

# APOSTILA

Python Impressor | versão 1.0



# O FORMATO

que vai fazer você relembrar tudinho



## Módulo X - Tema A - Assunto A

Qual módulo  
está sendo  
explicado

Qual tema está  
sendo explicado

Qual assunto  
está sendo  
explicado

## Print Screen

da aba que está sendo explicada

Página

**Explicação**  
Do tema trabalhado

# O CONTEÚDO

que você precisa para mandar bem



# Conteúdo

## Módulo 1 Introdução

|    |  |   |
|----|--|---|
| 01 | Introdução .....                                     | 1 |
| 02 | Qual é o nível preciso ter para aprender Python..... | 2 |

## Módulo 3 O que é Programação e o que é Python

|    |  |    |
|----|--|----|
| 06 | O que é Programação e o que é Python.....;         | 15 |
| 07 | O que é Lógica de Programação .....                | 17 |
| 08 | Qual a melhor forma de aprender a Programar? ..... | 19 |

## Módulo 5 Mais sobre variáveis

|    |                                      |    |
|----|--------------------------------------|----|
| 14 | Ordem de execução dos programas..... | 45 |
| 15 | Tipos de Variáveis .....             | 50 |
| 16 | Misturando tipos de variáveis.....   | 52 |
| 17 | Comando .Format.....                 | 54 |

## Módulo 2 Instalação

|    |                                     |    |
|----|-------------------------------------|----|
| 03 | Instalando o Python no Windows..... | 44 |
| 04 | Google Colab .....                  | 47 |
| 05 | Dúvidas Frequentes.....             | 50 |

## Módulo 4 Seus primeiros programas em Python

|    |                                      |    |
|----|--------------------------------------|----|
| 09 | Criando seu primeiro programa .....  | 21 |
| 10 | Operações Básicas.....               | 30 |
| 11 | Operações com texto (String) .....   | 36 |
| 12 | Variáveis.....                       | 38 |
| 13 | Input- Pegando dados do usuário..... | 41 |

## Módulo 6 Estrutura do If

|    |                                   |    |
|----|-----------------------------------|----|
| 18 | Condições no Python.....          | 57 |
| 19 | If dentro de if.....              | 62 |
| 20 | Elif.....                         | 65 |
| 21 | Comparadores.....                 | 70 |
| 22 | And e Or.....                     | 72 |
| 23 | Comparações contraintuitivas..... | 77 |

# Conteúdo

## Módulo 7 Strings – Textos e importância no Python

|  |    |
|--|----|
| 24 Por que aprender Strings e a<br>importância pro Python..... | 81 |
| 25 Índice e tamanho da String.....                             | 82 |
| 26 Índice Negativo e Pedaço de String.....                     | 84 |
| 27 Métodos String-Apresentação.....                            | 89 |

## Módulo 9 For – Estrutura de Repetição

|  |     |
|--|-----|
| 37 Estrutura de Repetição For.....;                | 135 |
| 38 For each-Percorrer cada item de uma lista.....  | 143 |
| 39 For e if.....                                   | 145 |
| 40 For com Item e Índice.....                      | 147 |
| 41 For dentro de For.....                          | 149 |
| 42 Break and Continue – Interrompendo um loop..... | 153 |

## Módulo 11 Tuplas Uma "lista" muito útil e imutável

|                             |     |
|-----------------------------|-----|
| 45 Tuplas.....              | 167 |
| 46 Unpacking em Tuplas..... | 169 |
| 47 Lista de Tuplas.....     | 174 |

## Módulo 8 Listas Python – Métodos e Usos

|   |     |
|---|-----|
| 28 Listas em Python.....  | 99  |
| 29 Índices em Lista, Consultando e Modificando<br>Valores ..... | 101 |
| 30 Adicionar e Remover itens da Lista.....                      | 107 |
| 31 Tamanho da lista, Maior, Menor Valor.....                    | 110 |
| 32 Juntar Listas e Ordenar.....                                 | 111 |
| 33 Print e Join em Listas.....                                  | 119 |
| 34 Alterações Incrementais de Variáveis.....                    | 122 |
| 35 Copiar e Igualdade de Listas.....                            | 126 |
| 36 Lista de Listas.....   | 131 |

## Módulo 10 While e Criando um Loop infinito

|                                |     |
|--------------------------------|-----|
| 43 Estrutura While.....        | 157 |
| 44 Loop Infinito no While..... | 161 |

# Conteúdo

## Módulo 12 Dicionários em Python

|  |     |
|--|-----|
| 48 Dicionários em Python.....                                | 178 |
| 49 Pegar item Dicionário e Verificar item Dicionário.....    | 181 |
| 50 Adicionar, Modificar e Remover Itens do Dicionário.....   | 183 |
| 51 For em Dicionários.....                                   | 186 |
| 52 Métodos Úteis Dicionários (Parte 1) –Items.....           | 188 |
| 53 Métodos Úteis Dicionários (Parte2) – Keys and values..... | 190 |
| 54 Zip e Transformando Listas em Dicionários.....            | 200 |

## Módulo 14 Criando suas funções em Python

|  |     |
|--|-----|
| 58 Functions no Python.....                            | 209 |
| 59 Retornar um valor na Function.....                  | 211 |
| 60 Argumentos e parâmetros em uma Function.....        | 212 |
| 61 Vários parâmetros e tipos de parâmetros.....        | 213 |
| 62 Valores padrões para argumentos.....                | 215 |
| 63 Falando um pouco mai dos returns das functions..... | 217 |
| 64 Return com mais de uma informação.....              | 218 |
| 65 Docstrings e annotations.....                       | 220 |
| 66 Exceções e erros em funções.....                    | 222 |
| 67 Múltiplos Argumentos para uma Function.....         | 226 |

## Módulo 13 Iterables

|                              |     |
|------------------------------|-----|
| 55 O que é um iterable?..... | 204 |
| 56 Range.....                | 206 |
| 57 Set.....                  | 207 |

## Módulo 15 Módulos, Bibliotecas e Introdução a Orientação a Objeto

|  |     |
|--|-----|
| 68 O que é Orientação a Objeto e Por que isso importa..... | 230 |
| 69 O que são módulos e qual a importância.....             | 161 |
| 70 Como ler um módulo.....                                 | 233 |
| 71 Módulo time.....  | 234 |
| 72 Módulo de Gráficos – Exibindo Gráficos no Python.....   | 236 |
| 73 Mais edições de Gráficos e Módulo Numpy.....            | 239 |
| 74 Como instalar módulos novos.....                        | 243 |

# Conteúdo

## Módulo 16 List Comprehension

|   |     |
|---|-----|
| 75 O que é List Comprehension.....                            | 246 |
| 76 Exemplo de List Comprehension.....                         | 249 |
| 77 List Comprehension filtrando com if.....                   | 250 |
| 78 Tratando casos em List Comprehension.....                  | 251 |
| 79 List Comprehension não serve apenas para criar listas..... | 252 |

## Módulo 18 Análise de Dados com o Pandas

|                                  |     |
|----------------------------------|-----|
| 84 O que é e para que serve..... | 261 |
| 85 Pandas e csv.....             | 262 |

## Módulo 20 Integração Python - Arquivos e Pastas do Computador

|  |     |
|--|-----|
| 89. Python para Navegar no seu Computador pathlib e shutil ..... | 307 |
| 90. Verificando se um arquivo existe no Computador.....          | 312 |
| 91. Criando pastas e movimentando arquivos.....                  | 314 |

## Módulo 17 Functions em Iterables e a função map

|  |     |
|--|-----|
| 80 Functions em iterables e a função map.....    | 254 |
| 81 Aplicando funcion em iterable no sort.....    | 257 |
| 82 Lambda expressions.....                       | 258 |
| 83 A grande utilidade de Lambda expressions..... | 260 |

## Módulo 19 Integração Python com Arquivos txt de Dados com o Pandas

|                                    |     |
|------------------------------------|-----|
| 86. Lendo Arquivos txt.....        | 302 |
| 87. Escrevendo em Arquivo txt..... | 303 |
| 88. Método With.....               | 305 |

## Módulo 21 Integração Python – E-mail

|   |     |
|---|-----|
| 92. Formas de Integrar Python com E-mail..... | 317 |
| 93. Integração de Python com Gmail.....       | 319 |
| 94. Ajeitando o corpo do E-mail.....          | 322 |
| 95. Integração Python com Outlook.....        | 324 |

# Conteúdo

## Módulo 22 Integração Python - SQL

|  |     |
|--|-----|
| 96. Configurações.....                                 | 326 |
| 97. Importando o banco de Dados para o SQL Server..... | 329 |
| 98. Biblioteca pyodbc .....                            | 331 |
| 99. Cuidados com o SQL e Python.....                   | 334 |

## Módulo 24 Integração Python com APIs e JSON

|  |     |
|--|-----|
| 105.Python e Interações de API.....                  | 350 |
| 106.API de cotação de Moeda.....                     | 354 |
| 107.Mais exercícios de API com Cotação de Moeda..... | 356 |
| 108.Envio de SMS com Python e Twilio.....            | 360 |

## Módulo 26 Ambientes Virtuais

|  |     |
|--|-----|
| 115.O que é um ambiente Virtual.....                     | 384 |
| 116.Navegando no Prompt de Comando.....                  | 385 |
| 117.Criando,Ativando e Removendo Ambientes Virtuais..... | 388 |
| 118.Observações Importantes de Ambientes virtuais.....   | 391 |

## Módulo 23 Integração Python - Web

|   |     |
|---|-----|
| 100. Integrando Python com Web usando Selenium..... | 336 |
| 101.Abrindo Navegador Escondido.....                | 340 |
| 102.Como funciona web-scraping.....                 | 341 |
| 103.Métodos úteis do Selenium.....                  | 343 |
| 104.Preenchendo formulários com Selenium.....       | 348 |

## Módulo 25 Integração Python para Finanças

|   |     |
|---|-----|
| 109.Python para finanças.....   | 365 |
| 110.Retorno IBOV e Média Móvel.....                                       | 368 |
| 111.Pegando Cotações Carteira com Python.....                             | 370 |
| 112.Tratando Problemas de Importação para o dataframe e Normalização..... | 375 |
| 113.Criando Valor da Carteira e Puxando Ibovespa.....                     | 378 |
| 114.Comparando a Carteira com Ibovespa.....                               | 380 |

## Módulo 27 Integração Python com ArcGIS

|  |     |
|--|-----|
| 119.Configurações para integração Python com ArcGIS..... | 393 |
| 120.Criando seus primeiros mapas com a API.....          | 395 |
| 121.Integrando com Usuário e Senha no ArcGIS.....        | 397 |

# Conteúdo

## Módulo 28 Integração Python com Power BI

|  |     |
|--|-----|
| 128.Como funciona a integração de Python com Power BI..... | 399 |
| 129.Configurações Importantes- Python e Power BI.....      | 400 |
| 130.Python para gerar Base de Dados no Power BI .....      | 404 |
| 131.Integração Python com Power BI.....                    | 406 |
| 132.Usando Python para criar Gráficos no Power BI.....     | 410 |
| 133.Gráficos Seaborn no Power BI.....                      | 416 |

## Módulo 30 Orientação a Objetos Completo – Classes e Métodos

|   |     |
|---|-----|
| 139. Instalando o PyCharm.....  | 436 |
| 140. Classes e Objetos.....   | 441 |
| 141. O que é uma Classe no nosso código.....                          | 445 |
| 142. Conceitos Importantes sobre Programação orientada a Objetos..... | 448 |
| 143.Criando nossa 1 <sup>a</sup> Classe.....                          | 450 |
| 144.O que é o self das Classes.....                                   | 454 |
| 145.Criando Métodos dentro das Classes.....                           | 455 |
| 146.Parâmetros nos Métodos das Classes.....                           | 456 |
| 147.Parâmetros no init da Classe.....                                 | 458 |
| 148.Atributos de Classe.....  | 461 |
| 149. Criando um Sistema de Conta Corrente.....                        | 462 |
| 150.Criando Métodos para a Nossa Classe.....                          | 464 |

## Módulo 29 Transformando Python em exe

|   |     |
|---|-----|
| 134.Instruções Importantes.....                     | 420 |
| 135.Python para exe com Arquivos Simples.....       | 421 |
| 136.Diminuindo Tamanho de Arquivos Executáveis..... | 425 |
| 137.Adaptando o código de Arquivos Complexos.....   | 426 |
| 138.Python para exe com Arquivos Complexos.....     | 430 |

|  |     |
|--|-----|
| 151.Métodos Auxiliares Regra de Métodos Simples e Direto ao Ponto.....       | 468 |
| 152.Underline e Métodos não Públicos.....                                    | 471 |
| 153.Reforço de Organização e Padrões de Classe.....                          | 474 |
| 154.Métodos Estáticos e Modificando Classe da Melhor Forma.                  | 475 |
| 155.Usando objetos e instâncias como Parâmetros.....                         | 480 |
| 156.Atributos Não Públicos.....  | 482 |
| 157.Docstring em Classes.....  | 486 |
| 158.Criando outra Classe e Relação entre Classes.....                        | 488 |
| 159.Atributos de Data e Aleatórios.....                                      | 494 |
| 160.Regra- Listar Métodos e atributos de uma Classe.....                     | 498 |
| 161.Separando Programa e arquivo de Classes e Importando nossas Classes..... | 499 |

# Conteúdo

## Módulo 30 Orientação a Objetos Completo – Classes e Métodos

---

|   |     |
|---|-----|
| 162. Criando uma nova classe - Agências.....        | 503 |
| 163. Criando SubClasses.....                        | 507 |
| 164. Init em SubClasses e Herança.....              | 509 |
| 165. Finalizando o init nas outras SubClasses.....  | 511 |
| 166. Métodos Específicos de Uma SubClasse.....      | 514 |
| 167. Personalizando um Método em uma SubClasse..... | 516 |

## Módulo 31 Interface Gráfica – Tkinter e Criando Sistemas com Python

---

|  |     |
|--|-----|
| 168. Criando uma Janela.....                     | 519 |
| 169. Caixa de Mensagem.....                      | 523 |
| 170. Edição de Caixa de Mensagem.....            | 526 |
| 171.Caixa de Texto para o Usuário.....           | 528 |
| 172.Organizando as Janelas do Tkinter.....       | 530 |
| 173.Botões.....                                  | 534 |
| 174.Interação entre Elementos do Tkinter.....    | 536 |
| 175.Listas Suspensas.....                        | 538 |
| 176.Caixa de Textos Grandes.....                 | 539 |
| 177.Checkbox(Checkbutton).....                   | 544 |
| 178.RadioButton(OptionButton).....               | 547 |
| 179.Pedir para o Usuário Selecionar Arquivo..... | 550 |

# BONS ESTUDOS

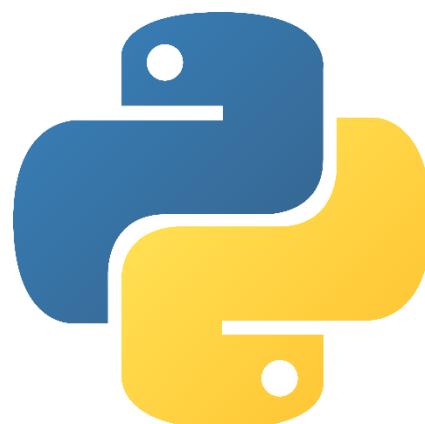
Dúvidas ou Feedbacks nos procure

# Módulo 1

# Introdução

O Python, é uma linguagem de programação que vem sendo utilizada cada vez mais no mercado de trabalho. Além disso, é uma tecnologia “Open Source” que significa dizer que seu código é colaborativo podendo ser melhorado por programadores do mundo todo.

Um dos grandes motivos para esse crescimento é a facilidade dessa linguagem. O Python, foi criado com objetivo de ser intuitivo e muito parecido com o inglês falado. Assim, os códigos são mais fáceis não só de serem elaborados, como também de serem entendidos.



A aplicabilidade do Python no mercado também se tornou um diferencial para essa linguagem de programação. O Python pode ser utilizado para diversas aplicações em todos os mercados, como por exemplo:

- Data Science;
- Criação de Sites;
- Automação de processos;
- Machine Learning;

Outro diferencial é a velocidade de processamento do Python. Muitas vezes tratamento de milhões de dados em planilhas Excel se tornam muito difíceis ou até mesmo inviáveis. Já com o Python esse processamento pode levar apenas alguns segundos.

O Python é uma linguagem de programação que não requer conhecimento prévio em outras linguagens de programação. É claro, que certa afinidade com a lógica de programação irá facilitar o aprendizado.

No entanto, será visto mais a frente, que é possível iniciar o curso do completo zero, passando pelo básico, intermediário e indo até o avançado.

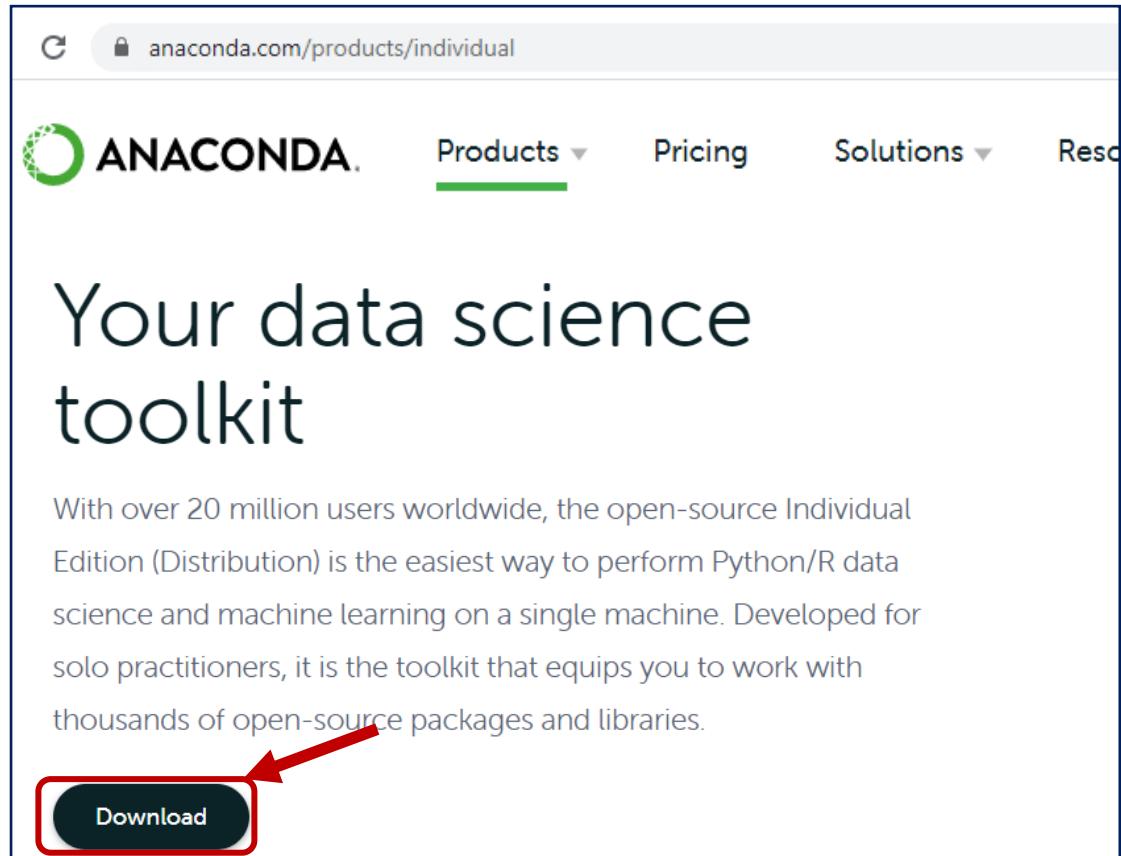
**Então pode ficar tranquilo que o conhecimento em Python ou conhecimento de programação, não é um pré-requisito.**

Você se sentirá totalmente capaz de aplicar os códigos conforme avançamos no conteúdo.



# Módulo 2

# Instalação



Finalmente, vamos dar início. O Python, não é um programa padrão do Windows. Portanto, vamos ter que instalar!

O Python pode ser instalado pelo link <https://www.python.org/downloads/> mas, para o curso, vamos usar o **Anaconda**, que nada não é nada mais que uma grande caixa de ferramentas do Python.

Além do Python, outras ferramentas que nos auxiliam no dia-a-dia já serão instaladas automaticamente. Para baixar, basta acessar o link <https://www.anaconda.com/products/individual> e clicar no botão DOWNLOAD conforme a imagem abaixo:

Após clicar no botão download, você pode seguir o passo a passo das imagens abaixo.

1

Escolher a opção adequada para seu computador

Windows

Python 3.8

64-Bit Graphical Installer (457 MB)

32-Bit Graphical Installer (403 MB)



### ATENÇÃO !

Caso a versão disponível seja superior a 3.8 não tem problema! É só seguir o mesmo passo a passo.

#### Como saber se meu computador é 32 ou 64-Bit?

- Pesquise **Painel de Controle** na barra de pesquisa;



Digite aqui para pesquisar

- Selecione o campo **Sistema e Segurança**;



Sistema e Segurança  
Verificar o status do computador  
Salvar cópias de backup dos arquivos com Histórico de Arquivos  
Backup e Restauração (Windows 7)

- Selecione o campo **Sistema**;



Sistema  
Exibir a quantidade de RAM e a velocidade do processador

- Verificar seu sistema;



Sistema  
Tipo de sistema: Sistema Operacional de 64 bits, processador com base em x64

2

Fazer Download



Anaconda3-2020.1....exe

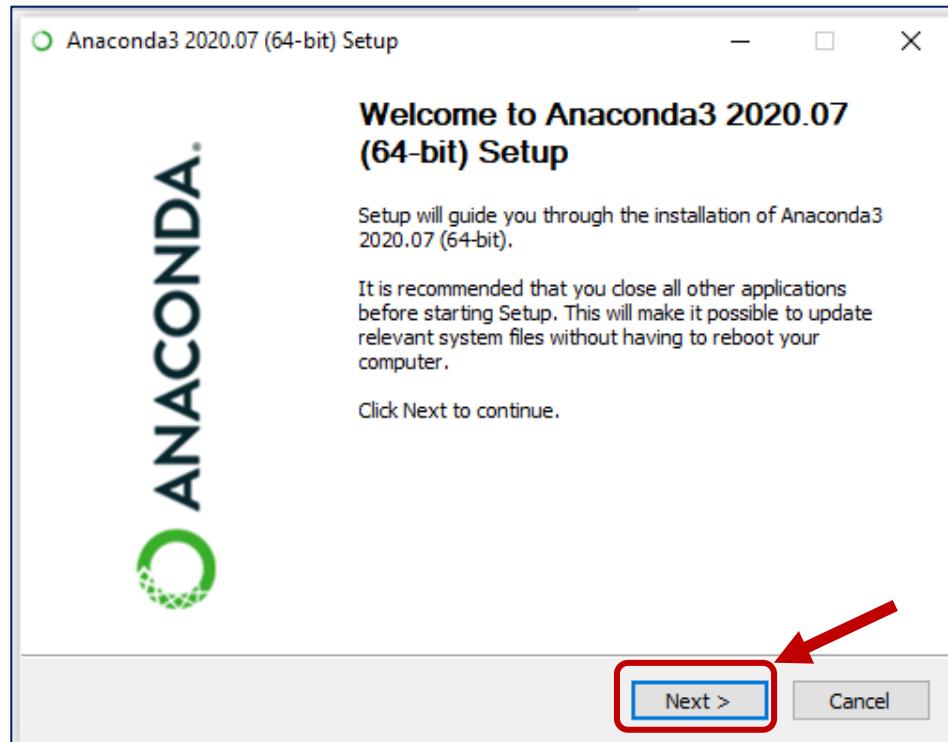
69,0/457 MB, 3 minutos resta...



Segue a instalação daquele jeito bem padrão. Next, Next...

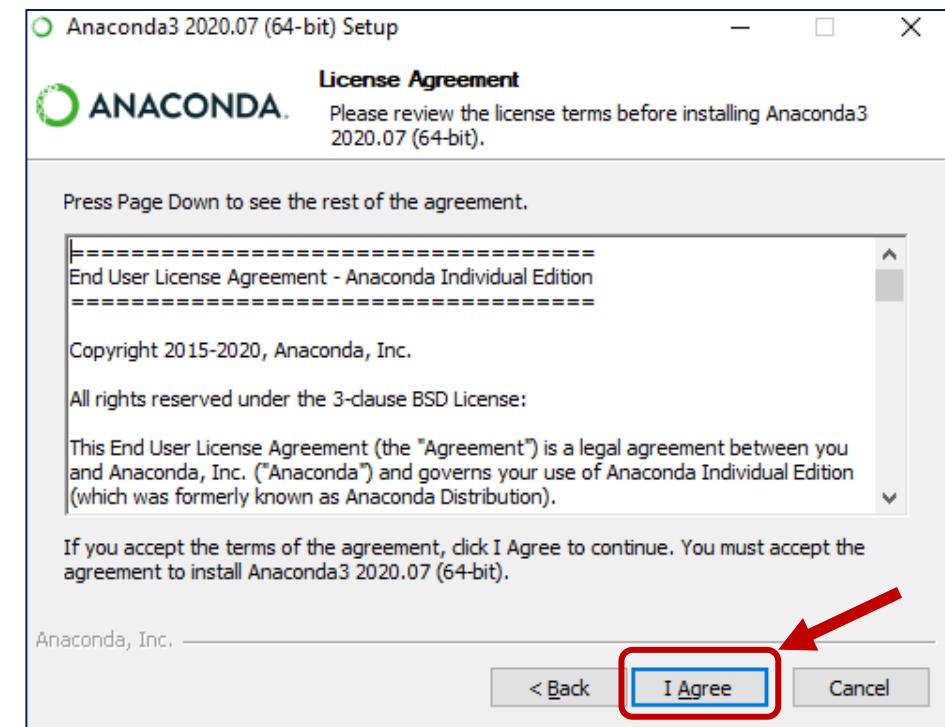
3

Abra o Instalador do Anaconda



4

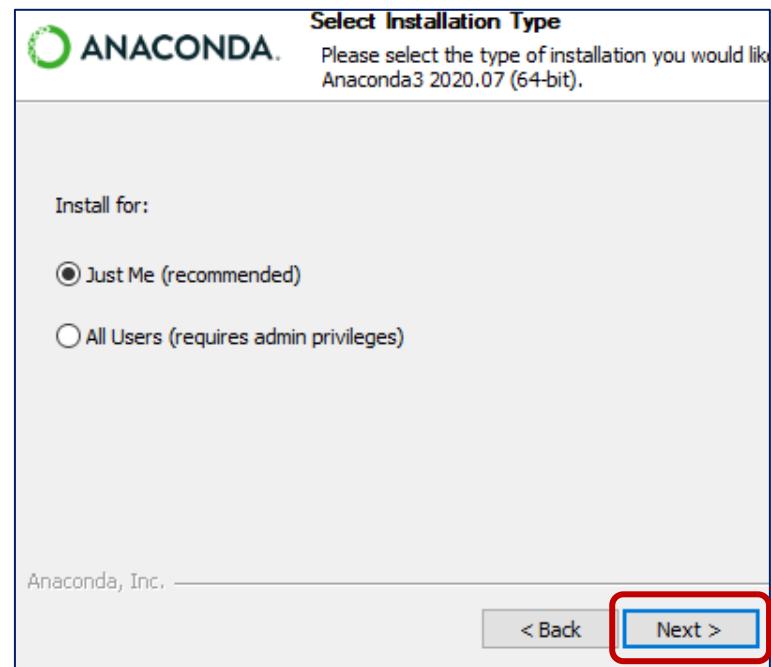
Aceite os termos de uso



Segue a instalação daquele jeito bem padrão. Next, Next...

5

### Selecionar tipo de Instalação



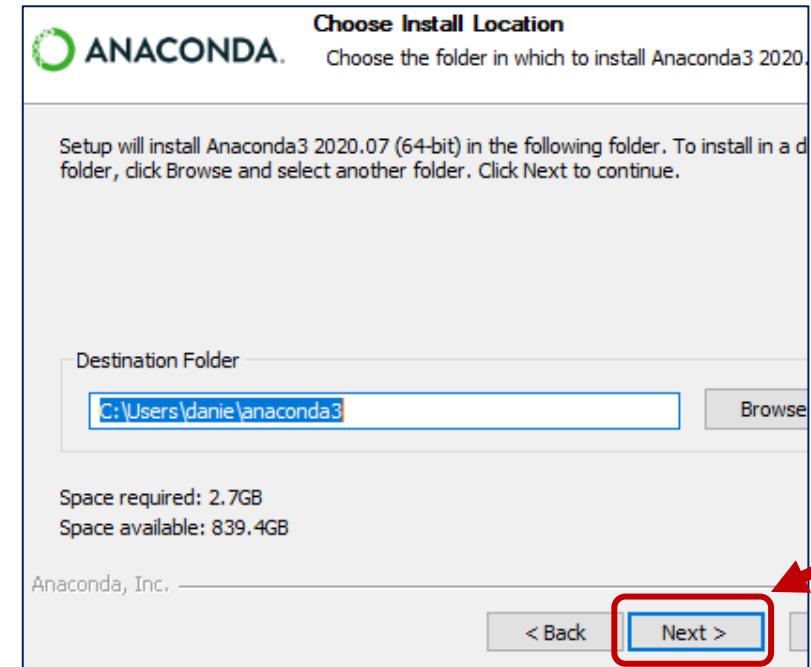
Nessa parte da instalação indicamos a opção **JUST ME** pois em teoria apenas o seu usuário precisa ter o Anaconda instalado.

PORÉM, em alguns casos, a instalação **JUST ME**, gera algumas falhas na inicialização do JUPYTER NOTEBOOK que vamos utilizar durante o curso.

Caso aconteça com você, reinstale e utilize a opção **ALL USERS**.

6

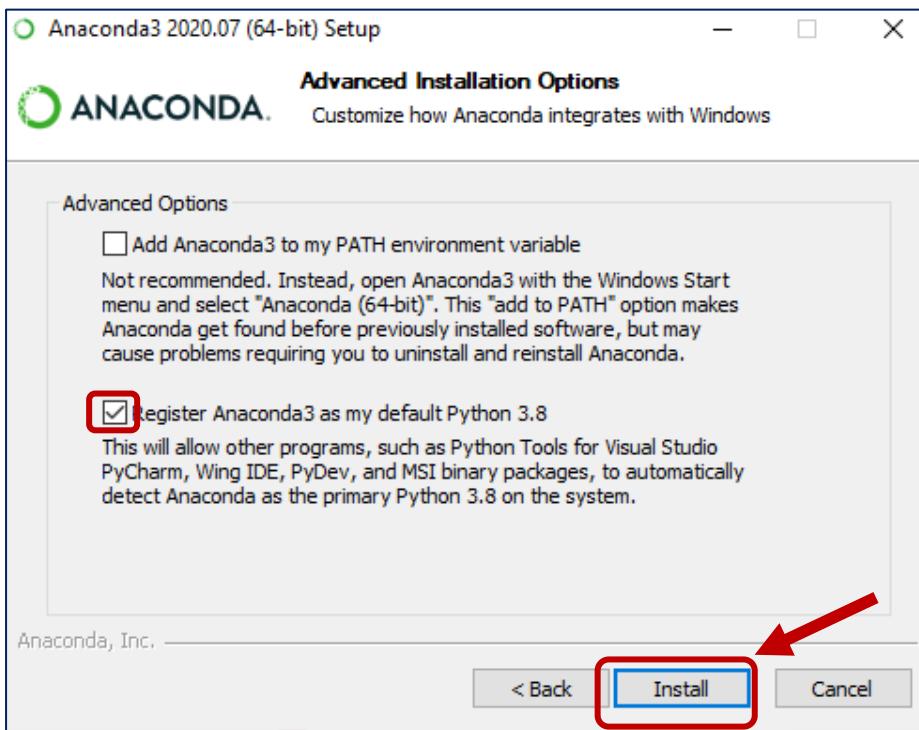
Aperte Next e siga a pasta padrão definida pelo Anaconda



Segue a instalação padrão. Next, Next...

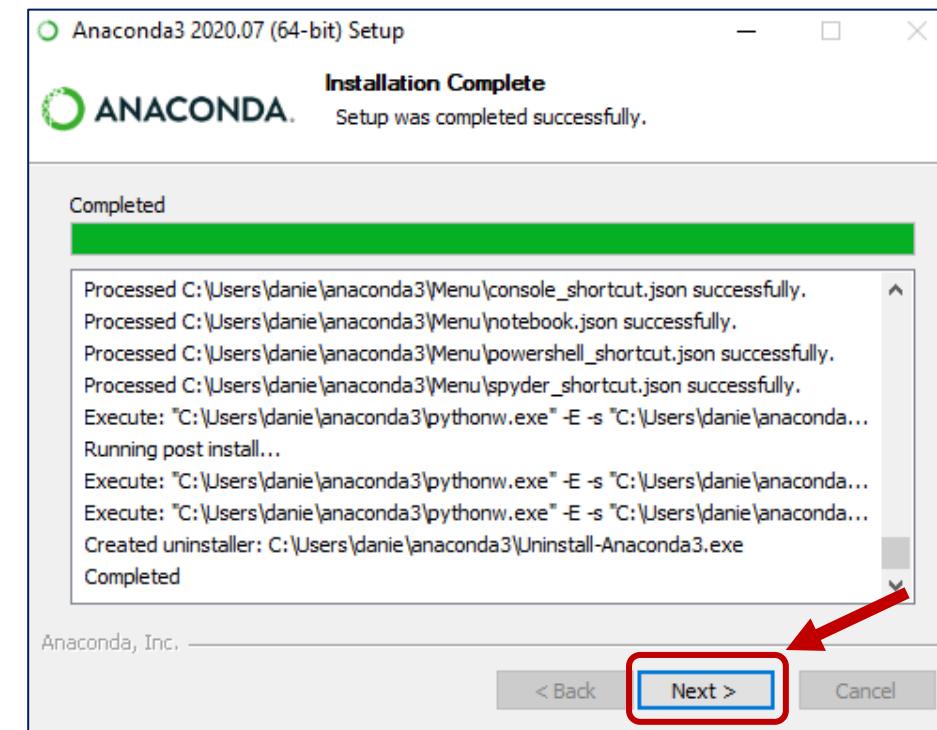
7

Defina o Anaconda como seu Python padrão e siga com a instalação clicando em *Install*



8

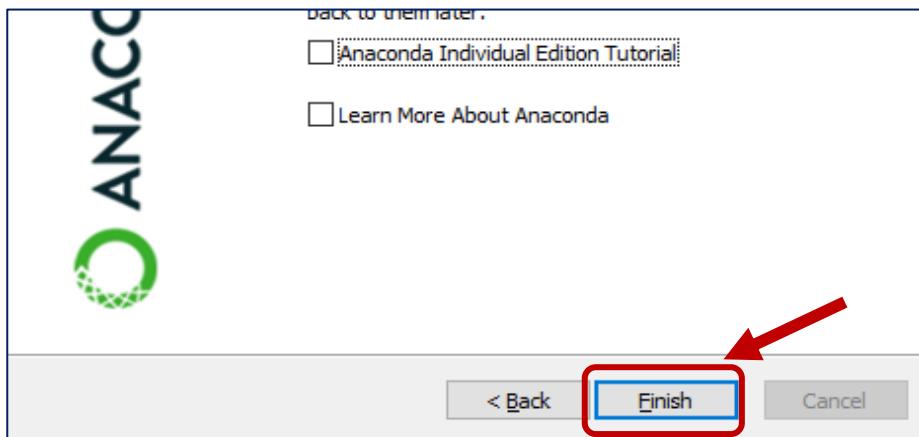
Ao fim da instalação clique em *Next*



Acabou...

9

*Next e Finish* nas próximas duas etapas

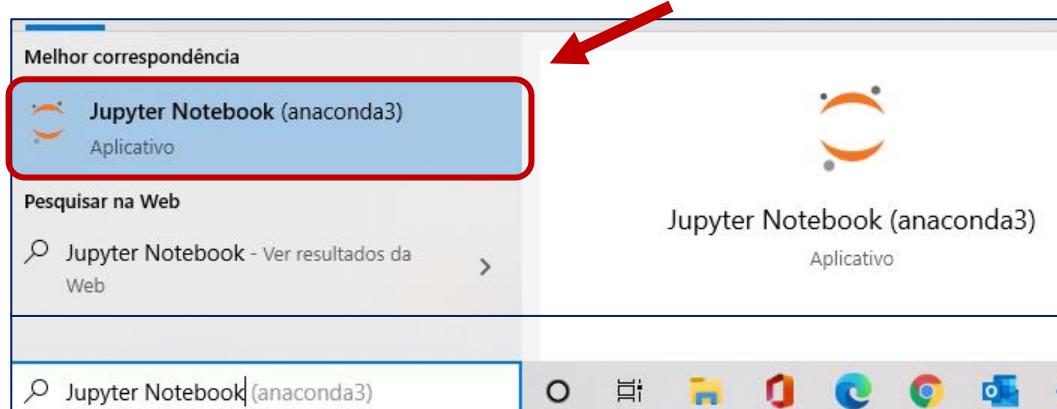


Pronto! Anaconda instalado. Agora vamos ver se está tudo OK para começarmos!

Testando...

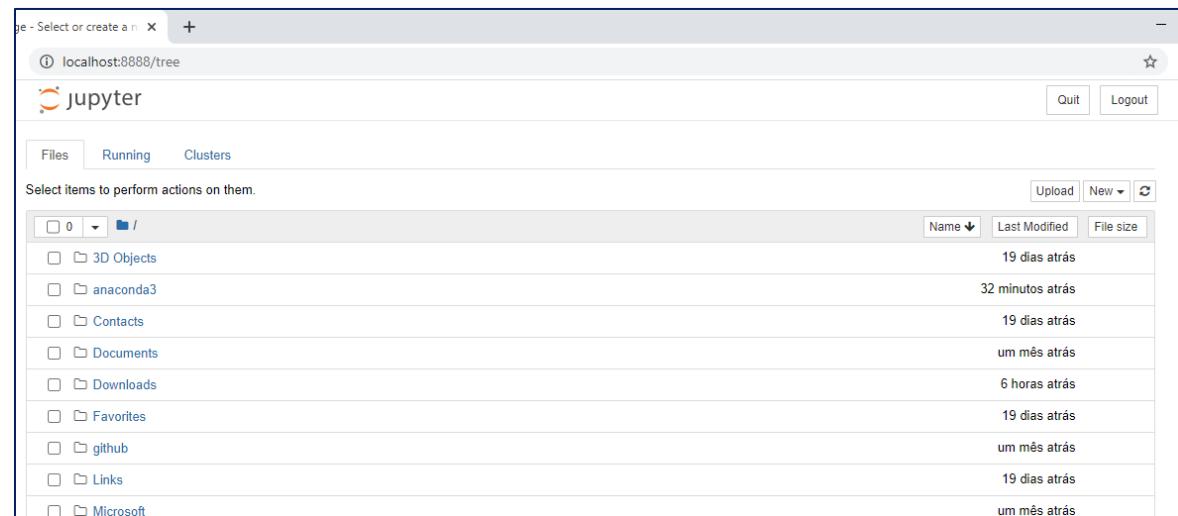
1

Pesquise por Jupyter Notebook na sua barra de tarefas



2

Abra o Jupyter Notebook



ATENÇÃO !

Ao clicar no ícone do Jupyter seu navegador padrão deverá abrir o Jupyter Notebook como no print 2. Além disso, uma janela preta com o símbolo do Jupyter irá abrir. **Não feche esta janela!** Ela é o Jupyter Notebook sendo rodado pelo seu computador.

Google

Google colab

colab research google c Traduzir esta página

**Google Colab**

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your ...

**Introduction to Colab and Python** [Google Drive](#)

Welcome to this Colab where you will get a quick introduction to ...

This notebook provides recipes for loading and saving data from ...

O Google Colab é uma alternativa para quem não quer baixar nenhum programa. Ele assim como o Jupyter Notebook é bem simples de usar.

**Os pré-requisitos são ter um conta gmail e ter acesso a internet**

Olá, este é o Colaboratory

Arquivo Editar Ver Inserir

Índice

- Primeiros passos
- Ciência de dados
- Machine learning
- Mais recursos
- Exemplos de machine learning
- Seção

Filtrar notebooks

| Titulo                     | Primeiro acesso | Aberto pela última vez |
|----------------------------|-----------------|------------------------|
| Olá, este é o Colaboratory | há 5 minutos    | há 0 minuto            |

NOVO NOTEBOOK CANCELAR

The screenshot shows the Google Colab interface. At the top, there's a toolbar with tabs: 'Exemplos' (selected), 'Recente' (highlighted with a red box and arrow), 'Google Drive' (highlighted with a red box and arrow), 'GitHub', and 'Upload'. Below the toolbar is a section titled 'Filtrar notebooks' with columns for 'Título', 'Primeiro acesso', and 'Aberto pela última vez'. A single notebook titled 'Olá, este é o Colaboratory' is listed. At the bottom, there's a dialog box with the text 'Criar um novo Notebook' and two buttons: 'NOVO NOTEBOOK' (highlighted with a red box and arrow) and 'CANCELAR'.

Seus arquivos utilizados recentemente

Seus arquivos salvos no Google Drive

Carregar arquivos. Aqui você pode carregar as cartilhas do curso

Exemplos

Recente

Google Drive

GitHub

Upload

Filtrar notebooks

| Título                     | Primeiro acesso | Aberto pela última vez |
|----------------------------|-----------------|------------------------|
| Olá, este é o Colaboratory | há 5 minutos    | há 0 minuto            |

Criar um novo Notebook

NOVO NOTEBOOK

CANCELAR

The screenshot shows the Google Colab interface. At the top, there is a navigation bar with back and forward arrows, a refresh icon, and a URL field containing 'colab.research.google.com/drive/19plhil0UZh1'. Below the URL is a title bar with the text 'Untitled0.ipynb' next to a file icon. The main menu includes 'Arquivo', 'Editar', 'Ver', 'Inserir', and 'Ambiente de execução'. On the left side, there is a sidebar with icons for search, code, and text. In the center, there is a large, empty workspace area.

Nome do arquivo no formato *.ipynb*

colab.research.google.com/drive/19plhil0UZh1

Untitled0.ipynb

Arquivo Editar Ver Inserir Ambiente de execução

+ Código + Texto

Para **criar um novo notebook** basta clicar no ícone Novo Notebook conforme o slide anterior.

Após a criação você terá uma tela como esta ao lado esquerdo.

Para **alterar o nome do arquivo** basta clicar o campo indicado e alterar.

**Atenção!** O formato *.ipynb* indica que se trata de um formato de notebook. Ele poderá ser lido tanto no Google Colab quanto no Jupyter Notebook.

Os campos **+ CÓDIGO** e **+ TEXTO** permitem que você insira mais linhas de código ou mais linhas de texto respectivamente.

Mas, fique tranquilo que vamos entender melhor isso mais para frente no curso!



### Por que o Anaconda?

O Anaconda é uma boa opção para o aprendizado do Python pois além do Python vários programas já são instalados automaticamente. Esses programas normalmente são chamados de **bibliotecas** e diminuem muito as linhas de código que precisam ser escritas. Além disso, o Anaconda também instala automaticamente o **Jupyter Notebook** que usaremos ao longo do curso.



### Posso usar outra IDE? Por exemplo, o Pycharm?

No curso usaremos o Jupyter Notebook, mas nada impede que você utilize outra IDE. **Atenção!** As cartilhas disponibilizadas no curso não estão no formato ".py" e sim no formato ".ipynb" que é originado pelo Jupyter Notebook. Será necessário copiar o código para a sua IDE de preferência.



### Não quero usar o Anaconda como *Default*, algum problema?

Durante a instalação do Anaconda será perguntado se você deseja ter o Anaconda como o Python Default. Nós recomendamos que sim (principalmente se for seu primeiro contato com o Python). Mas, se você já tinha o Python instalado e não quer selecionar o Anaconda como Default não tem problema nenhum!



### Meu *Jupyter Notebook* não abriu automaticamente. O que faço?

Em geral o Jupyter deveria abrir automaticamente, caso não seja o caso tente as 3 opções abaixo:

- Desinstale e Instale novamente o Anaconda;
- Verifique se você está usando seu navegador definido com o padrão;
- Verifique se a janela do CMD está aberta( janela preta), lá você encontrará <http://localhost:8888> . É só copiar e colar no seu navegador;

## Módulo 3

# O que é Programação e o que é Python



**ATENÇÃO !**

Se você já possui alguma experiência em programação  
este módulo é opcional.

Se é o seu caso, **clique aqui** para pular este módulo

Conjunto de comandos escritos em uma linguagem específica na qual o computador consegue compreender.

Vamos considerar um exemplo simples. Todo dia você prepara um relatório para o seu chefe com a atualização da produção do dia anterior. Ele sempre elogia seu trabalho mas gostaria de receber no primeiro horário.

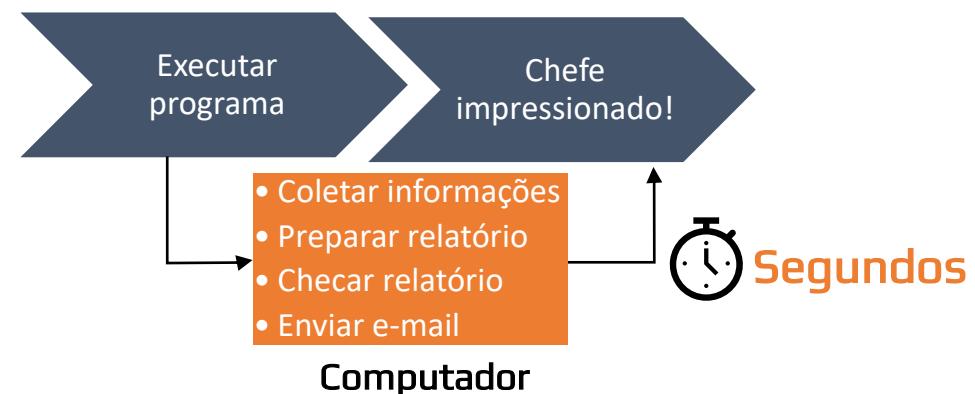
Manualmente, seu trabalho demora pelo menos 1 hora... Com programação, você pode descrever o que você faz para o computador, e assim, fazer com que ele faça para você em segundos. Deixando seu **chefe ainda mais impressionado!!**

### Processo manual:



1 hora

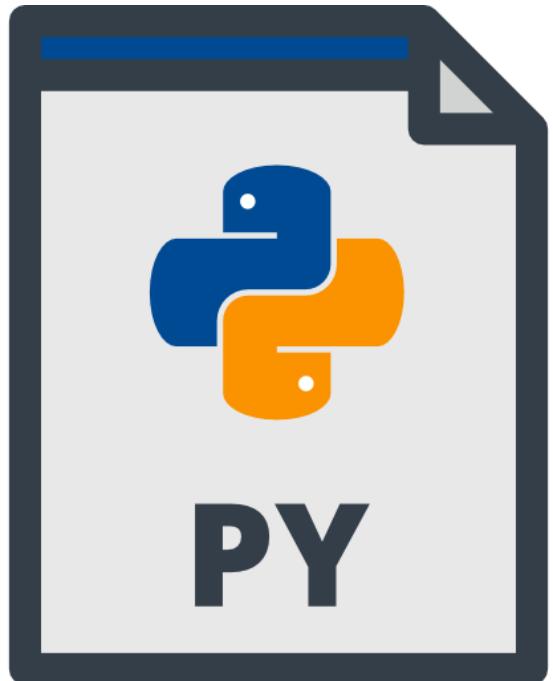
### Processo com programação



1 hora

Segundos

Computador



### O que é o Python? Por que o Python?

Uma das linguagens mais utilizadas no mundo.

Comunidade global de usuários do Python. Muitos “problemas” já resolvidos. Isso ajuda muito pois você não precisa quebrar a cabeça de como fazer e sim buscar um código que resolva o seu “problema” e personalizar para o seu caso.

