em Verdadeiro (TRUE) ou Falso (FALSE).
- São usadas em comandos condicionais e de repetição.
- Servem para analisar o estado de uma computação e permitir escolher o próximo passo.

Temos também os **tipos de alto nível**:

- **Listas**: vetor que pode armazenar todo tipo de valor.
- **Tuplas**: lista imutável. Qualquer operação sobre ela resulta em uma nova tupla.
- **Dicionários**: sequências que podem utilizar índices (imutáveis) de tipos variados.
- **Arquivo**: Python possui um tipo pré-definido para manipular arquivos.
- **Classes e instâncias**.





The screenshot shows a video player interface. On the left, there is a sidebar with a logo and a table of contents. The main area displays the title 'Introdução a Programação de Computadores - topico-04 - video_09' and the acronym 'UVV'. Below the title, there is a list of Python keywords: yield, assert, else, import, pass, break, except, in, and raise. At the bottom, a code editor shows a Python script with a syntax error: `True = "teste"`. The error message is 'SyntaxError: can't assign to keyword'.

4. Entrada e saída de dados



The screenshot shows a video player interface. On the left, there is a sidebar with a logo and a table of contents. The main area displays the title 'Introdução a Programação de Computadores - topico-04 - video_10' and the acronym 'UVV'. Below the title, there is a list of Python keywords: yield, assert, else, import, pass, break, except, in, and raise. At the bottom, a code editor shows a Python script with a syntax error: `True = "teste"`. The error message is 'SyntaxError: can't assign to keyword'.

Aqui vamos analisar como se dá a entrada e saída de dados em um código Python. Lembre-se de que o interpretador Python não compila o código, ele executa linha por linha, conforme mostramos a seguir:

“Python é uma linguagem interpretada. O interpretador Python executa um comando por vez para funcionar o

código”.

(Wes McKinney, 2ª edição: 2017, página 16).

Chamada de código

É possível chamar o código Python pela linha de comando, e isso é o suficiente para executá-lo. Imaginemos, por exemplo, que temos um arquivo chamado *hello_world.py*, cujo código seja:

```
print('Hello world')
```

Você pode executar o código apenas digitando:

```
Python hello_world.py
```

e terá como saída:

```
Hello world
```

Em um ambiente IPython, é possível executar o código com o comando `%run`. Exemplo, se temos um código chamado *iPython_teste.py*, podemos executá-lo com `%run iPython_teste.py`.



Comentários

Os comentários em Python são feitos com o operador `#`.

```
# isto é um comentário
```

Outra forma de fazer comentários é colocando três aspas duplas antes e depois dos comentários. Dessa forma, é possível fazer comentários em mais de uma linha, sem usar um `#` por linha

```
"""  
isto é um comentário  
muito maior  
"""
```

Indentação

“Python usa espaços em branco (tabulações ou espaços) para estruturar o código em vez de usar colchetes como em muitas outras linguagens”.

(Wes McKinney, 2ª edição: 2017, página 30).

Dessa forma, a linguagem Python procura exigir organização básica de quem programa.

```
for x in array:
    if x < pivot:
        less.append(x)
    else:
        greater.append(x)
```

Exemplo de estruturação de código Python.

SAÍDA DE DADOS

Podemos exibir um resultado por meio do comando print.



The screenshot shows a code editor with a play button icon on the left. The code is: `a = "Hello"`, `b = "World"`, and `print (a, b)`. Below the code, the output is displayed as `Hello World`.

Saída de dados pelo print.

Outra forma é fazer diretamente com o código print:



The screenshot shows a code editor with a play button icon on the left. The code is: `print ("Hello World")`. Below the code, the output is displayed as `Hello World`.

Saída de dados diretamente pelo PRINT.

Ou, até mesmo, sem o comando print:



Saída de dados sem o comando
print.