Links e Comunidades que valem a pena conhecer:

- [Stack Overflow](#);
- [Python Documentation](#);
- [Github](#);



### MODO DE PREPARO

- 1 Misture no liquidificador o ovo, o leite, o óleo, o açúcar, o fubá e bata bem.
- 2 Despeje a mistura em uma tigela, e misture com a farinha e o fermento em pó.
- 3 Despeje a massa em uma forma untada com manteiga e farinha, e acrescente uma camada de goiabada. Repita este processo até preencher toda a forma.
- 4 Leve ao forno médio (180° C), preaquecido, por 40 minutos.

Imagine que você é um computador cozinheiro por 1 minuto! Você não sabe o que é um bolo e nem como o mesmo deve ser feito.

Olhe esta receita ao lado e imagine que você tenha que seguir exatamente o que está sendo ordenado.

Vamos dar alguns exemplos:

#### Olhe o passo 1:

Para os humanos é fácil entender que para misturar algo no liquidificador precisamos pegar os ovos primeiro. Mas para o computador isso precisa ser dito! Nesse caso o você não conseguiria misturar o ovo, o leite, o açúcar e o fubá pois você não colocou nem mesmo pegou estes itens.



### MODO DE PREPARO

- 1 Misture no liquidificador o ovo, o leite, o óleo, o açúcar, o fubá e bata bem.
- 2 Despeje a mistura em uma tigela, e misture com a farinha e o fermento em pó.
- 3 Despeje a massa em uma forma untada com manteiga e farinha, e acrescente uma camada de goiabada. Repita este processo até preencher toda a forma.
- 4 Leve ao forno médio (180° C), preaquecido, por 40 minutos.

OK! Você deve estar pensando que é um exemplo bobo, mas é exatamente assim que seu computador pensa!

#### Por isso algumas dicas:

- Ser explícito no que deve ser feito é importante;
- Se você não der o comando ao computador ele não vai fazer;
- Se você der um comando ele irá executar, então, atenção com a sequência que os comandos são escritos. O computador sempre lerá de cima para baixo
- Um comando muito importante é o de encerrar o programa ou dar um resultado final. Não ter esse comando pode fazer seu pc entrar em looping infinito.

- 1 Errar é normal
- 2 Não se fruste
- 3 Não precisa decorar o código
- 4 Se divirta programando!
- 5 Aprenda a procurar respostas sozinho
- 6 QUALQUER COISA ESTAMOS AQUI!



Mestre Lira, João

## Módulo 4

# Seus primeiros programas em Python



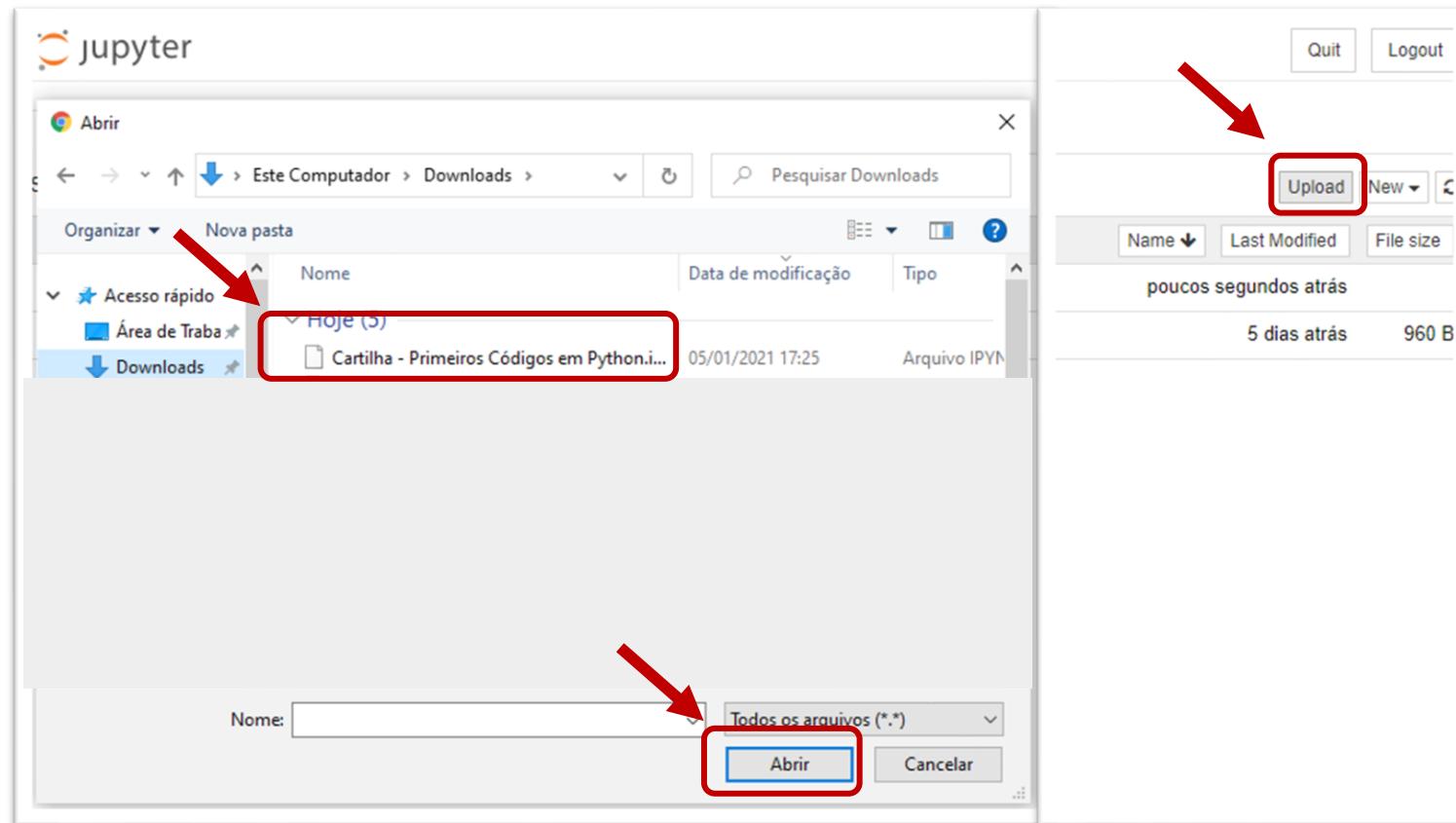
Primeiro vamos entender a **interface do Jupyter!**

Quando o Jupyter rodar no seu computador você verá uma tela como a tela ao lado.

Ela basicamente é um espelho das pastas do **SEU** computador.

Para começarmos basta você selecionar a pasta em que deseja salvar os arquivos.

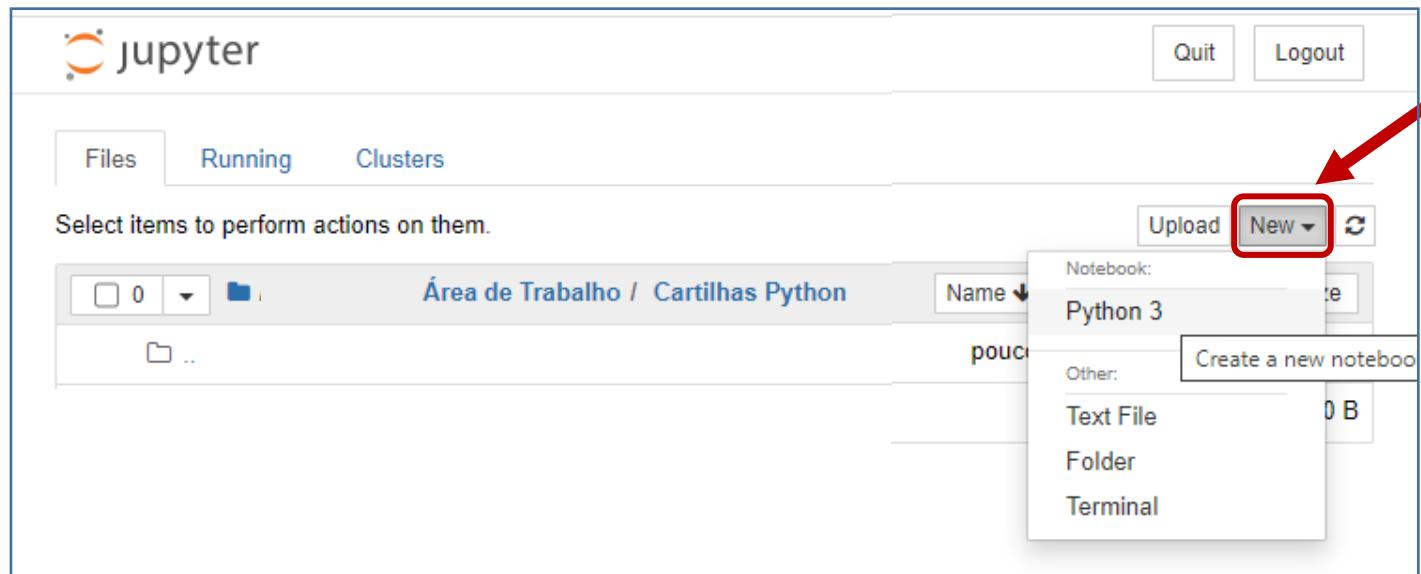
No nosso caso vamos salvar na Pasta **Área de Trabalho > Cartilhas Python**



Na pasta selecionada você poderá abrir Notebooks antigos, criar novos Notebooks ou carregar arquivos (por exemplo as cartilhas disponibilizadas no curso).

Para carregar um arquivo é bem fácil. Basta clicar no item **UPLOAD** indicado pela seta.

Já é interessante você carregar as cartilhas de exercícios disponibilizadas nas aulas. Lembrando que os arquivos são no formato **.ipynb** e por isso só podem ser abertos no Jupyter ou no Google Colab



A outra opção é criarmos do zero um novo arquivo do Jupyter.

Para isso basta clicar em **NEW > Python 3**

Show! Temos agora um novo arquivo Jupyter, no formato .ipynb.

Vamos conhecer agora os **principais ícones da barra de tarefas**.

**Nome do arquivo:**  
Para alterar basta clicar e escrever o nome que você preferir

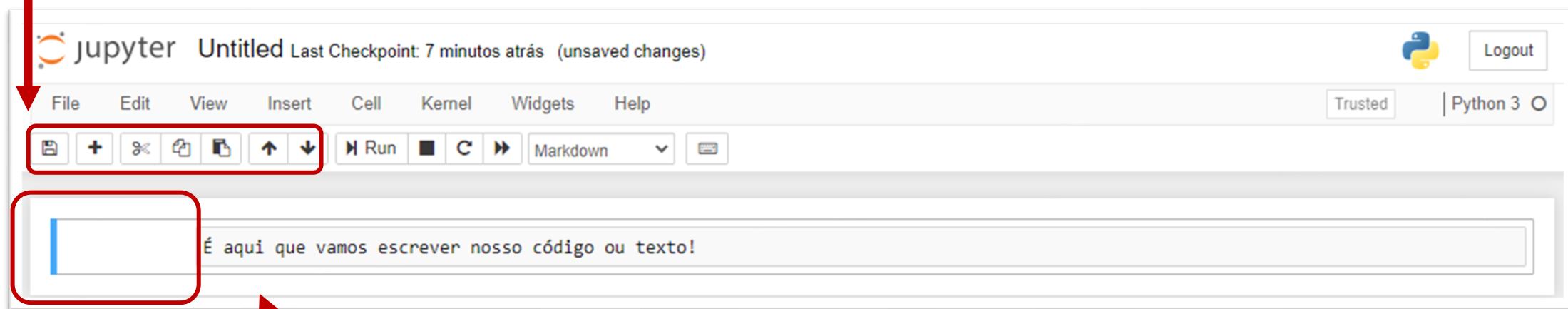
**Status do Autosave:**  
Status da última vez que o arquivo foi salvo



Vamos conhecer agora os **principais ícones da barra de tarefas**.

Da esquerda para a direita:

- **Save and Checkpoint**: Salva arquivo;
- **Insert Cell Below**: Insere uma célula abaixo da célula selecionada;
- **Cut selected Cells**: Corta células selecionadas
- **Copy select Cells**: Copia células selecionadas;
- **Paste cells below**: Cola células abaixo;
- **Move select cells up**: Move a célula selecionada para a linha acima;
- **Move select cells down**: Move a célula selecionada para a linha abaixo;



Indicação de seleção de célula:

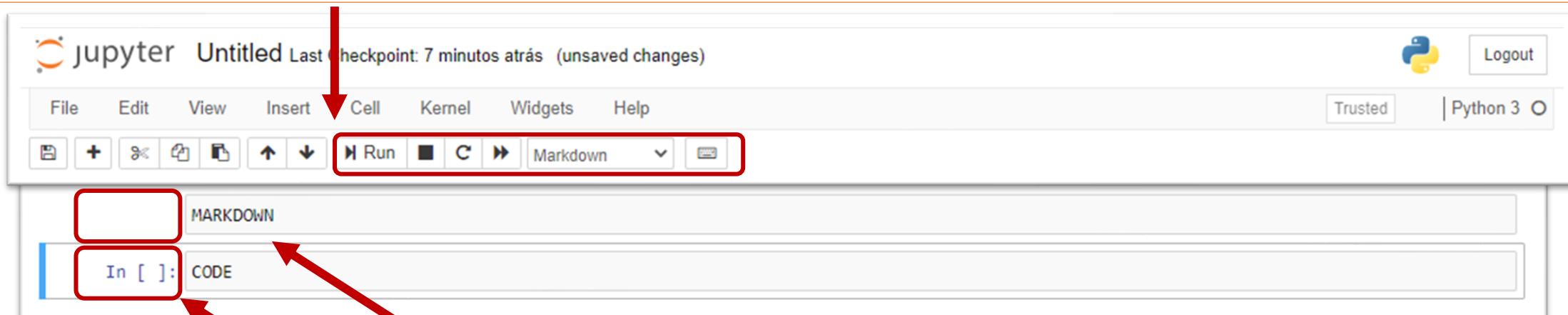
Azul: Célula selecionada(ATALHO = ESC);

Verde: Célula ativa para entrada de dados (ATALHO = ENTER);

Vamos conhecer agora os **principais ícones da barra de tarefas**.

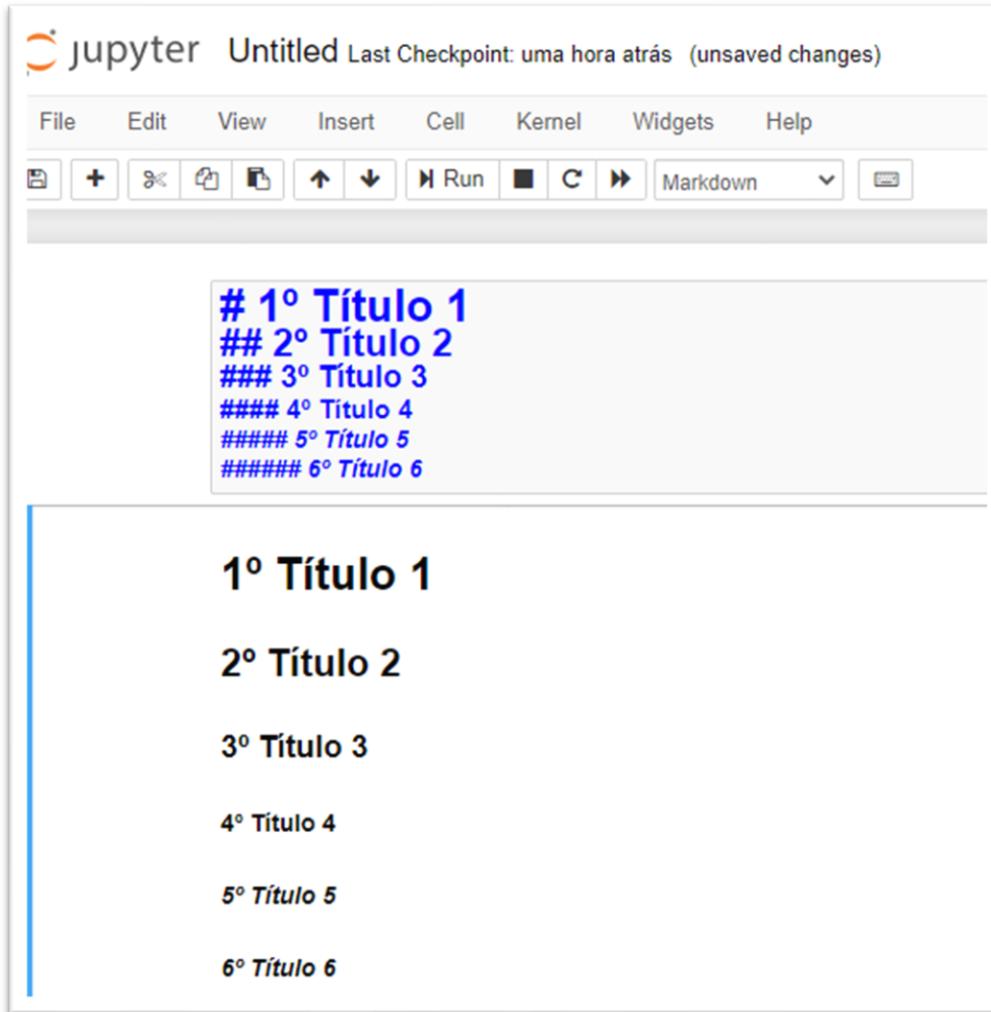
Da esquerda para a direita:

- **Run**: Roda o código (ATALHO = CTRL+ENTER)
- **Interrupt the kernel**: Pausa o processamento no kernel;
- **Restart the Kernel**: “Reseta” o Jupyter
- **Restart the Kernel, then re-run the whole notebook** : “Reseta” o Jupyter e roda o programa do início novamente.
- **Barra de “tipo de entrada”**: Define se será inserido um **Markdown (TEXTO)** ou **Code (CÓDIGO)** na célula selecionada;
- **Open the command palette**: Abre a lista de atalhos



Atenção! Células **CODE** são aquelas que possuem o indicador ao lado “**In [ ]**”. Já as **MARKDOWN** não possuem esse indicador.

Agora vamos entender um pouco mais a fundo como funciona o Jupyter Notebook.



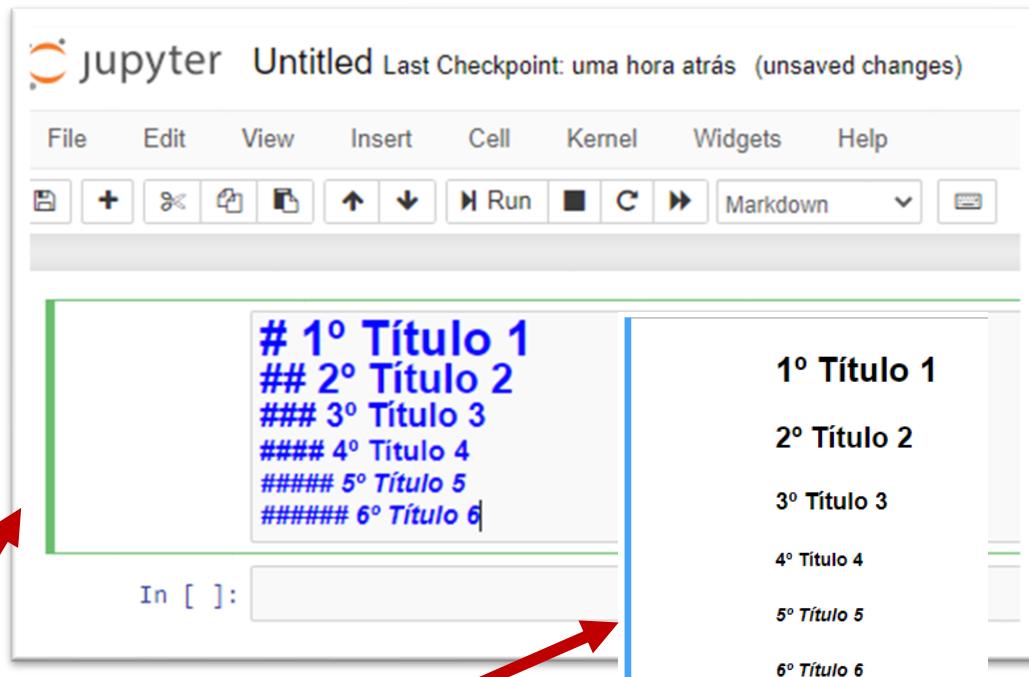
Primeiro vamos falar sobre o tipo **Markdown**:

Este tipo de célula será utilizada quando você quiser APENAS texto. Um título, uma introdução, etc.

O Markdown possui 6 tamanhos de fontes distintos para títulos. Para selecionar o tamanho desejado você deve usar o **"#"** quanto menor o número de # maior o tamanho da letra.

Outro detalhe é que o Jupyter já entende que existe um espaço entre linhas quando você usa esse recurso. Pode ver que o resultado sempre tem um espaço entre o texto mesmo que não esteja sinalizado na hora de escrever.

Agora vamos entender um pouco mais a fundo como funciona o Jupyter Notebook.



ATENÇÃO !

Lembre – se:

Verde : Célula ativa para entrada de dados;

Azul : Célula selecionada e não ativa para entrada de dados;

Para executar a célula selecionada basta clicar no ícone RUN ou usar o atalho CTRL+ ENTER.

Outra dicas útil é usar o atalho Shift + ENTER quando você quiser **executar uma célula e inserir uma nova célula** automaticamente.

Este aqui é um Título em Markdown.

A Célula de baixo é do tipo CODE. É lá que a gente vai escrever nosso primeiro código.

```
In [8]: print("Meu primeiro código em Python. Usando 2 aspas")
print('Meu primeiro código em Python. Usando 1 aspa')
print('Os 2 jeitos funcionam')
print('Mas sem as aspas não funciona!')
print( 'Se colocar print=também não vai funcionar!!!')
print(123)
print('Com número também funciona')
```

```
Meu primeiro código em Python. Usando 2 aspas
Meu primeiro código em Python. Usando 1 aspa
Os 2 jeitos funcionam
Mas sem as aspas não funciona!
Se colocar print=também não vai funcionar!!!
123
Com número também funciona
```

```
In [ ]:
```

Agora vamos entender as células CODE.

O primeiro código que vamos usar vai ser usando a função **PRINT**. Ela “imprime” o que estiver descrito dentro do parênteses.

Funciona tanto para texto como para números, mas para texto é muito importante lembrar de utilizar as **ASPAS**!

O Python vai entender tanto as duplas(“ ”) quanto as simples (‘ ’).

Para os números basta escrever dentro do parênteses.

Mais para frente vamos entender como imprimir variáveis e uma combinação entre texto e números.

Agora vamos entender um pouco mais a fundo como funciona o Jupyter Notebook.

```
In [2]: print('1+1')
print('para operações que só contenham números as aspas não são necessárias')
print(1+1)
print(1-1)
print(7/2)
print(3*2)
print(3**2)
print(7%2)
```

```
1+1
para operações que só contenham números as aspas não são necessárias
2
0
3.5
6
9
1
```

As operações matemáticas básicas podem ser realizadas no Python a partir dos símbolos abaixo:

SOMA (+)

SUBTRAÇÃO (-)

DIVISÃO (/)

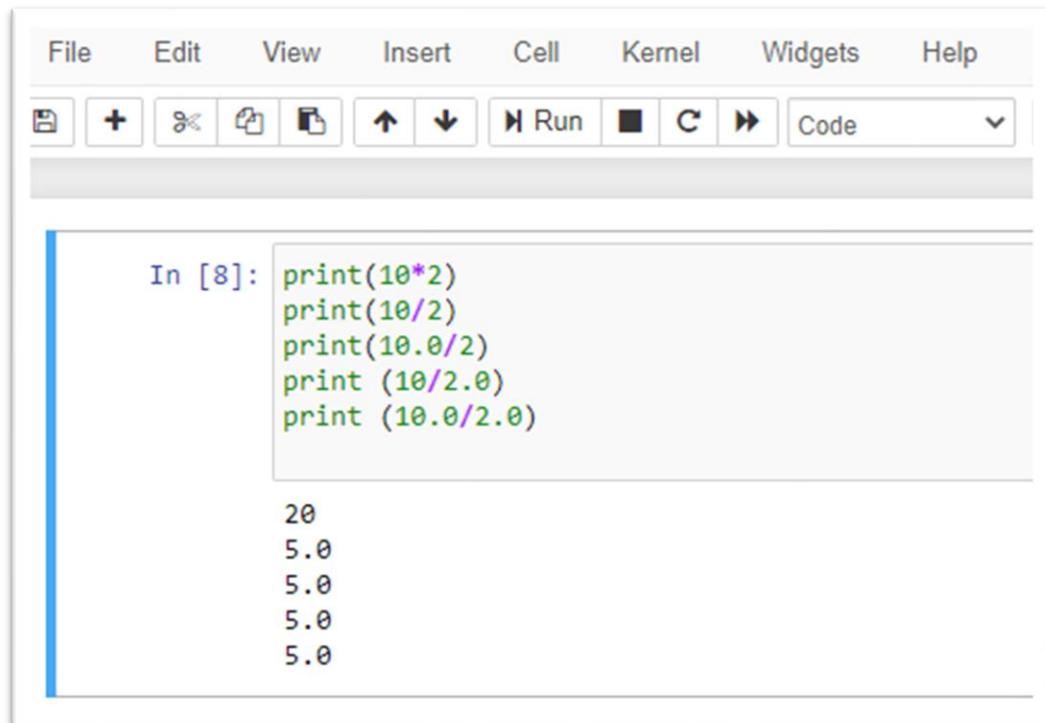
MULTIPLICAÇÃO (\*)

POTENCIAÇÃO (\*\*)

Além destas operações, o Python também possui uma operação que será muito útil mais para frente: o RESTO (%).

O RESTO (%) sempre terá como resultado o RESTO de uma DIVISÃO. Por exemplo:  
RESTO(%) da divisão 7 DIVIDO por 2 = 1

Agora vamos entender um pouco mais a fundo como funciona o Jupyter Notebook.



The screenshot shows the Jupyter Notebook interface. The top menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with various icons for file operations like opening, saving, and running cells. The main area is titled 'In [8]:' and contains the following Python code:

```
print(10*2)
print(10/2)
print(10.0/2)
print (10/2.0)
print (10.0/2.0)
```

Below the code, the output is displayed:

```
20
5.0
5.0
5.0
5.0
```



### ATENÇÃO !

Como o Python tenta ser o mais próximo possível do inglês, as casas decimais serão sempre representadas pelo “.” e não pela “,” como normalmente utilizamos no Brasil.

Vamos pegar o exemplo ao lado. O primeira operação é uma multiplicação e o resultado é o número, sem casa decimais, 20.

Já as divisões seguintes possuem o mesmo resultado , com casa decimal, 5.0.

Qual a diferença?

- 1) Uma divisão **sempre** será apresentada com casa decimais.
- 2) **Sempre** que um dos números envolvidos na operação tiver casas decimais o seu resultado terá casa decimais.

```
In [1]: print(10/2)
print(10/0)

5.0
-----
ZeroDivisionError: division by zero
Traceback (most recent call last)
<ipython-input-1-9da9367bc033> in <module>
      1 print(10/2)
----> 2 print(10/0)

ZeroDivisionError: division by zero
```

O Python sempre vai buscar entender o seu código. Mas, caso ele não entenda o **erro vai ser indicado para você**.

É comum em um primeiro momento que ele assuste mas vamos focar no que é essencial nesse início e vai ser bem fácil.

Como você pode ver fizemos 2 operações  $10 : 2$  e  $10:0$ . A primeira operação foi bem sucedida. O nossos resultado foi “printado”. Sendo ele o valor **5.0**

```
In [1]: print(10/2)
print(10/0)

5.0
ZeroDivisionError
<ipython-input-1-9da9367bc033> in <module>
      1 print(10/2)
----> 2 print(10/0)

ZeroDivisionError: division by zero
```

Agora, antes de entendermos a operação 2 vamos tentar ler a mensagem de erro:

**ZeroDivisionError:** Nesse campo, conseguimos entender que o erro é uma divisão por zero. Pois matematicamente isso não é correto.

```
In [1]: print(10/2)
print(10/0)

5.0

-----
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-9da9367bc033> in <module>
      1 print(10/2)
----> 2 print(10/0)

ZeroDivisionError: division by zero
```



Precisamos entender em qual das linhas esse erro está. Para isso, não é necessário procurar no código.

O Python sempre te dará uma dica de onde pode estar o problema. Nesse caso, está na **linha 2** da nossa célula.

Ela está indicada pela seta - - - → logo antes do número 2.

Sabendo disso é fácil identificar que de fato no nosso código, na linha 2, temos o cálculo  $10/0$  o que gera assim, um erro no programa.

No caso de operações maiores o Python seguirá a mesma ordem de operações da matemática.

Ou seja,

Multiplicação e Divisão são realizadas antes das operações de Adição e Subtração.

Caso seja necessária a adição antes da multiplicação como no exemplo ao lado, será necessário o uso do parênteses.

Como você pode perceber, o resultado foi alterado pois na **primeira linha temos o cálculo:**  $10*2=20 + 5 = 25$

**Enquanto na segunda linha,**  
Primeiro é somado 2 com 5 = 7 e depois é feita a multiplicação  $10*7 = 70$ .

```
In [2]: print(10*2 +5)
          print(10*(2+5))

25
70
```

```
In [4]: print("Daniel")
print("Candiotto")
print ("Daniel" + "Candiotto")
print ("Daniel" , "Candiotto")
```

```
Daniel
Candiotto
DanielCandiotto
Daniel Candiotto
```



### ATENÇÃO !

A decisão pelo uso do “+” ou “,” pode afetar seu resultado mais a frente. Uma outra forma de se obter o mesmo resultado da vírgula utilizando o sinal “+” é:  
print('Daniel' + ' ' + 'Candiotto')

Até agora utilizamos o termo texto para os códigos escritos dentro das aspas.

Na realidade, o termo mais correto é **STRING**.

Assim como nas operações matemáticas, podemos fazer operações com Strings.

Vamos ver o exemplo ao lado. As duas primeiras linhas são apenas prints “convencionais” sem nenhuma operação.

Já a terceira e quarta linha usam 2 operadores distintos para unir textos e formar uma string única.

Perceba que o sinal de + CONCATENA os textos escritos nas aspas.

Já a vírgula, concatena os dois textos mas faz uma separação por um espaço

```
In [3]: 'D' in 'Daniel'
```

```
Out[3]: True
```

```
In [5]: 'Da' in 'Daniel'
```

```
Out[5]: True
```

```
In [8]: 'J' in 'Daniel'
```

```
Out[8]: False
```

```
In [7]: 'd' in 'Daniel'
```

```
Out[7]: False
```



### ATENÇÃO !

A operação “in” é *case sensitive*, isso significa dizer que há diferenciação entre letras MAIÚSCULAS e MINÚSCULAS.

Nesse caso, por exemplo:  
daniel é diferente de Daniel

Outra operação possível com strings é a função **in**.

Essa função permite verificar se **algum caractere ou conjunto de caracteres** está contido em uma outra string.

Vamos dar uma checada nos exemplos ao lado.

‘D’ in ‘Daniel’ -> Resultado “TRUE”. Por que?

A função **in** retorna sempre se o que está sendo testado é VERDADEIRO (TRUE) ou FALSO (FALSE). Nesse caso como ‘D’ está em ‘Daniel’ o resultado é verdadeiro.

Já no teste ‘J’ in ‘Daniel’ o resultado é FALSE pois não existe a letra J no nome Daniel.

### O que são variáveis?

Variáveis são elementos que nos ajudam a guardar uma informação mesmo que esta varie com o tempo.

### Mas como eu crio uma variável?

No Python é muito comum que as variáveis tenham nomes muito parecidos com os nome falados. Por exemplo, vamos dizer que eu quero fazer um programa que calcula média de alunos.

Uma variável que posso criar será a variável NOTA = 9



### ATENÇÃO !

Quando estamos tratando de variáveis o termo “=” não significa IGUAL e sim, RECEBE!

Na programação, sempre o que está A ESQUERDA do “=” RECEBE ALGO que está escrito À DIREITA do “=”

Então, no exemplo ao lado não lemos NOTA igual a NOVE e sim, A variável NOTA recebe o valor NOVE

```
In [2]: nota1 = 10  
nota2 = 9.5  
nota3 = 9  
media_geral = ((nota1+nota2+nota3)/3)  
print (media_geral)
```

9.5

```
In [4]: nota1 = 5  
nota2 = 5.5  
nota3 = 6  
media_geral = ((nota1+nota2+nota3)/3)  
print (media_geral)
```

5.5

Agora que entendemos um pouco mais sobre variáveis vamos ver o exemplo ao lado escrito no Jupyter.

Ao invés de sempre fazermos a conta manualmente, podemos usar as variáveis **nota1**, **nota2** e **nota3** para calcular a média que seguirá a fórmula descrita na variável **média\_geral**

A vantagem da variável pode ser vista no exemplo ao lado.

Sem mudar nenhuma linha de código, apenas as notas, conseguimos recalcular a média.

Na criação de uma variável é importante levar em consideração algumas restrições e dicas:

### Restrições:

- Não pode ter o nome de uma função. Ex: Uma variável não pode se chamar “print”.
- A variável **não deve ter nenhum ESPAÇO**. O uso do **\_** é uma boa forma de separar as palavras como por exemplo `media_geral`. Outra possibilidade é o uso de letras maiúsculas. Ex: `MediaGeral`.

```
In [2]: nota1 = 10
nota2 = 9.5
nota3 = 9
media_geral = ((nota1+nota2+nota3)/3)
print (media_geral)

9.5
```



### Dicas:

- Padronize para todo o programa como você escreverá as variáveis. Isso facilitará o seu entendimento e das demais pessoas que usarão seus programas.

O \* indica que a célula está sendo processada pelo Jupyter. Nesse caso específico aguardando o usuário inserir no campo a informação solicitada

```
In [*]: input('Qual o seu nome')
Qual o seu nomeDaniel
```

```
In [1]: input('Qual o seu nome')
Qual o seu nomeDaniel
Out[1]: 'Daniel'
```

Muitas vezes no Python iremos precisar consultar o usuário sobre alguma informação.

Para isso, usamos a **função Input()**. Vamos ver o exemplo ao lado.

A estrutura do input é:

**Input( ' TEXTO QUE SERÁ LIDO PELO USUÁRIO' )**

Após a execução da célula um campo em branco será disponibilizado ao usuário. Após inserida a informação pelo usuário, o Python continuará a processar as demais linhas do código.

```
In [3]: input('Qual o seu nome?')  
Qual o seu nome?Daniel  
Out[3]: 'Daniel'  
  
In [5]: nome = input(' Qual o seu nome? ')  
Qual o seu nome? Daniel  
  
In [6]: print(nome)  
Daniel
```

No entanto, o input por si só não guarda esta informação... Para armazenar essa informação precisamos **atribuir o resultado do input à uma variável**.

No exemplo ao lado temos essa estrutura:

Variável **NOME** recebe resultado do **INPUT**

Muito importante!! Perceba que não usamos mais o termo IGUAL e sim RECEBE pois o sinal “=” deverá ser lido assim daqui para frente.

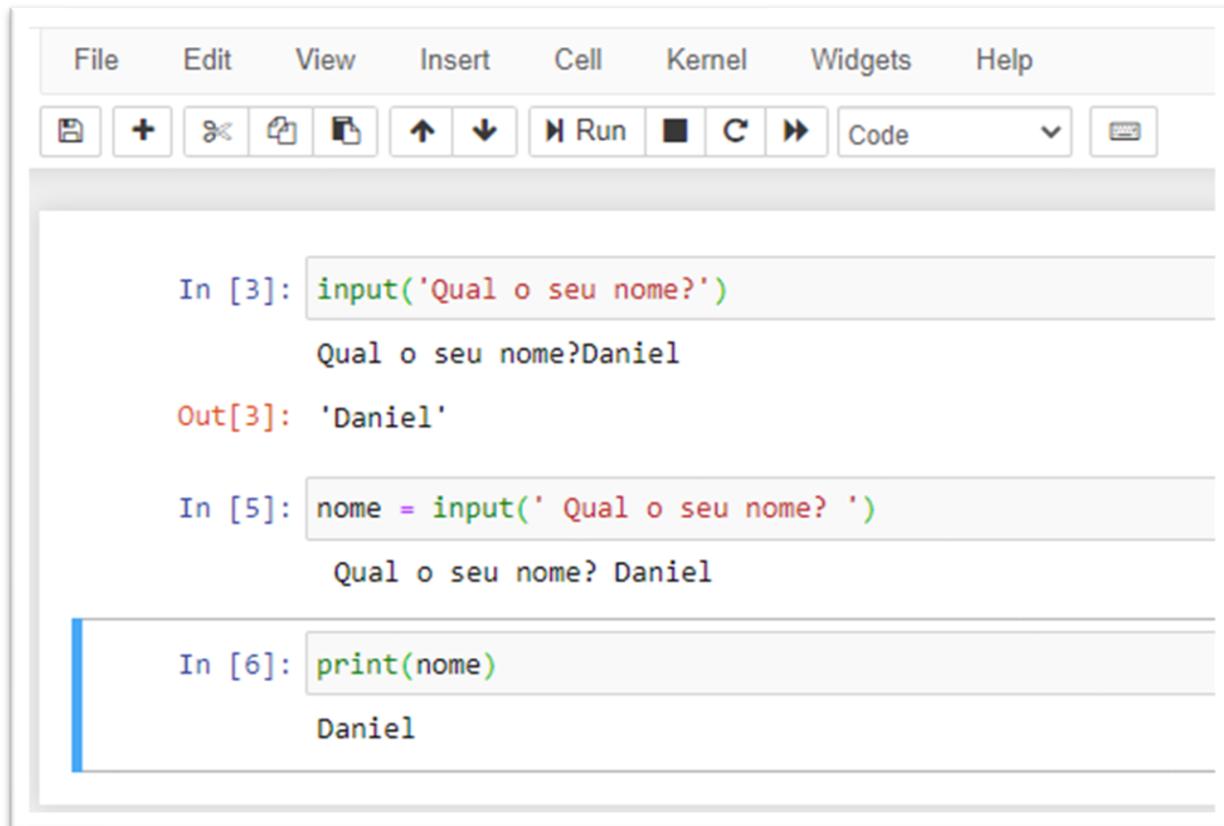
Outro ponto de atenção é a estrutura:

A variável sempre virá à esquerda do “=” e o que se quer atribuir na variável à direita.

**VARIÁVEL = O QUE SE QUER ATRIBUIR**

## Módulo 4 – Seus primeiros programas em Python – Input - Pegando informações do Usuário (3/3)

43



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
File Edit View Insert Cell Kernel Widgets Help  
Run Cell Kernel Help  
In [3]: input('Qual o seu nome?')  
Qual o seu nome?Daniel  
Out[3]: 'Daniel'  
  
In [5]: nome = input(' Qual o seu nome? ')  
Qual o seu nome? Daniel  
  
In [6]: print(nome)  
Daniel
```

Perceba que no primeiro caso o input apenas retornou a resposta dada pelo usuário mas o seu código não prevê uma “impressão” da variável.

Já no segundo caso, como atribuímos a função input a varável NOME, a informação “Daniel” fornecida pelo usuário está armazenada nesta variável.

Assim ao inserirmos a variável NOME dentro do argumento da função print () a mesma retornará o nome armazenado.

Módulo 5

Mais sobre variáveis

```
View Insert Cell Kernel Widgets Help  
Run Cell Code  
]: print(nome)  
nome='João'  
  
NameError Traceback (most recent call last)  
<ipython-input-1-f9edefc78603> in <module>  
----> 1 print(nome)  
      2 nome='João'  
  
NameError: name 'nome' is not defined
```

No Python, a ordem das suas linhas de código impactam no resultado.

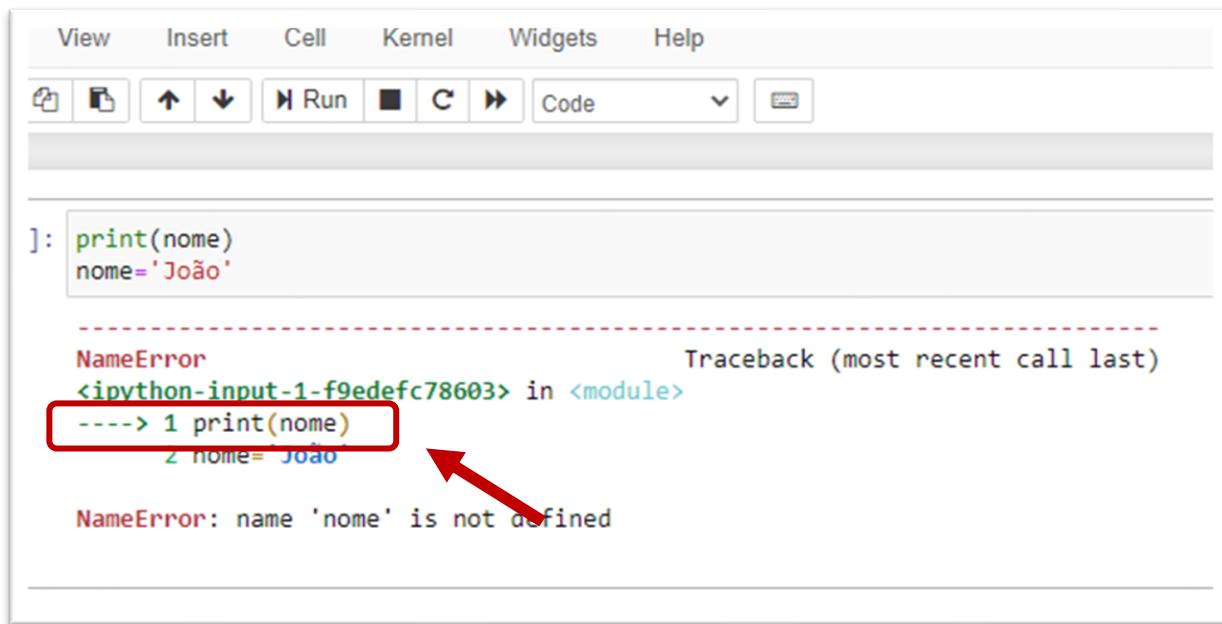
O que isso quer dizer? O Python sempre lerá o seu programa de cima para baixo.

Vamos dar uma olhada no exemplo ao lado.

Consegue entender esse erro?

Não? Tranquilo! A gente te explica!

O erro, como você pode ver na última linha, se refere a variável 'nome' que não está definida(*not defined*). Isso significa que o que está sendo executado não foi definido anteriormente.



```
View Insert Cell Kernel Widgets Help  
Run Cell Code  
]: print(nome)  
nome='João'  
  
NameError  
<ipython-input-1-f9edefc78603> in <module>  
----> 1 print(nome)  
      ^ nome= João  
  
NameError: name 'nome' is not defined
```

Você deve estar se perguntando:

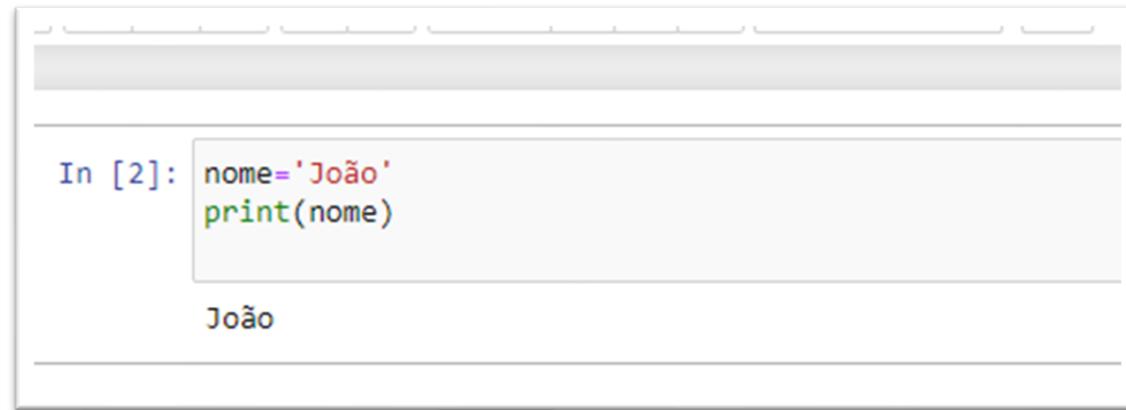
“-Mas está definido sim! Nome= João!”

Sim, você tem razão sobre isso mas vamos olhar mais uma vez a mensagem de erro.

A seta indica que o erro ocorreu na linha 1 onde a variável nome é “imprimida” pela função print.

Com essa dica da mensagem de erro fica mais simples de entender que na verdade o erro é a sequência do nosso código.

Imagine que você é o computador e não sabe o que virá na linha 2. Ao ler print(nome) que nome você “printaria”? Nenhum certo? Você ainda não tem essa informação.



In [2]:

```
nome='João'  
print(nome)
```

João

Ok. Como corrigir?

Alterando a ordem do nosso código. Vamos dar uma olhada no exemplo ao lado já corrigido.

Vamos fazer o mesmo exercício. Leia como o computador leria...

Variável nome recebe João.  
Imprimir a o valor da variável nome  
Como nome = João  
Resultado João

Portanto, sempre tenha atenção com a sequência do seu código!

```
: faturamento = 100
custo = 80

lucro = (faturamento - custo)

print (lucro)

custo = 50

print(lucro)

20
20
```

Vamos ver um exemplo um parecido mas envolvendo algumas operações matemáticas:

Apesar de termos “printado” duas vezes o lucro usando custos diferentes o resultado foi o mesmo.

Em teoria teríamos

$$\text{Faturamento}(100) - \text{Custo}(80) = 20$$

$$\text{Faturamento} (100) - \text{Custo}(50) = 50$$

O erro no código, neste caso, ocorreu pois a variável lucro manteve o cálculo inicial realizado na linha 3 do nosso código.

Mesmo fornecendo um novo custo, nesse caso, 50 o cálculo do novo valor do lucro não foi realizado. Logo, a variável manteve seu valor original

```
4]: faturamento = 100  
custo = 80  
  
lucro = (faturamento - custo)  
  
print (lucro)  
  
custo = 50  
  
lucro = (faturamento - custo)  
  
print(lucro)
```

```
20  
50
```

Para resolver esse problema, temos como uma solução bem simples o exemplo ao lado:

Inserindo uma nova linha para cálculo do lucro, podemos calcular o novo lucro considerando o novo custo fornecido.

Assim,

$$\text{Lucro} = \text{Faturamento} (100) - \text{Custo} (50)$$

```
In [1]: faturamento = 1000  
        type(faturamento)
```

```
Out[1]: int
```

```
In [2]: faturamento = 1000.00  
        type(faturamento)
```

```
Out[2]: float
```

```
In [3]: faturamento = '1.000'  
        type(faturamento)
```

```
Out[3]: str
```

```
In [4]: ganha_bonus = True  
        type(ganha_bonus)
```

```
Out[4]: bool
```

Até o momento, temos tratado as variáveis como iguais, mas na realidade no Python, cada variável possui um tipo. Para saber qual o tipo de uma variável usamos a função **Type()** como nos exemplos ao lado.

- **INT** -> Números inteiros (sem casa decimal)
- **FLOAT** -> Basicamente são números com casas decimais (Lembrando que no Python a casa decimal é representada pelo “.” e não pela “,”)
- **STR** -> Tipo string. Basicamente texto. Um ponto de atenção é que números entre aspas são considerados strings.
- **BOOL** -> Chamadas booleanas. São variáveis que só possuem 2 valores possíveis: TRUE(Verdadeiro) ou FALSE(Falso)

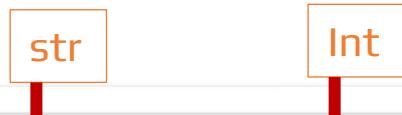
```
In [1]: dir(str)  
Out[1]: ['__add__',  
         '__class__',  
         '__contains__',  
         '__delattr__',  
         '__dir__',  
         '__doc__',  
         '__eq__',  
         '__format__',  
         '__ge__',  
         '__getattribute__',  
         '__getitem__',  
         '__getnewargs__',  
         '__gt__',  
         '__hash__']
```

Métodos da variável do tipo STR

Ou seja, todos os tipos de variáveis que vimos anteriormente possuem métodos específicos para cada um daqueles objetos.

Para descobrir quais são esses métodos podemos utilizar a função **dir(TIPO DA VARIÁVEL)** conforme a figura ao lado usando o tipo STR.

Não é necessário guardar ou decorar todos os métodos possíveis, mas é importante entender que esses métodos são ações que podem ser executadas nos objetos criados no programa.



```
In [7]: faturamento = 1000
        print('O faturamento da loja foi '+faturamento)

-----  
TypeError                                 Traceback (most recent call last)
<ipython-input-7-ff3f133db6d3> in <module>
      1 faturamento = 1000
      2 type(faturamento)
----> 3 print('O faturamento da loja foi'+faturamento)
      4
      5

TypeError: can only concatenate str (not "int") to str
```

In [8]: type(faturamento)  
Out[8]: int

Quando trabalhamos com variáveis de tipos distintos podemos ter erros como nesse caso.

Se analisarmos o erro, conseguimos entender que só é possível concatenar variáveis str com outras variáveis str.

No nosso código, estamos tentando concatenar o texto ' O faturamento da loja foi' que é um **string** com **faturamento** que é uma variável **int** já que atribuído na linha anterior o valor 1000.

```
In [10]: faturamento = 1000  
print('O faturamento da loja foi '+str(faturamento))  
O faturamento da loja foi 1000
```



```
In [12]: faturamento = 1000  
print('O faturamento da loja foi '+str(faturamento))  
O faturamento da loja foi 1000
```

```
In [13]: type(faturamento)  
Out[13]: int
```

Para tratarmos esse erro e conseguirmos ter apenas variáveis tipo str na hora de concatenar, vamos usar a função str() como no print ao lado.

Assim, o Python lerá o valor atribuído à variável faturamento e converterá esse valor em uma string.

Uma dúvida comum é:

**“Mas essa conversão é pontual ou é definitiva?”**

Vamos dar uma olhada no type da variável faturamento nesse segundo print:

Perceba que apesar de termos convertido a variável faturamento, **a mesma não foi alterada em definitivo apenas na linha onde foi feita a conversão**.

### Método Format com uma variável

```
faturamento = 1000
custo = 500

print('O faturamento da loja foi {}'.format(faturamento))
```

O faturamento da loja foi 1000

Outra forma de trabalharmos com mais de um tipo de variável é o método `.format`. Esse método, além de mais usual, é mais simples pois não é necessário especificar para qual tipo de variável será alterada.

Vamos entender como funciona com o primeiro exemplo:

Perceba que ao invés de usarmos `str(faturamento)` como no exemplo anterior, nós usamos apenas **duas chaves {}** e ao fim do texto usamos o método `.format(faturamento)`.

O python ao ler o `{}` entenderá que o valor de uma variável precisa ser incluída ali. **O Python buscará esse valor dentro dos argumentos do método format.** Nesse caso faturamento.

### Método Format com 2 ou mais variáveis

```
faturamento = 1000
custo = 500

print('O faturamento da loja foi {} e o custo {}'.format(faturamento,custo))
O faturamento da loja foi 1000 e o custo 500
```

**Atenção!! A ordem faz diferença!**

```
faturamento = 1000
custo = 500

print('O faturamento da loja foi {} e o custo {}'.format(custo,faturamento))
O faturamento da loja foi 500 e o custo 1000
```

Mas se ao invés de uma variável forem mais variáveis?

Vamos olhar o exemplo 2:

Nesse caso **temos dois sinais “{}”** o que significa que **serão necessários 2 valores**. Olhando os valores dentro do .format podemos ver que as variáveis faturamento e custo são esses valores.

A Ordem faz diferença?

**SIM!** Vamos dar uma olhada no último caso:

Aqui, ao invés de format(faturamento,custo) invertemos a ordem dessas variáveis. Logo format(custo,faturamento). Perceba no output que **os valores estão invertidos**. Portanto, **atenção pois a ordem faz diferença!**

# Módulo 6

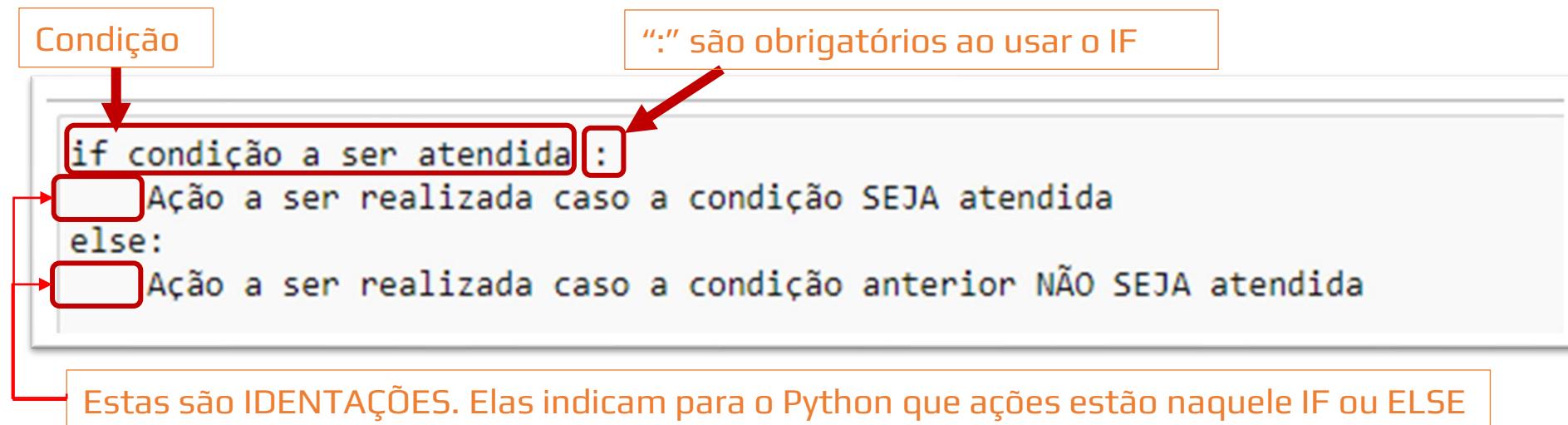
# Estrutura do If

Algo muito comum e quase certo de ser utilizado em algum código que você venha a fazer, é a necessidade de definir ações baseadas em condições específicas.

Se você já usou o Excel ou já programou em outras linguagens deve conhecer esse conceito como o SE ou IF em inglês.

Essencialmente no Python é a mesma coisa só mudando a forma como é feito.

Vamos entender a estrutura do IF:



Vamos olhar um exemplo aplicável.

O programa deverá ler a nota do aluno.

Nesse caso, NOTA =10

Após a leitura da nota o programa irá através do IF verificar uma condição pré-programada.

Nesse caso, a condição é:

O VALOR da VARIÁVEL NOTA é MAIOR ou IGUAL a 7?

Como NOTA =10 a resposta é VERDADEIRA.

```
nota=10
```

```
if nota>=7:  
    print('Aprovado')  
else:  
    print('Não aprovado')
```

Aprovado

```
: nota=10
```

```
if nota>=7:  
    print('Aprovado')  
else:  
    print('Não aprovado')
```

Aprovado

Como a CONDIÇÃO é VERDADEIRA vamos para o bloco do IF identificado pela INDENTAÇÃO.

Perceba que a linha indicada está com um alinhamento diferente do IF. Isso significa que é um código que só será lido caso a condição seja atendida.

Nesse código, o programa “printará” o termo Aprovado.

```
nota=10  
  
if nota>=7:  
    print('Aprovado')  
else:  
    print('Não aprovado')
```

Aprovado

Vamos agora para um caso onde a nota é 5.

Assim como no primeiro caso o Python lerá que a Variável NOTA recebe o valor 5.

Ao chegar na linha do IF a condição será verificada.

O VALOR da VARIÁVEL NOTA é MAIOR ou IGUAL a 7?

Como NOTA =5 a resposta é FALSA.

nota=5

```
if nota>=7:  
    print('Aprovado')  
else:  
    print('Não aprovado')
```

Não aprovado

nota=5

```
if nota>=7:  
    print('Aprovado')  
else:  
    print('Não aprovado')
```

Não aprovado

Como a resposta é FALSA perceba que o Python pulará a linha print('Aprovado') e irá diretamente para o caso do ELSE.

O ELSE funciona como um “Nenhuma das opções anteriores”. O que isso significa?

Caso **nenhuma condição tenha sido atendida**, o código irá para o bloco do **else**.

Nesse caso , o termo indentado que dará como resultado NÃO APROVADO



```
nota=5
if nota>=7:
    print('Aprovado')
else:
    print('Não aprovado')
```

Não aprovado

```
nota=5
if nota>=7:
    print('Aprovado')
else:
    print('Não aprovado')
```

Não aprovado

```
nota=6

if nota>=7:
    print('Aprovado')
else:
    if nota>=5:
        print('Não aprovado/Recuperação')
    else:
        print('Não aprovado/Reprovado direto')
```

Não aprovado/Recuperação



Em alguns casos temos condições que só existem caso uma condição prévia tenha sido atendida.

Vamos pegar nosso exemplo anterior.

Vamos dizer que ao invés de apenas “Não aprovado” também seja necessário dar um status se foi “reprovado” ou se “está de recuperação”.

Nesse caso vamos usar um **if dentro do if** como no print ao lado.

```
nota=6

if nota>=7:
    print('Aprovado')
else:
    if nota>=5:
        print('Não aprovado/Recuperação')
    else:
        print('Não aprovado/Reprovado direto')


```

Não aprovado/Recuperação

Perceba que usamos a indentação para diferenciarmos os 2 blocos IF que temos.

**BLOCO IF EXTERNO:** considera todo o código. Perceba o alinhamento do if e do else.

**BLOCO IF INTERNO AO ELSE:** considera todo o bloco interno. Só será lido caso a condição nota>=7 do bloco externo não seja atendida.



### ATENÇÃO !

Diferente de outras linguagens onde é necessário “fechar” o IF, no Python, isso é feito pela indentação. Perceba como o alinhamento dos blocos indica a que bloco o mesmo pertence e quando o mesmo inicia e termina.

```
nota=6

if nota>=7:
    print('Aprovado')
else:
    if nota>=5:
        print('Não aprovado/Recuperação')

    print('Não aprovado/Reprovado direto')


```

Não aprovado/Recuperação  
Não aprovado/Reprovado direto

Vamos ver esse outro exemplo.

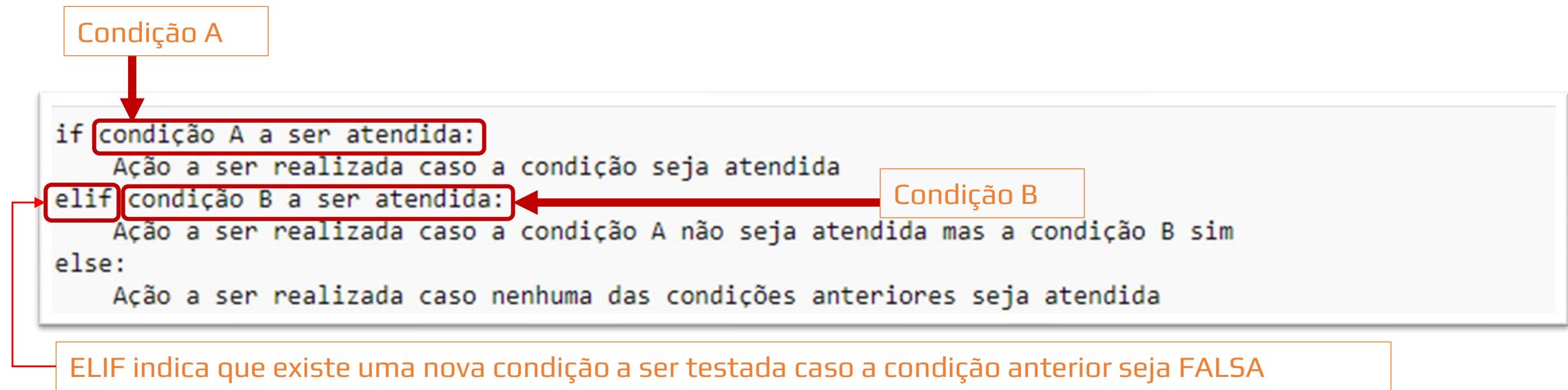
Aqui temos um erro comum que precisa ser entendido.

Perceba que foi retirado o ELSE do segundo bloco.

Isso afetou diretamente o resultado, pois para o Python, a última linha do código não está vinculada ao segundo bloco. Apenas ao primeiro.

Assim, ao entrar no ELSE do primeiro bloco, ele obrigatoriamente lerá o print ('Não aprovado/Reprovado direto') mesmo que a nota não atenda essa condição.

Agora que entendemos a estrutura do IF vamos entender um caso mais complexo onde não temos apenas 1 condição mas 2 ou mais.



Vamos avaliar o exemplo ao lado:

Nesse exemplo um bônus é calculado baseado no desempenho das vendas.

- 1) Caso a meta não seja atingida, não haverá bônus.
- 2) Caso as vendas superem a meta em 2x o bônus é calculado por: Vendas X 7%
- 3) Se a meta for superada mas inferior a 2x o bônus será calculado por : Vendas X 3%

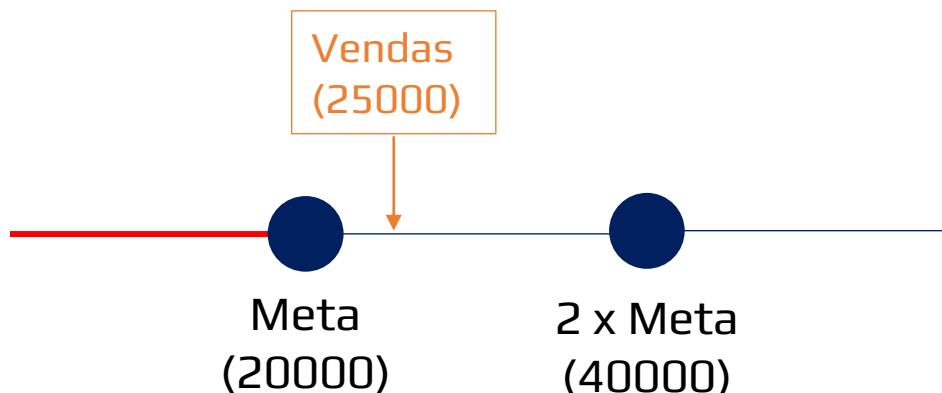
```
: meta = 20000
vendas = 25000

if vendas < meta:
    print('Não ganhou bônus')
elif vendas > (meta*2):
    bonus= 0.07*vendas
    print('Ganhou {} de bônus'.format(bonus))
else:
    bonus = 0.03 * vendas
    print('Ganhou {} de bônus' .format(bonus))

Ganhou 750.0 de bônus
```

Vamos avaliar o exemplo ao lado:

Primeiro testamos se a meta foi atingida. Como a condição não é atendida, pois, 25000 é MAIOR que 20000, vamos para a próxima condição.



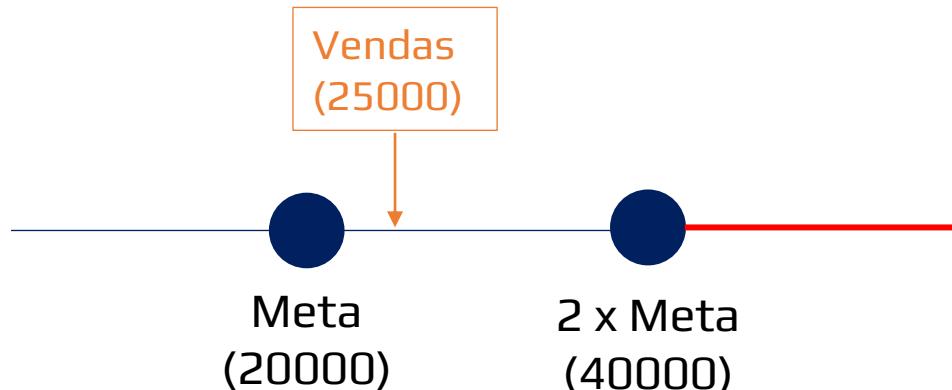
```
: meta = 20000
vendas = 25000

if vendas < meta:
    print('Não ganhou bônus')
elif vendas > (meta*2):
    bonus= 0.07*vendas
    print('Ganhou {} de bônus'.format(bonus))
else:
    bonus = 0.03 * vendas
    print('Ganhou {} de bônus' .format(bonus))

Ganhou 750.0 de bônus
```

Agora é onde temos novidade! Até agora, só tínhamos visto situações onde só existia uma condição. Nesse caso, temos mais de uma, logo, apenas o ELSE não seria suficiente para esse caso. Por isso, vamos usar ELIF + CONDIÇÃO.

Como a condição não é atendida, vamos para a próxima condição.



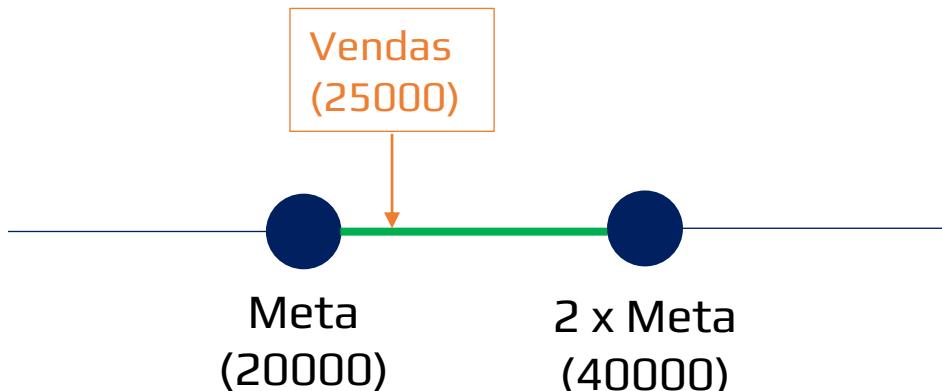
```
: meta = 20000
vendas = 25000

if vendas < meta:
    print('Não ganhou bônus')
elif vendas > (meta*2):
    bonus= 0.07*vendas
    print('Ganhou {} de bônus'.format(bonus))
else:
    bonus = 0.03 * vendas
    print('Ganhou {} de bônus' .format(bonus))

Ganhou 750.0 de bônus
```

Como nenhuma das condições anteriores foi atendida o código entrará no bloco do ELSE.

Aqui o cálculo do bônus será feito e através do método .Format vamos “printar” o resultado.



```
: meta = 20000
vendas = 25000

if vendas < meta:
    print('Não ganhou bônus')
elif vendas > (meta*2):
    bonus= 0.07*vendas
    print('Ganhou {} de bônus'.format(bonus))
else:
    bonus = 0.03 * vendas
    print('Ganhou {} de bônus' .format(bonus))

Ganhou 750.0 de bônus
```

Durante a descrição dos IFs anteriores usamos alguns comparadores como MAIOR ou IGUAL, MENOR ou IGUAL, etc. Abaixo colocamos os principais comparadores para você poder usar nos seus programas sem erro!

### IGUALDADE (==)

```
faturamento = 100
custo = 200
# IGUAL(==)
if faturamento == custo:
    print('IGUAL')
else:
    print('NÃO É IGUAL')
```

NÃO É IGUAL

### DIFERENTE (!=)

```
faturamento = 100
custo = 200
# DIFERENTE (!=)
if faturamento != custo:
    print('É DIFERENTE')
else:
    print('NÃO É DIFERENTE')
```

É DIFERENTE

### MAIOR (>)

```
faturamento = 100
custo = 200
# MAIOR(>)
if faturamento > custo:
    print('MAIOR')
else:
    print('NÃO É MAIOR')
```

NÃO É MAIOR

### MAIOR OU IGUAL(>=)

```
faturamento = 100
custo = 200
# MAIOR OU IGUAL(>=)
if faturamento >= custo:
    print('MAIOR OU IGUAL')
else:
    print('NÃO É MAIOR OU IGUAL')
```

NÃO É MAIOR OU IGUAL

### MENOR (<)

```
Faturamento = 100
custo = 200
# MENOR(<)
if faturamento < custo:
    print('MENOR')
else:
    print('NÃO É MENOR')
```

MENOR

### MENOR OU IGUAL (<=)

```
faturamento = 100
custo = 200
# MAIOR OU IGUAL(<=)
if faturamento <= custo:
    print('MENOR OU IGUAL')
else:
    print('NÃO É MENOR OU IGUAL')
```

MENOR OU IGUAL

Além dos itens de comparação anteriores temos também mais dois comparadores que serão muito usados:

O comparador IN permite identificar se algo existe ao menos uma vez em um texto, ou uma variável, lista etc. Mais para frente no curso vamos entender um pouco melhor a funcionalidade deste comparador.

### NÃO (not)

```
if not '#' in 'lira@hashtagtreinamentos.com.br':  
    print(' NÃO TEM #')  
else:  
    print('TEM #')
```

NÃO TEM #

### CONTÉM (in )

```
if '@' in 'lira@hashtagtreinamentos.com.br':  
    print('TEM @')  
else:  
    print('NÃO TEM @')
```

TEM @

Já o comparador NOT inverte o sentido da condição. No exemplo ao lado temos como condição:

SE “#” NÃO ESTIVER CONTIDO EM LIRA....

Como não há nessa string o caractere # o Python entenderá que a condição É VERDADEIRA. Pode ser estranho em um primeiro momento pois a inexistência atende a condição testada.

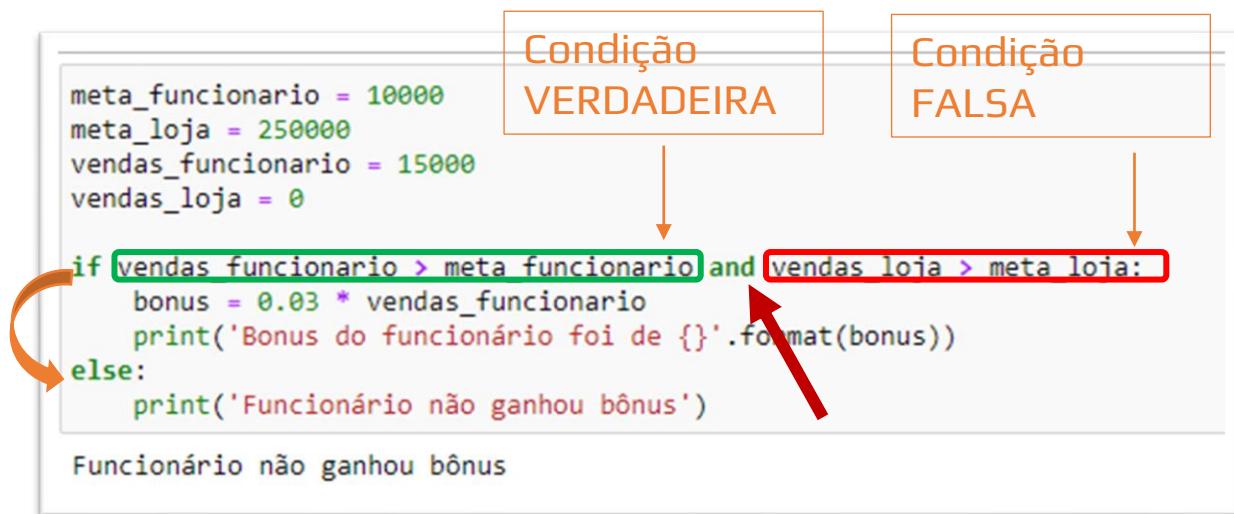
Usando os conectores AND e OR podemos acrescentar mais condições a um mesmo teste.

### AND

```
if (1ª condição a ser atendida) and (2ª condição a ser atendida)
    Ação a ser realizada caso a 1ª e 2ª condições forem atendidas
else:
    Ação a ser realizada em todos os outros casos. São eles:
    1) A 1ª condição é VERDADEIRA e a 2ª FALSA;
    2) A 1ª condição é FALSA e a 2ª VERDADEIRA;
    3) As duas condições são FALSAS;
```

### OR

```
if (1ª condição a ser atendida) or (2ª condição a ser atendida):
    Ação a ser realizada caso pelo menos a 1ª ou a 2ª condições forem atendidas. Ou seja:
    1) A 1ª condição é VERDADEIRA e a 2ª FALSA;
    2) A 1ª CONDIÇÃO é FALSA e a 2ª VERDADEIRA;
    3) As duas condições são VERDADEIRAS;
else:
    Ação a ser realizada caso as duas condições sejam FALSAS;
```



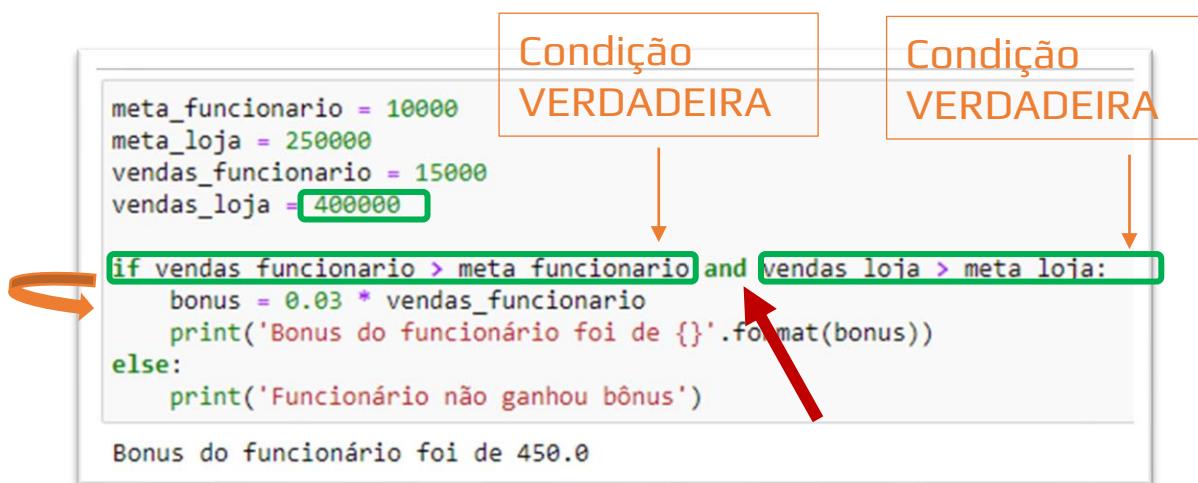
Vamos dar uma olhada nesse exemplo prático.

Aqui o cálculo do bônus depende de 2 condições para ocorrer.

Nesse caso, como usamos o AND necessariamente as duas condições precisam ser atendidas para o pagamento do bônus.

Considerando esse caso temos que `vendas_funcionario` é `> meta_funcionario` mas a outra condição `vendas_loja > meta_loja` não foi atendida.

Portanto, o Python, irá pular o bloco identado da condição IF e irá para o bloco ELSE.



Vamos considerar agora que o valor de venda ao invés de 0(exemplo anterior) passe a ser 40000.

Como ambas condições do IF são VERDADEIRAS, o bônus será calculado conforme descrito no bloco identado do IF.

```
meta_funcionario = 10000  
meta_loja = 25000  
vendas_funcionario = 15000  
vendas_loja = 0  
  
if vendas_funcionario > meta_funcionario or vendas_loja > meta_loja:  
    bonus = 0.03 * vendas_funcionario  
    print('Bonus do funcionário foi de {}'.format(bonus))  
else:  
    print('Funcionário não ganhou bônus')  
  
Bonus do funcionário foi de 450.0
```

Condição  
VERDADEIRA

Condição  
FALSA

Agora, vamos usar o OR:

Ao contrário do AND que precisava que ambas condições fossem verdadeiras no OR, caso PELO MENOS UMA for verdadeira a condição é atendida.

Nesse caso, como:

vendas\_funcionario > meta\_funcionario, é VERDADEIRO o Python entende que pelo menos uma das condições foi atendida e logo, pode rodar o bloco identado do IF.

Assim, o bônus é calculado conforme o código do bloco identado.

```
meta_funcionario = 10000  
meta_loja = 250000  
vendas_funcionario = 0  
vendas_loja = 0  
  
if vendas_funcionario > meta_funcionario or vendas_loja > meta_loja:  
    bonus = 0.03 * vendas_funcionario  
    print('Bonus do funcionário foi de {}'.format(bonus))  
else:  
    print('Funcionário não ganhou bônus')  
  
Funcionário não ganhou bônus
```

Condição FALSA

Condição FALSA

Vamos considerar agora que o vendas\_funcionario seja 0.

Como ambas condições do IF são FALSAS, o bônus não será calculado.

Assim, o Python pulará o bloco identado do IF e irá diretamente para o bloco ELSE.

```
faturamento = input('Qual foi o faturamento da loja nesse mês?')
custo = input('Qual foi o custo da loja nesse mês?')

lucro = int(faturamento) - int(custo)

print("O lucro da loja foi de {} reais".format(lucro))
```

Qual foi o faturamento da loja nesse mês?

Qual foi o custo da loja nesse mês?

```
-----  
ValueError                                              Traceback (most recent call last)
<ipython-input-2-9f67478e789d> in <module>
      2 custo = input('Qual foi o custo da loja nesse mês?')
      3
----> 4 lucro = int(faturamento) - int(custo)
      5
      6 print("O lucro da loja foi de {} reais".format(lucro))

ValueError: invalid literal for int() with base 10: ''
```

Nessa parte, vamos te explicar algumas comparações que em um primeiro momento não são tão simples de compreender.

Veja o exemplo ao lado:

O Usuário não forneceu nenhum valor para as duas perguntas feitas pelo Python através da função INPUT().

Ao invés de retornar algo como vazio, ou simplesmente nada o retorno foi de um erro no código. O que torna mais estranho é que o erro está representado na linha 4 onde é feito um cálculo simples de subtração entre as variáveis faturamento e custo.

Você sabe onde está o erro???

```
faturamento = input('Qual foi o faturamento da loja nesse mês?')
custo = input('Qual foi o custo da loja nesse mês?')

lucro = int(faturamento) - int(custo)

print("O lucro da loja foi de {} reais".format(lucro))
```

Qual foi o faturamento da loja nesse mês?

Qual foi o custo da loja nesse mês?

```
-----  
ValueError                                Traceback (most recent call last)
<ipython-input-2-9f67478e789d> in <module>
      2 custo = input('Qual foi o custo da loja nesse mês?')
      3
----> 4 lucro = int(faturamento) - int(custo)
      5
      6 print("O lucro da loja foi de {} reais".format(lucro))

ValueError: invalid literal for int() with base 10: ''
```

Vamos passo a passo:

O usuário não forneceu valores. Portanto, as variáveis faturamento e custo que recebem esses inputs estão com valor VAZIO.

(Atenção! VAZIO é diferente de ZERO.)

Na linha 4 o lucro é calculado transformando os valores das variáveis faturamento e custo em um número inteiro e subtraindo-os.

Como transformar VAZIO em um número inteiro?

Aí está o problema... O fato de não existir um valor atribuído a essas variáveis impede que o Python faça essa transformação e consequentemente o cálculo da variável LUCRO.

```
faturamento = input('Qual foi o faturamento da loja nesse mês?')
custo = input('Qual foi o custo da loja nesse mês?')

if faturamento!="" and custo!="":
    lucro = int(faturamento) - int(custo)
    print("O lucro da loja foi de {} reais".format(lucro))
else:
    print('Algum valor não foi fornecido')
```

Qual foi o faturamento da loja nesse mês?  
Qual foi o custo da loja nesse mês?  
Algum valor não foi fornecido

```
faturamento = input('Qual foi o faturamento da loja nesse mês?')
custo = input('Qual foi o custo da loja nesse mês?')

if faturamento and custo :
    lucro = int(faturamento) - int(custo)
    print("O lucro da loja foi de {} reais".format(lucro))
else:
    print('Algum valor não foi fornecido')
```

Qual foi o faturamento da loja nesse mês?  
Qual foi o custo da loja nesse mês?  
Algum valor não foi fornecido

Temos algumas formas para casos como esses:

Verificação com IF se os dois valores estão preenchidos e portanto, não são vazios.

Aqui é uma forma um pouco mais elegante que a forma anterior e será comum ver códigos como esses.

Aqui, o faturamento e o custo sem nenhuma operação associada significam:

O valor de faturamento é não Nulo?  
O valor de custo é não Nulo?

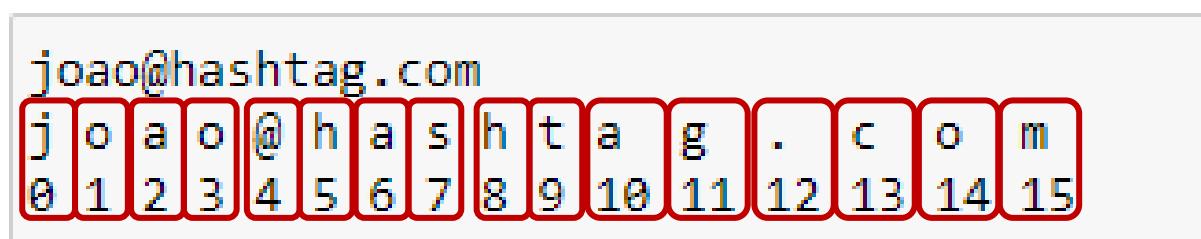
Módulo 7

# Strings – Textos e importância no Python

No módulo variáveis descobrimos que os textos no Python são em geral variáveis tipo STRING. No entanto, ainda temos alguns outros conhecimentos muito importante sobre esse tipo de variáveis que são fundamentais. Veja abaixo:

## STRINGS no Python são listas:

Ainda não vimos no detalhe o que são listas no Python, mas por enquanto, guarde essa informação e entenda que para o Python cada caractere é um item de uma lista.



Se pegarmos um e-mail genérico 'joao@hashtag.com' podemos dividir todos seus caracteres em uma lista.

Perceba que o primeiro caractere 'j' é o [0] e o 'm' o da posição [15].

Algo que pode gerar confusão no início, é que o número de caracteres é sempre um número a mais que a posição.

Como vemos na figura acima, como o Python inicia na posição 0 o número de caracteres sempre será 1 a mais que o número de posições.

Usando o mesmo exemplo, vamos agora ver como utilizar as posições no Python.

```
joao@hashtag.com  
j o a o @ h a s h t a g . c o m  
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
email='joao@hashtag.com'  
print(email[0]) # PEGANDO A POSIÇÃO [0] do valor atribuído a variável email
```

j

```
print(email[0:5]) # PEGANDO A POSIÇÃO [0] até a posição [5]* do valor atribuído a variável email
```

joao@

Perceba, que podemos acessar qualquer posição ou range de posições através da estrutura VARIÁVEL[POSIÇÃO]!

Outro ponto de atenção é o caso de range. Ao colocarmos [0:5] não estamos pegando a posição 5. Para o Python, este intervalo se inicia na posição [0] mas finaliza na posição [4].

Podemos associar métodos a nossa string que facilitarão muito o tratamento desses dados. Um dos métodos muito usados é o LEN()

```
joao@hashtag.com
j o a o @ h a s h t a g . c o m
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

In [1]: email='joao@hashtag.com'
len(email) # Fornece o tamanho da variável LEN. ATENÇÃO! CONSIDERA TODOS OS CARACTERES

Out[1]: 16
```



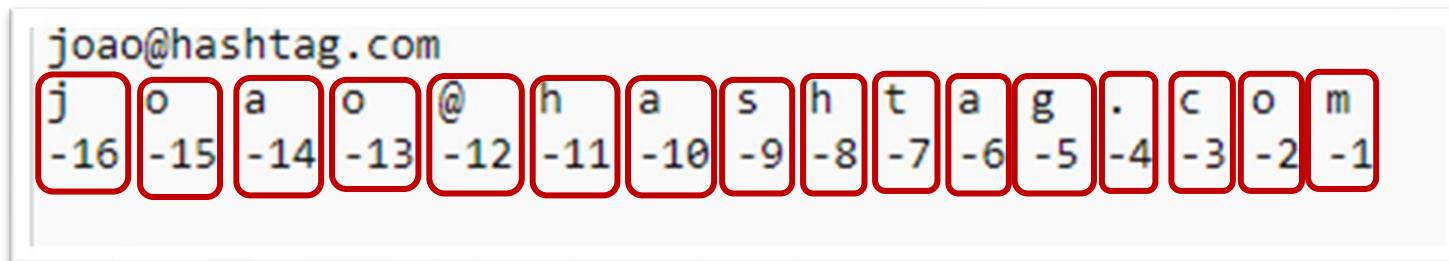
### ATENÇÃO !

O LEN() sempre contará (TODOS os caracteres da sua STRING. Ou seja, ESPAÇO(' '), VÍRGULAS (','), PONTOS('.'), etc serão considerados!

Como vimos anteriormente, o índice da posição dos caracteres segue o modelo abaixo.



Outra forma possível, é a posição com índice negativo. Veja o exemplo abaixo:



## Fica a dica!

Em geral, vamos usar o índice negativo em casos que é sabido que o buscamos está mais próximo do fim da string. MAS, nada impede que você use só o positivo ou só o negativo.

Perceba que as duas formas são válidas e coexistem no Python.  
Se quisermos o caractere '@' podemos usar tanto [4] como [-12].

Como vimos anteriormente, o índice da posição dos caracteres segue o modelo abaixo.

The string "joao@hashtag.com" is shown in blue. Below it, its characters are enclosed in red-bordered boxes, indexed from 0 to 15. The characters and their indices are: j(0), o(1), a(2), o(3), @(4), h(5), a(6), s(7), h(8), t(9), a(10), g(11), .(12), c(13), o(14), m(15).

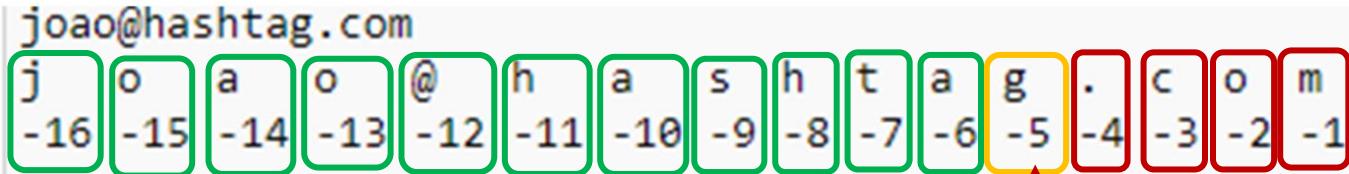
Outra forma possível, é a posição com índice negativo. Veja o exemplo abaixo:

The same string "joao@hashtag.com" is shown again. This time, the characters are indexed from -16 to -1. The characters and their indices are: j(-16), o(-15), a(-14), o(-13), @(-12), h(-11), a(-10), s(-9), h(-8), t(-7), a(-6), g(-5), .(-4), c(-3), o(-2), m(-1).

### Fica a dica!

Em geral, vamos usar o índice negativo em casos que é sabido que o buscamos está mais próximo do fim da string. MAS, nada impede que você use só o positivo ou só o negativo.

Perceba que as duas formas são válidas e coexistem no Python.  
Se quisermos o caractere '@' podemos usar tanto [4] como [-12].

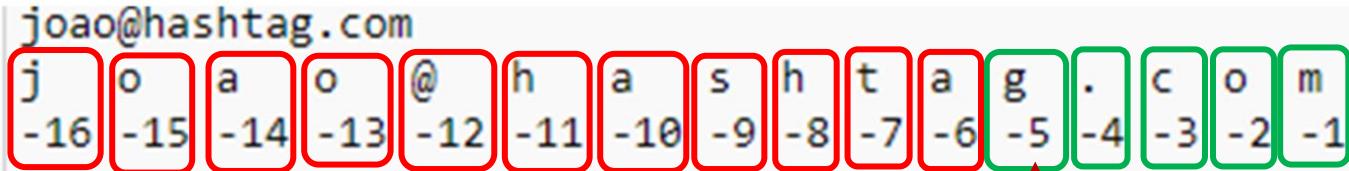


```
In [8]: email[:-5]  
Out[8]: 'joao@hashta'
```

':' Indica que se trata de um intervalo

Assim como vimos nos índices positivos, podemos utilizar a mesma lógica para pegarmos pedaços da string.

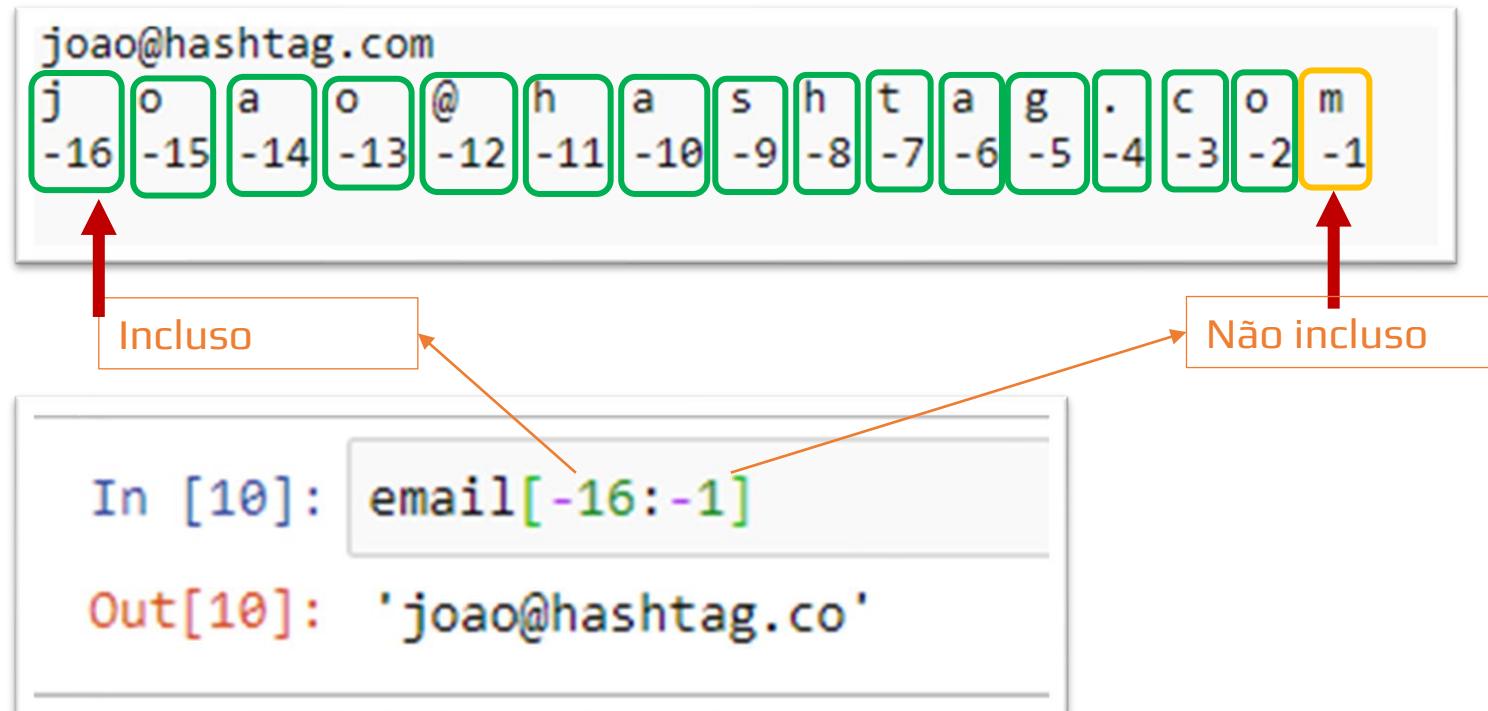
Todos os dados até o caractere [-5](não incluso)



```
In [9]: email[-5:]  
Out[9]: 'g.com'
```

Assim como vimos nos índices positivos, podemos utilizar a mesma lógica para pegarmos pedaços da string.

Todos os dados a partir do caractere [-5](nesse caso inclusivo)



Assim como vimos nos índices positivos, podemos utilizar a mesma lógica para pegarmos pedaços da string.

Todos os caracteres do índice [-16] até o [-1] (não inclusivo)

```
In [12]: dir(str)
Out[12]: ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Você deve se lembrar que no módulo XX falamos sobre métodos e o que eles são.

As strings, assim como os outros tipos de variáveis possuem uma série de métodos que podem ser utilizados para ajudar no tratamento dos dados.

Aqui do lado temos todos os métodos da string e no próximo slide, vamos ver os mais importantes.

## .CAPITALIZE

```
texto = 'lira'  
print(texto.capitalize())
```

Lira

Transforma APENAS a primeira letra de uma STRING  
em MAIÚSCULA

Transforma todas as letras MAIUSCULAS em  
MINÚSCULAS

## .CASEFOLD

```
texto = 'Lira'  
print(texto.casefold())
```

lira

**.COUNT**

```
texto = 'lira@yahoo.com.br'  
print(texto.count('.'))
```

2

CONTA o número de vezes que um caractere específico aparece na STRING. No caso ao lado :'

Retorna TRUE( VERDADEIRO) ou FALSE(FALSO) para um teste SE a string termina com um STRING específico. No exemplo ao lado, como 'lira@gmail.com' termina com gmail.com o resultado é TRUE.

**.ENDSWITH**

```
texto = 'lira@gmail.com'  
print(texto.endswith('gmail.com'))
```

True

## .FIND

```
texto = 'lira@gmail.com'  
print(texto.find('@'))
```

4

Encontra a posição do termo procurado.

Atenção! Lembre-se que a contagem de posição se inicia em [0]

Já falamos sobre ela anteriormente, lembra?

Ela insere o valor de uma variável no termo indicado por {}. Muito útil para evitar ter que transformar o formato de cada variável individualmente

## .FORMAT

```
faturamento = 1000  
print('O faturamento da loja foi de {} reais'.format(faturamento))
```

O faturamento da loja foi de 1000 reais

**.ISALNUM**

```
: texto = 'João123'  
print(texto.isalnum())
```

True

Verifica se um texto é todo feito com caracteres alfanuméricos (letras e números) -> letras com acento ou ç são considerados letras para essa função.

**.ISALPHA**

```
texto = 'João'  
print(texto.isalpha())
```

True

Verifica se um texto é todo feito de letras.  
Caso o texto fosse 'João123' o retorno seria FALSE visto que 123 não são letras.

**.ISNUMERIC**

```
: texto = '123'  
print(texto.isnumeric())
```

True

Verifica se um texto é todo feito por números.

Substitui um caractere escolhido por outro.

No exemplo ao lado temos que o símbolo PONTO('.)  
foi alterado por VÍRGULA(',') .

Atenção! Nesse caso, veja que temos 2 argumentos  
no método. 2 pontos são importantes:

- 1) A ordem faz diferença;
- 2) A VÍRGULA indicada em vermelho é o separador  
dos dois argumentos

**.REPLACE**

```
texto = '1000.00'  
print(texto.replace('.','.',','))
```

1000,00



Separador dos  
argumentos

### .SPLIT

```
texto = 'lira@gmail.com'  
print(texto.split('@'))  
  
['lira', 'gmail.com']
```

Separa o texto da STRING baseado em algum caractere indicado. No caso ao lado temos a separação do texto antes e depois do '@'. Perceba, que o split já criou uma lista ao fazer essa separação. Isso será bem útil para você

### .SPLITLINES

```
texto = '''Olá, bom dia  
Venho por meio desse e-mail lhe informar o faturamento da loja no dia de hoje.  
Faturamento = R$2.500,00  
'''  
  
print(texto.splitlines())  
  
['Olá, bom dia', 'Venho por meio desse e-mail lhe informar o faturamento da loja no dia de hoje.', 'Faturamento = R$2.500,00']
```

Cria uma lista, onde cada item é o texto de uma linha. Cada "ENTER" é criado um novo item na lista.

**.TITLE**

```
texto = 'joão paulo lira'  
print(texto.title())
```

João Paulo Lira

Coloca todas as letras iniciais das palavras  
MAIÚSCULAS

Retira os caracteres indesejados, como por exemplo,  
espaços que não agregam valor.

Perceba que no resultado fornecido pelo Python, não  
existem os espaços indesejados.