O mesmo pode ser feito com a criação de um MÓDULO (como
são chamados os programas em Python):

```
# Primeiro programa Python: hello.py  
a = "Hello"  
b = "World"  
print (a, b)
```

Saída de dados através de um módulo.

Se executamos o comando <Python hello.py>, teremos:

- Hello World

Outra forma de usar o print é mostrando que dado que vamos
imprimir juntamente com a string que vamos imprimir.

Exemplo:

dia = 20

mes = "novembro"

print ("Eu nasci no dia %d de %s" %(dia,mes))

[OUT]:

Eu nasci no dia 20 de novembro

Assim, mostramos com o %d que será inserido um inteiro (dia) e
com o %s que será inserida uma string (mês).

Podemos utilizar outras formas como %f (float); %x
(hexadecimal); %c (caractere).

Outra forma, caso não queiramos especificar o tipo de dado a ser
inserido:

dia = 20

mes = "novembro"

print(f"Eu nasci no dia {dia} de {mes}")



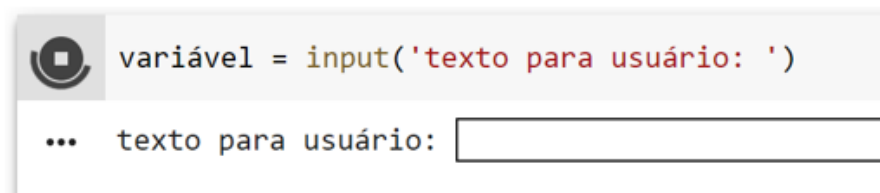
[OUT]:

Eu nasci no dia 20 de novembro

ENTRADA DE DADOS

Para entrada de dados pelo usuário, utilizamos o comando INPUT, fazendo a atribuição de valor diretamente para uma variável.

A sintaxe é:



Entrada de dado do usuário.

Assim, a <variável> terá uma variável do tipo STRING salva.

```
variável = input('texto para usuário: ')  
print(variável)
```

```
[OUT]:  
texto para usuário: texto  
texto
```



Vamos, agora, aprender um pouco sobre a atribuição de valores, para dar sequência na entrada de dados. Para atribuir um dado ou argumento, modificamos a variável usando o comando de atribuição (=).

- Var = expressão

Por exemplo:



Atribuição de um
dado a uma

VARIÁVEL.

É possível também atribuir a várias variáveis, simultaneamente:

```
a = 1
a,b = 3*a,a
print(a,b)
a,b=b,a
print(a,b)
```

```
3 1
1 3
```

Atribuição de dados a mais de uma VARIÁVEL.

Python possui tipagem dinâmica, ou seja, a tipagem pode mudar a cada nova entrada de dados em uma variável.

```
a = 1
print(type(a))
a = 'banana'
print(type(a))
a = 1.0
print(type(a))
```

```
<class 'int'>
<class 'str'>
<class 'float'>
```



TIPAGEM DINÂMICA.

E quais são as possíveis formas de **TIPAGEM**?



Tipar um dado é dizer a que grupo ele pertence

- **Tipagem Forte:** é necessário declarar explicitamente o tipo da variável que você está

alocando.

- **Tipagem fraca:** não é necessário explicitar o tipo de dado (há autotipagem).
- **Tipagem Dinâmica:** além de não ser necessário explicitar o tipo de dado, é possível modificar o tipo quando quisermos.

No entanto, a tipagem do Python tem suas limitações, o que chamamos de “REFERÊNCIA DINÂMICA, TIPO FORTE”.





“Alguns observadores podem concluir rapidamente que Python não é uma “Linguagem tipada.” Isso não é verdade”

Wes McKinney, 2ª edição: 2017, página 34.



A sequência abaixo resulta em um erro de tipagem pois não é possível somar uma string a um inteiro.

```
 '5' + 5
```

```
 -----  
TypeError                                Traceback  
<ipython-input-14-4dd8efb5fac1> in <module>()  
----> 1 '5' + 5  
  
TypeError: must be str, not int
```

ERRO DE TIPAGEM.