**.STRIP**

```
: texto = ' BEB123453 '  
print(texto.strip())
```

BEB123453

Espaços  
indesejados

**.STARTSWITH**

```
texto = 'BEB123453'  
print(texto.startswith('BEB'))
```

True

Retorna TRUE ou FALSE para se uma STRING se inicia com um texto específico.

No caso ao lado temos que BEB123453 se inicia com BEB, logo, o Python retorna TRUE.

**.UPPER**

```
texto = 'beb12343'  
print(texto.upper())
```

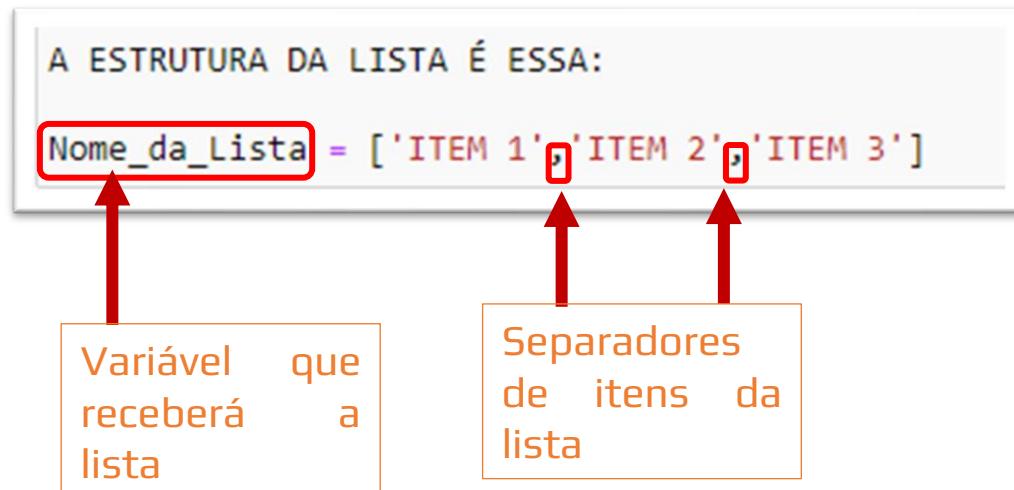
BEB12343

Altera todo o texto para MAIÚSCULAS. Números ficam inalterados.

# Módulo 8

## Listas Python – Métodos e Usos

Esse módulo vamos tratar das Listas que são estruturas muito importantes e serão muito utilizadas no curso. Vamos dar uma olhada na estrutura de uma lista e algumas variações que você pode acabar encontrando por aí em outros códigos.



Os marcadores de uma lista são os colchetes [ ]. Dentro de uma lista teremos itens separados por VÍRGULAS ','.

Caso o item seja um texto, é importante usar as aspas. Caso seja um número ou variável não são necessárias.

Esse módulo vamos tratar das Listas que são estruturas muito importantes e serão muito utilizadas no curso. Vamos dar uma olhada na estrutura de uma lista e algumas variações que você pode acabar encontrando por aí em outros códigos.

**Nome\_da\_Lista = [ ]**

As listas podem ser vazias. Em breve vamos explicar como adicionar itens a elas.

As listas podem conter outras listas. Perceba no Output que temos uma lista (todas\_listas) que contém as 3 listas.

Percebeu também que não usamos strings nos itens? Usamos diretamente a variável

```
lista1=[1,2,3,4,5]
lista2=[6,7,8,9,10]
lista3=[11,12,13,14,15]
todas_listas = [lista1,lista2,lista3]
print(todas_listas)
```

```
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
```

```
Nome_da_Lista = [
    'ITEM 1',
    'ITEM 2',
    'ITEM 3'
]
```

Outra forma de definir uma lista é usando uma linha para cada item. Isso vai variar de programador para programador, mas é importante que você saiba que o resultado é o mesmo. É apenas uma forma de apresentar o código.

Agora vamos entender como acessar os itens da lista.

Lembra como fizemos com a STRING, onde o caractere podia ser acessado usando a estrutura TEXTO[POSIÇÃO]?

Nas listas vamos usar o mesmo conceito:

```
produtos=['tv', 'celular', 'mouse', 'teclado', 'tablet']
          0           1           2           3           4
```

```
print(produtos[1])
```

celular

```
produtos=['tv', 'celular', 'mouse', 'teclado', 'tablet']
          0           1           2           3           4
```

```
print(produtos[4])
```

tablet

```
produtos = ['tv', 'celular', 'mouse', 'teclado', 'tablet']
           0         1         2         3         4
vendas =   [1000, 1500,    350,     270,     900]

print('As vendas de {} foram de {}'.format(produtos[1],vendas[1]))
```

As vendas de celular foram de 1500



### ATENÇÃO !

No caso acima, a posição das vendas e produtos estão coincidindo, mas não obrigatoriamente isso ocorrerá sempre que estivermos trabalhando com DUAS listas distintas. Fique atento! Mais para frente no curso vamos conhecer outros recursos para tratarmos esses casos.

Vamos para um outro caso agora onde temos duas listas distintas mas que possuem informações que são complementares entre si.

Vamos ver o exemplo ao lado.

Aqui temos duas listas que não possuem vínculo entre si, mas são dados complementares.

Podemos usar o método .FORMAT para acessarmos os dados de produto e venda de forma simples.

Agora que sabemos como acessar os itens da lista pelo índice dele, vamos aprender como fazer o inverso. Ou seja, descobrir qual o índice de um item conhecido.

Nós vamos chamar esse índice de *i* e vamos usar a estrutura abaixo para buscar esse valor:

```
i = nome_da_lista.index('o_que_você_procura')
```

Vamos para o exemplo:

```
produtos = ['tv', 'celular', 'mouse', 'teclado', 'tablet']

i=produtos.index('mouse')
print('O valor de i é ' + str(i))
print('O produto da posição i é ' + str(produtos[i]))
```

```
O valor de i é 2
O produto da posição i é mouse
```

E se o item que estou buscando não está na lista?  
Como saber, ou como checar essa informação?

Aqui, vamos juntar vemos conhecimentos que vimos até agora.

IF, conector IN, INPUT. Preparado? Vamos lá!

Imagine um programa que te dá a posição de um produto em uma lista baseado na informação fornecida por um usuário.

```
produtos = ['tv', 'celular', 'tablet', 'mouse', 'teclado', 'geladeira', 'forno']
estoque = [100, 150, 100, 120, 70, 180, 80]
```

```
produto = input('Insira o nome do produto em letra minúscula')
i = produtos.index(produto)
print(i)
```

Insira o nome do produto em letra minúscula geladeeeeira

Informação fornecida pelo usuário não está contida na lista.

Como podemos ver, caso não seja escrito no código, sempre que um usuário informar algo que não esteja na lista o programa irá acusar um erro.

Isso não é interessante tanto em termos de funcionalidade quanto de usabilidade.

Sabe como resolver esse problema?

```
produtos = ['tv', 'celular', 'tablet', 'mouse', 'teclado', 'geladeira', 'forno']  
estoque = [100, 150, 100, 120, 70, 180, 80]
```

```
produto = input('Insira o nome do produto em letra minúscula')  
i = produtos.index(produto)  
print(i)
```

```
Insira o nome do produto em letra minúscula geladeeeeira
```

```
-----  
ValueError                                                 Traceback (most recent call last)  
<ipython-input-6-c0de6eb6700e> in <module>  
      1 produto = input('Insira o nome do produto em letra minúscula')  
----> 2 i = produtos.index(produto)  
      3 print(i)
```

```
ValueError: 'geladeeeeira' is not in list
```

Primeiro é muito importante entender a condição que utilizamos no IF.

Perceba que ali usamos PRODUTO in PRODUTOS.

Apesar de serem variáveis com nomes muito parecidos estamos tratando de coisas diferentes.

A variável PRODUTOS é UMA lista onde todos os produtos estão listado.

Já a variável PRODUTO é uma variável criada localmente no IF para que ela possa percorrer todos os itens da lista PRODUTOS.

```
produto = input('Insira o nome do produto em letra minúscula')
if produto in produtos:
    i = produtos.index(produto)
    qtde_estoque = estoque[i]
    print('Temos {} unidades de {} no estoque'.format(qtde_estoque, produto))
else:
    print('{} não existe no estoque'.format(produto))

Insira o nome do produto em letra minúscula geladeira
geladeira não existe no estoque
```

Vamos entender agora como excluir ou remover um item à nossa lista:

#### ADICIONAR -> .APPEND

```
produtos = ['tv', 'celular', 'tablet', 'mouse', 'teclado', 'geladeira', 'forno']

produtos.append('iphone')
print(produtos)

['tv', 'celular', 'tablet', 'mouse', 'teclado', 'geladeira', 'forno', 'iphone']
```

#### REMOVER -> .REMOVE

```
produtos = ['tv', 'celular', 'tablet', 'mouse', 'teclado', 'geladeira', 'forno']

: produtos.remove('mouse')
print(produtos)

['tv', 'celular', 'tablet', 'teclado', 'geladeira', 'forno', 'iphone']
```



#### ATENÇÃO !

Apesar de .REMOVE e POP parecem iguais elas são bem diferentes.

O .REMOVE DELETA o valor da lista.

Já o .POP retira esse valor da lista, mas não o DELETA podendo SER ARMAZENADO em uma variável auxiliar como no exemplo ao lado.

Outra diferença é que o .POP se utiliza do índice diferentemente do remove.

Vamos agora fazer o mesmo exercício que resolvemos alguns slides atrás.

Se o usuário tentar remover um item que não existe na lista. Como evitar o erro?

Se você se lembra bem, podemos usar o IF/ELSE para casos como esses. No entanto, existe outra forma.

Essa forma é o TRY e o EXCEPT!

Vamos dar uma olhada na estrutura dele:

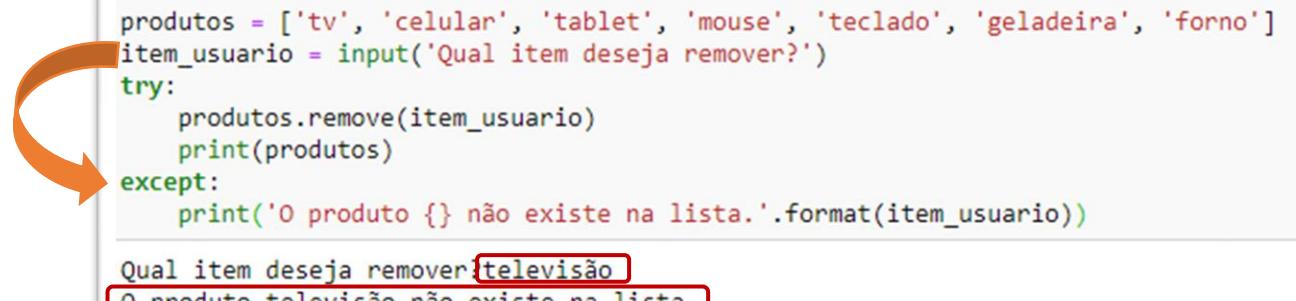
```
try:  
    lista.remove(Item que você quer tentar remover)  
except:  
    Ação que você deseja ter caso não seja possível remover o item.|
```

“:” são obrigatórios ao usar o TRY

Estas são IDENTAÇÕES. Elas indicam para o Python que ações estão no TRY ou no EXCEPT

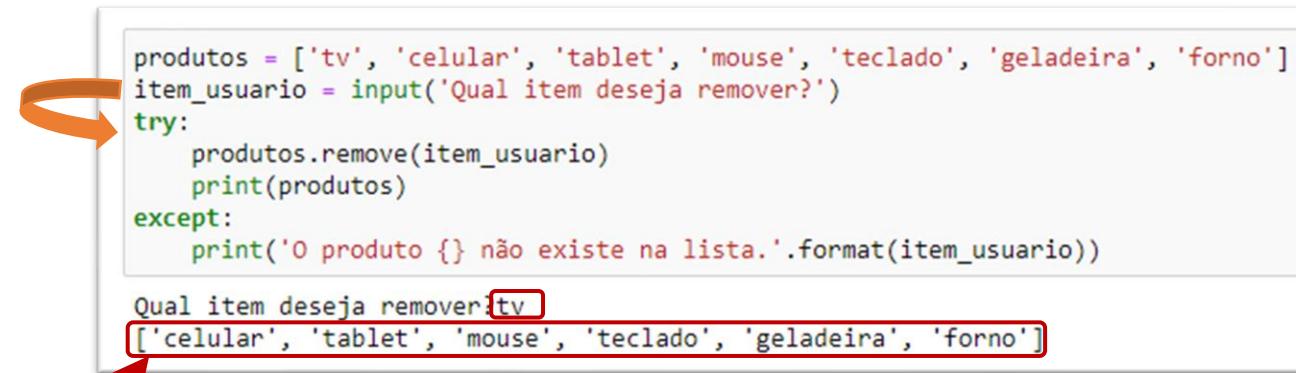
Voltando para nosso exemplo anterior, vamos usar o TRY para evitar que ocorra um erro ao tentarmos remover um item fornecido pelo usuário que não existe na lista.

No segundo exemplo, mostramos como o TRY ocorre quando a condição é atendida. Nesse caso, removendo o item TV da lista



```
produtos = ['tv', 'celular', 'tablet', 'mouse', 'teclado', 'geladeira', 'forno']
item_usuario = input('Qual item deseja remover?')
try:
    produtos.remove(item_usuario)
    print(produtos)
except:
    print('O produto {} não existe na lista.'.format(item_usuario))

Qual item deseja remover?televisão
O produto televisão não existe na lista.
```



```
produtos = ['tv', 'celular', 'tablet', 'mouse', 'teclado', 'geladeira', 'forno']
item_usuario = input('Qual item deseja remover?')
try:
    produtos.remove(item_usuario)
    print(produtos)
except:
    print('O produto {} não existe na lista.'.format(item_usuario))

Qual item deseja remover?tv
['celular', 'tablet', 'mouse', 'teclado', 'geladeira', 'forno']
```

Lista sem o item 'tv'

Vamos agora aprender 3 métodos que nos ajudarão a ter informações sobre as nossas listas.

**LEN()** – Indica o tamanho da lista(Nº de itens)

**MAX()** – Indica o maior item da lista

**MIN()** – Indica o menor item da lista

```
vendas = [10,50,670,16,46,87,99]
print(len(vendas))
print(max(vendas))
print(min(vendas))
```

```
7
670
10
```

Outra atividade muito comum no dia a dia é juntar listas em uma única lista. Para fazermos isso no Python é bem simples. Basta a utilização do método **.EXTEND**

```
produtos = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']  
novos_produtos = ['chromecast', 'windows phone']
```

```
produtos.extend(novos_produtos)  
print(produtos)
```

```
['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods', 'chromecast', 'windows phone']
```

Outra forma bastante usual é a utilização do sinal '+'. No entanto, perceba que nesse caso é necessário a criação de uma nova variável para receber a lista. As listas originais permanecem inalteradas

```
produtos = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']  
novos_produtos = ['chromecast', 'windows phone']
```

```
produtos_compilado = produtos + novos_produtos  
print(produtos_compilado)
```

Lista produtos

Lista novos\_produtos

```
[ 'apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods', 'chromecast', 'windows phone']
```

É possível que alguns de vocês tenha se perguntado: “Posso usar o **.APPEND** para adicionar a lista?

Vamos ver quais são as diferenças.

```
produtos = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']
novos_produtos = ['chromecast', 'windows phone']

print('Usando +:')
produtos_compilado = produtos + novos_produtos
print(produtos_compilado)
print('Usando Append:')
produtos.append(novos_produtos)
print(produtos)
```

Usando +:

['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods', 'chromecast', 'windows phone']

Usando Append:

['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods', ['chromecast', 'windows phone']]

Lista  
dentro  
de uma  
lista

Perceba que usando ‘+’ foram adicionados 2 itens a uma NOVA LISTA PRODUTOS\_COMPIADOS

Já usando o método **.APPEND**, é criado um ÚNICO ITEM que é uma LISTA DENTRO DE UMA LISTA.

E se ocorrer de termos na lista nova um valor IGUAL ao já existente na lista original? Vamos ver esse caso:

```
produtos = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']
novos_produtos = ['chromecast', 'airpods']

produtos.extend(novos_produtos)
print(produtos)

['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods', 'chromecast', 'airpods']
```

Valores duplicados



Perceba que quando estamos falando de listas é sim possível duplicarmos os valores.

E se estivermos tratando de uma lista numérica? Até o momento só tratamos de listas não numéricas.

Perceba que o efeito é exatamente o mesmo, seja com o método .EXTEND ou com o uso do +.

Lembrando que a diferença entre os dois casos é que para o uso do + é necessária a criação de uma nova variável.

```
lista1 = [100 , 50 , 25, 75, 80]
lista2 = [700, 10, 80, 90, 20]

lista1.extend(lista2)
print(lista1)

lista1 = [100 , 50 , 25, 75, 80]
lista2 = [700, 10, 80, 90, 20]

lista3 = lista1+lista2
print(lista3)

[100, 50, 25, 75, 80, 700, 10, 80, 90, 20]
[100, 50, 25, 75, 80, 700, 10, 80, 90, 20]
```

Usando o método `.SORT()` podemos ordenar uma lista. Abaixo temos um exemplo de uma lista não numérica. Algo chama atenção?

```
produtos = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']  
vendas = [ 1000 , 1500 , 15000 , 270 , 900 , 100 , 1200 ]
```

```
#Ordenando a Lista produtos  
produtos.sort()  
print(produtos)
```

```
['IPad', 'airpods', 'apple tv', 'apple watch', 'iphone x', 'mac', 'mac book']
```

É possível que você esteja estrenhando 'Ipad' antes de 'airpods'. Isso acontece pois o Python segue a lista ASCII que entende letras maiúsculas como anteriores as minúsculas. Portanto, atenção!

Vamos agora usar o nosso método **.SORT()** na lista vendas que é totalmente numérica.

```
produtos = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']
vendas = [ 1000 , 1500 , 15000 , 270 , 900 , 100 , 1200 ]
```

```
#Ordenando lista vendas
vendas.sort()
print(vendas)
```

```
[100, 270, 900, 1000, 1200, 1500, 15000]
```



Aqui, por se tratarem de números, a lista é ordenada do menor para o maior.

```
produtos = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']  
vendas = [ 1000 , 1500 , 15000 , 270 , 900 , 100 , 1200 ]
```

#Ordenando a lista produtos

```
produtos.sort()  
print(produtos)
```

```
['IPad', 'airpods', 'apple tv', 'apple watch', 'iphone x', 'mac', 'mac book']
```

#Ordenando lista vendas

```
vendas.sort()  
print(vendas)
```

```
[100, 270, 900, 1000, 1200, 1500, 15000]
```



### ATENÇÃO!

Como estamos tratando de 2 listas distintas (mesmo que sejam complementares) ao usar o método .SORT as posições não são mais equivalentes. Ou seja, na lista original tínhamos que produtos[0]==apple tv e vendas[0]== 1000. Após o uso do sort temos que 'apple tv' está em produtos [2] e 1000 em vendas[3].

Outra forma de utilizarmos nosso método **.SORT()** é usando o argumento **REVERSE**. Vamos para o exemplo:

```
produtos = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']
vendas = [ 1000      , 1500 , 15000      , 270   ,    900      ,    100      , 1200 ]
```

```
#Ordenando lista vendas
vendas.sort(reverse=True)
print(vendas)
```

```
[15000, 1500, 1200, 1000, 900, 270, 100]
```

Perceba que ao usarmos reverse=True, a ordem foi invertida do maior para menos.

Usando o reverse=False temos o output default que é do menor para o maior.

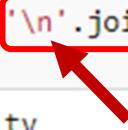
Continuando com nossas listas anteriores, vamos entender como apresentar nossas listas de forma um pouco mais simples.

Imagine que você esteja mandando a lista produtos para seu chefe e um e-mail. Como seria melhor ele receber? Todos os dados em uma linha ou como no exemplo ao lado?

Para fazermos isso vamos usar um método que concatena textos, o **JOIN()**.

Outra coisa que ainda não usamos é o '\n' esse caractere para o Python significa QUEBRA DE LINHA. Por isso conseguimos esse formato de um item da lista por linha.

```
produtos = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']
vendas = [ 1000 , 1500 , 15000 , 270 , 900 , 10000 , 100000 ]\n\nprint( '\n'.join(produtos) )
```



```
apple tv
mac
iphone x
IPad
apple watch
mac book
airpods
```

Vamos agora juntar conhecimentos no módulo de Strings e Listas. Vamos considerar que temos o texto atribuído a lista produtos(Atenção! Não é uma lista e sim uma string)

```
produtos = 'apple tv, mac, iphone x, iPad, apple watch, mac book, airpods'
```

Vamos transformar esse texto em uma lista. Transformar STRINGS em LISTAS facilita MUITO o trabalho!

```
lista_produtos = produtos.split(',')  
print(lista_produtos)
```

```
['apple tv', 'mac', 'iphone x', 'iPad', 'apple watch', 'mac book', 'airpods']
```

Conseguimos! MAS algo ainda está estranho... Percebe que temos um espaço antes dos produtos? Vamos precisar tratar isso também!

```
lista_produtos = produtos.split(',')
print(lista_produtos)

['apple tv', ' mac', ' iphone x', ' iPad', ' apple watch', ' mac book', ' airpods']
```

Vamos adicionar um ' '(ESPAÇO) ao nosso split. Isso fará com que o Python entenda esse 2 caracteres como separadores e não apenas a ','(VÍRGULA)

```
produtos = 'apple tv, mac, iphone x, iPad, apple watch, mac book, airpods'
lista_produtos = produtos.split(',')
print(lista_produtos)

['apple tv', 'mac', 'iphone x', 'iPad', 'apple watch', 'mac book', 'airpods']
```

Até agora sempre vimos variáveis que possuem valores estáticos como:

Faturamento = 1000

Essas linhas são bem comuns mas algumas vezes você verá linhas como:

faturamento = faturamento + 500

faturamento += 500

faturamento -= 500

Não se assuste! Esses códigos são formas diferentes de se alterar uma variável. Serão bastante utilizadas quando usarmos as repetições.

```
faturamento = 1000
faturamento = faturamento +1000
print(faturamento) # Valor anterior de faturamento + 1000
faturamento += 500
print(faturamento) # valor anterior de faturamento + 500
faturamento -= 500
print(faturamento) # valor anterior de faturamento - 500
```

2000

2500

2000

Vamos entender o que acontece aqui:

Inicialmente, nossa variável faturamento vale 1000.

Ao usarmos `faturamento + 1000`, estamos alterando o valor de faturamento para um novo valor.

Isso pode parecer estranho pois temos faturamento dos dois lados, mas é bem simples .

Veja sempre o lado esquerdo como RECEBE e o lado direito como ALGO QUE SERÁ ATRIBUÍDO.

Portanto, temos:

faturamento RECEBE o valor de faturamento( Aqui usaremos o valor 1000) + o valor 1000 descrito na linha de código.

Logo, faturamento RECEBE  $1000+1000$ .

```
faturamento = 1000
faturamento = faturamento +1000 # Valor anterior de faturamento + 1000
print(faturamento)
faturamento += 500
print(faturamento) # valor anterior de faturamento + 500
faturamento -= 500
print(faturamento) # valor anterior de faturamento - 500
```

2000  
2500  
2000

Vamos para a linha de código faturamento +=500.

Aqui podemos perceber que não há uma duplicidade da célula faturamento, no entanto, é exatamente a mesma coisa do exemplo anterior escrito de forma diferente.

Aqui, o segundo 'faturamento' é suprimido.

Portanto, temos:

faturamento RECEBE o valor de faturamento( Aqui usaremos o valor 2000) + o valor 500 descrito na linha de código.

Logo, faturamento RECEBE 2000+500.

```
faturamento = 1000
faturamento = faturamento +1000
print(faturamento) # Valor anterior de faturamento + 1000
faturamento += 500
print(faturamento) # valor anterior de faturamento + 500
faturamento -= 500
print(faturamento) # valor anterior de faturamento - 500
```

2000  
2500  
2000



Da mesma forma temos a última expressão:  
faturamento -= 500

faturamento RECEBE o valor de faturamento( Aqui usaremos o valor 2500) - o valor 500 descrito na linha de código.

Logo, faturamento RECEBE 2500-500.

```
faturamento = 1000
faturamento = faturamento +1000
print(faturamento) # Valor anterior de faturamento + 1000
faturamento += 500
print(faturamento) # valor anterior de faturamento + 500
faturamento -= 500 # valor anterior de faturamento - 500
print(faturamento) # valor anterior de faturamento - 500
```

2000

2500

2000

Vamos entender agora uma particularidade das listas.

Até então quando fazíamos :

```
faturamento = 1000  
Faturamento2 = 2000
```

Entendíamos isso como 2 variáveis distintas que não possuem relação entre si.

O que é verdade!

```
faturamento = 1000  
faturamento2 = 2000  
print(faturamento)  
print(faturamento2)
```

```
1000  
2000
```

No entanto, quando tratamos de listas é um pouco diferente. No exemplo ao lado temos que a lista1=list2.

Ao alterarmos o item [2] da lista1 é esperado que lista1 não seja mais IGUAL a lista2...

No entanto, elas são...

```
lista1 = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']
lista2=lista1
```

```
lista1[2]='iphone12'
print(lista1)
print(lista2)
```

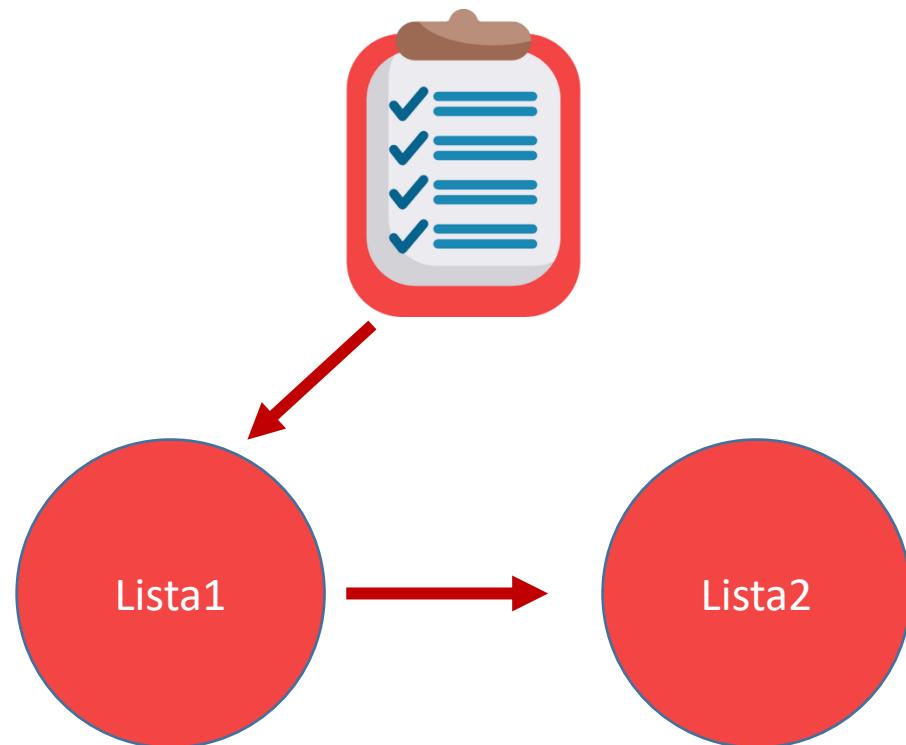
```
['apple tv', 'mac', 'iphone12', 'IPad', 'apple watch', 'mac book', 'airpods']
['apple tv', 'mac', 'iphone12', 'IPad', 'apple watch', 'mac book', 'airpods']
```

Vamos para o próximo slide para entender porque.

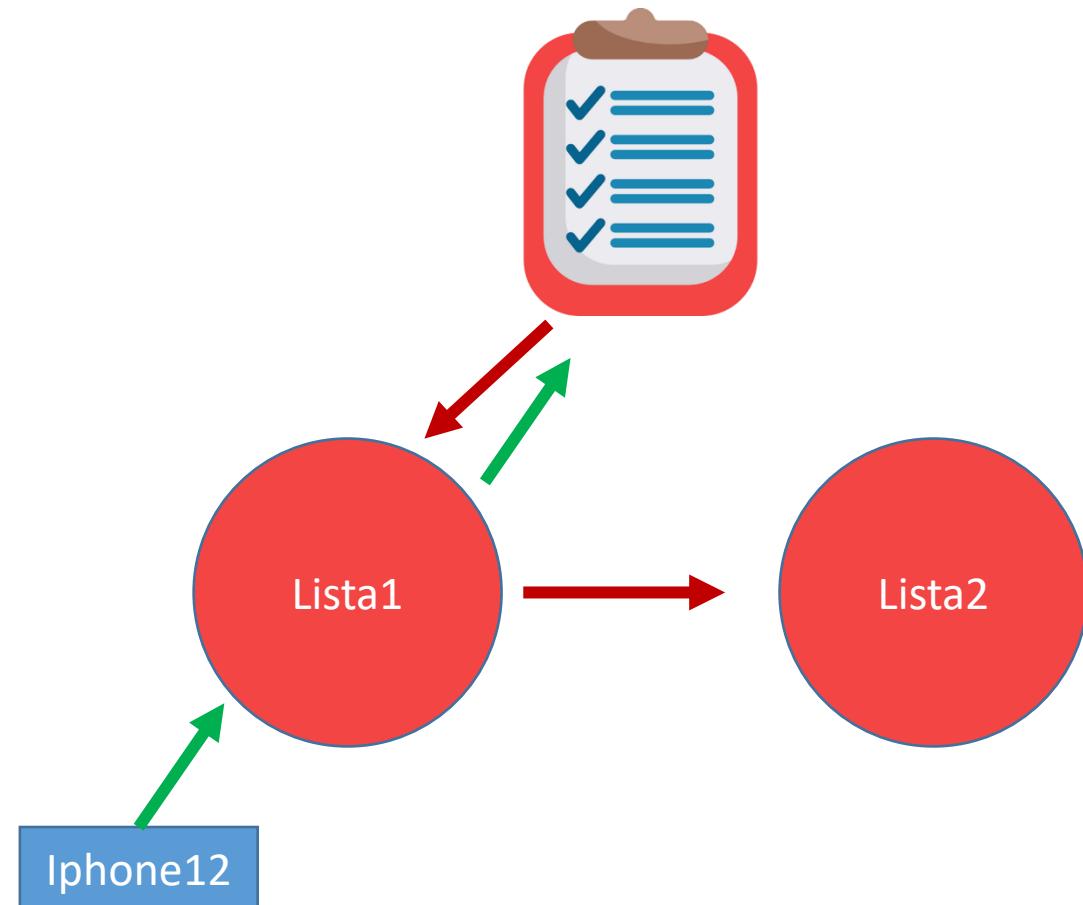
Quando se trata de listas nossa visão precisa mudar um pouco. Considere que a lista existe independentemente das variáveis.

Ao criarmos a variável 'lista1' o que fazemos é dizer para que essa variável puxe suas informações dessa lista.

Ao criarmos a lista2, estamos puxando todas as informações da lista1.



Ao alterarmos a lista1, essa informação é fornecida a lista original e consequentemente a lista2.



Portanto, ao trabalharmos com cópia de listas precisamos usar um método específico para isso.

Esse método é o **.COPY()**.

Este método realmente copia a lista e cria uma nova lista, sem relação com a lista anterior.

No exemplo podemos perceber que ao “printarmos” a lista 1 e lista 2 temos valores distintos.

```
lista1 = ['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']
lista2 = lista1.copy()
lista1[2] = 'iphone 12'
print(lista1)
print(lista2)

['apple tv', 'mac', 'iphone 12', 'IPad', 'apple watch', 'mac book', 'airpods']
['apple tv', 'mac', 'iphone x', 'IPad', 'apple watch', 'mac book', 'airpods']
```

Vamos pegar o seguinte caso:

4 vendedores de uma loja que vende Iphone e Ipad.



Lira:  
IPAD: 100  
IPHONE:200



João:  
IPAD: 300  
IPHONE:500



Diego:  
IPAD: 50  
IPHONE:1000



Alon:  
IPAD: 900  
IPHONE:10

Antes de ir para o próximo slide pense:

- Em quantas listas podemos dividir as informações ao lado?
- Como poderíamos acessar os dados de vendas IPAD feitas pelo Diego?

Caso não tenha conseguido, não se desespere! Faz parte. O mais importante é ter esse momentos onde saímos do código e conseguimos estruturar um problema real em linhas de código.

Vamos para o próximo slide e entender como poderíamos fazer 😊

Primeiro vamos entender de quantas listas estamos falando...

Uma primeira lista é a lista de vendedores onde temos:

- Lira;
- João;
- Diego;
- Alon

Item 2 da lista vendas

Item 4 da lista vendas

```
vendedores = ['Lira', 'João', 'Diego', 'Alon']
produtos = ['ipad', 'iphone']
vendas = [
    [100, 200], ← Item 1 da lista vendas
    [300, 500], ← Item 2 da lista vendas
    [50, 1000], ← Item 3 da lista vendas
    [900, 10], ← Item 4 da lista vendas
]
```

Nossa segunda lista é a de produtos vendidos na loja:

- Ipad;
- Iphone;

A terceira já é um pouco menos intuitiva. Cada item da nossa lista será uma lista. Vamos entender melhor:

- Item 1 da lista(referente ao vendedor Lira): 100(vendas de Ipad), 200(vendas de iphone);
- Item 2 da lista(referente ao vendedor João): 300(vendas de Ipad), 500(vendas de iphone);
- Item 3 da lista(referente ao vendedor Diego): 50(vendas de Ipad), 1000(vendas de iphone);
- Item 4 da lista(referente ao vendedor Alon): 900(vendas de Ipad), 10(vendas de iphone);

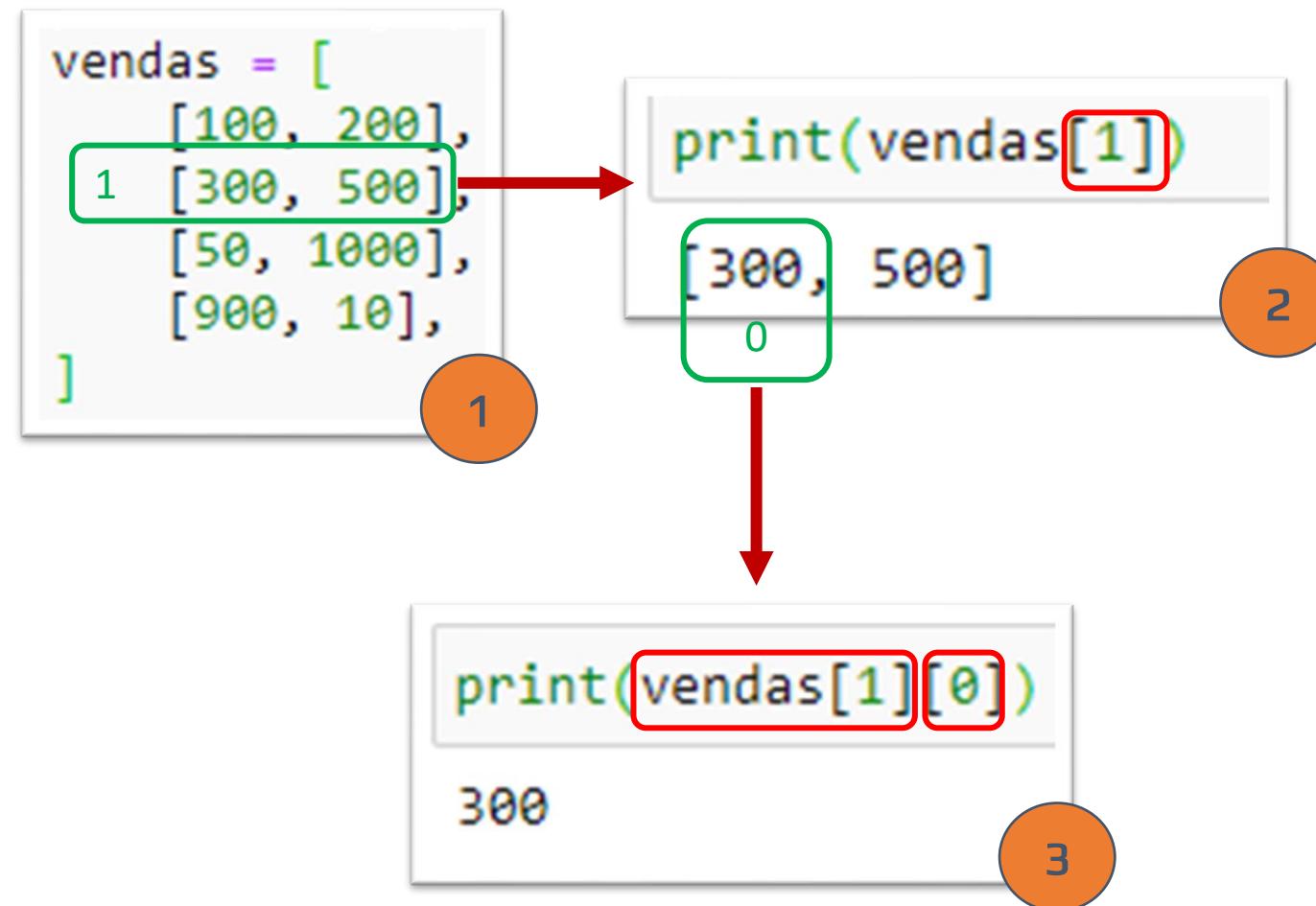
Ok. Mas como acessar esse itens da lista que está dentro da lista vendas?

Vamos por partes.

Ao fazermos `print(vendas[1])` iremos acessar todas as vendas do vendedor João. 300 corresponde as vendas de ipad e 500 as vendas de iphone.

Isso seria suficiente se fosse esse o nosso objetivo. Mas se quiséssemos acessar somente as vendas de iphone do vendedor João?

Usando `print(vendas[1][0])` podemos acessar essa informação. Vendas[1]significa acessar o segundo item da lista vendas. Já o [0] significa acessar o PRIMEIRO item da lista que está contida dentro deste item



# Módulo 9

# For – Estrutura de Repetição

Muitas vezes precisamos repetir ações não apenas 1 ou 2 vezes mas n vezes. Sendo n o número de linhas de uma planilha, número de dias no ano, número de clientes, número de cidades no estado do Rio de Janeiro, etc...

Replicar isso em um código seria inviável. Em geral, temos uma regra que sempre que você está escrevendo linhas de código iguais, possivelmente existe um jeito melhor de fazer. PS: Não se preocupe com isso agora. O tempo e a prática vão te fazer mais eficiente 😊

Para esses casos temos as estrutura de de repetição. Uma das estruturas mais úteis e usadas é a FOR.

Vamos dar uma olhada em como é essa estrutura:

`for inicio até nº de vezes que se pretende repetir :`

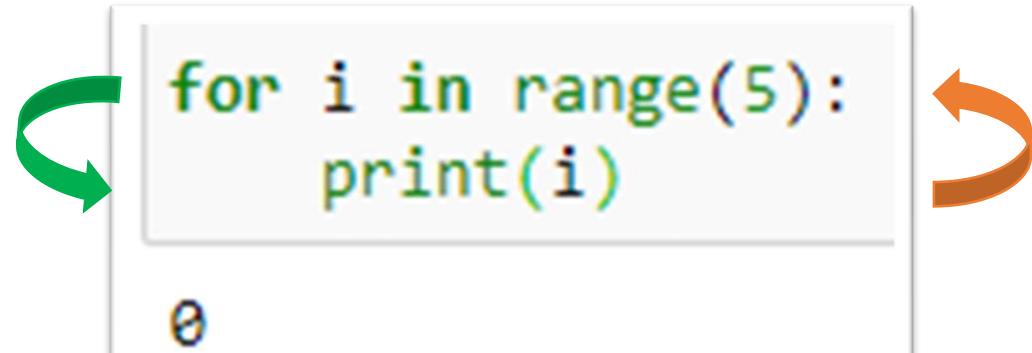
→ Ação 1 a ser realizada

→ Ação 2 a ser realizada

Indentação indica que essas linhas pertence ao FOR

→ Ação fora do for. Sem indentação!!!

Falta de Indentação indica que essa linha NÃO pertence ao FOR

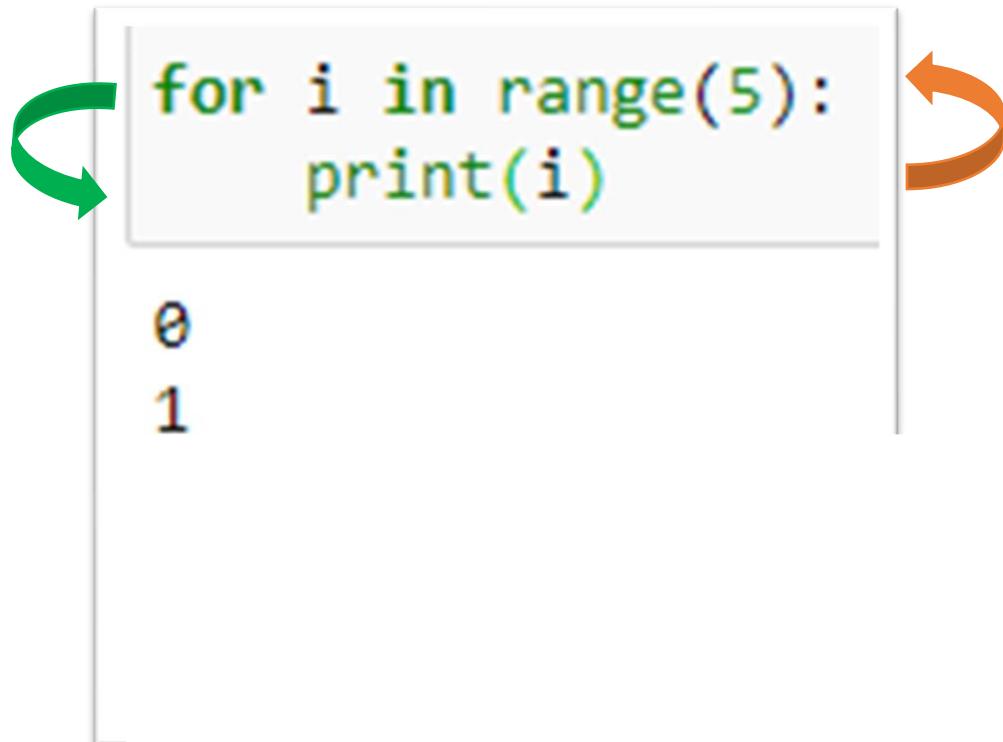


Aqui usamos a estrutura FOR para “printarmos” a variável i 5 vezes.

O uso do i é bem usual na programação e funciona como um contador. Como não definimos seu valor anteriormente, o Python assumirá que seu valor inicial é 0.

Logo, indo para primeira interação (indicada pela seta verde) o output é 0(ZERO)

Após esta ação, a estrutura for volta a linha superior(indicado pela seta laranja)



Após a primeira interação o valor de i será acrescido de uma unidade. Logo, seu novo valor é 1.

A seta verde indica a entrada na estrutura for que “printa” o valor de i (1)

Assim como na interação anterior, após a execução da identação, o python retorna para a linha do for para a terceira interação

A diagram illustrating the state of the variable `i` during the third iteration of a `for` loop. On the left, a code block contains the following Python code:

```
for i in range(5):
    print(i)
```

On the right, the value of `i` is shown as `2`, indicated by a green curved arrow pointing from the code to the value. A red curved arrow points from the value back to the code, indicating the flow of control.

Assim como nos casos anteriores é feita a TERCEIRA interação

A diagram illustrating the state of the variable `i` during the fourth iteration of a `for` loop. On the left, a code block contains the same Python code as the first diagram:

```
for i in range(5):
    print(i)
```

On the right, the value of `i` is shown as `3`, indicated by a green curved arrow pointing from the code to the value. A red curved arrow points from the value back to the code, indicating the flow of control.

Assim como nos casos anteriores é feita a TERCEIRA interação



```
for i in range(5):
    print(i)
```

0  
1  
2  
3  
4

Na quinta e última interação, é “printado” o valor de *i* mas não há retorno para o início do FOR. O código continuará a ser lido.

Uma dúvida comum é:  
“Por que o valor de *i* só vai até 4 se `range` é (5)?”

Lembre-se que o valor inicial de *i* era 0(ZERO) e que o `range(5)` significa “repetir 5 vezes” e não repetir até 5.

```
produtos = ['coca', 'pepsi', 'guaraná', 'sprite', 'fanta']
producao = [15000, 12000, 13000, 5000, 250]

print('Produção do ' + produtos[0] + ' é de ' + str(producao[0]))
print('Produção do ' + produtos[1] + ' é de ' + str(producao[1]))
print('Produção do ' + produtos[2] + ' é de ' + str(producao[2]))
print('Produção do ' + produtos[3] + ' é de ' + str(producao[3]))
print('Produção do ' + produtos[4] + ' é de ' + str(producao[4]))
```

Produção do coca é de 15000  
Produção do pepsi é de 12000  
Produção do guaraná é de 13000  
Produção do sprite é de 5000  
Produção do fanta é de 250

Vamos agora para um caso mais aplicável:  
Imagine que temos 2 listas produtos e produção.

```
produtos = ['coca', 'pepsi', 'guaraná', 'sprite', 'fanta']
producao = [15000, 12000, 13000, 5000, 250]
```

Se fosse pedido para elaborar um relatório que informasse a produção de todos os produtos poderíamos usar o código ao lado.

Conseguimos o resultado... Seria a melhor forma?  
Se ao invés de 5 produtos fossem 500 seria simples?

Pensa em alguma alternativa para esse caso?

```
produtos = ['coca','pepsi','guaraná','sprite','fanta']
producao = [15000,12000,13000,5000,250]

for i in range(5):# repete 5 vezes pois são 5 produtos!!
    print('Produção do {} é de {}'.format(produtos[i],producao[i]))
```

Produção do coca é de 15000 → Primeira Interação. i==0  
Produção do pepsi é de 12000 → Segunda Interação. i==1  
Produção do guaraná é de 13000 → Terceira Interação. i==2  
Produção do sprite é de 5000 → Quarta Interação. i==3  
Produção do fanta é de 250 → Quinta Interação. i==4

Vamos lá! Sabemos que temos o FOR que nos permite repetir o número de vezes que precisarmos

Além disso, vamos precisar de uma variável auxiliar para contar o número de interações. Vamos chama-la de i. Seu valor inicial será 0(ZERO).

Dentro da indentação do FOR ao invés de replicarmos 5 vezes a frase vamos usar o .FORMAT e o índice das nossas listas para printar o resultado!

```
produtos = ['coca', 'pepsi', 'guaraná', 'sprite', 'fanta', 'mineirinho']
producao = [15000, 12000, 13000, 5000, 250, 800]

tamanho = len(produtos)

for i in range(tamanho):
    print('Produção do {} é de {}'.format(produtos[i], producao[i]))
```

Produção do coca é de 15000  
Produção do pepsi é de 12000  
Produção do guaraná é de 13000  
Produção do sprite é de 5000  
Produção do fanta é de 250  
Produção do mineirinho é de 800



Conseguimos reduzir nosso código e otimizar bastante. No entanto, da forma que está descrito temos uma limitação... Caso a lista cresça para um número superior a 5 itens nosso for não contemplará todos os itens da lista.

Para isso vamos usar a função **LEN()**. Isso permitirá que mesmo que a lista aumente nosso for sempre conte cole todos os itens.

No exemplo anterior utilizamos o **Range()** para podermos informar o número de repetições que serão executadas no for.

No entanto, no Python é muito usual usarmos a própria lista para termos essa informação.

Como podemos ver no exemplo, criamos localmente uma variável item que irá percorrer todos os itens da lista produtos.

```
produtos = ['coca', 'pepsi', 'guaraná', 'sprite', 'fanta', 'mineirinho']

for produto in produtos:
    print('Produção do {}'.format(produto))
```

```
Produção do coca
Produção do pepsi
Produção do guaraná
Produção do sprite
Produção do fanta
Produção do mineirinho
```

```
produtos = ['coca', 'pepsi', 'guaraná', 'sprite', 'fanta', 'mineirinho']

for item in produtos:
    print('Produção do {}'.format(item))
```

```
Produção do coca
Produção do pepsi
Produção do guaraná
Produção do sprite
Produção do fanta
Produção do mineirinho
```

Dica: Em geral, no Python o que mais é utilizado são estruturas como a do exemplo ao lado.

Utilizamos a palavra referente a lista no singular PRODUTO e na lista a palavra no plural PRODUTOS.

Podemos ler essa estrutura como:  
“Para cada PRODUTO na lista PRODUTOS, “printar” o texto”

Podemos combinar o uso do for com o IF por exemplo. Como já vimos anteriormente o uso da indentação é muito importante para que o código seja lido pelo Python corretamente.

No modelo abaixo apresentamos uma estrutura genérica. Perceba que o If possui uma indentação em relação ao FOR e as linhas 3 e 4 possuem uma indentação “dupla” pois pertencem ao IF e ao FOR.

```
for item in listas :  
    if item ATENDE a CONDIÇÃO:  
        Ação 1 a ser realizada  
        Ação 2 a ser realizada
```

Indentação indica que essas linhas pertence ao FOR

Vamos imaginar agora que temos uma lista de vendas de um produto realizadas por diferentes vendedores. A empresa espera que a meta de vendas dos funcionários seja de 100 unidades vendidas.

Vamos criar um código que verifique de forma automática quantas vendedores conseguiram atingir a meta.

Para isso vamos precisar usar 2 conhecimentos.

- 1) FOR para poder passar por todos os itens da lista;
- 2) IF para verificarmos se o item é  $\geq 100$

```
vendas = [100,50,80,190,200,210,45,37,99,105,130,111,44,24,67,93,157,25]
meta = 100

for venda in vendas:
    if venda >= meta:
        print(venda)

100
190
200
210
105
130
111
157
```

**ATENÇÃO!**

As indentações precisam ser respeitadas.

Conseguimos “printar” todos os valores de venda  $\geq 100$ , mas ainda não é a informação que queríamos...

O ideal seria termos um “contador” que acumulasse o número de funcionários que bateram a meta. Com esse contador será possível por exemplo dizer automaticamente á % de atingimento de meta.

No exemplo ao lado fizemos essa melhoria.

Perceba que a variável `qtde_bateu_meta` precisa ser previamente definida com valor = 0, caso contrário, o Python não entenderá a linha de comando `+=1`.

```
vendas = [100, 50, 80, 190, 200, 210, 45, 37, 99, 105, 130, 111, 44, 24, 67, 93, 15]
meta = 100

qtde_bateu_meta = 0
for venda in vendas:
    if venda >= meta:
        qtde_bateu_meta += 1

qtde_funcionarios = len(vendas)
print('O percentual de pessoas que bateram a meta foi de {:.1%}'
      .format(qtde_bateu_meta / qtde_funcionarios))
```

O percentual de pessoas que bateram a meta foi de 44.4%

Aqui usamos um novo formato para o input do format.  
Ao invés de {} vamos usar o termo {:.1%}  
Vamos ver o que esses caracteres significam:

- : - indica que será formato de uma forma específica
- .ALGARISMO – indica o número de casas decimais. Neste caso UMA
- % - indica que se trata de uma PORCENTAGEM

```
funcionarios = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo']
for i, funcionario in enumerate(funcionarios):
    print('{} é o funcionário {}'.format(i, funcionario))
```

0 é o funcionário Maria  
1 é o funcionário José  
2 é o funcionário Antônio  
3 é o funcionário João  
4 é o funcionário Francisco  
5 é o funcionário Ana  
6 é o funcionário Luiz  
7 é o funcionário Paulo

Uma terceira forma de usarmos o FOR além do RANGE ou da própria lista, é usar o ENUMERATE(). Esta é uma forma que de certa forma combina as outras duas. Será usada principalmente em situações que é necessário saber a posição de um item.

No exemplo temos uma lista de funcionários e precisamos saber qual a posição desses funcionários. Usando o ENUMERATE, podemos coletar tanto a posição quanto o nome do funcionário.

Perceba que usamos duas variáveis no for ao mesmo tempo, **i** e **funcionário**.

Vamos considerar um novo exemplo mais aplicável. Vamos criar um código que nos permita monitorar os níveis de estoque de vários produtos e nos avise caso este produto esteja menor que o estoque mínimo definido.

Primeiro, vamos criar nosso for para “printar” todos as quantidades (qtde) em estoque.

No entanto, ainda não é suficiente para o que desejamos. Vamos precisar colocar algum teste que nos permita verificar se essas quantidades são Maiores ou menores que a variável nível\_mínimo.

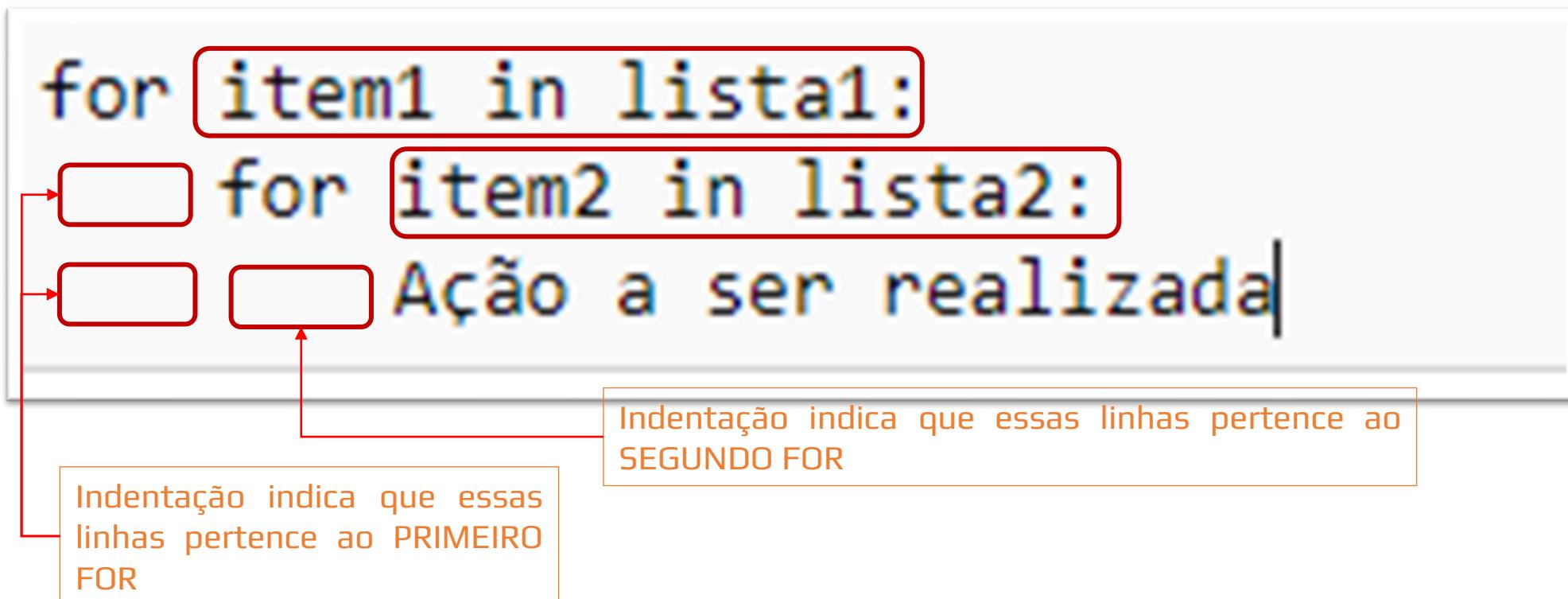
```
estoque = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100
produtos = ['coca', 'pepsi', 'guarana', 'skol', 'brahma', 'agua', 'del val'
nivel_minimo = 50

for qtde in estoque:
    print(qtde)
```



```
1200
300
800
1500
1900
2750
400
```

Assim como fizemos com o IF também podemos colocar um FOR dentro de um FOR. Assim como dito anteriormente, é muito importante ter atenção com a indentação. Ela identificará a ordem da execução das atividades



Usando o IF indentado dentro do For podemos realizar o teste citado no slide anterior. Ou seja, toda vez que uma quantidade(qtde) for MENOR(<) que o nível\_minimo vamos “printar” esse valor.

No entanto, ainda falta saber a que produto esses valores se referem...

```
estoque = [1200, 300, 800, 1500, 1900, 27
produtos = ['coca', 'pepsi', 'guarana',
nivel_minimo = 50

for i, qtde in enumerate(estoque):
    if qtde < nivel_minimo:
        print(produtos[i], qtde)

dolly 20
red bull 23
```

```
estoque = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90
produtos = ['coca', 'pepsi', 'guarana', 'skol', 'brahma', 'agua'
nivel_minimo = 50

for qtde in estoque:
    if qtde < nivel_minimo:
        print(qtde)
```

20  
23

Para sabermos qual o produto, vamos precisar saber qual o índice desse valor e assim aplicarmos esse índice na lista produtos para termos o produto.

Vamos usar o ENUMERATE() visto anteriormente juntamente do i na estrutura do FOR.

Assim, conseguimos a posição i, da qtde, da lista estoque, para usarmos na lista produtos.

Vamos para um tema prático. Imagine que temos 5 fábricas distintas e assim como fizemos anteriormente gostaríamos de saber se temos produtos com níveis de estoque inferior ao nível mínimo.

No exemplo que fizemos anteriormente só usamos 1 For pois só havia uma fábrica. Agora precisamos ter um For que percorre todos os itens de estoque e outro que percorre cada uma das fábricas.

Vamos dar uma olhada no exemplo. Comecaremos percorrendo todo os as listas de estoque das fábricas.

```
estoque = [
    [294, 125, 269, 208, 783, 852, 259, 371, 47, 102, 386, 87, 685, 686, 697, 941, 163, 631, 7, 714, 218, 670, 453],
    [648, 816, 310, 555, 992, 643, 226, 319, 501, 23, 239, 42, 372, 441, 126, 645, 927, 911, 761, 445, 974, 2, 549],
    [832, 683, 784, 449, 977, 705, 198, 937, 729, 327, 339, 10, 975, 310, 95, 689, 137, 795, 211, 538, 933, 751, 522],
    [837, 168, 570, 397, 53, 297, 966, 714, 72, 737, 259, 629, 625, 469, 922, 305, 782, 243, 841, 848, 372, 621, 362],
    [429, 242, 53, 985, 406, 186, 198, 50, 501, 870, 781, 632, 781, 105, 644, 509, 401, 88, 961, 765, 422, 340, 654],
]
```

```
fabricas = ['Lira Manufacturing', 'Fábrica Hashtag', 'Python Manufaturas', 'Produções e Cia', 'Manufatura e Cia']
nivel_minimo = 50
```

```
for lista in estoque:
    print(lista)
```

```
[294, 125, 269, 208, 783, 852, 259, 371, 47, 102, 386, 87, 685, 686, 697, 941, 163, 631, 7, 714, 218, 670, 453]
[648, 816, 310, 555, 992, 643, 226, 319, 501, 23, 239, 42, 372, 441, 126, 645, 927, 911, 761, 445, 974, 2, 549]
[832, 683, 784, 449, 977, 705, 198, 937, 729, 327, 339, 10, 975, 310, 95, 689, 137, 795, 211, 538, 933, 751, 522]
[837, 168, 570, 397, 53, 297, 966, 714, 72, 737, 259, 629, 625, 469, 922, 305, 782, 243, 841, 848, 372, 621, 362]
[429, 242, 53, 985, 406, 186, 198, 50, 501, 870, 781, 632, 781, 105, 644, 509, 401, 88, 961, 765, 422, 340, 654]
```

Beleza! Conseguimos “printar” todas as listas da minha lista ESTOQUE. Agora precisamos criar um teste que identifique os níveis de estoque < que o nível\_mínimo. Para isso vamos criar um FOR que percorrerá cada uma dessas listas combiná-lo com o IF para a verificação da condição.

```
estoque = [
    [294, 125, 269, 208, 783, 852, 259, 371, 47, 102, 386, 87, 685, 686, 697, 941, 163, 631, 7, 714, 218, 670, 453],
    [648, 816, 310, 555, 992, 643, 226, 319, 501, 23, 239, 42, 372, 441, 126, 645, 927, 911, 761, 445, 974, 2, 549],
    [832, 683, 784, 449, 977, 705, 198, 937, 729, 327, 339, 10, 975, 310, 95, 689, 137, 795, 211, 538, 933, 751, 522],
    [837, 168, 570, 397, 53, 297, 966, 714, 72, 737, 259, 62, 625, 469, 922, 305, 782, 243, 841, 848, 372, 621, 362],
    [429, 242, 53, 985, 406, 186, 198, 50, 501, 870, 781, 632, 781, 105, 644, 509, 401, 88, 961, 765, 422, 340, 654]
]

fabricas = ['Lira Manufacturing', 'Fábrica Hashtag', 'Python Manufaturas', 'Produções e Cia', 'Manufatura e Cia']
nivel_minimo = 50

for lista in estoque:
    for qtde in lista:
        if qtde < nivel_minimo:
            print(qtde)
```

47  
7  
23  
42  
2  
10



## ATENÇÃO !

Apesar do print ter colocado cada QTDE em uma linha, perceba que algumas delas pertencem a uma mesma lista.

Agora que conseguimos identificar todas as quantidades que são menores que o nível\_mínimo, precisamos informar qual a fábrica que ela pertence. Sabemos que a ordem das listas está relacionada com a ordem das fábricas. Ou seja, a primeira lista da lista ESTOQUE se refere a fábrica 'Lira Manufacturing'. Usando uma outra variável i vamos conseguir essa posição e usá-la na lista FABRICAS(fabricas[i])

```
estoque = [
    [294, 125, 269, 208, 783, 852, 259, 371, 47, 102, 386, 87, 685, 686, 697, 941, 163, 631, 7, 714, 218, 670, 453],
    [648, 816, 310, 555, 992, 643, 226, 319, 501, 23, 239, 42, 372, 441, 126, 645, 927, 911, 761, 445, 974, 2, 549],
    [832, 683, 784, 449, 977, 705, 198, 937, 729, 327, 339, 10, 975, 310, 95, 689, 137, 795, 211, 538, 933, 751, 522],
    [837, 168, 570, 397, 53, 297, 966, 714, 72, 737, 259, 629, 625, 469, 922, 305, 782, 243, 841, 848, 372, 621, 362],
    [429, 242, 53, 985, 406, 186, 198, 50, 501, 870, 781, 632, 781, 105, 644, 509, 401, 88, 961, 765, 422, 340, 654],
]
fabricas = ['Lira Manufacturing', 'Fábrica Hashtag', 'Python Manufaturas', 'Produções e Cia', 'Manufatura e Cia']
nivel_minimo = 50

for i, lista in enumerate(estoque):
    #se dentro daquela lista tem alguém abaixo do nível minimo
    for qtde in lista:
        if qtde < nivel_minimo:
            print(fabricas[i])
```

```
Lira Manufacturing
Lira Manufacturing
Fábrica Hashtag
Fábrica Hashtag
Fábrica Hashtag
Python Manufaturas
```

Por enquanto, nossas listas sempre foram pequenas. Com poucos itens, mas imagine que você tenha uma lista de 5000 itens. Rodar o for demandará mais do seu pc. As vezes não precisamos percorrer toda a lista para acharmos o que procuramos. Muitas vezes o que queremos saber é se pelo menos 1 dos 5000 itens atende uma condição. Se ele for atendido já é suficiente e o for poderia ser pausado.

Para esses casos temos o BREAK. Ele permite pausar um FOR caso a condição seja atendida. Isso transforma o nosso código mais eficiente! Ao invés de percorrer 5000 linhas ele irá percorrer apenas o suficiente para atender a condição.

```
for item in lista:
```

```
    if condição:
```

```
        Ação se condição é verdadeira
```

```
    BREAK #Pausa o FOR e pula todas as demais interações
```

Indentações indicam que essas linhas pertence ao IF

Indentações indicam que essas linhas pertencem ao FOR

BREAK Pausará o for. Ou seja, se sua lista possui por exemplo 10 itens e a condição foi atendida no item 2, o break não permitirá que o for passe pelos itens 3 ao 10.

Vamos para um exemplo simples. Imagine uma chamada virtual. Temos uma lista de pessoas e queremos saber se uma pessoa específica está presente ou não.

No exemplo abaixo é exatamente o que acontece. Perguntamos: "Alon está presente?" O primeiro diz, "Não, eu sou João!", o segundo diz "Não, eu sou o Lira!", já o terceiro diz "Sim, estou aqui!". Não faz sentido perguntar ao Diego se ele é o Alon, visto que já sabemos que o Alon está lá. Ganhamos tempo!

Essa pausa para o PYTHON se chama BREAK conforme colocamos o código.

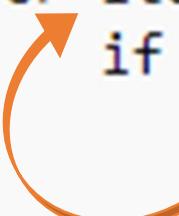
```
pessoas_presentes = ['João', 'Lira', 'Alon', 'Diego', 'Sergio', 'Marcos']
#Quero saber se uma pessoa específica está presente
chamada = 'Alon'

for pessoa in pessoas_presentes:
    if pessoa == chamada:
        print('{} está presente.'.format(chamada))
        break
    else:print('Só um print para mostrar que o for passou por essa pessoa:' + str(pessoa))
```

```
Só um print para mostrar que o for passou por essa pessoa:João
Só um print para mostrar que o for passou por essa pessoa:Lira
Alon está presente.
```

Por outro lado temos uma situação inversa que usaremos o CONTINUE. Ele ao invés de pausar e sair do FOR, ele pausa e pula diretamente para a nova interação do FOR.

```
for item in lista:  
    if condição:  
        Ação se condição é verdadeira  
        CONTINUE#Pausa o FOR e pula para a próxima interação  
        Ação que não será realizada ✗  
    else:  
        condição se falso
```



Voltando para nosso exemplo vamos inserir um CONTINUE e ações de print abaixo dessa linha de código. Perceba que apesar de termos linhas de código abaixo do CONTINUE elas são ignoradas e não “printadas”. Já a linha acima do CONTINUE é executada normalmente.

```
pessoas_presentes = ['João', 'Lira', 'Alon', 'Diego', 'Sergio', 'Marcos']
#Quero saber se uma pessoa específica está presente
chamada = 'Alon'

for pessoa in pessoas_presentes:
    if pessoa == chamada:
        print('{} está presente.'.format(chamada))
        break
    else:
        print('Só um print para mostrar que o for passou por essa pessoa:' + str(pessoa))
        continue
        print('Só para mostrar que isso não vai ser printado') ✘
        print('Isso também não') ✘
        print('Acho que deu para entender né?') ✘
```

Só um print para mostrar que o for passou por essa pessoa:João  
Só um print para mostrar que o for passou por essa pessoa:Lira  
Alon está presente.

# Módulo 10

## While e Criando um Loop infinito

Neste módulo vamos falar do WHILE. Ele, assim como o FOR, também é uma estrutura de repetição.

A diferença é que o WHILE será executado ENQUANTO algo for verdade. Ou seja, não é definido o tamanho previamente. Por exemplo, “Executar enquanto  $i < 5$ ” ou “Executar enquanto venda  $< 10000$ ”.

**while condição:**

Ação que deverá ser feita

Só será executado quando a condição não for mais atendida

Sem indentação. Só será executada após todas as interações do WHILE

Indentação indica que essa linha pertencem ao WHILE

Vamos considerar uma lista de compras. Usaremos o **WHILE** para inserir produtos na lista até um dado momento onde não há mais produtos a serem inseridos. Ou seja, podemos colocar 1 produto na lista ou 200. O programa só será interrompido quando ordenarmos para que isso seja feito.

```
venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')
vendas = []
#crie aqui o programa

while venda != '':
    vendas.append(venda)
    venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')
```

```
print('Registro Finalizado. As vendas cadastradas foram: {}'.format(vendas))
```

```
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia
```

Podemos inserir produtos eternamente nesta lista. Enquanto a condição != de '' for TRUE(VERDADE) sempre aparecerá um inputbox para que o usuário possa inserir o próximo produto.

Vamos ver no próximo slide o que acontece quando o usuário preenche ''(VAZIO) na caixa de entrada.

```
venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')
vendas = []
#crie aqui o programa

while venda != '':
    vendas.append(venda)
    venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')
```

```
print('Registro Finalizado. As vendas cadastradas foram: {}'.format(vendas))
```

```
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vaziaarroz
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vaziatatata
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vaziafeijão
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vaziamacarrão
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vaziacenoura
```

```
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia
```

Perceba que ao usarmos '' no na caixa de entrada recebemos a informação ‘Registro Finalizado’, assim como a lista de itens inseridos anteriormente.

```
venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')
vendas = []
#crie aqui o programa

while venda != '':
    vendas.append(venda)
    venda = input('Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia')
```

```
print('Registro Finalizado. As vendas cadastradas foram: {}'.format(vendas))
```

Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia  
arroz  
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia  
batata  
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia  
feijão  
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia  
macarrão  
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia  
cenoura  
Registre um produto. Para cancelar o registro de um novo produto, basta apertar enter com a caixa vazia  
Registre Finalizado. As vendas cadastradas foram: ['arroz', 'batata', 'feijão', 'macarrão', 'cenoura']

Se você se lembra bem falamos que o While rodará eternamente até a condição ser atendida.

E se ela não for? O que acontece?

Entramos no que chamamos de LOOP INFINITO.

Qual o problema deste loop. Como ele rodará eternamente, 2 cenários são possíveis:

- 1) Seu PC irá travar;
- 2) Seu código não chegará ao fim e você não poderá continuar seu trabalho.

Vamos entender primeiro o que é e depois como resolver. Vamos para o exemplo.

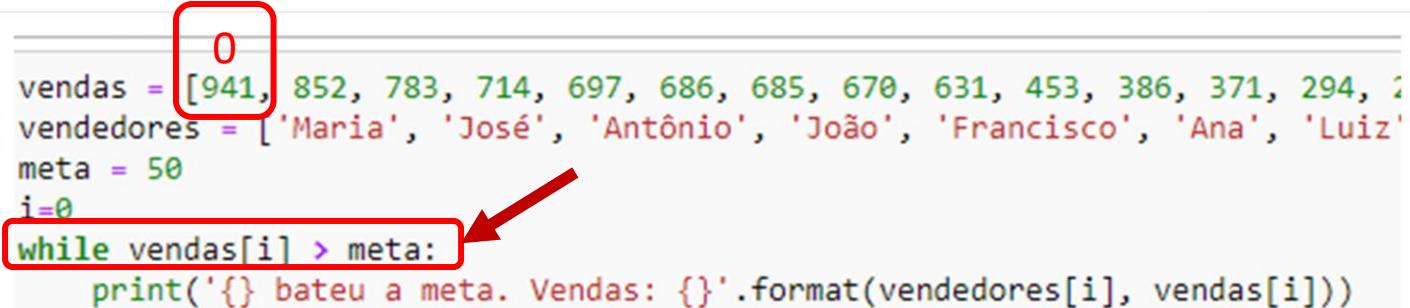
```
vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 269, 259, 218, 208, 163, 125, 102, 87, 47, 7]
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'Carlos', 'Manoel', 'Pedro', 'Francisca',
meta = 50
i=0
while vendas[i] > meta:
    print('{} bateu a meta. Vendas: {}'.format(vendedores[i], vendas[i]))
```

Olhando a condição do while temos que nosso loop será pausado quando a venda for < que a meta.

Ok! Parece razoável... No entanto, quando olhamos o código dentro da indentação não temos nada que aumente o valor de i.

Logo, i sempre será 0 (ZERO)! Isso fará com que vendas sempre seja 941 e consequentemente sempre maior que a meta que é 50.

Estamos em um LOOP INFINITO...



```
vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 1]
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz']
meta = 50
i=0
while vendas[i] > meta:
    print('{} bateu a meta. Vendas: {}'.format(vendedores[i], vendas[i]))
```

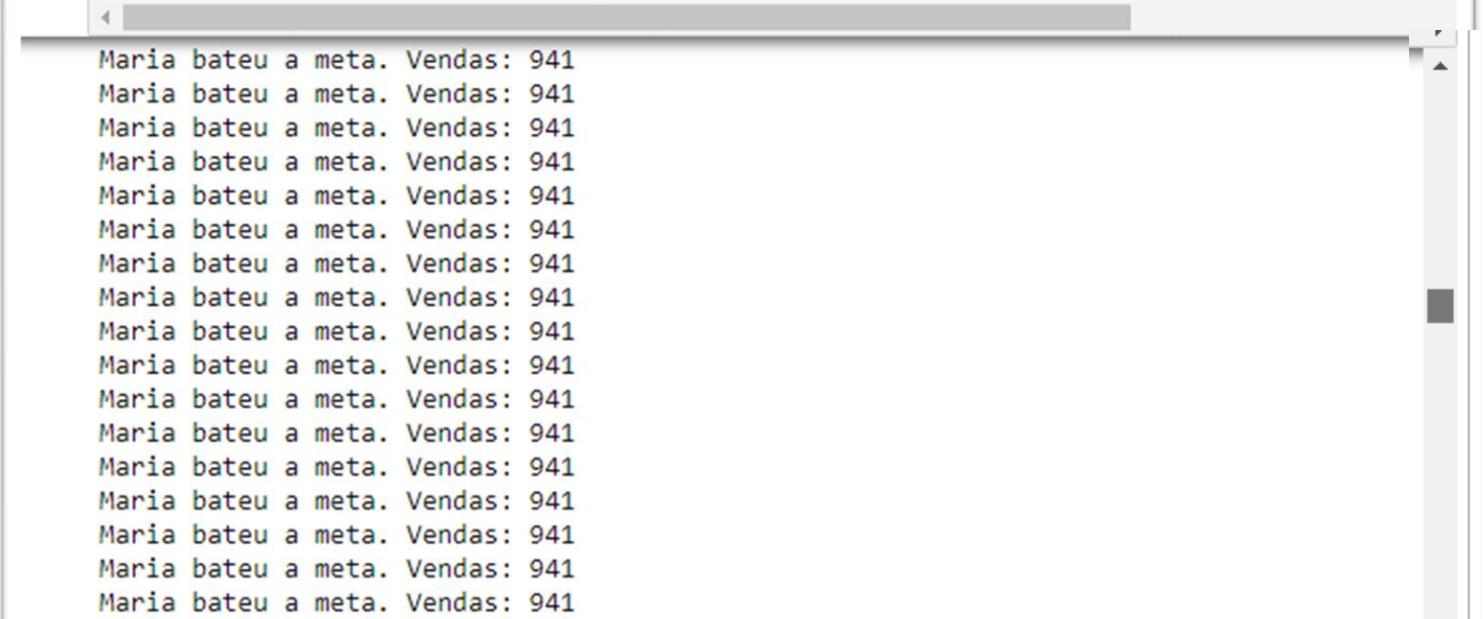
Veja ao lado o que acontece ao rodar o programa.

Perceba que temos uma repetição da linha "Maria bateu a meta. Vendas:941".

Outro ponto é que temos o asterisco ao lado da célula sinalizando que o Jupyter continua a processar a informação.



```
[ * ]: vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 453, 386, 371, 294, 269, 259, 218, vendedores = ['Maria', 'José', 'Antônio', 'João', 'Francisco', 'Ana', 'Luiz', 'Paulo', 'meta = 50 i=0 while vendas[i] > meta: print('{} bateu a meta. Vendas: {}'.format(vendedores[i], vendas[i]))
```



```
Maria bateu a meta. Vendas: 941
```

Como Solucionar? Quando entramos em um Looping Infinito não devemos nos desesperar. Isso irá acontecer mais vezes do que você gostaria infelizmente 😞 Faz parte.

Existem 2 formas para resolver o problema:

- 1) Pausar (BOTÃO DE STOP) o Jupyter Notebooks e corrigir o erro

Indica que o Kernel está sendo pausado

```
vendas = [941, 852, 783, 714, 697, 686, 685, 670, 631, 163, 125, 102, 87, 47, 7  
vendedores = ['Maria', 'José', 'Antônio', 'João', 'Frans', 'Manoel', 'Pedro', 'Fr  
meta = 50  
i=0  
while vendas[i] > meta:  
    print('{} bateu a meta. Vendas: {}'.format(vendedores[i], vend  
    i+=1
```

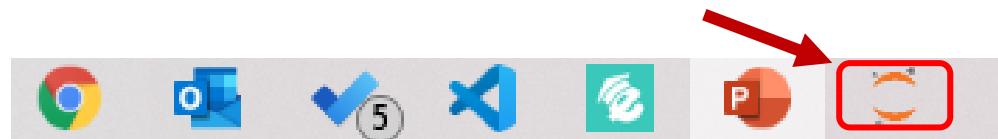
Maria bateu a meta. Vendas: 941  
Maria bateu a meta. Vendas: 941

Como Solucionar? Quando entramos em um Looping Infinito não devemos nos desesperar. Isso irá acontecer mais vezes do que você gostaria infelizmente 😞 Faz parte.

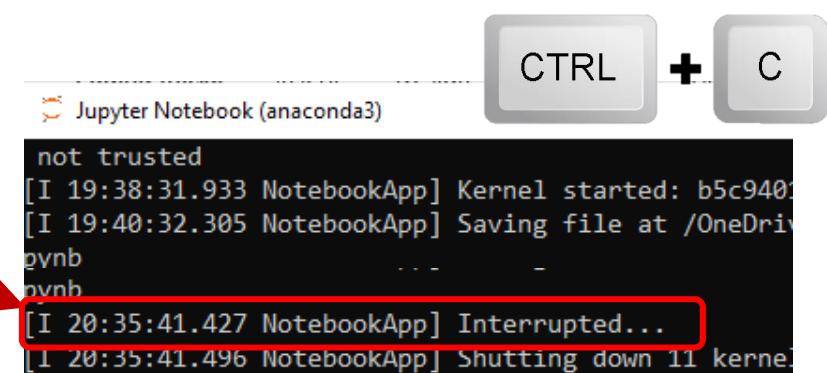
Existem 2 formas para resolver o problema:

- 1) Pausar (BOTÃO DE STOP) o Jupyter Notebooks e corrigir o erro
- 2) Pausar o Jupyter via Prompt de comando

1 Abra o Prompt de comando do Jupyter



2 No Prompt de comando use CTRL + C e retorno para seu navegador



## Módulo 11

# Tuplas - Uma "lista" muito útil e imutável

```
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
print(vendas)

nome=vendas[0]
data_contratacao=vendas[1]
data_nascimento = vendas[2]
salario = vendas[3]
cargo = vendas[4]

('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
```

↑  
Item não numérico

A estrutura das Tuplas são bem parecidas com listas. A grande diferença entra as duas é que as Tuplas são imutáveis.

Outra diferença está na estrutura, enquanto as listas usam [] as tuplas usam ().

É bem mais comum usarmos tuplas quando temos dados heterogêneos. Nas listas, usávamos listas apenas numéricas, ou apenas não numéricas. Nas tuplas vai ser comum vermos dados numéricos e não numéricos juntos.

```
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
print(vendas)

nome=vendas[0]
data_contratacao=vendas[1]
data_nascimento = vendas[2]
salario = vendas[3]
cargo = vendas[4]

#tentando alterar a posição [0]
vendas[0]='João'

('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-3-8bdb98cdb68a> in <module>
      9
     10 #tentando alterar a posição [0]
--> 11 vendas[0]='João'

TypeError: 'tuple' object does not support item assignment
```



Como falamos anteriormente, as tuplas são imutáveis.

Veja o que ocorre ao tentarmos mudar o nome 'Lira' para 'João' na tupla.

Conforme indicado temos um erro que indica que uma tupla não pode ter seus valores alterados.

```
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')  
nome, data_contratacao, data_nascimento, salario, cargo = vendas
```

```
print(nome)  
print(data_contratacao)  
print(data_nascimento)  
print(salario)  
print(cargo)
```

```
Lira  
25/08/2020  
15/02/1994  
2000  
Estagiário
```



No exemplo anterior fizemos a atribuição das variáveis através da sua posição. Assim como fazíamos com as listas.

Uma forma mais simples e usual de se fazer esta atribuição é através do **unpacking**.

O que isso significa? Em uma única linha de código iremos atribuir os valores das tuplas para cada uma das variáveis. Assim como demonstramos no exemplo ao lado.

Perceba que a ordem da atribuição corresponde a sequência da tupla.

```
vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
nome, data_contratacao, data_nascimento, cargo = vendas

-----
ValueError                                              Traceback (most recent call last)
<ipython-input-4-30772ab2438b> in <module>
      1 vendas = ('Lira', '25/08/2020', '15/02/1994', 2000, 'Estagiário')
----> 2 nome, data_contratacao, data_nascimento, cargo = vendas
      3
ValueError: too many values to unpack (expected 4)
```

Vamos ver o que acontece quando esquecemos alguma das variáveis e tentamos fazer o *unpacking* da tupla vendas.

No nosso exemplo, estamos atribuindo uma tupla com 5 itens a 4 variáveis. Esse descasamento gera um erro no Python.

Nesse caso, como o número de itens da tupla é MAIOR que o número de variáveis, temos o erro que são muito valores para *unpack*.

Se estivéssemos em uma situação onde o número de variáveis é maior que o número de itens da tupla, seria informado que não há itens o suficiente para *unpack*.

```
vendas = [1000, 2000, 300, 300, 150]
funcionarios = ['João', 'Lira', 'Ana', 'Maria', 'Paula']

for item in enumerate(vendas):
    print(item)

(0, 1000)
(1, 2000)
(2, 300)
(3, 300)
(4, 150)
```

Você se lembra do ENUMERATE()? Se lembra que tínhamos como resultados os itens entre PARENTESES?

Pois é, eram TUPLAS que agora estamos aprendendo como usar.

```
vendas = [1000, 2000, 300, 300, 150]
funcionarios = ['João', 'Lira', 'Ana', 'Maria', 'Paula']

for item in enumerate(vendas):
    i,vendas = item
    print(i)
    print(vendas)
```

```
0
1000
1
2000
2
300
3
300
4
150
```



Bem, se sabemos que item é uma TUPLA podemos fazer o *unpacking* desta tupla da mesma forma que fizemos anteriormente conforme indicado na figura.

Agora, ao invés de uma tupla, “printamos” apenas os valores “separados”

```
vendas = [1000, 2000, 300, 300, 150]
funcionarios = ['João', 'Lira', 'Ana', 'Maria', 'Paula']

for i,venda in enumerate(vendas):
    print('O funcionário {} vendeu {}'.format(funcionarios[i],venda))
```

```
O funcionário João vendeu 1000.
O funcionário Lira vendeu 2000.
O funcionário Ana vendeu 300.
O funcionário Maria vendeu 300.
O funcionário Paula vendeu 150.
```



## ATENÇÃO !

A estrutura do unpacking é comumente esta utilizada no nosso exemplo.

- venda in vendas
- produto in produtos
- funcionario in funcionários

Mas, não parece ser muito eficiente criarmos uma variável item que será atribuída por i e vendas.

Vamos melhorar esse código...

Veja o exemplo:

Substituimos a variável ITEM diretamente pelo *unpacking*. Aproveitamos também para melhorar nossa saída usando .FORMAT ao invés de apenas o print com valores avulsos.

Perceba como em apenas 2 linhas de código conseguimos uma informação bem fácil de ser compreendida.

O *unpacking* tem exatamente essa função, de forma simples coletar informações de TUPLAS e LISTAS com o auxílio do FOR.

```
vendas = [  
    ('20/08/2020', 'iphone x', 'azul', '128gb', 350, 4000),  
    ('20/08/2020', 'iphone x', 'prata', '128gb', 1500, 4000),  
    ('20/08/2020', 'ipad', 'prata', '256gb', 127, 6000),  
    ('20/08/2020', 'ipad', 'prata', '128gb', 981, 5000),  
    ('21/08/2020', 'iphone x', 'azul', '128gb', 397, 4000),  
    ('21/08/2020', 'iphone x', 'prata', '128gb', 1017, 4000),  
    ('21/08/2020', 'ipad', 'prata', '256gb', 50, 6000),  
    ('21/08/2020', 'ipad', 'prata', '128gb', 4000, 5000),  
]
```

Vamos agora para um desafio. Imagine que você está no seu trabalho e receba esta informação ao lado com a venda de vários produtos em datas distintas.

Seu chefe, te pergunta:

- 1) qual foi o faturamento do iphone no dia 20/08/2020?
- 2) Qual o produto mais vendido (em unidades) no dia 21/08/2020?

```
vendas = [  
    ('20/08/2020', 'iphone x', 'azul', '128gb', 350, 4000),  
    ('20/08/2020', 'iphone x', 'prata', '128gb', 1500, 4000),  
    ('20/08/2020', 'ipad', 'prata', '256gb', 127, 6000),  
    ('20/08/2020', 'ipad', 'prata', '128gb', 981, 5000),  
    ('21/08/2020', 'iphone x', 'azul', '128gb', 397, 4000),  
    ('21/08/2020', 'iphone x', 'prata', '128gb', 1017, 4000),  
    ('21/08/2020', 'ipad', 'prata', '256gb', 50, 6000),  
    ('21/08/2020', 'ipad', 'prata', '128gb', 4000, 5000),  
]
```

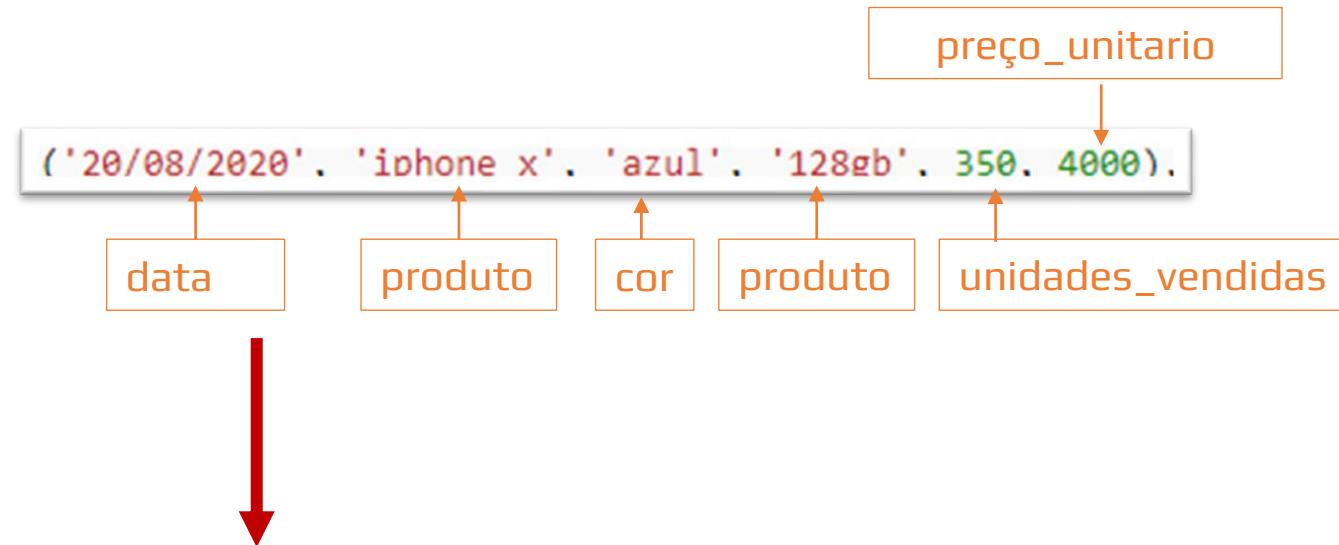
Antes do desespero, temos sempre que analisar a situação...

Como já vimos, [] simbolizam listas e () tuplas. Logo, o que temos aqui é uma lista com 8 tuplas...

Dessa forma, é um pouco complicado de trabalharmos com a informação, portanto, vamos fazer o *Unpacking* destas tuplas de forma que as informações fiquem mais simples de serem trabalhadas.

Para fazer o *unpacking* precisamos de 2 coisas:

- 1) Identificar número de itens por tupla e seus significados;
- 2) Criar um for para o *unpacking*



```
for item in vendas:  
    data, produto, cor, capacidade, unidades_vendidas, valor_unitario = item
```

Analisando os dados fornecidos da lista VENDAS podemos criar as variáveis apresentadas ao lado para cada um dos itens da tupla.

Elas serão usadas dentro do FOR para a realização do *unpacking*.

Perceba que após a execução do FOR teremos todos os itens alocados nas variáveis que acabamos de criar.

Agora já fizemos a estrutura para o unpacking. Vamos relembrar as perguntas do seu chefe...

- 1) Qual foi o faturamento do iphone no dia 20/08/2020?
- 2) Qual o produto mais vendido (em unidades) no dia 21/08/2020?

Para respondermos a pergunta 1 vamos precisar usar conhecimentos já conhecidos como o IF.  
Aqui podemos criar uma condição que caso seja atendida calcula o faturamento.

```
faturamento = 0
for item in vendas:
    data, produto, cor, capacidade, unidades, valor_unitario = item
    if produto == 'iphone x' and data == '20/08/2020':
        faturamento += unidades * valor_unitario

print('O faturamento de IPhone no dia 20/08/2020 foi de {}'.format(faturamento))
```

O faturamento de IPhone no dia 20/08/2020 foi de 7400000

Agora já fizemos a estrutura para o unpacking. Vamos relembrar as perguntas do seu chefe...

- 1) Qual foi o faturamento do iphone no dia 20/08/2020?
- 2) Qual o produto mais vendido (em unidades) no dia 21/08/2020?

Para respondermos a pergunta 2 vamos usar a mesma estrutura do *unpacking*. Usando o mesmo conceito do IF, vamos criar a condição para o dia 21/08/2020.

Além disso, vamos precisar de duas variáveis auxiliares (**produto\_mais\_vendido** e **qtde\_produto\_mais\_vendido**) para podermos armazenar o valor do produto mais vendido.

```
faturamento = 0
qtde_produto_mais_vendido = 0
for item in vendas:
    data, produto, cor, capacidade, unidades, valor_unitario = item
    if data == '21/08/2020':
        if unidades > qtde_produto_mais_vendido:
            produto_mais_vendido = produto
            qtde_produto_mais_vendido = unidades
print ('Meu produto mais vendido no dia 21/08/2020 foi o {} com {} unidades'.format(produto_mais_vendido, qtde_produto_mais_vendido))
```



```
Meu produto mais vendido no dia 21/08/2020 foi o ipad com 4000 unidades
```

Módulo 12

# Dicionários em Python

Já vimos até agora 2 formas de estruturar dados:

- 1) Listas – Usam [], são mutáveis;
- 2) Tuplas = Usam (), não são mutáveis;

Agora vamos conhecer uma das estruturas mais potentes. **Os dicionários.**

A estrutura desses dicionários são baseados em **chaves(key)** e **valores(value)**.

Aqui, ao invés de usarmos a posição para acessar uma informação, vamos usar a **CHAVE**.

Outra característica que os dicionários não possuem uma ordem padrão (nas versões mais antigas do Python)! Por isso não é recomendado pensarmos em posição ao usarmos dicionários.

```
Listas são assim : lista = [item1,item2,item3]
```

```
Tuplas são assim : tupla = (item1,item2,item3)
```

```
Dicionários são assim : dicionário = {chave1:'valor1', chave2: 'valor2', chave3: 'valor3'}
```



Vamos entender melhor a estrutura e o que são as chaves e valores que falamos anteriormente.

A estrutura sempre será **chave : valor**

No exemplo ao lado temos uma lista de produtos mais vendidos por categoria.

- Todas as categorias (**sublinhadas de amarelo**) são Chaves (*Keys*);
- Todos os produtos são (**sublinhados de verde**) são Valores(*values*);

O Python sempre buscará um valor dado uma chave.Ex:

Dicionário produtos mais vendidos, chave livros?

Resposta: o alquimista

```
tecnologia:iphone
refrigeracao:ar consul 12000 btu
livros:o alquimista
eletrodoméstico:geladeira
lazer:prancha surf
```

tecnologia:iphone  
refrigeracao:ar consul 12000 btu  
livros:o alquimista  
eletrodoméstico:geladeira  
lazer:prancha surf



Vamos ver o mesmo exemplo mas agora no Python!

Usamos a mesma estrutura do slide anterior para descrever nos itens do dicionário.

Atenção, que como se tratam de strings precisamos usar as aspas!

Criamos uma variável livro para receber o VALOR do dicionário **mais\_vendidos**.

Como a CHAVE fornecida foi ‘livros’ o valor atribuído é ‘o alquimista’.

```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'ge  
livro = mais_vendidos['livros']  
print(livro)  
<--  
o alquimista
```

Como vimos no exemplo anterior podemos buscar um VALOR usando **NOMEDICIONARIO['CHAVE']**.

Além dessa forma também temos um método chamado **.GET()**. Aqui vamos usar a mesma premissa anterior, só mudando o formato do código.

```
mais_vendidos = {'tecnologia': 'iphone', 'refrigeracao': 'ar consul 12000 btu', 'livros': 'o alquimista', 'eletrodoméstico': 'ge  
vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000,  
print(mais_vendidos['livros'])  
  
print(mais_vendidos.get('livros'))  
o alquimista  
o alquimista
```



```
#chave:  
print(mais_vendidos['geladeiras'])
```

```
-----  
KeyError Traceback (most recent call  
<ipython-input-19-dd3cacc920ad> in <module>  
----> 1 print(mais_vendidos['geladeiras'])
```

```
KeyError: 'geladeiras'
```

```
#Método .GET  
print(mais_vendidos.get('geladeiras'))
```

```
None
```



O que aconteceria se procurássemos por uma chave que não existe?

Usando apenas chave ao procurar uma chave inexistente, o Python retorna um erro

Já o método .GET retornará NONE como resultado. Ou seja, não existe nenhuma CHAVE 'geladeiras' no dicionário.



**ATENÇÃO!**

O fato do método GET não retornar erro não significa que seja melhor para todas as situações!

Assim como as listas, também podemos atualizar dicionários, adicionando, modificando ou removendo itens.

No exemplo ao lado, temos um dicionário vazio que vamos inserir dados de duas formas distintas.

```
vendas_vendedores = {}  
vendas_vendedores['João'] = 100  
print(vendas_vendedores)  
  
vendas_vendedores.update({'Lira': 50, 'Alon': 80})  
print(vendas_vendedores)  
  
{'João': 100}  
{'João': 100, 'Lira': 50, 'Alon': 80}
```

E quando inserimos uma CHAVE que já existe?

Perceba que ao usarmos as mesma estrutura anterior e atualizamos os valores das CHAVES anteriores.

Quando adicionamos novos valores em CHAVES EXISTENTES, SUBSTITUIMOS os valores anteriores. Não há duplicidade nas chaves.

```
vendas_vendedores = {}

vendas_vendedores['João'] = 100
print(vendas_vendedores)

vendas_vendedores.update({'Lira': 50, 'Alon': 80})
print(vendas_vendedores)

{'João': 100}
{'João': 100, 'Lira': 50, 'Alon': 80}

vendas_vendedores['João'] = 70
print(vendas_vendedores)

vendas_vendedores.update({'Lira': 67, 'Alon': 12})
print(vendas_vendedores)

{'João': 70, 'Lira': 50, 'Alon': 80}
{'João': 70, 'Lira': 67, 'Alon': 12}
```

Para remover um conjunto chave/valor de um dicionário, nós temos 3 formas distintas.