Por fim, uma vez que o comando INPUT sempre insere o dado na variável no formato STRING, é necessário converter o tipo da variável por conta de um processo chamado casting. Exemplo:

```
idade_cachorro = input('idade do cachorro: ')\nidade_humano = idade_cachorro*7\nprint(idade_humano)
```

```
[OUT]:\nidade do cachorro: 3\n3333333
```

Repare que, em vez de multiplicar a idade por 7, o código replicou a idade 7 vezes. Isso se dá porque a idade do cachorro é uma STRING.

Devemos colocar o input dentro de INT(), para que a idade seja devidamente convertida:

```
idade_cachorro = int(input('idade do cachorro: '))\nidade_humano = idade_cachorro*7\nprint(idade_humano)
```

```
[OUT]:\nidade do cachorro: 3\n21
```

Dessa vez, a idade foi devidamente convertida. O mesmo ocorre, por exemplo, para FLOAT – float(), entre outras.



5. Operadores





Introdução a Programação de Computadores - topico-04 - video_12

Con: UUV tem ser escritas com a mesma notação usada na linguagem C

- Hexadecimal: preceder os dígitos com 0x
- Octal: preceder os dígitos com 0o
- Binário: preceder os dígitos com 0b

```
[46] 0o22 # Octal
```

```
↳ 18
```

```
[47] 0x1f # Hexadecimal
```

```
↳ 31
```

```
[48] 0b1101 # Binário
```

```
↳ 13
```

FIGURA No. 1

Podemos usar o código Python como uma calculadora, por exemplo:

- + Adição
- - Subtração
- * Multiplicação
- / Divisão
- ** Potência
- // Divisão que retorna o quociente inteiro
- % Retorna o resto inteiro da divisão



<pre>[31] 10 + 5</pre>	<pre>[34] 10 / 3</pre>
<pre>↳ 15</pre>	<pre>↳ 3.3333333333333335</pre>
<pre>[32] 10 - 15</pre>	<pre>[35] # Quociente</pre>
<pre>↳ -5</pre>	<pre>10 // 3</pre>
<pre>[33] 10 * 3</pre>	<pre>↳ 3</pre>
<pre>↳ 30</pre>	<pre># Resto</pre>
	<pre>10 % 3</pre>
	<pre>↳ 1</pre>

OPERAÇÕES ARITMÉTICAS.

Constantes podem ser escritas com a mesma notação usada na linguagem C.

– *Hexadecimal* : *preceder os dígitos com 0x*

– *Octal* : *preceder os dígitos com 0o*

– *Binário* : *preceder os dígitos com 0b*

```
[46] 0o22 # Octal
```

```
↳ 18
```

```
[47] 0x1f # Hexadecimal
```

```
↳ 31
```

```
[48] 0b1101 # Binário
```

```
↳ 13
```

BASES OCTAIS,
HEXADECIMAIS E
BINÁRIAS.



Números de ponto flutuante

Para gerar uma constante de ponto flutuante, é preciso escrever com um ponto

decimal, ou usar a letra e (ou E), precedendo a potência de 10

```
[49] 10 # Inteiro
```

```
↳ 10
```

```
[50] 10.0 # Ponto flutuante
```

```
↳ 10.0
```

```
[51] 99e3
```

```
↳ 99000.0
```

```
[52] 99E-3
```

```
↳ 0.099
```

Ponto flutuante.

Números complexos

- Representados com dois números: um para a parte real e outro para a parte imaginária.
- Constantes são escritas como uma soma sendo que a parte imaginária tem o sufixo **j** ou **J**.

```
[53] 1+2j      [56] (1+2j)*3j
    ↪ (1+2j)    ↪ (-6+3j)

[54] 1+2j*3    [57] 2j
    ↪ (1+6j)    ↪ 2j

[55] (1+2j)*3
    ↪ (3+6j)
```

Números complexos.