- **DEL()** - Exclui um conjunto chave/valor de um dicionário;
- **.POP()** - Apaga um conjunto chave/valor de um dicionário, mas armazena essa informação em outra variável;
- **.CLEAR()** - Apaga todos os dados de um dicionário.

```
lucro_1tri = {'janeiro': 100000, 'fevereiro': 120000, 'março': 90000}
lucro_2tri = {'abril': 88000, 'maio': 89000, 'junho': 120000}

#removendo o mês de junho do dicionário e armazenando em uma variável Lucro_jun:
del lucro_2tri['junho']
print(lucro_2tri)
```

```
{'abril': 88000, 'maio': 89000}
```

```
#removendo o conjunto chave valor de junho e atribuindo a variavel Lucro_jun:
lucro_jun=lucro_2tri.pop('junho')
print(lucro_2tri)
print(lucro_jun)

{'abril': 88000, 'maio': 89000}
120000
```

```
#removendo todos os dados de um dicionário:
lucro_2tri.clear()
print(lucro_2tri)
```

```
{}
```

```
vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 5000, 'ipad': 8000, 'tv philco': 3500, 'notebook hp': 20000, 'notebook dell': 18000, 'notebook asus': 15000}

#demonstrando o for
for chave in vendas_tecnologia:
    print(chave)
```



```
iphone
samsung galaxy
tv samsung
ps5
tablet
ipad
tv philco
notebook hp
notebook dell
notebook asus
```

Vamos entender como associar o FOR para fazer um *unpacking* dos conjuntos chave/valor.

Usando o mesmo princípio que vimos anteriormente vamos criar uma variável chave que irá percorrer todo o dicionário vendas\_tecnologia.

Perceba que ao fazermos print(chave) não temos o conjunto chave/valor, apenas a chave.

```
vendas_tecnologia = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000, 'philco': 2500, 'notebook hp': 1000, 'notebook dell': 17000, 'notebook asus': 2450}

#demonstrando o for
for chave in vendas_tecnologia:
    print('O produto{} vendeu {} unidades.'.format(chave,vendas_tecnologia[chave]))
```

O produtoiphone vendeu 15000 unidades.  
O produtosamsung galaxy vendeu 12000 unidades.  
O produtotv samsung vendeu 10000 unidades.  
O produtops5 vendeu 14300 unidades.  
O produztablet vendeu 1720 unidades.  
O produzipad vendeu 1000 unidades.  
O produztv philco vendeu 2500 unidades.  
O produznotebook hp vendeu 1000 unidades.  
O produznotebook dell vendeu 17000 unidades.  
O produznotebook asus vendeu 2450 unidades.



Sabendo que conseguimos acessar o valor com uma chave específica, podemos usar a estrutura **vendas\_tecnologia[chave]** a fim de termos o resultado esperado.

```
vendas_tecnologia = {'notebook asus': 2450, 'iphone': 15000, 'samsung galaxy':  
    for item in vendas_tecnologia.items():  
        print(item)  
  
('notebook asus', 2450)  
('iphone', 15000)  
('samsung galaxy', 12000)  
('tv samsung', 10000)  
('ps5', 14300)  
('tablet', 1720)  
('notebook dell', 17000)  
('ipad', 1000)  
('tv philco', 2500)  
('notebook hp', 1000)
```

Assim como as listas e tuplas os dicionários possuem alguns métodos que nos ajudam a tratar os dados de forma simples:

Neste primeiro exemplo iremos usar o método .items().

Aqui ao invés de usarmos no for chave, vamos usar item.

Perceba que o que temos de retornos não é apenas as chaves mas todos as tuplas do dicionário com seus resultados chave/valor.

```
vendas_tecnologia = {'notebook asus': 2450, 'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'notebook dell': 17000, 'ipad': 1000, 'tv philco': 2500, 'notebook hp': 1000}

for produto, vendas in vendas_tecnologia.items():
    print('{}: {} de unidades.'.format(produto, vendas))
```

notebook asus: 2450 de unidades.  
iphone: 15000 de unidades.  
samsung galaxy: 12000 de unidades.  
tv samsung: 10000 de unidades.  
ps5: 14300 de unidades.  
tablet: 1720 de unidades.  
notebook dell: 17000 de unidades.  
ipad: 1000 de unidades.  
tv philco: 2500 de unidades.  
notebook hp: 1000 de unidades.

Podemos aproveitar as tuplas para fazermos o unpacking como já vimos no módulo 11.

A diferença aqui é que não usamos item no for e sim 2 variáveis que descrevem melhor o que representam o conjunto chave/valor

Mais 2 métodos de dicionários usualmente utilizados são:

- keys()
- values()

```
vendas_tecnologia = {'notebook asus': 2450, 'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'notebook dell': 17000, 'ipad': 1000, 'tv philco': 2500, 'notebook hp': 10000}

print(vendas_tecnologia.keys())
print(vendas_tecnologia.values())

dict_keys(['notebook asus', 'iphone', 'samsung galaxy', 'tv samsung', 'ps5', 'tablet', 'notebook dell', 'ipad', 'tv philco', 'notebook hp'])
dict_values([2450, 15000, 12000, 10000, 14300, 1720, 17000, 1000, 2500, 10000])
```



**ATENÇÃO !**

O `dict_keys` ou o `dict_values` são listas um pouco das que vimos até agora. Elas são listas associadas diretamente ao dicionário. Portanto, qualquer modificação no conjunto CHAVE/VALOR irá alterar a lista `dict_keys` ou `dict_values`.

```
produtos = ['iphone', 'samsung galaxy', 'tv samsung', 'ps5', 'tablet', 'ipad']
vendas = [15000, 12000, 10000, 14300, 1720, 1000]

lista_tuplas = zip(produtos, vendas)
print(lista_tuplas)

<zip object at 0x0000023541920780>
```

Função ZIP criará um objeto por isso ao “printarmos” lista\_tuplas não temos a lista e sim o código desse objeto

Até agora sempre trabalhamos com 2 listas que eram complementares entre sim porém não possuíam vínculo de estrutura.

O que vamos ver agora é como transformar listas em dicionários. Isso é bastante útil, pois como sabemos o dicionário faz um link entre informações de chave e valor.

Nesse caso, as chaves, são os itens da lista produtos e os valores os itens da lista vendas.

No entanto, para fazermos essa conversão precisamos transformar essas duas listas em uma lista de tuplas usando a função ZIP() conforme apresentado no print ao lado.

```
produtos = ['iphone', 'samsung galaxy', 'tv samsung', 'ps5', 'tablet', 'ipad']
vendas = [15000, 12000, 10000, 14300, 1720, 1000]

lista_tuplas = zip(produtos, vendas)
print(lista_tuplas)
for item in lista_tuplas:
    print(item)

<zip object at 0x00000235419205C0>
('iphone', 15000)
('samsung galaxy', 12000)
('tv samsung', 10000)
('ps5', 14300)
('tablet', 1720)
('ipad', 1000)
```

Lista\_tuplas pós unpacking

Como vimos, printar a `lista_tuplas` não nos trouxe o resultado esperado.

Para podermos acessar os itens dessa lista vamos usar um conhecimento que já vimos: o *unpacking*.

Usando o `for` vamos printar todos os itens da `lista_tuplas`.

Perceba que o resultado são tuplas !

Agora, nos falta criarmos o dicionário a partir da nossa lista de tuplas.

Para isso, vamos usar a função **dict()**.

Veja que agora a lista e produtos e vendas foram agora agrupadas em **dicionário\_vendas**.

```
produtos = ['iphone', 'samsung galaxy', 'tv samsung', 'ps5', 'tablet', 'ipad']
vendas = [15000, 12000, 10000, 14300, 1720, 1000]
```

```
lista_tuplas = zip(produtos, vendas)
dicionario_vendas = dict(lista_tuplas)
print(dicionario_vendas)
```

```
{'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000, 'ps5': 14300, 'tablet': 1720, 'ipad': 1000}
```



**ATENÇÃO !**

As listas, **produtos** e **vendas** não deixaram de existir!!  
Apenas foi criado um dicionário '**dicionário\_vendas**' a partir dessas listas

# Módulo 13

# Iterables

Até agora, sempre ao tratarmos de listas, tuplas, strings, dicionários nos referimos como estrutura de dados.

No entanto, existe um conceito (que aqui será explicado de forma mais simples) que define as características dessas estruturas.

Os **iterables** são estruturas que permitem iterações recorrentes com seus dados.

O que são essas iterações?

Geralmente essas iterações são códigos que percorrem os dados de forma repetitiva. Usualmente usaremos o FOR ou WHILE.

Esse conceito é importante porque várias funções do python usam isso para explicar como as coisas funcionam.

É importante que ao ler o termo "iterable" você entenda o que estão falando:

"é tipo uma lista de coisas que eu posso percorrer e fazer alguma ação com cada uma das coisas dentro dessa lista"

Vamos ver alguns Iterables já conhecidos:

## LISTAS

```
produtos = ['iphone', 'samsung galaxy', 'tv samsung', 'ps5']

for produto in produtos:
    print(produto)
```

```
iphone
samsung galaxy
tv samsung
ps5
```

## TUPLAS

```
produtos = ('iphone', 'samsung galaxy', 'tv samsung', 'ps5')

for produto in produtos:
    print(produto)
```

```
iphone
samsung galaxy
tv samsung
ps5
```

## DICIONÁRIO

```
vendas_produtos = {'iphone': 15000, 'samsung galaxy': 12000, 'tv samsung': 10000}

for produto in vendas_produtos:
    #print(produto)
    print('{}: {} unidades'.format(produto, vendas_produtos[produto]))
```

```
iphone: 15000 unidades
samsung galaxy: 12000 unidades
tv samsung: 10000 unidades
```

## STRING

```
texto = 'joao lira'

for ch in texto:
    print(ch)
```

```
j
o
a
o

l
i
r
a
```

## RANGE

```
#range com inicio e fim
print(range(1, 10))

#vamos olhar no for para entender
for i in range(2, 10):
    print(i)
```

```
range(1, 10)
2
3
4
5
6
7
8
9
```

```
for i in range(5):
    print(i)

0
1
2
3
4
Iterações iniciam no ZERO vão até 4 (n-1).
Número total de itens IGUAL ao valor indicado no parênteses

for i in range (5,10):
    print(i)

5
6
7
8
9
Iterações de +1 iniciando em 5 até 9(n-1).

for i in range(5,20,2):
    print(i)

5
7
9
11
13
15
17
19
Iterações de +2 iniciando em 5 até 19(n-1).
```

5 corresponde ao Tamanho (nº de itens)

Número terminará o FOR  
(10 não incluso)

Número que o FOR iniciará

Passo: Aumento de 2 em 2

Número terminará o FOR  
(10 não incluso)

Número que o FOR iniciará

Vamos falar agora de outro iterable, o **RANGE()**. Nós já usamos ele anteriormente mas ainda não vimos a fundo como ele funciona e como pode ser utilizado.

O Range será utilizado de 3 formas distintas:

- range(tamanho);
- range(início, fim);
- range(início, fim, passo);

Ao lado damos alguns exemplos onde usamos esses três tipos distintos do uso do RANGE().

```
set_produtos = {'arroz', 'feijao', 'macarrao', 'atum', 'azeite'}
print(set_produtos)
{'arroz', 'azeite', 'macarrao', 'atum', 'feijao'}
```

Ordem diferente da escrita no código

```
set_produtos = {'arroz', 'feijao', 'macarrao', 'arroz', 'atum', 'azeite'}
print(set_produtos)
{'feijao', 'atum', 'azeite', 'macarrao', 'arroz'}
```

Apenas 1 item 'arroz'

```
cpf_clientes = ['762.196.080-97', '263.027.380-67', '827.363.930-40', '925.413.640-91', '870.565.160-33', '892.080.930-50', '462.126.030-81', '596.125.830-05', '393.462.330-10', '925.413.640-91', '762.196.080-97']

print('São {} CPFs na lista.'.format(len(cpf_clientes)))
set_cpf_clientes = set(cpf_clientes)
cpf_clientes_unicos = list(set_cpf_clientes)
print(cpf_clientes_unicos)
print('No entanto, temos apenas {} clientes diferentes na loja'.format(len(set_cpf_clientes)))
```

São 14 CPFs na lista.

[ '827.363.930-40', '990.236.770-48', '870.565.160-33', '988.305.810-11', '263.027.380-67', '892.080.930-50', '462.126.030-81', '596.125.830-05', '393.462.330-10', '925.413.640-91', '762.196.080-97' ]

No entanto, temos apenas 11 clientes diferentes na loja

Um novo iterable que não vimos até agora é o SET.

Eles parecem bem bastante com listas mas possuem algumas particularidades:

- Não possuem uma ordem específica;
- Não aceitam valores repetidos;
- Usam {} como formato, mas não possuem CHAVE como os dicionários.

Todas as duplicatas foram removidas

Módulo 14

# Criando suas funções em Python

```
#Operação rotineira: Cálculo da média
nota1 = input('Insira nota 1:')
nota2 = input('Insira nota 2:')
media = (float(nota1) + float(nota2))/2
print('A média das notas é : {}'.format(media))
```

```
Insira nota 1:7
Insira nota 2:6
A média das notas é : 6.5.
```

```
#Criando uma função calcular_media:
def calcular_media(nota1,nota2):
    nota1 = input('Insira nota 1:')
    nota2 = input('Insira nota 2:')
    media = (float(nota1) + float(nota2))/2
    return print('A média das notas é : {}'.format(media))
```

```
#Chamando a função que acabamos de criar:
```

```
calcular_media(10,5)
```

```
Insira nota 1:5
Insira nota 2:6
A média das notas é : 5.5.
```

Diversas vezes usamos funções nos nossos códigos. O Python, já nos fornece dezenas delas!

Mas, e se eu faço uma operação frequentemente? Será que poderíamos criar uma função que atenda essa atividade?

Veja o exemplo ao lado. Um simples, apenas para entendermos o conceito.

Imagine que sempre precisamos calcular a media de 2 notas. Vamos criar uma função **calcular\_media** que nos permita realizar essa operação.

Função **calcular\_media** criada para esse programa que necessita de 2 parâmetros para rodar, nota1(Ex:10) e nota2(Ex:5).

Vamos entender a estrutura no Python para a criação de uma função.

**def** indica que uma função será **definida**.

**def nome\_da\_função(argumento1, argumento2, argumentoN):**

Ação 1 a ser realizada pela função;

Ação 2 a ser realizada pela função;

Ação 3 a ser realizada pela função;

**return** 0 que a função retornará ao rodar a função

Bloco de ações que serão executadas dentro da função

Indica que o que será retornado pela função

Indentação indica que essas linhas pertencem ao **def**

```
#Operação rotineira: Cálculo da média
nota1 = input('Insira nota 1:')
nota2 = input('Insira nota 2:')
media = (float(nota1) + float(nota2))/2
print('A média das notas é : {}'.format(media))
```

```
Insira nota 1:7
Insira nota 2:6
A média das notas é : 6.5.
```

```
#Criando uma função calcular_media:
def calcular_media(nota1,nota2):
    nota1 = input('Insira nota 1:')
    nota2 = input('Insira nota 2:')
    media = (float(nota1) + float(nota2))/2
    return print('A média das notas é : {}'.format(media))
```

```
#Chamando a função que acabamos de criar:
calcular_media(10,5)
```

```
Insira nota 1:5
Insira nota 2:6
A média das notas é : 5.5.
```

Vamos voltar para nosso exemplo anterior onde calculamos a média de duas notas.

Primeiros vamos entender o que é “Retornar um valor”.

Como estamos falando em fuctions, estamos falando de criarmos um código que a partir de algum input ela processa os dados e retorna um resultado.

No nosso caso por exemplo:

Inputs-> nota1 e nota2

Resultado retornado -> Média das notas

Se olharmos o código da nossa função **calcular\_media**, veremos que ao fim temos um linha **RETURN**.

Essa linha informa o que será retornado nessa função. No nosso exemplo, um print com a média das notas.

Essa resposta geralmente são: Listas, variáveis, dicionários, etc...

```
#Operação rotineira: Cálculo da média
nota1 = input('Insira nota 1:')
nota2 = input('Insira nota 2:')
media = (float(nota1) + float(nota2))/2
print('A média das notas é : {}'.format(media))
```

```
Insira nota 1:7
Insira nota 2:6
A média das notas é : 6.5.
```

```
#Criando uma função calcular_media:
def calcular_media(nota1,nota2):
    nota1 = input('Insira nota 1:')
    nota2 = input('Insira nota 2:')
    media = (float(nota1) + float(nota2))/2
    return print('A média das notas é : {}'.format(media))
```

```
#Chamando a função que acabamos de criar:
calcular_media(10,5)
```

```
Insira nota 1:5
Insira nota 2:6
A média das notas é : 5.5.
```

Como citamos no slide anterior para que essa função funcione são necessários dois inputs:

- nota1;
- nota2.

Esses inputs são chamados de **argumentos** ou **parâmetros**.

Importante dizer que os argumentos não são obrigatórios.

Os argumentos serão sempre definidos dentro dos parênteses da nossa linha de código que se inicia com def.

Na nossa função **Calcular\_media**, já usamos mais de um parâmetro.

Mas vamos entender alguns detalhes de como trabalhar com mais de um parâmetro de uma função.

Existem duas formas distintas de informar parâmetros para funções:

- 1) Em ordem(**positional argument**);
- 2) Com o nome do argumento (**keyword argument**)

Vamos entender essa diferença melhor olhando o exemplo ao lado:

Aqui temos uma função que classifica produtos baseado na sua categoria. Em especial, as bebidas, se são alcóolicas ou não.

```
def eh_da_categoria(bebida, cod_categoria):
    bebida = bebida.upper()
    if cod_categoria in bebida:
        return True
    else:
        return False

#Usando ordem:
for produto in produtos:
    if eh_da_categoria(produto, 'BEB'):
        print('Enviar {} para setor de bebidas alcóolicas'.format(produto))
    elif eh_da_categoria(produto, 'BSA'):
        print('Enviar {} para setor de bebidas não alcóolicas'.format(produto))

#Usando palavra chave:
for produto in produtos:
    if eh_da_categoria(cod_categoria ='BEB', bebida=produto):
        print('Enviar {} para setor de bebidas alcóolicas'.format(produto))
    elif eh_da_categoria(produto, 'BSA'):
        print('Enviar {} para setor de bebidas não alcóolicas'.format(produto))

#Só podemos usar um método, o caso abaixo retornará erro!
for produto in produtos:
    if eh_da_categoria(cod_categoria ='BEB', produto):
        print('Enviar {} para setor de bebidas alcóolicas'.format(produto))
    elif eh_da_categoria(produto, 'BSA'):
        print('Enviar {} para setor de bebidas não alcóolicas'.format(produto))
```

**Argumentos na mesma ordem**

**Argumentos por palavra chave**

**ERRO!**

Vamos entender um pouco melhor o erro do slide anterior.

Quando usamos um parâmetro de keyword no início da declaração de parâmetros, devemos seguir usando Keywords para todos os outros argumentos.

Vamos entender um caso onde podemos usar tanto os argumentos de posição quanto de palavra chaves em uma mesma linha de códigos. Vamos aproveitar para aprender um novo argumento:

**sep =**

Este argumento nos ajuda a apresentar os dados separando-os conforme a informação que seja fornecida por nós.  
Vamos para o exemplo:



Já reparou que em algumas funções existem padrões já estabelecidos no Python?

Não? Vamos para um caso:

Método `.sort()`. Sempre que fazemos este método, o padrão do Python é ordenar os dados de forma crescente como na figura ao lado.

No entanto, podemos forçar o Python a não usar o padrão fornecendo a ele uma palavra-chave:

`sort(reverse=True)`

Assim, a nossa lista será ordenada de forma decrescente.

O que acontece aqui é que o padrão do Python omite sua configuração padrão:

`sort(reverse=False)`

Isso acontece pois a função `sort` foi programada dessa forma!

```
1 vendas = [100,30,70,94,15,65,35]
2 vendas.sort() ← Reverse=False omitido (Padrão)
3 print('Ordem crescente')
4 print(vendas)
5 vendas.sort(reverse=True) ← Keyword
6 print('Ordem decrescente')
7 print(vendas)
```

Ordem crescente

[15, 30, 35, 65, 70, 94, 100]

Ordem decrescente

[100, 94, 70, 65, 35, 30, 15]

O mais interessante, é que podemos fazer exatamente a mesma coisa nas funções criadas por nós!

Vamos para um exemplo que cria códigos de produtos. No entanto, temos duas possibilidades de código:

- Todo em letras maiúsculas (M);
- Todo em letras minúsculas (m).

Nosso padrão será minúsculas, mas o usuário poderá usar a palavra chave `padrão='M'` para alterar para letras maiúsculas.

Perceba que usamos o IF para definir o que acontecerá dependendo do padrão fornecido pelo usuário.

Ao lado apresentamos a função e o resultado dos casos. Perceba que no primeiro caso o usuário não fornece nenhum argumento e o padrão MINÚSCULO é atendido.

Definição do padrão

```

1 def padronizar_codigos(lista_codigos, padrao='m'):
2     for i,item in enumerate(lista_codigos):
3         item = item.replace(' ', ' ')
4         item = item.strip()
5         if padrao == 'm':
6             item = item.casefold()
7         elif padrao == 'M':
8             item = item.upper()
9         lista_codigos[i] = item
10    return lista_codigos
11
12 cod_produtos = ['ABC12', 'abc34', 'AbC37']
13 print(padronizar_codigos(cod_produtos))

```

Argumento 'm' omitido

```
['abc12', 'abc34', 'abc37']
```

```

12 cod_produtos = ['ABC12', 'abc34', 'AbC37']
13 print(padronizar_codigos(cod_produtos, padrao='M'))

```

```
['ABC12', 'ABC34', 'ABC37']
```

Argumento 'M' fornecido,  
forçando a não usar o padrão

Como vimos anteriormente o return nos retorna sempre o “resultado” da nossa função.

É possível retornar booleans, integers, floats, variáveis, etc.

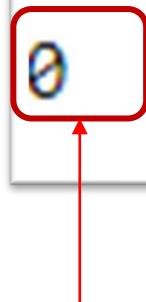
No entanto, é muito importante explicar que o return funciona como um FIM de uma função.

Ou seja, se a função chegar até a linha de código RETURN ela irá encerrar a função mesmo que se trate de um FOR como no exemplo ao lado.

Essa função não faz muito sentido de existir, mas podemos ver que apesar de termos definido que nossa função retorna o valor i com um range de 5, apenas o valor 0(PRIMEIRA ITERAÇÃO DO FOR) foi retornado.

Por que? Porque ao chegar return a nossa função chega ao FIM

```
1 def fazer_for():
2     for i in range(5):
3         return i
4
5 print(fazer_for())|
```



Ao chegarmos nessa linha de código, a função será interrompida. Ou seja, só haverá UMA iteração no FOR e não CINCO como definimos no RANGE

Retorno esperado: 0 ,1,2,3,4

Retorno obtido: 0 -> Apenas a primeira iteração do FOR

Vamos aprofundar um pouco mais no RETURN.

Nossas funções até agora, em geral, são simples e possuem retornos igualmente simples com apenas um valor por exemplo.

No entanto, é possível que o retorno necessário seja um pouco mais complexo com mais resultados.

Ao lado temos uma função ainda bem simples, mas que nos permite entender o conceito.

Dentro da nossa função **operações\_básicas** calculamos resultados como soma, diferença, multiplicação e divisão.

Uma informação importante é que o retorno é uma TUPLA. Isso nos permitirá trabalhar melhor com esses dados.

```
1 def operacoes_basicas(num1,num2):
2     soma = num1+num2
3     diferenca = num1-num2
4     mult = num1*num2
5     divisao = num1/num2
6     return (soma, diferenca, mult, divisao)
7
8 print(operacoes_basicas(10,2))
```

(12, 8, 20, 5.0)

Ao invés de apenas 1 resultado, temos 4 resultados da nossa função

Retorno de todas as variáveis calculadas dentro da função em uma TUPLA

Vamos para um caso um pouco mais complexo.

Nesse caso estamos avaliando quais vendedores bateram a meta.

Podemos perceber que nossa função retorna uma tupla com 2 valores:

- % que bateram a meta(perc\_baterammeta);
- Pessoas que bateram a meta (vendedores\_acima\_media)

Retorno da função será uma TUPLA

% de vendedores que bateram a meta  
LISTA de vendedores que bateram a meta

```

1 meta = 10000
2 vendas = {
3     'João':15000,
4     'Julia':27000,
5     'Marcus':9900,
6     'Maria':3750,
7     'Ana':10300,
8     'Alon':7870,
9 }
10
11 def calculo_meta(meta,vendas):
12     bateram_meta = []
13     for vendedor in vendas:
14         if vendas[vendedor]>=meta:
15             bateram_meta.append(vendedor)
16     perc_baterammeta = len(bateram_meta)/len(vendas)
17     return perc_baterammeta,bateram_meta
18
19 p_meta,vendedores_acima_meta = calculo_meta(meta,vendas)
20 print(p_meta)
21 print(vendedores_acima_meta)

```

0.5  
['João', 'Julia', 'Ana']

Unpacking a tupla

Não obrigatoriamente todas as funções criadas serão usadas apenas por você, por isso é importante que elas sejam organizadas e compreensíveis por outras pessoas.

Essa organização passa pelos:

**Docstrings** : Iniciados por "" permitem strings de mais de uma linha. Aqui você pode descrever o que sua função faz, argumentos que ela possui etc. Você pode criar o seu padrão, mas caso não possua um, temos um exemplo no print ao lado.

Informações podem ser acessadas usando SHIFT+TAB quando o cursor estiver na função desejada

```
1 def minha_funcao(arg1,arg2,arg3):
2     ''' O que a minha função faz.
3
4     Meus argumentos:
5     arg1(tipo do meu argumento):Comentário sobre o argumento
6     arg2(tipo do meu argumento):Comentário sobre o argumento
7     arg3(tipo do meu argumento):Comentário sobre o argumento
8
9     return
10    O retorno da minha função
11    ...
12
13    soma = arg1+arg2+arg3
14    return soma
15
16 minha_funcao()
```

Docstring

Signature: minha\_funcao(arg1, arg2, arg3)

Docstring:

O que a minha função faz.

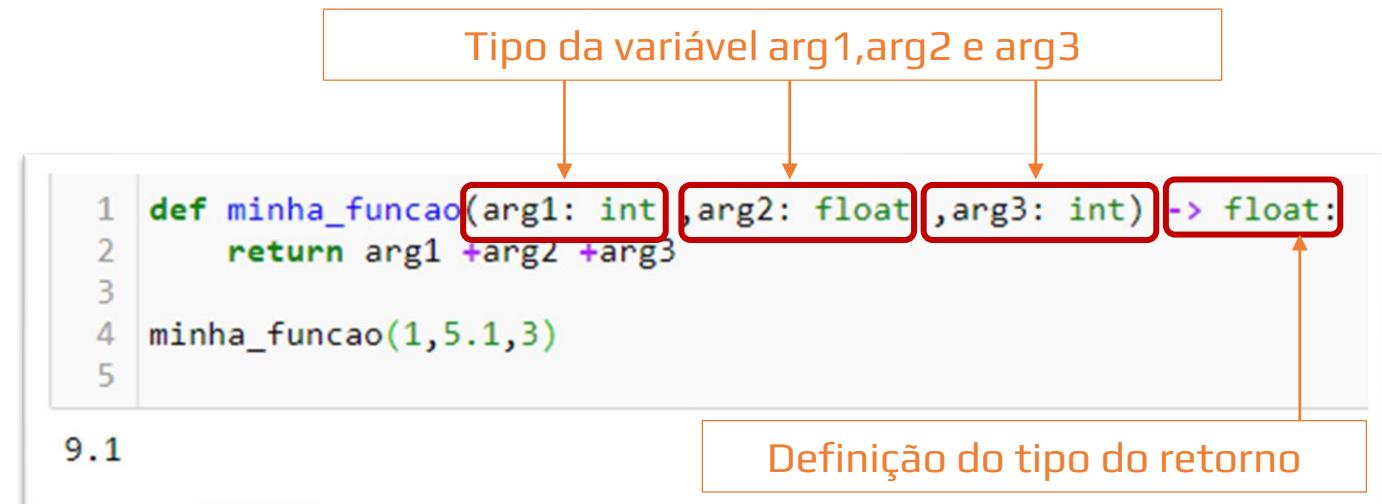
Meus argumentos:

arg1(tipo do meu argumento):Comentário sobre o argumento  
arg2(tipo do meu argumento):Comentário sobre o argumento  
arg3(tipo do meu argumento):Comentário sobre o argumento

Não obrigatoriamente todas as funções criadas serão usadas apenas por você, por isso é importante que elas sejam organizadas e compreensíveis por outras pessoas.

Essa organização passa pelos:

**Annotations** : É uma versão mais simplificada de um docstrings. As informações sobre a função são escritas dentro da própria função ao ser definida.



Até agora, sempre que nossos códigos acusavam algum erro, o que significava que seu programa será encerrado.

No entanto, pode ocorrer uma situação que o resultado ou o valor imputado pelo usuário está errado mas você não quer ele pare nesse momento, e sim que o corrija.

Para esses casos podemos usar o **TRY** e o **EXCEPT**.

**SEMPRE** que o TRY for utilizado deverá haver um EXCEPT.

```
try:  
    o que eu quero tentar fazer  
except:  
    o que vou fazer caso dê erro
```

Indentação indica que essas linhas pertencem ao try e except

Primeiro vamos entender o que são os erros na realidade.

Sabemos que são um pouco assustadores, principalmente no começo. Milhões de linhas de código que não são fáceis de entender.

Vamos pegar um exemplo para analisar e depois vamos voltar para nosso TRY e EXCEPT e como usá-lo.

```
1 print(erro)
```

-----

NameError

Nome do tipo de erro

<ipython-input-5-a751457e33f5> in <module>

----> 1 print(erro)

Traceback (most recent call last)

NameError: name 'erro' is not defined

Detalhamento do erro



Bem, agora que entendemos um pouco mais sobre a estrutura de um erro fornecido pelo Python, vamos usar o Try e Except para criar o nosso próprio erro, que nos permita informar ao usuário.

Vamos criar uma função que indica qual o tipo de e-mail utilizado pelo usuário.

Perceba que no bloco EXCEPT usamos ao invés do return o **RAISE**. Isso fará com que um erro seja exibido ao usuário.

Veremos isso melhor no próximo slide.

```
def descobrir_servidor(email):
    try:
        posicao_a = email.index('@')
    except:
        raise ValueError('Email digitado não tem @, digite novamente')
    else:
        servidor = email[posicao_a:]
        if 'gmail' in servidor:
            return 'gmail'
        elif 'hotmail' in servidor or 'outlook' in servidor or 'live' in servidor:
            return 'hotmail'
        elif 'yahoo' in servidor:
            return 'yahoo'
        elif 'uol' in servidor or 'bol' in servidor:
            return 'uol'
        else:
            return 'não determinado'
```

```
email = input('Qual o seu e-mail?')
print(descobrir_servidor(email))
```

Docstring

“Chamando” a função descobrir\_servidor

Caso o try seja atendido,  
entraremos no bloco else

Bem, agora que entendemos um pouco mais sobre a estrutura de um erro fornecido pelo Python, vamos usar o Try e Except para criar o nosso próprio erro, que nos permita informar ao usuário.

Vamos criar uma função que indica qual o tipo de e-mail utilizado pelo usuário.

Perceba que no bloco EXCEPT usamos ao invés do return o **RAISE**. Isso fará com que um erro seja exibido ao usuário.

Veremos isso melhor no próximo slide.

```
def descobrir_servidor(email):
    try:
        posicao_a = email.index('@')
    except:
        raise ValueError('Email digitado não tem @, digite novamente')
    else:
```

Qual o seu e-mail?hashtagmail.com

Usuário digita e-mail sem @

```
-----  
ValueError                                     Traceback (most recent call last)
<ipython-input-6-e31cf5c841f2> in descobrir_servidor(email)
      2     try:
----> 3         posicao_a = email.index('@')
      4     except:
  
ValueError: substring not found
```

During handling of the above exception, another exception occurred:

```
-----  
ValueError                                     Traceback (most recent call last)
<ipython-input-9-882fae42d366> in <module>
      1 email = input('Qual o seu e-mail?')
----> 2 print(descobrir_servidor(email))
  
<ipython-input-6-e31cf5c841f2> in descobrir_servidor(email)
      3     posicao_a = email.index('@')
      4     except:
----> 5         raise ValueError('Email digitado não tem @, digite novamente')
      6     else:
      7         servidor = email[posicao_a:]
```

Erro personalizado

ValueError: Email digitado não tem @, digite novamente

Voltando para o nosso exemplo de cálculo de média das notas. No caso anterior criamos 2 argumentos `nota1` e `nota2`, existe uma outra forma que podemos fazer essa declaração de variáveis.

Você verá em alguns lugares durante pesquisas o termo **\*args**.

Usando o **\* antes da variável** indicamos que não estamos tratando de um argumento único e sim um argumento que poderá receber um número ilimitado de valores.

Podemos ver no exemplo que ao usarmos **\*notas** permitimos não só ter apenas 2 notas mas 3, 4 ,5 ou 10...

Ao invés de limitar o número de notas, criamos um argumento de tamanho variável

```
1 def calcular_media(*notas):  
2     soma=0  
3     for nota in notas:  
4         soma += nota  
5     return soma/(len(notas))  
6  
7 print(calcular_media(10,0,5,6,7,8,9,1,0,4,8,6,4,8,9,4,2,2,2,3,4,5,8,9))  
8 print(calcular_media(10,0))  
9  
5.166666666666667  Apenas 2 argumentos  
5.0  Diversos argumentos
```

Como vimos, temos tanto argumentos de posição como argumentos **Keywords**(palavras-chave).

O conceito é exatamente o mesmo usado no exemplo anterior. Ao usarmos o **\*\*** antes da palavra-chave, temos uma função que aceita receber diversas palavras chaves na função.

Esse conceito, é comumente visto com **\*\*kwargs**.

Ao lado, temos um exemplo deste caso. Aqui, o usuário poderá(ou não) passar informações para que influenciem no preço final do produto.

No caso dos **\*\*kwargs**, é importante entender o que a função faz. Olhando dentro da nossa função `preco_final`, vemos que existem três variáveis que influenciam o preço:

- desconto;
- garantia\_extra;
- Imposto.

\*\* indicando que temos Keywords variáveis

```
1 def preco_final(preco, **adicionais):
2     print(adicionais)
3     if 'desconto' in adicionais:
4         preco *= (1 - adicionais['desconto'])
5     if 'garantia_extra' in adicionais:
6         preco += adicionais['garantia_extra']
7     if 'imposto' in adicionais:
8         preco *= (1 + adicionais['imposto'])
9     return preco
```

```
1 print(preco_final(1000, desconto=0.1, imposto=0.3))
```

```
{'desconto': 0.1, 'imposto': 0.3}
```

1170.0

Perceba que mesmo sem usar todos os argumentos nosso valor final foi calculado.

Ok. Entendi que tenho liberdade de ter \*args ou \*\*kwargs mas por enquanto só vimos isso separadamente.

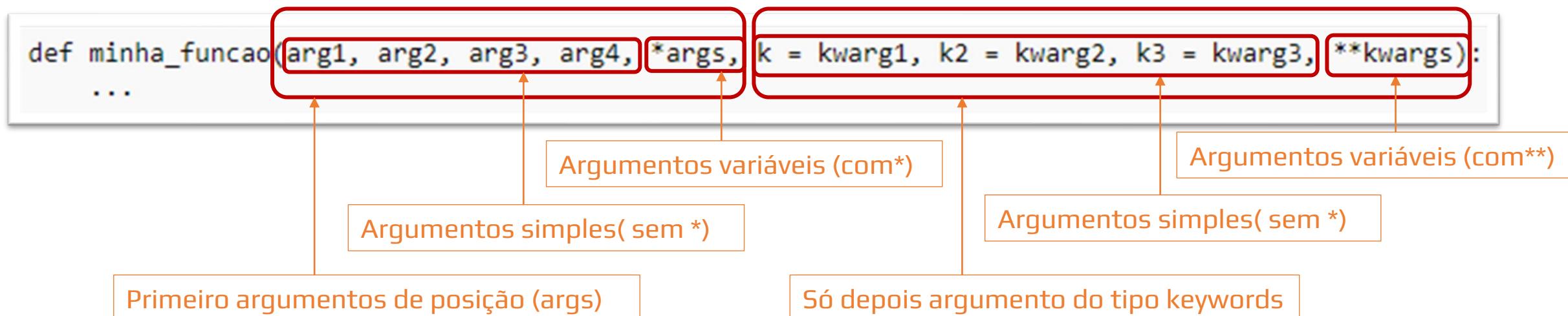
E se eu tiver um caso mais confuso, com 3 kwargs, 2 args?

Tem um jeito certo de fazer? Ou é só sair colocando?

Temos 2 regras para seguir:

- 1) Primeiro colocamos os argumentos de posição(args) depois os de palavra chave(kwargs)
- 2) Seguindo a ordem anterior primeiro fazemos os argumentos simples (sem \* ou \*\*) e depois os múltiplos (com \* ou \*\*)

Vamos dar uma olhada no estrutura abaixo para entendermos melhor.



## Módulo 15

# Módulos, Bibliotecas e Introdução a Orientação a Objeto

# Módulo 15 – Módulos, Bibliotecas e Introdução a Orientação a Objeto

## – O que é Orientação a Objeto e Por que isso importa

230

É bem provável que você já tenha escutado a expressão orientado a objetos. Assim como outras linguagens de programação o Python também é orientada a objetos.

E o que isso significa?

Vamos primeiro usar um exemplo da vida real

O que pode ser feito com um objeto como o Celular?

Ligar, mandar mensagens, ver vídeos, etc

Com uma bicicleta?

Andar, empurrar, etc

Na programação essas ações que podem ser executadas com os objetos são chamados de **Métodos**. Então para cada um dos objetos que temos é possível fazer algumas ações.

A gente vai ver isso melhor mais para frente. Fica tranquilo!



Ligar  
Mandar Mensagens  
Ver vídeos



Andar  
Empurrar  
Frear

# Módulo 15 – Módulos, Bibliotecas e Introdução a Orientação a Objeto

## – O que são módulos e qual a importância

231

Até agora só vimos os objetos e métodos que existem dentro do próprio Python, mas uma das principais “armas” dessa linguagem são as chamadas bibliotecas.

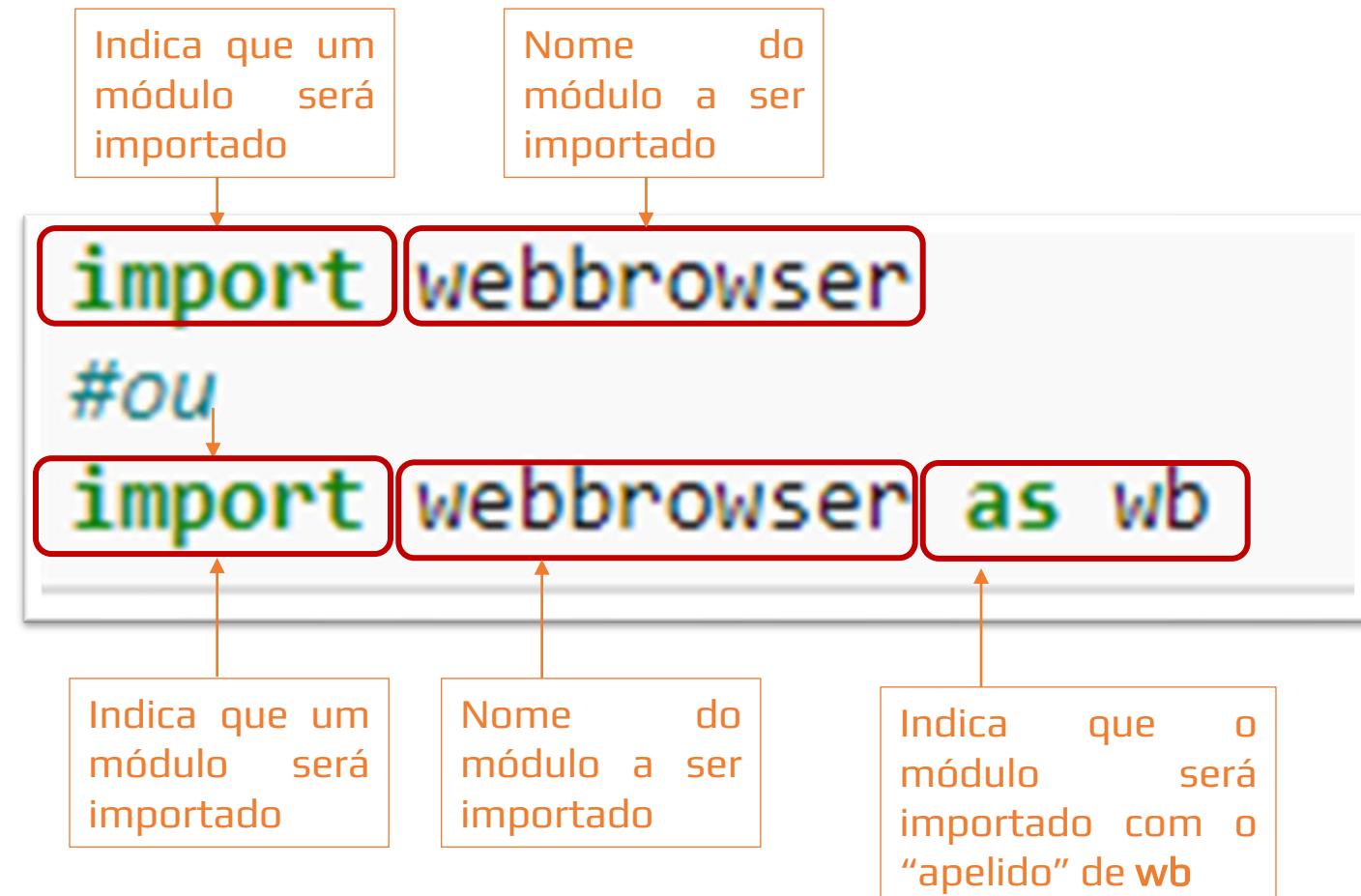
O que são bibliotecas? Elas são módulos prontos que podem ser importados para nossos projetos. Elas nos permitem executar códigos que nos demandariam muito tempo de desenvolvimento para executá-los.

Existem milhares de módulos disponíveis, ter todos eles integrados no Python nos demandaria muita memória.

Por isso, vamos precisar, sempre que necessário, importar esses módulos dentro de nossos projetos.

Para essa importação usaremos o comando:

`import`



Em alguns casos não é interessante importar toda a biblioteca mas apenas uma parte dela.

Para isso vamos usar a estrutura abaixo:

**From biblioteca import módulo**

Vamos ver um exemplo ainda usando o webbrowser.



Muito provavelmente o que você está se perguntando, é:

- 1) Mas o que faz o webbrowser?
- 2) Como saber que o módulo que preciso é o webbrowser?

É impossível apresentarmos no curso todas as bibliotecas existentes.

O webbrowser é uma das milhares bibliotecas existentes. Essa, especificamente nos ajuda a interagir com a internet(abrir uma página Web por exemplo).

No curso vamos apresentar as principais/mais famosas, mas uma ótima forma de descobrir bibliotecas é usando o nosso amigo GOOGLE.

A comunidade do Python é global e conseguimos achar resposta para qualquer dúvida que a gente venha a ter. Além disso, estamos aqui também para te ajudar OBVIAMENTE !😊

## Browser Controller Objects

Browser controllers provide these methods which parallel three of the module-level convenience functions:

`controller.open(url, new=0, autoraise=True)`

Display *url* using the browser handled by this controller. If *new* is 1, a new browser window is opened if possible. If *new* is 2, a new browser page ("tab") is opened if possible.

`controller.open_new(url)`

Open *url* in a new window of the browser handled by this controller, if possible, otherwise, open *url* in the only browser window. Alias `open_new()`.

`controller.open_new_tab(url)`

Open *url* in a new page ("tab") of the browser handled by this controller, if possible, otherwise equivalent to `open_new()`.

Métodos do pacote  
Webbrowser

Vamos aprofundar em um módulo bem recorrente em projetos de Python.

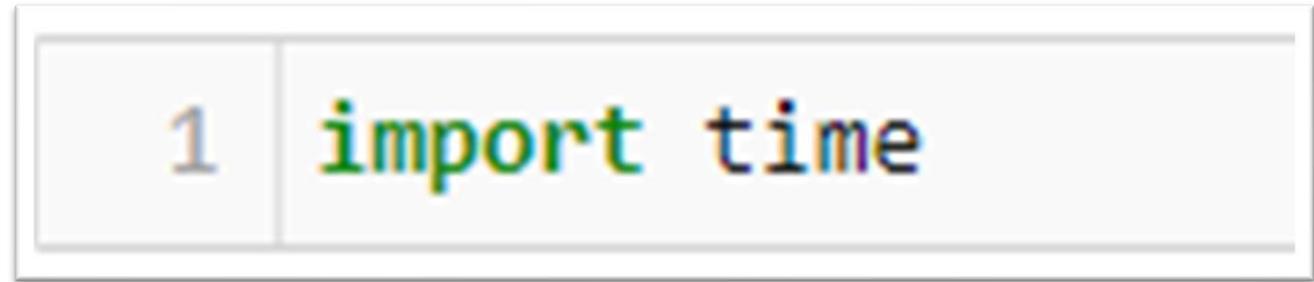
O módulo **time** é um dos vários módulos que nos ajudam a trabalhar com tempo e datas, mas é uma das mais famosas.

Essa função, possui diversos métodos. Aqui, vamos ver algumas delas mas caso você tenha interesse em conhecer mais sobre ela você pode acessar a documentação no link abaixo:

<https://docs.python.org/3/library/time.html>

Vamos primeiro, importar nossa biblioteca

Usando o **import time**



Vamos ver os principais métodos da biblioteca time :

- **EPOCH** - É o marco zero do Python. 01/01/1970;
- **time ()** - Nos retorna a diferença entre o tempo de hoje até a EPOCH em segundos
- **ctime()** - retorna uma **string** com a data no modelo UTC. UTC é um formato padrão utilizado pelo Python;
- **sleep()** - faz com que o python aguarde um tempo dado (em segundos) para executar a próxima linha de código;
- **gmtime()** - retorna as informações de data de forma detalhada;

Perceba que para usarmos os métodos sempre chamamos primeiramente a biblioteca.

Ou seja **time.método**

```
1 data_atual = time.gmtime()
```

```
2 print(data_atual)
```

```
time.struct_time(tm_year=2020, tm_mon=10, tm_mday=6, tm_hour=19, tm_min=42, tm_sec=38, tm_wday=1, tm_yday=280, tm_isdst=0)
```

```
1 segundos_hoje = time.time()
2 print(segundos_hoje)
```

1602013004.6264832

```
1 #para esperar 5 segundos fazemos:
2 print('Começando')
3 time.sleep(5)
4 print('Rodou 5 segundos após')
```

Começando

Rodou 5 segundos após

```
1 data_hoje = time.ctime()
2 #ou então data_hoje = time.ctime(time())
3 print(data_hoje)
```

Tue Oct 6 16:37:36 2020

Vamos para outra biblioteca, dessa vez para criação de gráficos. Uma das bibliotecas mais famosas é o **matplotlib**. Caso tenha mais interesse sobre a biblioteca, aqui está o link da documentação da biblioteca:

<https://matplotlib.org/stable/contents.html#>

Vamos importar o **matplotlib** como **plt**. Para criarmos um gráfico basta usarmos os métodos abaixo:

- **.plot(eixox,eixoy)** – indica quais são os dados que formarão os gráficos;
- **.ylabel()** – Define o rótulo de dados do eixo Y;
- **.xlabel()** – Define o rótulo de dados do eixo X;
- **.axis()** – Define mínimo e máximo dos eixos X e Y. Nessa ordem.
- **.show()** – Plota o gráfico

```
1 vendas_meses = [1500, 1727, 1350, 999, 1050, 1027, 1022, 1500, 2000, 2362, 2100, 2762]
2 meses = ['jan', 'fev', 'mar', 'abr', 'mai', 'jun', 'jul', 'ago', 'set', 'out', 'nov', 'dez']
3
4 #plotar o gráfico da forma mais simples
5 import matplotlib.pyplot as plt
6
7 plt.plot(meses, vendas_meses)          ← Métodos
8 plt.ylabel('Vendas')                  ←
9 plt.xlabel('Meses')                   ←
10 plt.axis([0, 12, 0, max(vendas_meses)+500]) ←
11 plt.show()                           ←
```

| Mês | Vendas |
|-----|--------|
| jan | 1500   |
| fev | 1727   |
| mar | 1350   |
| abr | 999    |
| mai | 1050   |
| jun | 1027   |
| jul | 1022   |
| ago | 1500   |
| set | 2000   |
| out | 2362   |
| nov | 2100   |
| dez | 2762   |

Vamos entender um pouco melhor como são usados este métodos.