STRING

Os caracteres não imprimíveis só desempenham sua função se estiverem dentro do *CÓDIGO PRINT*.



```
[58] "ab\r d"
    ↪ 'ab\r d'

[59] print("ab\r d")
    ↪ d

[60] print("abc\td")
    ↪ abc    d

[62] print("abc\nd")
    ↪ abc
    ↪ d
```

CARACTERES NÃO
IMPRIMÍVEIS.

A notação barra invertida (\) pode ser desabilitada quando a constante string é precedida pela letra r minúscula.

```
[63] print("abc\nd")  
      print(r"abc\nd")
```

```
↳ abc  
   d  
   abc\nd
```

Desabilitar a barra invertida.

Podemos escrever constantes `STRING` usando três aspas duplas (`"""`), para não termos de escrever as quebras de linha, ou uma barra invertida, ao fim de cada linha.

```
▶ print("uma maçã\nduas maçãs")  
  print("""  
    uma maçã  
    duas maçãs""")
```

```
↳ uma maçã  
   duas maçãs  
  
   uma maçã  
   duas maçãs
```

O uso das três aspas duplas (`"""`).



índices

- Notação: ***string***[***índice***] .
- O primeiro caractere tem índice **0**.
- O último caractere tem índice **-1**.

```
▶ a = "teste de índice"  
  print(a[0])  
  print(a[-1])
```

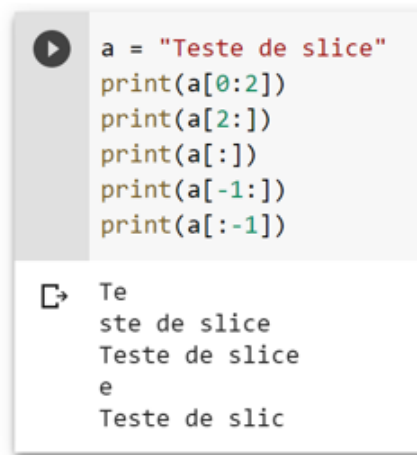
```
↳ t  
   e
```

Uso de índice em string (`"""`).

fatias (slices)

- **Notação:** `string[índice1 : índice2 : iteração]` .
- Retorna os caracteres desde o de ÍNDICE 1 (inclusivo) até o de ÍNDICE 2 (exclusivo).
- Se o primeiro índice é omitido, é assumido como 0
- Se o segundo índice é omitido, é assumido o fim da STRING.

ITERAÇÃO ou passo é como caminha a fatia.



```
a = "Teste de slice"
print(a[0:2])
print(a[2:])
print(a[:])
print(a[-1:])
print(a[:-1])
```

Te
ste de slice
Teste de slice
e
Teste de slic

Uso da função slice



Expressões booleanas

- Operadores mais usados
 - **Relacionais:** `>`, `<`, `==`, `!=`, `>=`, `<=`
 - **Booleanos:** *and*, *or*, *not*.
- Avaliação feita em “Curto-circuito”
 - Expressão avaliada da esquerda para a direita.
 - Se o resultado (*V* ou *F*) puder ser determinado sem avaliar o restante, ele é retornado imediatamente.
- As constantes `TRUE` e `false` são apenas símbolos convenientes.
- Qualquer valor não nulo é visto como verdadeiro, enquanto 0 (ou `FALSE`) é visto como falso.

- O operador OR retorna o primeiro operando se for visto como *verdadeiro*, caso contrário, retorna o segundo.
- O operador AND retorna o primeiro operando, se for visto como *falso*, caso contrário, retorna o segundo.
- Operadores relacionais são avaliados antes de NOT, que é avaliado antes de AND, que é avaliado antes de OR.