O método `plot`. Ao usarmos a ordem (`meses, vendas_meses`), definimos que:

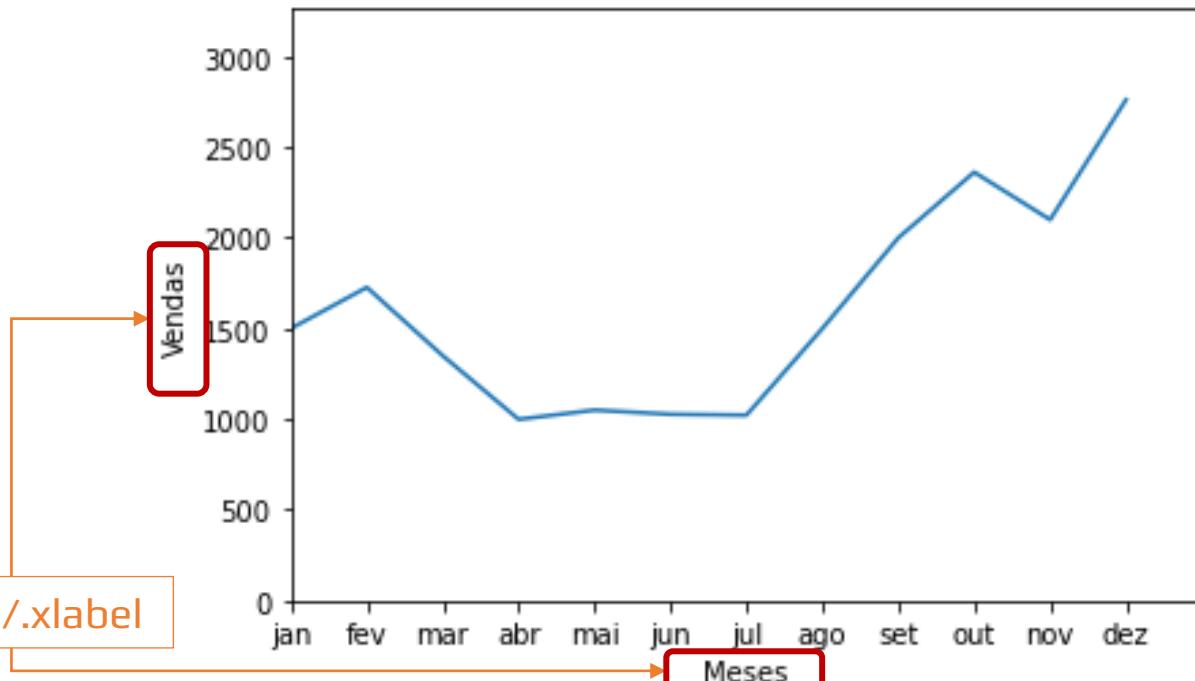
- `meses` estará no eixo X;
- `venda_meses` no eixo y;

Isso ocorre pois os argumentos deste método estabelece essa ordem.

Assim como vimos no módulo de funções, precisamos sempre lembrar que a ordem é importante.

Já os métodos `.ylabel` e `.xlabel` funcionam da mesma forma, só alterando o eixo a que se referem. Respectivamente eixo y e eixo x. Perceba que '`Vendas`' e '`Meses`', são STRINGS e não se referem as variáveis `meses` e `vendas_meses`. São apenas rótulos do gráfico.

```
6  
7 plt.plot(meses, vendas_meses)  
8 plt.ylabel('Vendas')  
9 plt.xlabel('Meses')  
10 plt.axis([0, 12, 0, max(vendas_meses)+500])  
11 plt.show()
```



O próximo método utilizado é o `.axis()`. Este método nos permite definir os valores mínimos e máximo dos eixos x e y.

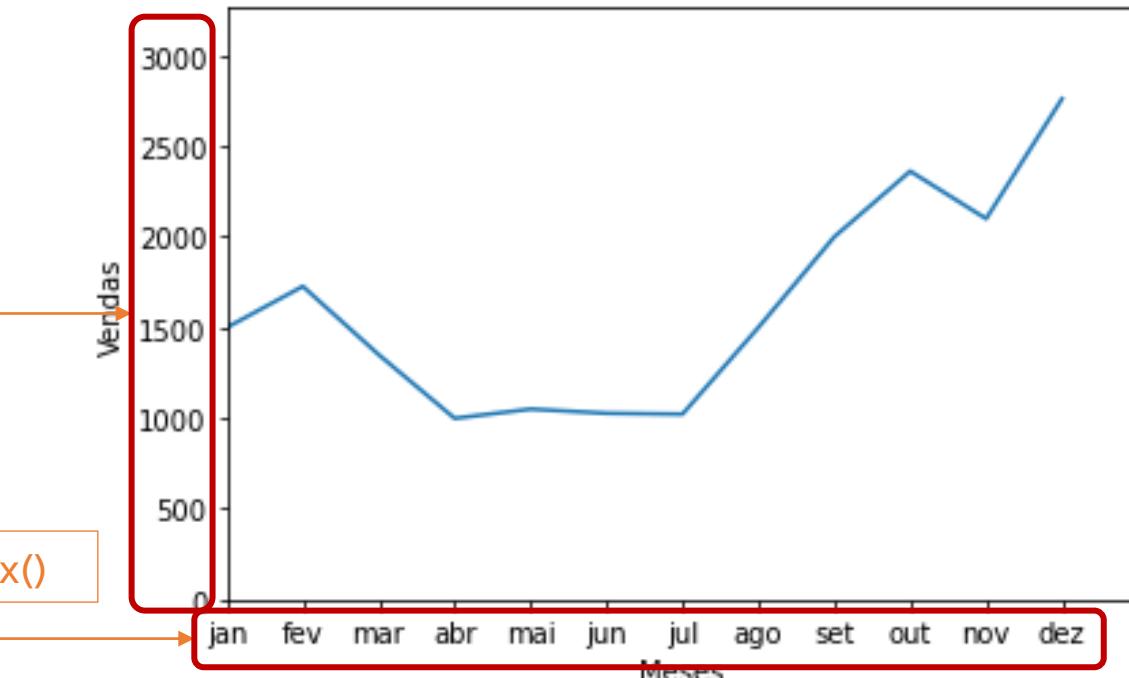
**IMPORTANTE!** Assim, como vimos no módulo de definição de funções, este método segue uma ordem específica.

- 1) Min eixo x;
- 2) Max eixo x;
- 3) Min eixo y;
- 4) Max eixo y.

Perceba que usamos a própria variável `vendas_meses` como parâmetro máximo do nosso eixo y ao fazermos `max(vendas_meses)+500`.

Por fim, usamos o método `show()` que nos fornece o gráfico definido por nós.

```
6
7 plt.plot(meses, vendas_meses)
8 plt.ylabel('Vendas')
9 plt.xlabel('Meses')
10 plt.axis([0, 12, 0, max(vendas_meses)+500])
11 plt.show()
```



Outro método muito comum, principalmente quando estamos tratando dados estatisticamente, é o numpy. Segue abaixo o link da documentação: [https://numpy.org/doc/stable/user/tutorials\\_index.html](https://numpy.org/doc/stable/user/tutorials_index.html)

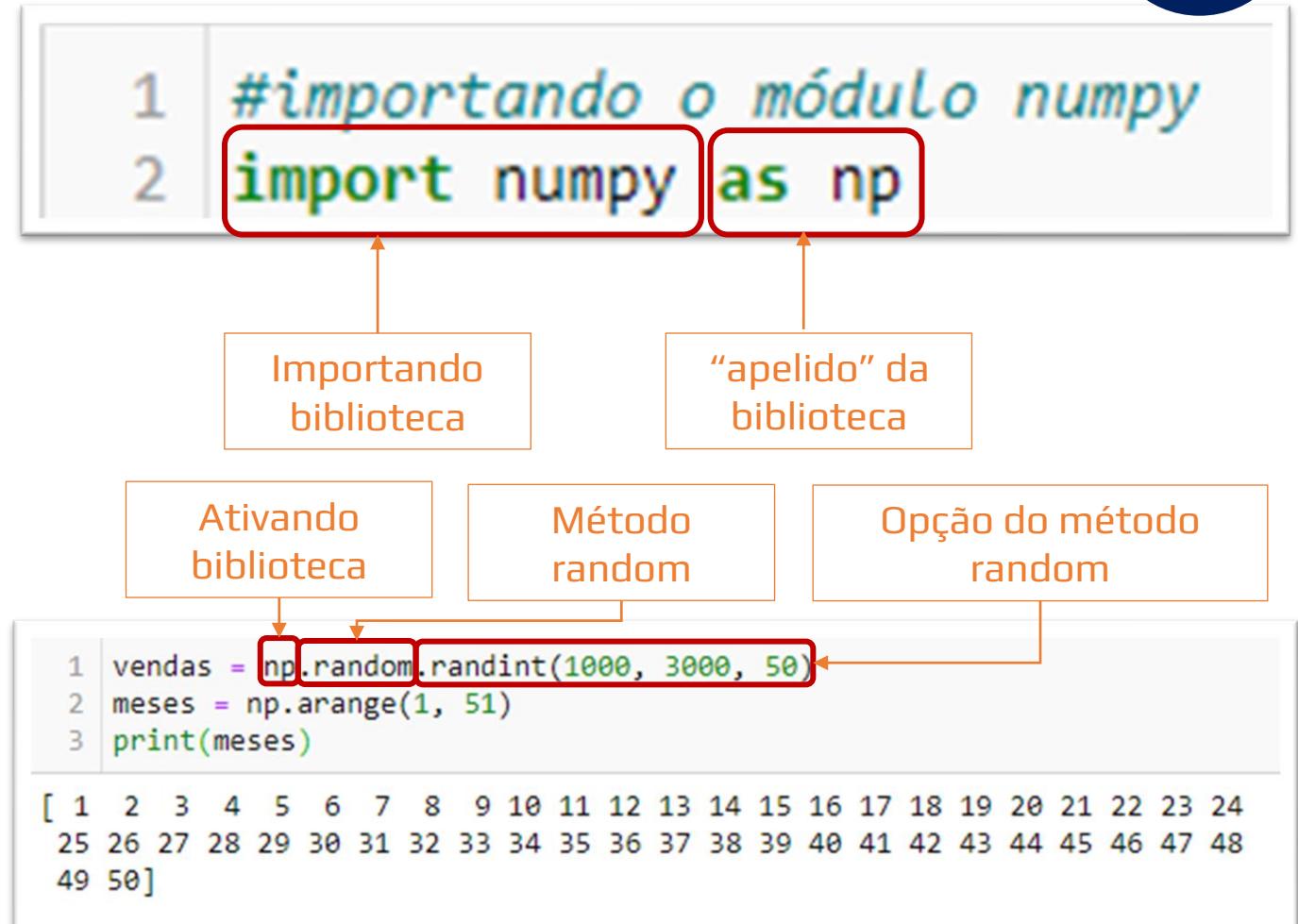
Vamos usar o Numpy juntamente do Matplotlib que vimos anteriormente.

Para importar o Numpy vamos usar o comando abaixo:

**Import numpy as np**

Aqui vamos replicar o exemplo anterior, mas utilizando o numpy para gerar números aleatórios de vendas (linha 1).

Já na linha 2 vamos usar o método `.arange()` para criação de 50 meses que coreesponderão as vendas geradas aleatoriamente.



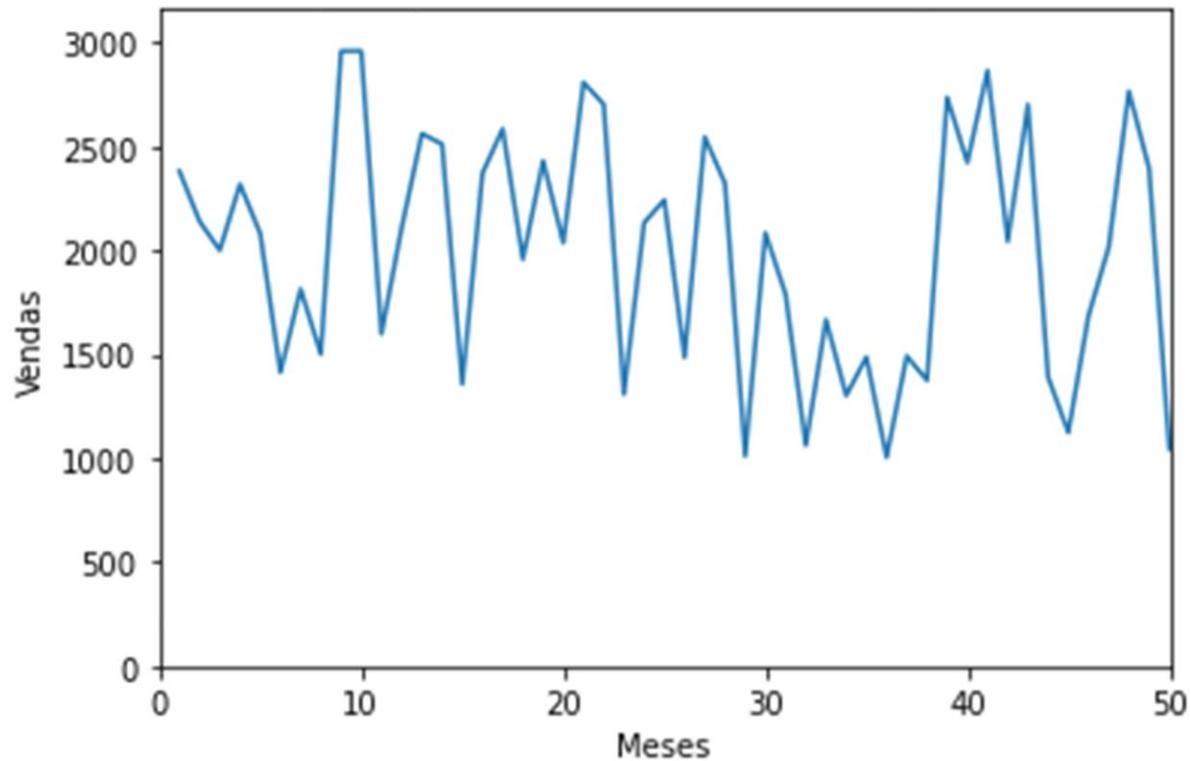
Vamos agora usar outras funcionalidades da nossa biblioteca matplotlib.

Usando a mesma linha de código do gráfico anterior temos este gráfico apresentado ao lado.

No entanto, conforme falamos, nossa biblioteca tem dezenas, centenas de funcionalidades que podemos usar para melhorar a apresentação do gráfico.

É isso que vamos fazer nas próximas páginas.

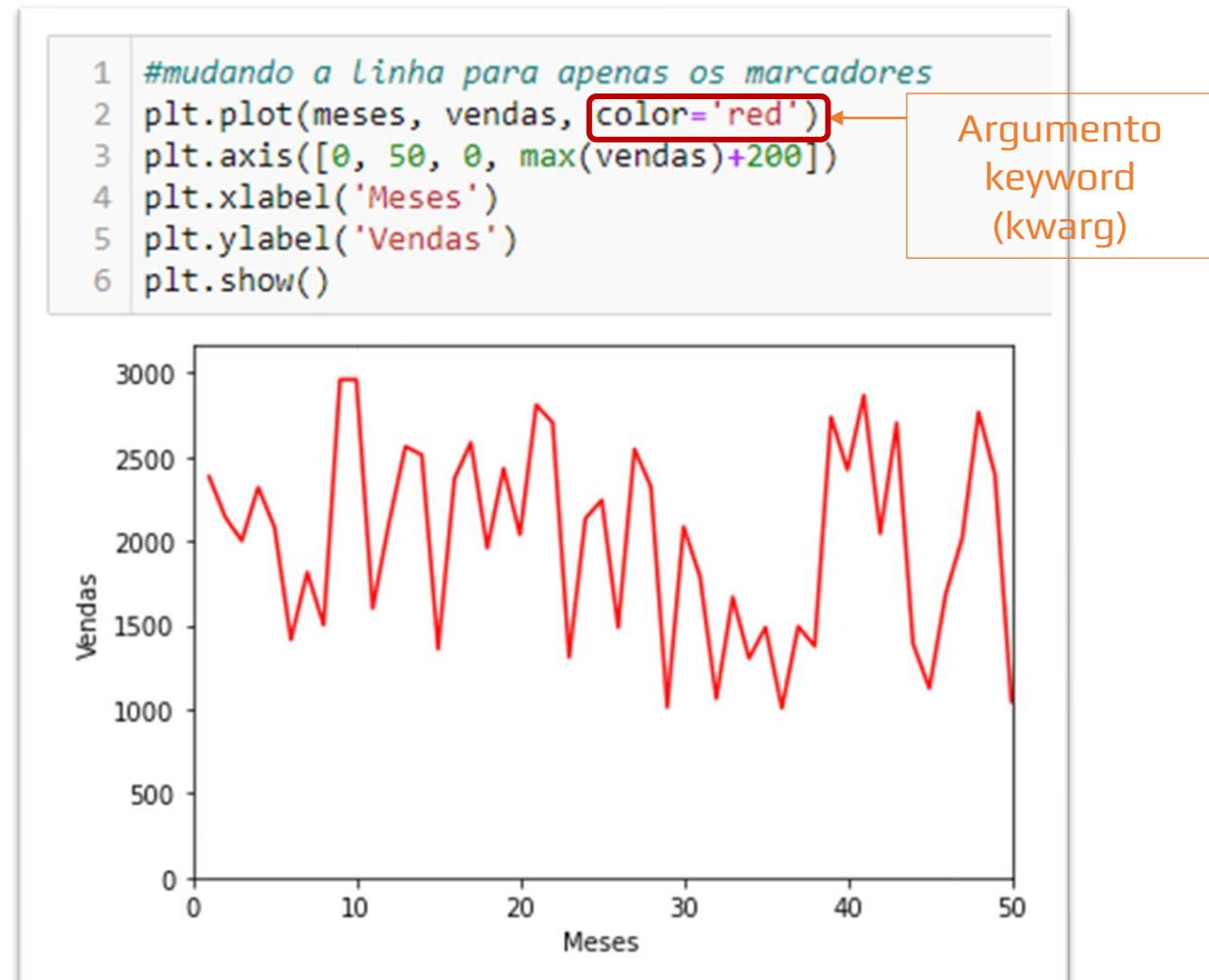
```
1 plt.plot(meses, vendas)
2 plt.axis([0, 50, 0, max(vendas)+200])
3 plt.xlabel('Meses')
4 plt.ylabel('Vendas')
5 plt.show()
```



Começando pelo método `.plot()`, vamos alterar a cor da linha do gráfico usando a keyword `color =`.

Perceba que apenas acrescentando uma keyword nos argumentos podemos alterar a cor.

Lembre-se!! Ao fazermos isso, estamos alterando a cor padrão (azul). Muito importante também que só foi possível usar a palavra chave **RESPEITANDO** que este argumento só foi utilizado **APÓS** a definição dos eixos X e Y.



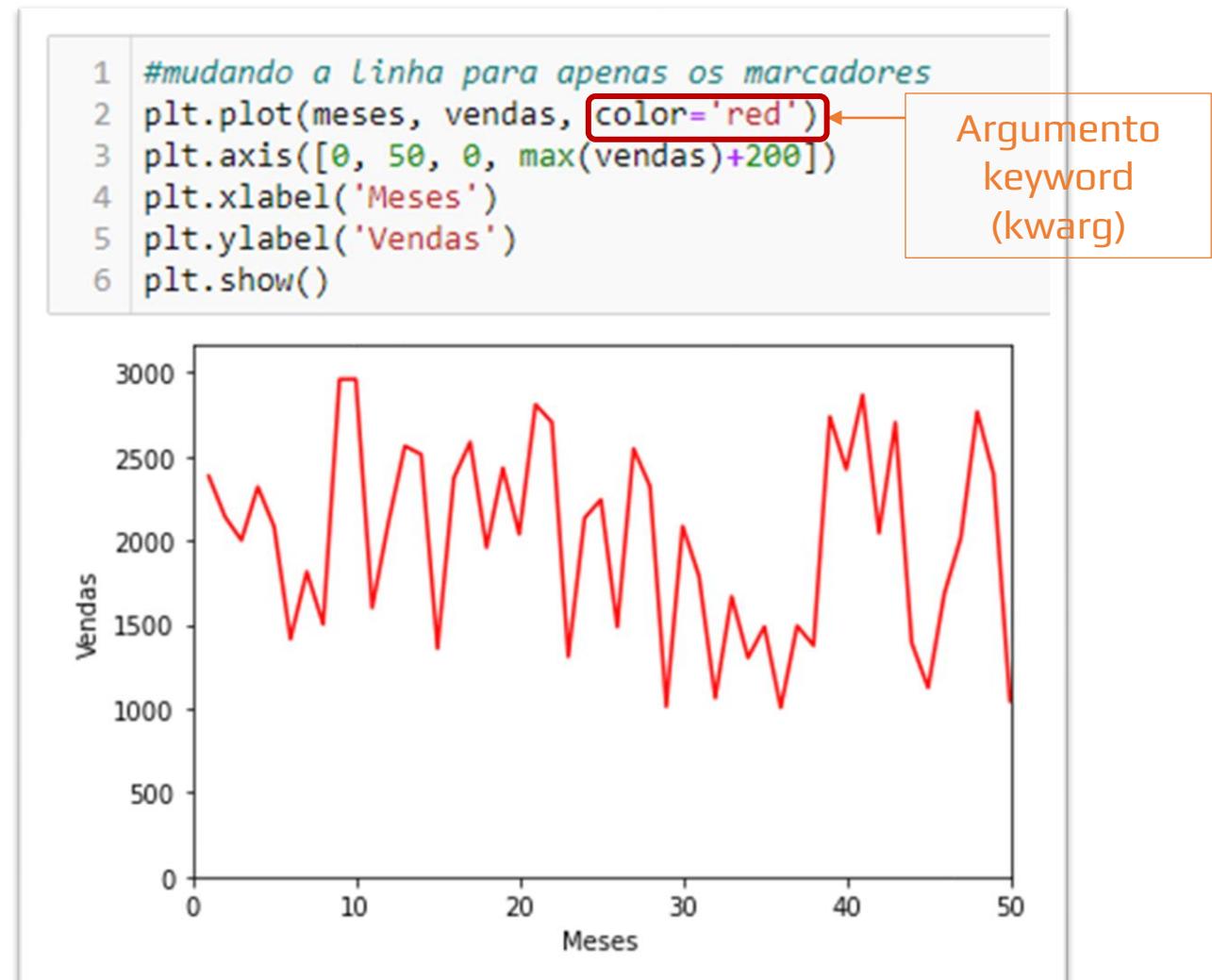
Aqui são inúmeras as possibilidades de alteração.

Talvez você se pergunte:

Preciso decorar isso tudo??

A resposta é NÃO! As documentações existem para serem consultadas. Obviamente quanto mais você exercitar, mais você saberá executar sem pesquisar mas FAZ PARTE do processo usar as documentações durante a programação.

Seja ela em Python ou qualquer linguagem.



Como temos diversos pacotes existentes, pode ser que durante uma pesquisa, um pacote que você achou interessante não está instalado no seu computador.

Como saber que a biblioteca não está instalada.

Vamos pegar o exemplo ao lado.

Ao tentar importar a biblioteca keyboard, recebi este erro **ModuleNotFoundError**.

Se você digitou corretamente o nome da biblioteca, isso significa que não temos isso instalado.

Para resolver esse problema, vamos usar um instalador que já existe “embutido” dentro do Python, o **PIP**.

```
1 import keyword
-----
ModuleNotFoundError: No module named 'keyword'                                     Traceback (most recent call last)
<ipython-input-1-f7cee866d5f5> in <module>
----> 1 import keyword

Argumento
keyword
(kwarg)
```

A captura de tela mostra uma janela de terminal com o seguinte conteúdo:

```
1 import keyword
-----
ModuleNotFoundError: No module named 'keyword'                                     Traceback (most recent call last)
<ipython-input-1-f7cee866d5f5> in <module>
----> 1 import keyword
```

O erro **ModuleNotFoundError: No module named 'keyword'** é destacado com um retângulo vermelho. Abaixo desse retângulo, uma caixa laranja contém o texto **Argumento keyword (kwarg)**, com uma seta apontando para o erro.

Usando o PIP, é possível, instalar, desinstalar pacotes.

Caso, você queira saber se um pacote está instalado, basta usar o comando abaixo em uma das células do Jupyter:

**pip freeze**

Caso não encontre na lista o pacote desejado, use o comando abaixo para instalar:

**pip install nome da biblioteca**

```
pip freeze
argh==0.26.2
asn1crypto==1.3.0
astroid==2.4.2
astropy==4.0.1.post1
atomicwrites==1.4.0
attrs==19.3.0
autopep8 @ file:///tmp/build/80754af9/autopep8_1592412889138/work
Babel==2.8.0
backcall==0.2.0
backports.functools-lru-cache==1.6.1
scipy @ file:///C:/ci/scipy_1592916963468/work
seaborn==0.10.1
selenium==3.141.0
Send2Trash==1.5.0
simplegeneric==0.8.1
singledispatch==3.4.0.3
sip==4.19.13
six==1.15.0
snowballstemmer==2.0.0
sortedcollections==1.2.1
sortedcontainers==2.2.2
soupsieve==2.0.1
Sphinx @ file:///tmp/build/80754af9/sphinx_1594223420021/work
```

Comando pip freeze

Lista de pacotes instalados no  
Python

Módulo 16

# List Comprehension

O que você aprenderá nesse módulo não é algo OBRIGATÓRIO. Esse módulo tem por objetivo apresentar práticas comuns na construção de códigos em Python. No início, mais importante que escrever seus códigos usando o List Comprehension é saber entender um código que esteja escrito dessa forma.

### Por que?

Boa parte do desafio em se programar, seja em Python ou qualquer outra linguagem é saber:

- O que pesquisar;
- Como pesquisar;
- Entender o que se achou;
- Adequar o que já existe para seu projeto.

É muito comum que programadores mais antigos se usem desses artifícios para que seu código fique mais elegante. Por isso é muito importante entender o que são e quais são os “pulos” de código que são usados no mercado.

Vamos para um caso que queremos calcular os impostos de produtos distintos.

O imposto é fixado em 30% do valor do produto.

O que faremos aqui é apresentar 2 soluções distintas que estão CORRETAS, mas escritas de forma distintas.

A primeira utiliza a lógica que temos usado até aqui(TRAIDICIONAL), já a segunda, usa o conceito de list comprehension.

Abaixo temos as duas formas de se resolver esse problema.

```
preco_produtos = [100, 150, 300, 5500]
produtos = ['vinho', 'cafeiteira', 'microondas', 'iphone']
```

“Tradicional”

```
1 impostos = []
2 for item in preco_produtos:
3     impostos.append(item * 0.3)
4 print(impostos)
```

```
[30.0, 45.0, 90.0, 1650.0]
```

List Comprehension

```
1 impostos = [preco * 0.3 for preco in preco_produtos]
2 print(impostos)
```

```
[30.0, 45.0, 90.0, 1650.0]
```

Você é capaz de entender o que está sendo feito no list comprehension comparando os dois códigos? Achou estranho? Fique tranquilo, não é trivial no início. Vamos entender melhor.

```
preco_produtos = [100, 150, 300, 5500]
produtos = ['vinho', 'cafeiteira', 'microondas', 'iphone']
```

```
1 impostos = []
2 for item in preco_produtos:
3     impostos.append(item * 0.3)
4 print(impostos)
```

```
[30.0, 45.0, 90.0, 1650.0]
```

```
1 impostos = [preco * 0.3 for preco in preco_produtos]
2 print(impostos)
```

```
[30.0, 45.0, 90.0, 1650.0]
```

Essencialmente o que ocorre aqui é:

- Na primeira solução (“Tradicional”):

Criamos uma variável item que percorre a lista preco\_produtos, que explicitamente adiciona(.append) o valor calculado na lista impostos.

- Na segunda solução :

O método .append não é explícito. A leitura da linha de código seria: “para cada preço da lista preco\_produtos será calculado um item da lista imposto. Esse valor será calculado por preço \* 0.3”

Vamos para outro exemplo.

Aqui, temos 2 listas separadas mas que se relacionam entre si.

Sabemos que se por exemplo ordenarmos a lista de vendas\_produtos do maior para o menor nada acontecerá na nossa lista produtos.

Ou seja, as posições que antes estavam “casadas” agora estão distintas.

Antes de qualquer tratamento, precisamos UNIR essas duas listas. Uma boa forma é transformar essas duas listas em uma única lista de tuplas.

Para isso, usaremos a função ZIP() que já vimos anteriormente no módulo de dicionários.

Possuindo a lista de tuplas é só aplicarmos o unpacking usando o List Comprehension como apresentado na figura ao lado.

```
1 vendas_produtos = [1500, 150, 2100, 1950]
2 produtos = ['vinho', 'cafeiteira', 'microondas', 'iphone']
3
4 lista_aux = list(zip(vendas_produtos, produtos))
5 lista_aux.sort(reverse=True)
6 produtos = [produto for vendas, produto in lista_aux]
7 print(produtos)
```

```
['microondas', 'iphone', 'vinho', 'cafeiteira']
```

List comprehension que pode ser lida como:  
“para cada item venda e produto da lista\_aux, será criado um item produto na lista produtos”

Mais um caso onde o list comprehension nos auxilia a reduzir as linhas de código. Este caso, é ainda menos intuitivo, principalmente no início da jornada na programação.

LEMBRANDO, nada disso é obrigatório, são formas distintas de se escrever onde o resultado final é o mesmo.

```
1 meta = 1000
2 vendas_produtos = [1500, 150, 2100, 1950]
3 produtos = ['vinho', 'cafeiteira', 'microondas', 'iphone']
```

```
1 produtos_acima_meta = [produto for i, produto in enumerate(produtos) if vendas_produtos[i] > meta]
2 print(produtos_acima_meta)
['vinho', 'microondas', 'iphone']
```

Nova lista que receberá os produtos que estão acima da meta

Valor que será incluído na lista “produtos\_acima\_meta”

For que permite percorrer toda a lista produtos

Condição a ser atendida.

Mais um caso onde o list comprehension nos auxilia a reduzir as linhas de código.

Nesse caso, vamos usar as estruturas de IF/ELSE para criar uma lista de vendedores que bateram uma meta pré-definida.

LEMBRANDO, nada disso é obrigatório, são formas distintas de se escrever onde o resultado final é o mesmo.

```
1 vendedores_dic = {'Maria': 1200, 'José': 300, 'Antônio': 800, 'João': 1500, 'Francisco': 1900, 'Ana': 2750, 'Luiz': 400, 'Pa  
2 meta = 1000
```

```
1 bonus = [vendedores_dic[item] * 0.1 if vendedores_dic[item] > meta else 0 for item in vendedores_dic]  
2 print(bonus)
```

[120.0, 0, 0, 150.0, 190.0, 275.0, 0, 0, 0, 0, 0, 0, 110.0, 0, 0, 0, 0, 0, 111.1000000000001, 0, 0, 0, 0]

Nova lista que receberá os produtos que estão acima da meta

Valor que será incluído na lista “bonus”

Condição a ser atendida

Caso a condição não seja atendida

For que percorre os itens do dicionário “vendedores\_dic”

Mais um caso onde o list comprehension nos auxilia a reduzir as linhas de código.

Nesse caso, não vamos criar uma lista, mas sim retornar um valor da soma das vendas dos produtos listados como "TOP5".

LEMBRANDO, nada disso é obrigatório, são formas distintas de se escrever onde o resultado final é o mesmo.

```
1 produtos = ['coca', 'pepsi', 'guarana', 'skol', 'brahma', 'agua', 'del valle', 'dolly', 'red bull', 'cachaça', 'vinho tinto'
2 vendas = [1200, 300, 800, 1500, 1900, 2750, 400, 20, 23, 70, 90, 80, 1100, 999, 900, 880, 870, 50, 1111, 120, 300, 450, 800]
3 top5 = ['agua', 'brahma', 'skol', 'coca', 'leite de castanha']

1 total_top5 = sum(vendas[i] for i, produto in enumerate(produtos) if produto in top5)
2 print(total_top5)
3 print('Top 5 representou {:.01%} das vendas'.format(total_top5/sum(vendas)))
```

8461

Top 5 representou 50.6% das vendas

Variável que receberá a SOMA das vendas dos top5 produtos

Função SOMA.

Valores que serão somados CASO a condição seja atendida

Percorre a lista produtos testando a condição definida na parte final da linha

Testa a condição se o produto da lista produtos da vez está na lista top5.

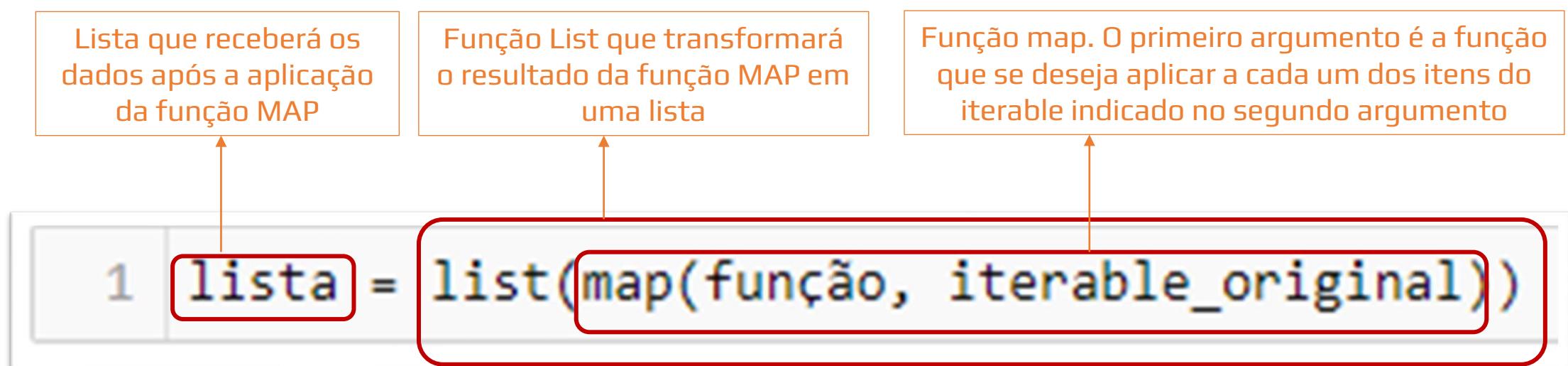
Módulo 17

# Functions em Iterables e a função map

Veremos agora alguns casos que nos permitem usarmos funções dentro de funções. Na prática eles farão o papel do List Comprehension. Novamente, não são linhas de código obrigatórias, pois existem formas de escrever o código da mesma forma sem o uso destas funções.

A primeira função que veremos é o MAP. Ela nos permite aplicar uma função a cada um dos itens de um iterable. Lembrando que os iterables são listas, dicionários, strings etc.

Abaixo definimos o formato padrão de uma expressão MAP:



Voltando para nossa exercício de padronização de códigos, vamos entender como poderíamos usar a função MAP para responder este caso.

Perceba que primeiro definimos uma função que padroniza os códigos.

Usamos a função definida DENTRO do argumento do MAP juntamente da lista PRODUTOS que é um iterable.

Perceba que assim como vimos com a função ZIP, a função MAP nos retorna um OBJETO que precisa ser transformado em uma lista, assim como é realizado na última célula do jupyter.

```
def padronizar_texto(texto):
    texto = texto.casefold()
    texto = texto.replace(" ", " ")
    texto = texto.strip()
    return texto
```

Definindo uma função

- Agora queremos padronizar uma lista de códigos:

```
produtos = ['ABC12 ', 'abc34', 'AbC37', 'beb12', 'BSA151', 'BEB23']
```

```
produtos = map(padronizar_texto, produtos)
print(produtos)
```

Função map()

```
<map object at 0x0000010DA8F8FDF0>
```

Resultado da função MAP é um objeto

```
produtos = list(map(padronizar_texto, produtos))
print(produtos)
```

Conversão do objeto MAP em uma lista

```
['abc12', 'abc34', 'abc37', 'beb12', 'bsa151', 'beb23']
```

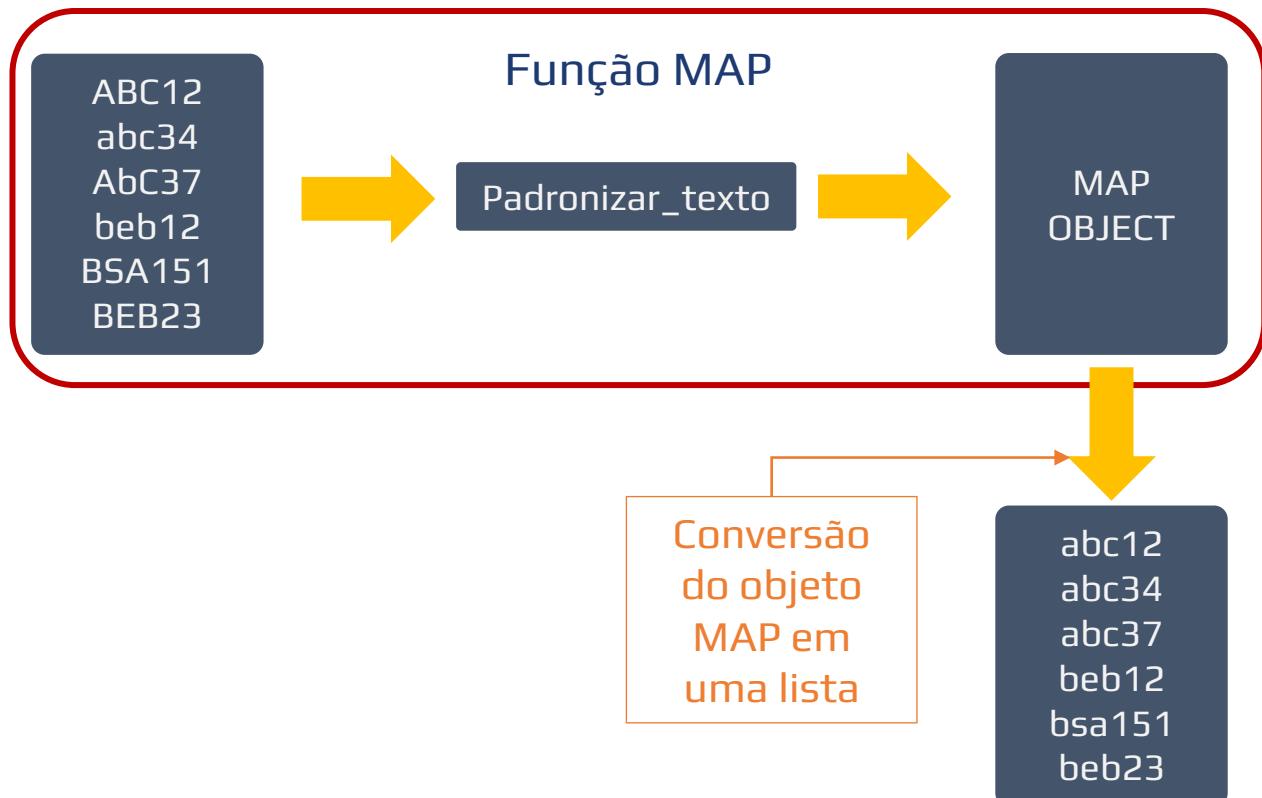
```
produtos = ['ABC12', 'abc34', 'AbC37', 'beb12', 'BSA151', 'BEB23']

produtos = map(padronizar_texto, produtos)
print(produtos)
```

Vamos olhar mais a fundo o que está ocorrendo dentro da função map do exemplo anterior.

Veja a função MAP como uma espécie de linha de produção.

O que ela faz é pegar cada um dos itens do ITERABLE(no nosso caso a lista produtos) e passar esse item pela função indicada(nesse caso padronizar\_texto). O resultado dessa linha de produção, é um grande pacote de items transformados em algo de acordo com a função indicada.



Já conhecemos o método `.sort()` aplicável a listas.

Quando usamos esse método para lista de strings por exemplo, a ordem não será alfabética e sim seguindo a tabela ASCII. (exemplo ao lado)

O Python nos permite fornecer algum critério chave para que a ordem seja realizada.

Par aíssso usaremos o argumento `key=` dentro da nossa função `.sort`.

Key indica que um critério específico será fornecido. Nesse caso, `casefold`(letras minúsculas).

Perceba que foi necessário o `str`, pois `casefold` é um método de STRINGS



As lambda expressions, são mais um conceito que não é obrigatório, mas é bem comum no mercado de trabalho e certamente irá surgir em códigos que você venha a ler no futuro.

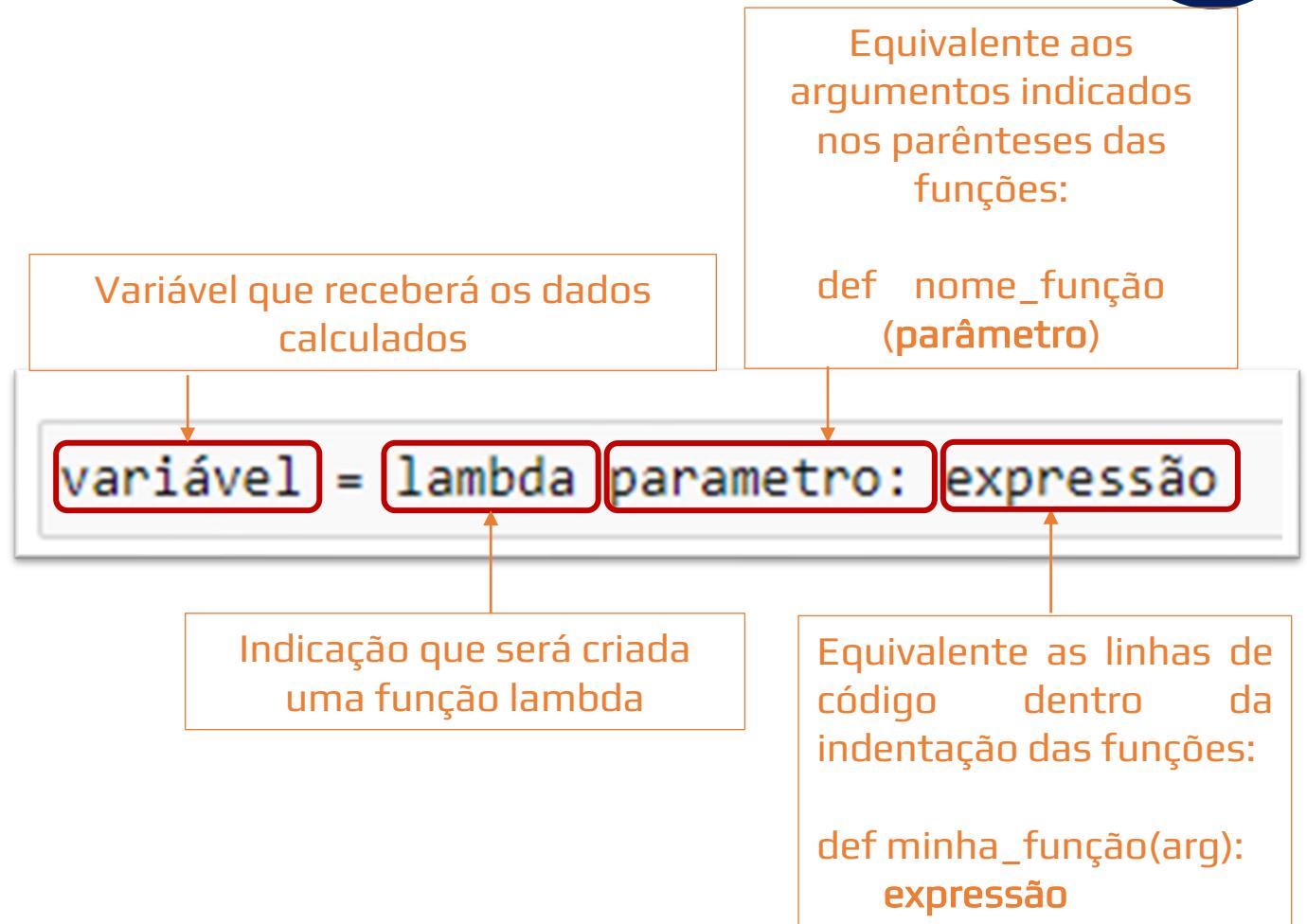
Ela funciona como uma versão simplificada da função que nós criamos usando def etc...

A Lambda é uma função Anônima.

Ela não é chamada como as funções como vimos até agora. Ela é uma função local, que só funciona dentro de uma variável que definirmos.

Chamamos de anônima, pois não é necessário chama-la como fazíamos com as funções “normais”.

Antes de irmos para um exemplo, vamos entender sua estrutura.



Vamos analisar agora um exemplo real.

O desafio, é criar uma função que calcule o imposto devido baseado no preço de um produto.

Criamos uma função “normal” e uma lambda para que seja possível compará-las.

Perceba que a função lambda nos permite declarar toda a função em apenas uma linha.

No entanto, o mesmo resultado pode ser obtido usando a função convencional.

```
imposto = 0.3

#Criando uma função "normal"
def preco_imposto(preco):
    return preco * (1 + 0.3)
print(preco_imposto(100))

#Criando uma função Lambda
preco_imposto2 = lambda preco: preco * (1.0 + imposto)

print(preco_imposto2(100))
130.0  Indicação que será criada
130.0  uma função lambda
```

Módulo 18

# Análise da Dados com o Pandas

O Pandas é mais uma biblioteca disponível que nos auxilia a ser mais eficiente.

Sem dúvida, é uma das mais importantes bibliotecas do Python.

O Pandas, nos permite importar, tratar uma quantidade enorme de dados em poucas linhas de código e muito rápido.

É uma das ferramentas mais usadas para análise de dados e Data Science.

Se tornou uma “convenção” importar esta biblioteca como pd. Portanto, não estranhe se sempre ler pd ao invés de pandas.



Quando estamos tratando muitos dados é muito comum que esses arquivos estejam em arquivos .csv.

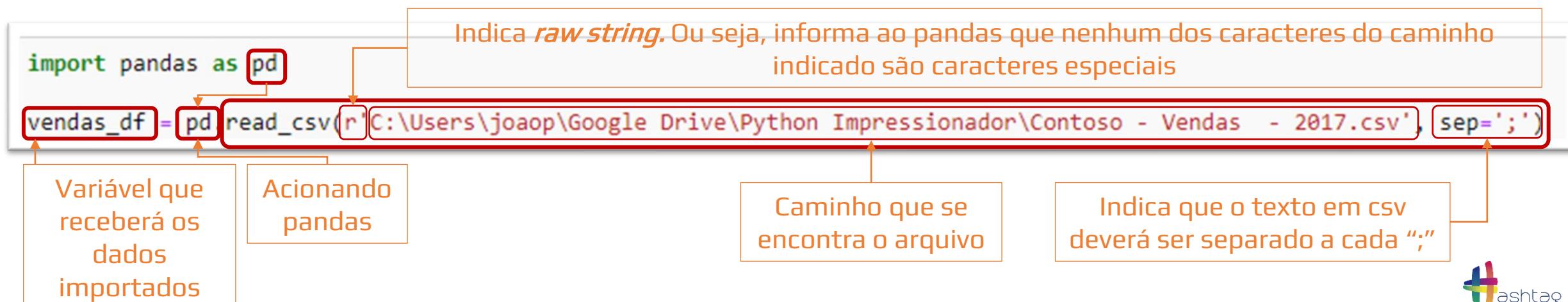
Esse formato é muito comum ao se extrair dados de sistemas.

Existem algumas bibliotecas que permitem importar esses dados para o Python, mas o Python é uma das mais conhecidas e é extremamente simples de se fazer a extração.

Para importarmos os dados usaremos a função abaixo:

`pd.read_csv()`

Além disso, precisamos criar uma variável que receberá esses dados e informar o caminho de onde se encontra o arquivo que será importado no nosso exemplo chamaremos essa variável de `vendas_df`.