[68] <code>1==1</code> ☞ True	[74] <code>False and 3</code> ☞ False
[69] <code>1<2 or 3<2</code> ☞ True	[75] <code>"abc" or 1</code> ☞ 'abc'
[70] <code>1<2 and 3<2</code> ☞ False	[76] <code>1 and not 0</code> ☞ True
[71] <code>0 or 2 and 3</code> ☞ 3	▶ <code>"primeiro" and 1</code> ☞ 1
▶ <code>0 and 2 and 3</code> ☞ 0	

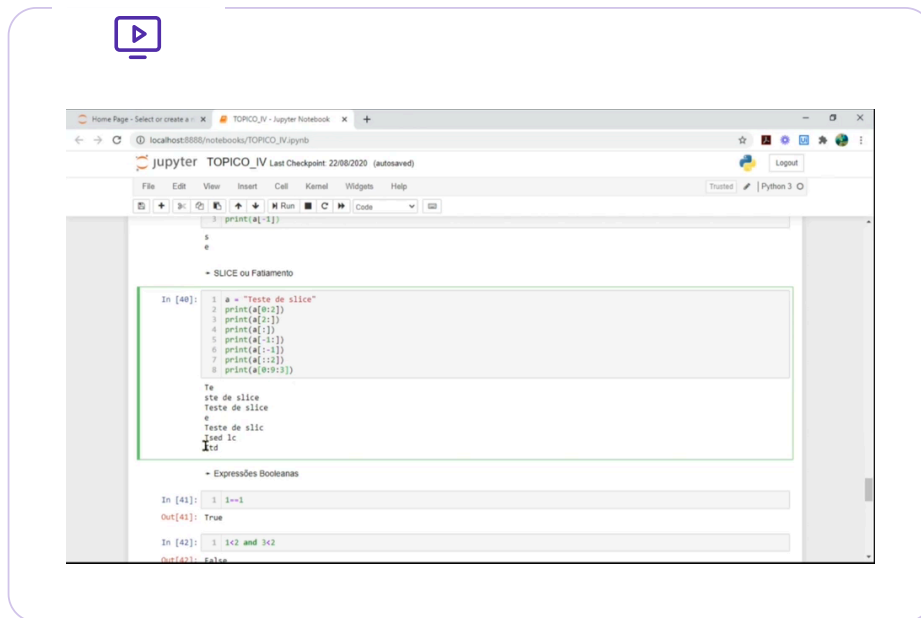


Operadores booleanos

Módulos

Além dos operadores, podemos usar funções para fazer determinadas operações. Aqui vão alguns exemplos e formas de uso:

- As funções ***abs()***, ***chr()*** e ***ord()*** são funções embutidas do Python
 - ***abs(x)*** retorna o valor absoluto ***x***
 - ***chr(y)*** retorna o caractere cujo código ASCII é ***y***
 - ***ord(z)*** retorna o código ASCII do caractere ***z***



6. Conclusão

Este tópico procurou mostrar alguns exemplos básicos de como programar usando a linguagem Python, e a importância dessa linguagem.



Buscamos aprender conceitos e regras sobre a linguagem e sua estrutura. A partir desses conceitos, vamos poder construir códigos mais complexos, e até mesmo levar isso para outras linguagens de programação, quando necessário.

Assim, procuramos fundamentar as bases da compreensão de um tópico importante, que será utilizado frequentemente daqui para frente em todas as linguagens de programação.

7. Referências

McKinney, Wes. Python for Data Analysis. United States of America: O'Reilly, 2017.

MENEZES, Nilo Ney Coutinho. **Introdução à programação com Python:** algoritmos e lógica de programação para

iniciantes. 3. ed. rev. amp. São Paulo, SP: Novatec, 2019. 328 p.
ISBN 9788575227183.



Parabéns, esta aula foi
concluída!

O que achou do conteúdo estudado?



Péssimo

Ruim

Normal

Bom

Excelente

Deixe aqui seu comentário

Mínimo de caracteres: 0/150

Enviar