Podemos ver que a importação foi feita. O pandas por padrão, já nos fornece uma tabela formatada que nos permite visualizar melhor os dados. Além disso, nos informa o número de linhas e colunas da tabela no canto inferior esquerdo.

```
import pandas as pd  
  
vendas_df = pd.read_csv(r'C:\Users\joaop\Google Drive\Python Impressionador\Contoso - Vendas - 2017.csv', sep=';')  
  
vendas_df
```

|        | Numero da Venda | Data da Venda | Data do Envio | ID Canal | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida | Quantidade Devolvida |
|--------|-----------------|---------------|---------------|----------|---------|------------|-------------|------------|--------------------|----------------------|
| 0      | 1               | 01/01/2017    | 02/01/2017    | 1        | 86      | 981        | 2           | 6825       | 9                  | 1                    |
| 1      | 2               | 01/01/2017    | 06/01/2017    | 5        | 308     | 1586       | 2           | 18469      | 9                  | 1                    |
| 2      | 3               | 01/01/2017    | 01/01/2017    | 0        | 294     | 1444       | 5           | 19730      | 13                 | 1                    |
| 3      | 4               | 01/01/2017    | 01/01/2017    | 0        | 251     | 1468       | 5           | 29326      | 6                  | 1                    |
| 4      | 5               | 01/01/2017    | 07/01/2017    | 6        | 94      | 1106       | 2           | 22617      | 4                  | 1                    |
| ...    | ...             | ...           | ...           | ...      | ...     | ...        | ...         | ...        | ...                | ...                  |
| 980637 | 980638          | 31/12/2017    | 31/12/2017    | 0        | 194     | 2490       | 4           | 10353      | 120                | 0                    |
| 980638 | 980639          | 31/12/2017    | 06/01/2018    | 6        | 32      | 2488       | 4           | 31750      | 120                | 0                    |
| 980639 | 980640          | 31/12/2017    | 03/01/2018    | 3        | 210     | 2511       | 10          | 12003      | 80                 | 0                    |
| 980640 | 980641          | 31/12/2017    | 31/12/2017    | 0        | 53      | 436        | 4           | 25550      | 39                 | 0                    |
| 980641 | 980642          | 31/12/2017    | 31/12/2017    | 0        | 309     | 2510       | 10          | 28707      | 80                 | 0                    |

980642 rows × 10 columns

Indica que temos 980642 linha e 10 colunas

Essa tabela do pandas é chamada de **dataframe**. O poder do pandas passa por ela.

Podemos ver as colunas do Dataframe como as chaves de um Dicionário.

Veja o exemplo ao lado, usando o 'nome da coluna 'ID Cliente' o pandas nos dá os resultados da coluna, além de alguns dados sobre ela que nos ajudam a entende-la.

| vendas_df['ID Cliente'] |       |
|-------------------------|-------|
| 0                       | 6825  |
| 1                       | 18469 |
| 2                       | 19730 |
| 3                       | 29326 |
| 4                       | 22617 |
| ...                     | ...   |
| 980637                  | 10353 |
| 980638                  | 31750 |
| 980639                  | 12003 |
| 980640                  | 25550 |
| 980641                  | 28707 |

Name: ID Cliente, Length: 980642, dtype: int64

Nome da coluna entre [ ] nos permite filtrar do dataframe apenas a informação que nos interessa

Índice das linhas

ID dos clientes

Informações sobre a coluna 'ID Cliente'

Da mesma forma, as linhas podem ser vistas como Listas. Essa comparação possui alguns detalhes, mas são bem parecidas.

Se quisermos apenas 1 linha ou diversas linhas podemos usar a estrutura apresentada abaixo:

```
: import pandas as pd  
  
vendas_df = pd.read_csv(r'Contoso - Vendas - 2017.csv', sep=';')
```

```
vendas_df[:3]
```

Intervalo [:3] indica que as linhas 0,1 e 2 serão apresentadas.  
Perceba que o título também é printado

|   | Numero da Venda | Data da Venda | Data do Envio | ID Canal   | ID Loja | ID Produto | ID Promocao |
|---|-----------------|---------------|---------------|------------|---------|------------|-------------|
| 0 |                 | 1             | 01/01/2017    | 02/01/2017 | 1       | 86         | 981         |
| 1 |                 | 2             | 01/01/2017    | 06/01/2017 | 5       | 308        | 1586        |
| 2 |                 | 3             | 01/01/2017    | 01/01/2017 | 0       | 294        | 1444        |

Índice das linhas

Se quisermos apenas uma linha precisamos usar a estrutura um pouco distinta da utilizada nas listas. O Pandas precisa sempre de um intervalo, logo precisamos de um início e um fim. Como no exemplo abaixo, onde o início é a **linha 2** e o fim **linha 3**.

```
: import pandas as pd  
  
vendas_df = pd.read_csv(r'Contoso - Vendas - 2017.csv', sep=';')  
  
vendas_df[2:3]
```

Intervalo [2:3] indica que as linhas 0,1 e 2 serão apresentadas.  
Perceba que o título também é printado

|   | Numero da Venda | Data da Venda | Data do Envio | ID Canal | ID Loja | ID Produto | ID Promocao | ID Cliente | Q |
|---|-----------------|---------------|---------------|----------|---------|------------|-------------|------------|---|
| 2 | 3               | 01/01/2017    | 01/01/2017    | 0        | 294     | 1444       | 5           | 19730      |   |

Índice da linha  
apresentada

Também é possível pegarmos várias colunas de um dataframe. Inclusive, se quisermos criar outro dataframe a partir dessas colunas basta atribuirmos ela a uma nova variável.

No exemplo ao lado, vamos criar um novo dataframe apenas com as colunas ‘ID Cliente’ e ‘Quantidade Vendida’.

Perceba que ao indicarmos ao pandas quais colunas buscamos, aproveitamos para armazena-las em um novo dataframe chamado “df\_vendas\_cliente”

Perceba que criamos uma lista dentro do colchetes que utilizamos para sinalizar as colunas que serão filtradas.

```
import pandas as pd  
  
vendas_df = pd.read_csv(r'Contoso - Vendas - 2017.csv', sep=';')  
  
df_vendas_clientes = vendas_df[['ID Cliente', 'Quantidade Vendida']]  
df_vendas_clientes
```

|       | ID Cliente | Quantidade Vendida |
|-------|------------|--------------------|
| 0     | 6825       | 9                  |
| 1     | 18469      | 9                  |
| 2     | 19730      | 13                 |
| 3     | 29326      | 6                  |
| 4     | 22617      | 4                  |
| ...   | ...        | ...                |
| 66667 | 100000     | 100                |

Colchetes indicam uma lista

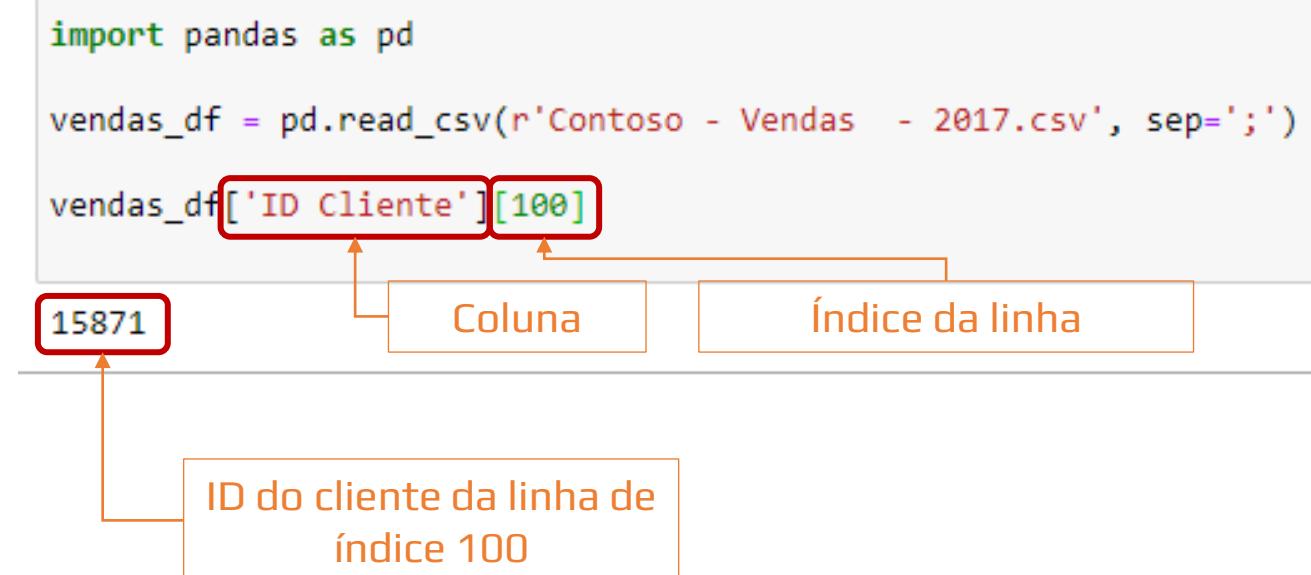
Colunas “filtradas”

Novo dataframe criado

Até agora, pegamos uma linha, uma coluna, várias linhas, várias colunas, mas além disso, também podemos acessar um valor específico de um dataframe.

No exemplo ao lado, pegamos o valor específico da linha de índice 100 da coluna 'ID Cliente'.

Perceba que assim como ocorrem nas listas, o índice a o número da linha são diferentes pois a tabela começa com índice 0(ZERO)



Agora que entendemos como fazer esses filtros vamos entender como podemos ter uma visão geral do nosso dataframe antes de iniciarmos mais profundamente nossas análises.

O pandas nos fornece um método que resume todas as colunas nos fornecendo informações importantes que podem acabar direcionando as próximas etapas de análise.

### .info()

Este método nos permite entender os tipos dos dados de cada uma das colunas, nome das colunas, se as linhas destas colunas são não nulas, etc.

```
import pandas as pd  
  
vendas_df = pd.read_csv(r'Contoso - Vendas - 2017.csv', sep=';')  
  
vendas_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 980642 entries, 0 to 980641  
Data columns (total 10 columns): ↓  
 #   Column           Non-Null Count  Dtype     
---  --  
 0   Numero da Venda  980642 non-null  int64    
 1   Data da Venda    980642 non-null  object    
 2   Data do Envio    980642 non-null  object    
 3   ID Canal          980642 non-null  int64    
 4   ID Loja           980642 non-null  int64    
 5   ID Produto        980642 non-null  int64    
 6   ID Promocao      980642 non-null  int64    
 7   ID Cliente        980642 non-null  int64    
 8   Quantidade Vendida 980642 non-null  int64    
 9   Quantidade Devolvida 980642 non-null  int64  
dtypes: int64(8), object(2)  
memory usage: 74.8+ MB
```

| # | Column               | Non-Null Count  | Dtype  |
|---|----------------------|-----------------|--------|
| 0 | Numero da Venda      | 980642 non-null | int64  |
| 1 | Data da Venda        | 980642 non-null | object |
| 2 | Data do Envio        | 980642 non-null | object |
| 3 | ID Canal             | 980642 non-null | int64  |
| 4 | ID Loja              | 980642 non-null | int64  |
| 5 | ID Produto           | 980642 non-null | int64  |
| 6 | ID Promocao          | 980642 non-null | int64  |
| 7 | ID Cliente           | 980642 non-null | int64  |
| 8 | Quantidade Vendida   | 980642 non-null | int64  |
| 9 | Quantidade Devolvida | 980642 non-null | int64  |

Informa quantos dos dados da coluna são não nulos

Tipo de dados em cada uma das colunas

Antes de qualquer análise, temos uma etapa muito importante. A verdade que é uma etapa um pouco chata e trabalhosa, mas o Pandas nos ajuda muito fazer dessa etapa algo mais fácil e direto.

Antes de qualquer alteração vamos importar novamente nossas bases e usar o método `.display()` que nos permitirá visualizar todas as bases.

Importante, lembrar que o caminho para importar os arquivos vai depender de onde os seus arquivos estão. No exemplo ao lado, como os arquivos estão DENTRO da MESMA pasta do nosso arquivo do Jupyter, não precisamos indicar a pasta.

Outro ponto importante é o encoding. É provável, que as bases que você usará possuem caracteres do português. O Python não reconhece esses símbolos por padrão. Por isso, algumas vezes precisamos usar o encoding para “traduzir” nossas bases para o Python.

```
import pandas as pd

#às vezes precisaremos mudar o encoding. Possíveis valores para testar:
#encoding='latin1', encoding='ISO-8859-1', encoding='utf-8' ou então encoding='cp1252'
vendas_df = pd.read_csv(r'Contoso - Vendas - 2017.csv', sep=';')
produtos_df = pd.read_csv(r'Contoso - Cadastro Produtos.csv', sep=';')
lojas_df = pd.read_csv(r'Contoso - Lojas.csv', sep=';')
clientes_df = pd.read_csv(r'Contoso - Clientes.csv', sep=';')

#usaremos o display para ver todos os dataframes
display(vendas_df)
display(produtos_df)
display(lojas_df)
display(clientes_df)
```

Opções de encoding. Nesse exemplo ele não é necessário.

Display dos dataframe importados

| Numero da Venda | Data da Venda | Data do Envio | ID Canal   | ID Loja | ID Produto | ID Promocao | ID Clier |
|-----------------|---------------|---------------|------------|---------|------------|-------------|----------|
| 0               | 1             | 01/01/2017    | 02/01/2017 | 1       | 86         | 981         | 2 68     |
| 1               | 2             | 01/01/2017    | 06/01/2017 | 5       | 308        | 1586        | 2 184    |
| 2               | 3             | 01/01/2017    | 01/01/2017 | 0       | 294        | 1444        | 5 197    |
| 3               | 4             | 01/01/2017    | 01/01/2017 | 0       | 251        | 1468        | 5 293    |
| 4               | 5             | 01/01/2017    | 07/01/2017 | 6       | 94         | 1106        | 2 226    |
| ...             | ...           | ...           | ...        | ...     | ...        | ...         | ...      |

Quando usamos o PANDAS, é muito comum importarmos bases muito maiores do que precisamos. O ideal na fase de tratamento de dados é termos apenas o necessário para nossa análise pois isso fará com que nosso programa seja mais eficiente.

Vamos olhar uma extração da base “clientes\_df”. Podemos ver que, temos uma série de colunas vazias que não agregam nenhum valor a nossa análise.

Uma forma de fazer esse tratamento é retirando do dataframe as informações inúteis. Para essa retirada, utilizamos o método abaixo:

`.drop()`

No próximo slide, vamos entender melhor como esse método funciona e seus argumentos.

|       | ID Cliente | Primeiro Nome | Sobrenome    | E-mail                        | Genero    | Numero de Filhos | Data de Nascimento | Unnamed: 7 | Unnamed: 8 | Unnamed: 9 | Unnamed: 10 |
|-------|------------|---------------|--------------|-------------------------------|-----------|------------------|--------------------|------------|------------|------------|-------------|
| 0     | 1          | Garnet        | Lanfranchi   | glanfranchi0@mayoclinic.com   | Feminino  | 2                | 12/05/1995         | NaN        | NaN        | NaN        | NaN         |
| 1     | 2          | Lurette       | Roseblade    | iroseblade1@bigcarfel.com     | Feminino  | 2                | 30/06/1943         | NaN        | NaN        | NaN        | NaN         |
| 2     | 3          | Glenden       | Ishchenko    | gishchenko2@moonfruit.com     | Masculino | 5                | 09/04/1989         | NaN        | NaN        | NaN        | NaN         |
| 3     | 4          | Baron         | Jedrzejewsky | bjedrzejewsky3@e-recht24.de   | Masculino | 4                | 17/11/1998         | NaN        | NaN        | NaN        | NaN         |
| 4     | 5          | Sheree        | Bredbury     | sbredbury4@sitemeter.com      | Feminino  | 5                | 08/09/1975         | NaN        | NaN        | NaN        | NaN         |
| ...   | ...        | ...           | ...          | ...                           | ...       | ...              | ...                | ...        | ...        | ...        | ...         |
| 39499 | 39500      | Brandy        | Malhotra     | brandy0@adventure-works.com   | Masculino | 3                | 27/07/1980         | NaN        | NaN        | NaN        | NaN         |
| 39500 | 39501      | Alicia        | Raje         | alicia12@adventure-works.com  | Feminino  | 5                | 10/05/2001         | NaN        | NaN        | NaN        | NaN         |
| 39501 | 39502      | Connie        | Rai          | connie4@adventure-works.com   | Feminino  | 0                | 13/02/1997         | NaN        | NaN        | NaN        | NaN         |
| 39502 | 39503      | Shawn         | Raji         | shawn23@adventure-works.com   | Masculino | 1                | 29/03/2001         | NaN        | NaN        | NaN        | NaN         |
| 39503 | 39504      | Lindsey       | Sharma       | lindsey10@adventure-works.com | Feminino  | 0                | 19/11/1961         | NaN        | NaN        | NaN        | NaN         |

Colunas sem valor nenhum para análise

O método `.drop()` funciona como uma espécie de “delete”. Ele tira do nosso dataframe as colunas ou linhas indicadas.

Vamos dizer que nossa intenção é retirar as colunas indicadas no nosso print. Usando o `drop`, vamos passar o intervalo dessas colunas e indicar se o que queremos é retirar as colunas ou as linhas, conforme indicado na linha de código ao lado.

Para esse método, temos 2 valores para o argumento `axis`:

- `AXIS=1` -> Retira COLUNAS;
- `AXIS=0` -> Retira LINHAS;



## ATENÇÃO !

O default do método `.drop()` é retirar LINHAS, portanto, se não indicarmos `axis=1` o PANDAS irá retirar todas as linhas das colunas indicadas, OU SEJA, TODAS AS COLUNAS DA PLANILHA.

| ID    | Cliente | Primeiro Nome | Sobrenome    | E-mail                        | Genero    | Numero de Filhos | Data de Nascimento | Unnamed: 7 | Unnamed: 8 | Unnamed: 9 | Unnamed: 10 |
|-------|---------|---------------|--------------|-------------------------------|-----------|------------------|--------------------|------------|------------|------------|-------------|
| 0     | 1       | Garnet        | Lanfranchi   | glanfranchi0@mayoclinic.com   | Feminino  | 2                | 12/05/1995         | NaN        | NaN        | NaN        | NaN         |
| 1     | 2       | Lurette       | Roseblade    | lroseblade1@bigcartel.com     | Feminino  | 2                | 30/06/1943         | NaN        | NaN        | NaN        | NaN         |
| 2     | 3       | Glenden       | Ishchenko    | gishchenko2@moonfruit.com     | Masculino | 5                | 09/04/1989         | NaN        | NaN        | NaN        | NaN         |
| 3     | 4       | Baron         | Jedrzejewsky | bjedrzejewsky3@eracht24.de    | Masculino | 4                | 17/11/1998         | NaN        | NaN        | NaN        | NaN         |
| 4     | 5       | Sheree        | Bredbury     | sbredbury4@sitemeter.com      | Feminino  | 5                | 08/09/1975         | NaN        | NaN        | NaN        | NaN         |
| ...   | ...     | ...           | ...          | ...                           | ...       | ...              | ...                | ...        | ...        | ...        | ...         |
| 39499 | 39500   | Brandy        | Malhotra     | brandy0@adventure-works.com   | Masculino | 3                | 27/07/1980         | NaN        | NaN        | NaN        | NaN         |
| 39500 | 39501   | Alicia        | Raje         | alicia12@adventure-works.com  | Feminino  | 5                | 10/05/2001         | NaN        | NaN        | NaN        | NaN         |
| 39501 | 39502   | Connie        | Rai          | connie4@adventure-works.com   | Feminino  | 0                | 13/02/1997         | NaN        | NaN        | NaN        | NaN         |
| 39502 | 39503   | Shawn         | Raji         | shawn23@adventure-works.com   | Masculino | 1                | 29/03/2001         | NaN        | NaN        | NaN        | NaN         |
| 39503 | 39504   | Lindsey       | Sharma       | lindsey10@adventure-works.com | Feminino  | 0                | 19/11/1961         | NaN        | NaN        | NaN        | NaN         |

Colunas que queremos retirar

```
clientes_df = clientes_df.drop(['Unnamed: 7','Unnamed: 8','Unnamed: 9','Unnamed: 10'],axis=1)
display (clientes_df)
```

Colunas a serem retiradas

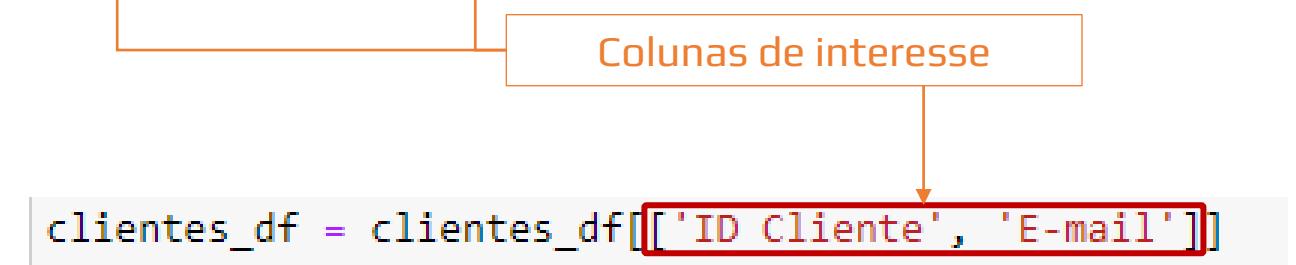
Indica que colunas serão retiradas

Outra forma que podemos tratar um dataframe para termos apenas as informações necessárias é criar um novo dataframe apenas com as colunas que queremos.

É mais comum usar essa abordagem em casos onde queremos poucos dados em relação ao todo da base de dados.

Para isso, vamos usar a estrutura, indicada na linha de código apresentada ao lado.

| ID Cliente | Primeiro Nome | Sobrenome | E-mail       | Genero                        | Numero de Filhos | Data de Nascimento | Unnamed: 7 | Unnamed: 8 | Unnamed: 9 | Unnamed: 10 |
|------------|---------------|-----------|--------------|-------------------------------|------------------|--------------------|------------|------------|------------|-------------|
| 0          | 1             | Garnet    | Lanfranchi   | gianfranchi0@mayoclinic.com   | Feminino         | 2                  | 12/05/1995 | NaN        | NaN        | NaN         |
| 1          | 2             | Lurette   | Roseblade    | iroseblade1@bigcartel.com     | Feminino         | 2                  | 30/06/1943 | NaN        | NaN        | NaN         |
| 2          | 3             | Glenden   | Ishchenko    | gishchenko2@moonfruit.com     | Masculino        | 5                  | 09/04/1989 | NaN        | NaN        | NaN         |
| 3          | 4             | Baron     | Jedrzejewsky | bjedrzejewsky3@errecht24.de   | Masculino        | 4                  | 17/11/1998 | NaN        | NaN        | NaN         |
| 4          | 5             | Sheree    | Bredbury     | sbredbury4@sitemeter.com      | Feminino         | 5                  | 08/09/1975 | NaN        | NaN        | NaN         |
| ...        | ...           | ...       | ...          | ...                           | ...              | ...                | ...        | ...        | ...        | ...         |
| 39499      | 39500         | Brandy    | Malhotra     | brandy0@adventure-works.com   | Masculino        | 3                  | 27/07/1980 | NaN        | NaN        | NaN         |
| 39500      | 39501         | Alicia    | Raje         | alicia12@adventure-works.com  | Feminino         | 5                  | 10/05/2001 | NaN        | NaN        | NaN         |
| 39501      | 39502         | Connie    | Rai          | connie4@adventure-works.com   | Feminino         | 0                  | 13/02/1997 | NaN        | NaN        | NaN         |
| 39502      | 39503         | Shawn     | Raji         | shawn23@adventure-works.com   | Masculino        | 1                  | 29/03/2001 | NaN        | NaN        | NaN         |
| 39503      | 39504         | Lindsey   | Sharma       | lindsey10@adventure-works.com | Feminino         | 0                  | 19/11/1961 | NaN        | NaN        | NaN         |



```
clientes_df = clientes_df[['ID Cliente', 'E-mail']]
```

Vamos considerar que no nosso exemplo, usaremos essa mesma lógica para cada uma das 4 bases que foram importadas.

No entanto, trabalhar com 3 dataframes, pode não ser a melhor opção em alguns casos. Portanto, vamos juntar esses dataframes em um único dataframe utilizando o método abaixo:

.merge()



## ATENÇÃO !

O método .MERGE() necessita que colunas IGUAIS possuam TÍTULOS IGUAIS, caso contrário serão consideradas colunas distintas. Caso seja necessário renomear uma coluna usaremos o método .RENAME()

```
clientes_df = clientes_df[['ID Cliente', 'E-mail']]
produtos_df = produtos_df[['ID Produto', 'Nome do Produto']]
lojas_df = lojas_df[['ID Loja', 'Nome da Loja']]
```

ON indica em qual dataframe produtos\_df será “agregado”

#juntando os dataframes

```
vendas_df = vendas_df.merge(produtos_df, on='ID Produto')
vendas_df = vendas_df.merge(lojas_df, on='ID Loja')
vendas_df = vendas_df.merge(clientes_df, on='ID Cliente')
```

Novo dataframe criado para receber a união

Juntando os dataframes em um novo dataframe vendas\_df

|   | Numero da Venda | Data da Venda | Data do Envio | ID Canal | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida | Quantidade Devolvida | Nome do Produto                                   | Nome da Loja               | E-mail                 |
|---|-----------------|---------------|---------------|----------|---------|------------|-------------|------------|--------------------|----------------------|---|----------------------------|------------------------|
| 0 | 1               | 01/01/2017    | 02/01/2017    | 1        | 86      | 981        | 2           | 6825       | 9                  | 1                    | A. Datum Advanced Digital Camera M300 Pink        | Loja Contoso Austin        | rbrumfieldmy@ameblo.jp |
| 1 | 880458          | 23/11/2017    | 23/11/2017    | 0        | 306     | 235        | 10          | 6825       | 8                  | 0                    | Litware Home Theater System 7.1 Channel M710 B... | Loja Contoso Europe Online | rbrumfieldmy@ameblo.jp |
| 2 | 191019          | 20/03/2017    | 21/03/2017    | 1        | 172     | 376        | 2           | 6825       | 9                  | 0                    | Adventure Works Laptop12 M1201 Silver             | Loja Contoso Hartford      | rbrumfieldmy@ameblo.jp |

Agora que temos nosso dataframe apenas com as informações que queremos, vamos começar nossa análise.

Vamos dizer que nossa análise se baseia em saber quais dos clientes mais vezes comprou. Além da contagem vamos aproveitar e fazer um gráfico que nos permita analisar visualmente esse tema.

Para a contagem de vezes usaremos o método:

.value\_counts()

Já para o gráfico usaremos o método :

.plot()

Perceba que apenas esses métodos não são o suficiente para uma análise boa ... O gráfico ainda está estranho, o eixo x do gráfico está ilegível...

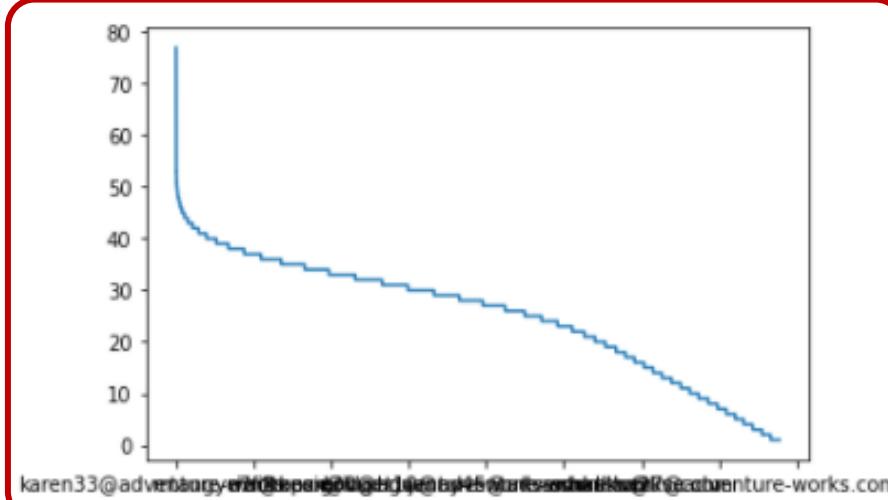
Vamos entender como melhorar esses temas!

```
frequencia_clientes = vendas_df['E-mail do Cliente'].value_counts()  
display(frequencia_clientes)  
frequencia_clientes.plot()
```

|                                 |    |
|---------------------------------|----|
| karen33@adventure-works.com     | 77 |
| chloe77@adventure-works.com     | 74 |
| julia43@adventure-works.com     | 69 |
| karen16@adventure-works.com     | 68 |
| gilbert9@adventure-works.com    | 68 |
| ...                             | .. |
| charles33@adventure-works.com   | 1  |
| marco9@adventure-works.com      | 1  |
| irma0@adventure-works.com       | 1  |
| alexandra36@adventure-works.com | 1  |
| mpaxfordeoz@netlog.com          | 1  |

Name: E-mail do Cliente, Length: 58907, dtype: int64

<matplotlib.axes.\_subplots.AxesSubplot at 0x2a02ce61880>



Value\_counts está agrupando as linhas baseado na coluna "E-mail do Cliente"

Gráfico plotado mas com visualização nada amigável

Para melhorarmos a análise, primeiro, vamos olhar para os dados que importam. Perceba que ao usarmos o `.value_counts()` criamos um rank de emails do maior para o menor. Ou seja, temos uma lista rankeada onde a posição [0] corresponde ao maior, [1] ao segundo maior, etc....

Se usarmos isso juntamente ao método `.plot()` podemos plotar o gráfico apenas dos clientes mais relevantes.

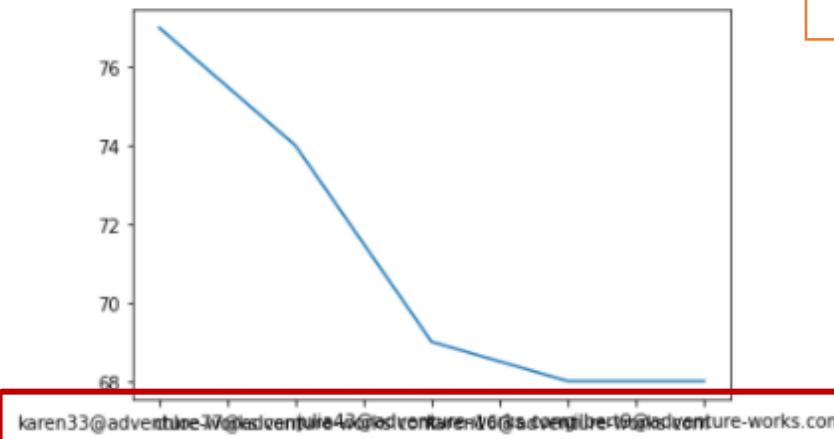
Ao lado fizemos a etapa citada acima. Perceba como a informação é um pouco melhor, mas o gráfico, ainda não é o ideal. O eixo x ainda apresenta valores sobrepostos o que não nos permite entender exatamente o que está acontecendo.

```
frequencia_clientes = vendas_df['E-mail do Cliente'].value_counts()
display(frequencia_clientes)
frequencia_clientes.plot()
```

```
karen33@adventure-works.com      77
chloe77@adventure-works.com      74
julia43@adventure-works.com      69
karen16@adventure-works.com      68
gilbert9@adventure-works.com     68
...
Name: E-mail do Cliente, Length: 38907, dtype: int64
```

```
charles33@adventure-works.com    1
marco9@adventure-works.com       1
irma0@adventure-works.com       1
alexandra36@adventure-works.com 1
mpaxfordeoz@netlog.com          1
Name: E-mail do Cliente, Length: 38907, dtype: int64
```

```
: frequencia_clientes = vendas_df['E-mail do Cliente'].value_counts()
: display(frequencia_clientes)
: frequencia_clientes[:5].plot() ←
: <matplotlib.axes._subplots.AxesSubplot at 0x2a02cecbb30>
```



Clientes mais relevantes: maior frequência

Clientes menos relevantes :com frequência baixa

Usando [:5] conseguimos plotar apenas os 5 primeiros clientes que mais compraram.

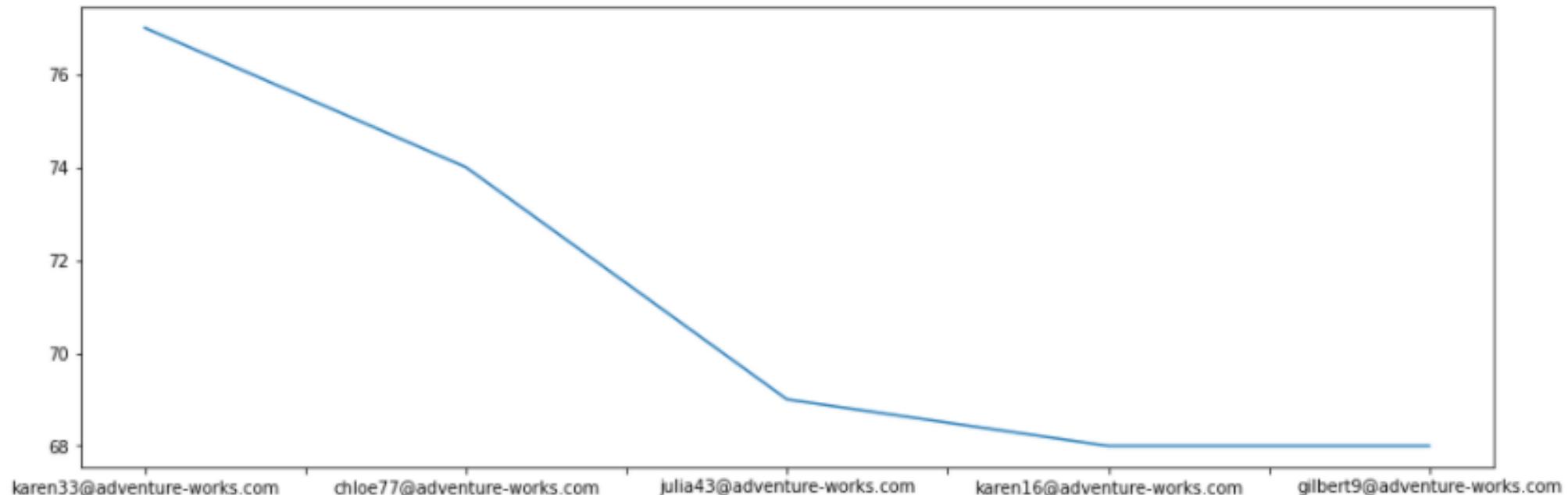
Eixo x ainda confuso...

Vamos agora melhorar a visualização do nosso gráfico usando alguns dos argumentos disponíveis dentro do método `.plot()`.

No nosso caso, usaremos o argumento `figsize` que requer uma tupla de valores (largura,altura). Isso permitirá estendermos o gráfico e termos uma melhror visualização do que ocorre no eixo X

```
frequencia_clientes = vendas_df['E-mail do Cliente'].value_counts()
frequencia_clientes[:5].plot(figsize=(15, 5))  
<matplotlib.axes._subplots.AxesSubplot at 0x2a02cf488e0>
```

Figsize sendo utilizado para aumentar a largura do gráfico apresentado.



Vamos analisar agora as lojas que apresentaram a melhor performance em vendas.

A lógica é muito parecida com o anterior, no entanto, antes contamos quantas vezes o e-mail aparecia para sabermos quem mais comprou.

Agora, vamos precisar somar todos as vendas das lojas.

Para isso, usaremos outros 2 métodos `.GROUPBY()` e `.SUM()` conforme apresentado ao lado:

- `.GROUPBY()` -> Agrupa os dados iguais da coluna indicada ([doc](#))
- `.SUM()` -> Soma todos os dados indicados ([doc](#))

Perceba que utilizamos o `groupby` antes do `sum` para que o Python entenda que precisa somar baseado no agrupamento

```
vendas_lojas = vendas_df.groupby('Nome da Loja').sum()  
display(vendas_lojas)
```

| Nome da Loja            | Numero da Venda | ID Canal | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida | Quantidade Devolvida |
|-------------------------|-----------------|----------|---------|------------|-------------|------------|--------------------|----------------------|
| Loja Contoso Albany     | 1202431493      | 5537     | 279756  | 2256167    | 5099        | 48257399   | 26353              | 298                  |
| Loja Contoso Alexandria | 1180450098      | 5336     | 449696  | 2254764    | 5041        | 48586319   | 26247              | 326                  |
| Loja Contoso Amsterdam  | 1463641203      | 6295     | 721440  | 2855945    | 21345       | 58902735   | 28294              | 401                  |
| Loja Contoso Anchorage  | 1214566953      | 5441     | 458964  | 2328863    | 5252        | 49081995   | 27451              | 361                  |
| Loja Contoso Annapolis  | 1196644689      | 5401     | 466995  | 2243342    | 5131        | 47511944   | 26065              | 300                  |
| ...                     | ...             | ...      | ...     | ...        | ...         | ...        | ...                | ...                  |
| Loja Contoso Yokohama   | 1277269788      | 5423     | 681377  | 2331010    | 10524       | 50296371   | 28023              | 367                  |
| Loja Contoso York       | 1447746885      | 6402     | 600372  | 2736443    | 20990       | 58801825   | 27164              | 365                  |
| Loja Contoso klon No.1  | 1473231489      | 6712     | 699944  | 2802822    | 21812       | 60117285   | 29046              | 356                  |
| Loja Contoso klon No.2  | 1495673435      | 6758     | 699430  | 2846553    | 21777       | 60060540   | 29650              | 417                  |
| Loja Contoso obamberg   | 1443837839      | 6544     | 709240  | 2756377    | 21049       | 59153076   | 29336              | 378                  |

306 rows × 8 columns

Colunas somadas mas que não agregam valor

Colunas somadas corretamente

Apenas as colunas “somáveis” foram apresentadas. Colunas formatadas como TEXTO ou DATA por exemplo não foram exibidas

Como vimos no slide anterior, ao somarmos nosso dataframe vendas\_df agrupando por nome de Loja, algumas informações foram descartadas pois não eram “somáveis” visto o formato e outras foram somadas, mas gerando uma informação sem sentido  
Ex: ID LOJA.

Perceba que as informações descartadas não foram perdidas visto que criamos uma variável vendas\_lojas que recebeu essa extração do dataframe original.

Visto que não perdemos informação, podemos descartar também as informações que não agregam valor, ficando apenas com “Quantidade Vendida”, conforme o print ao lado. Para isso, vamos filtrar nosso df vendas\_lojas usando a estrutura apresentada ao lado.

```
: vendas_lojas = vendas_df.groupby('Nome da Loja').sum()  
vendas_lojas = vendas_lojas[['Quantidade Vendida']]  
display(vendas_lojas)
```

| Nome da Loja            | Quantidade Vendida |
|-------------------------|--------------------|
| Loja Contoso Albany     | 26353              |
| Loja Contoso Alexandria | 26247              |
| Loja Contoso Amsterdam  | 28294              |
| Loja Contoso Anchorage  | 27451              |
| Loja Contoso Annapolis  | 28065              |
| ...                     | ...                |
| Loja Contoso Yokohama   | 28023              |
| Loja Contoso York       | 27164              |
| Loja Contoso koln No.1  | 29046              |
| Loja Contoso koln No.2  | 29650              |
| Loja Contoso obamberg   | 29338              |

306 rows × 1 columns

Filtrando apenas a coluna “Quantidade Vendida”

vendas\_lojas após tratamento dos dados

Soma de todas as vendas da loja Contoso Koln Nº 1

Agora já temos uma tabela bem mais interessante e intuitiva, mas podemos melhorar ainda mais por exemplo colocando-a em ordem decrescente de vendas.

Para isso, usaremos o método `.sort_values()`.

**ATENÇÃO** este `sort_values` que usaremos não é o mesmo que vimos até aqui. Este `sort_values`, é o sort do PANDAS e apesar de o resultado ser o mesmo, possui argumentos distintos. Em caso de dúvidas, consulte a [documentação](#).

Caso o argumento `ascending` não seja fornecido, será considerado a ordem **crescente**. Como não é o que queremos, vamos inserí-lo após a coluna que desejamos ranquear usando a sintaxe:

`ascending = False`

```
#ordenando o dataframe
vendas_lojas = vendas_lojas.sort_values('Quantidade Vendida', ascending = False)
display(vendas_lojas)
```

| Nome da Loja                        | Quantidade Vendida |
|-------------------------------------|--------------------|
| Loja Contoso Catalog                | 1029117            |
| Loja Contoso North America Online   | 701961             |
| Loja Contoso Europe Online          | 616845             |
| Loja Contoso Asia Online            | 578458             |
| Loja Contoso North America Reseller | 520176             |
| ...                                 | ...                |
| Loja Contoso Berlin                 | 379                |
| Loja Contoso Paterson               | 371                |
| Loja Contoso Marseille              | 370                |
| Loja Contoso Racine No.2            | 253                |
| Loja Contoso Venezia                | 234                |

306 rows × 1 columns

Ascending =false, indica que será realizado do maior para o menor

Já temos nossa tabela apresentada de forma bem intuitiva, agora vamos usar essa informação para gerar um gráfico com essas informações.

Novamente, usaremos o método `.plot()` ([doc](#))

Como temos uma tabela já ordenada, podemos usar a estrutura `[:5]` por exemplo, para pegarmos apenas o TOP 5 de lojas em vendas. Se nossa intenção fosse pegar as últimas 5 lojas, por exemplo, poderíamos usar `[-5:]`

Outra possibilidade, é usar o argumento `kind='bar'` para plotarmos o gráfico no modelo de barras verticais.

```
#ordenando o dataframe
vendas_lojas = vendas_lojas.sort_values('Quantidade Vendida', ascending = False)
```

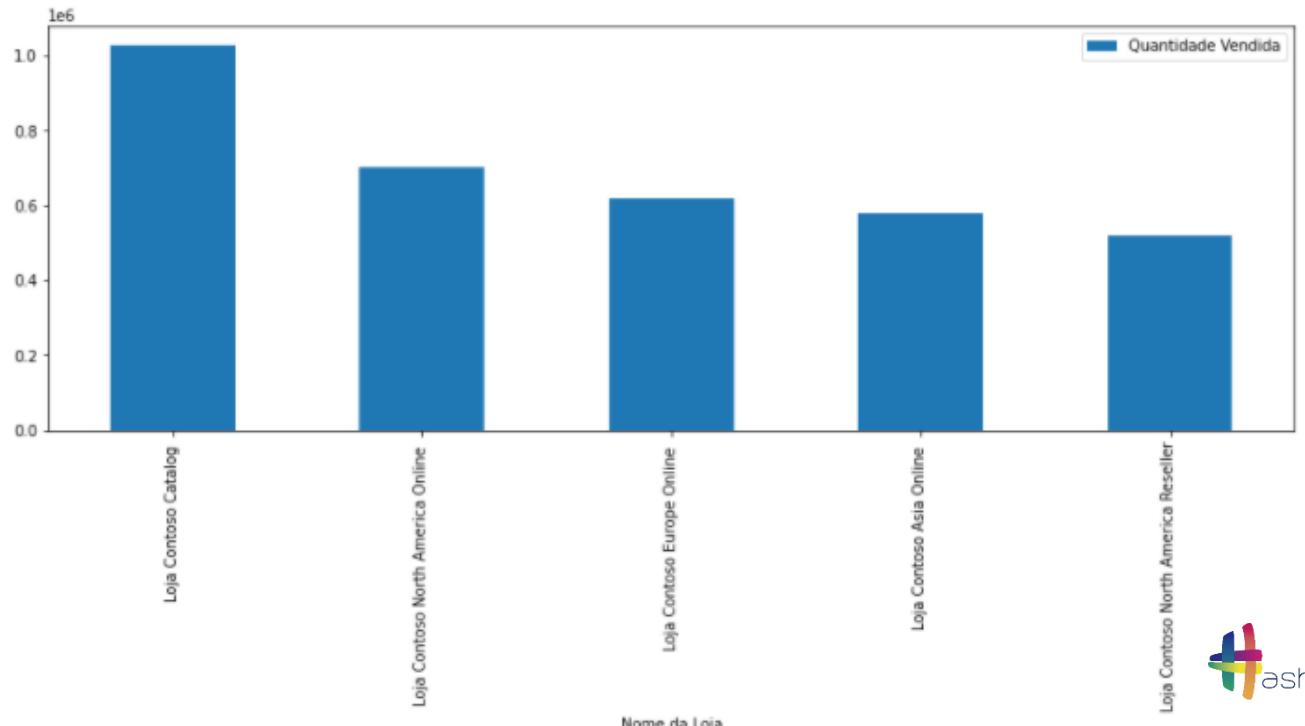
#podemos plotar em um gráfico

```
vendas_lojas[:5].plot(figsize=(15, 5), kind='bar')
```

Indica o tipo do gráfico

Indica o colunas a serem plotadas

Indica o tamanho do gráfico

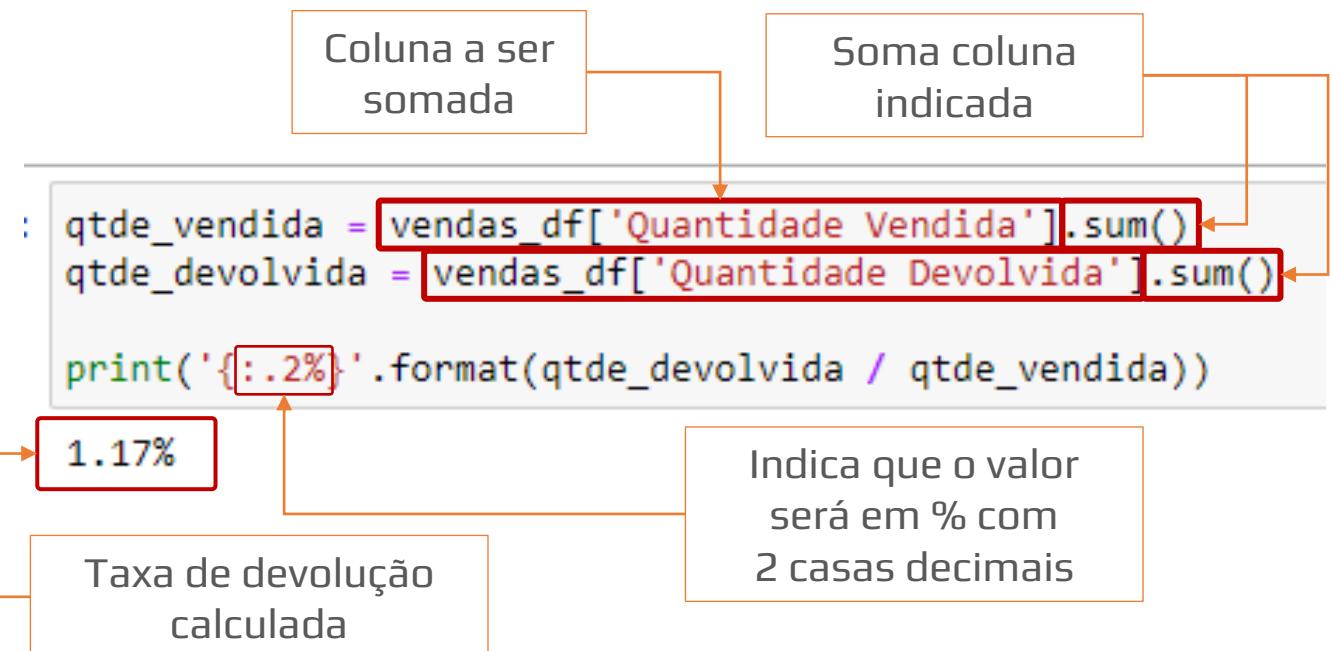


Algo bastante comum nas análises, é precisarmos filtrar as informações baseado em alguns critérios.

Vamos voltar a usar nosso dataframe original `vendas_df` para aprendermos como filtrar informações utilizando o pandas.

Primeiro, vamos avaliar qual a taxa de devoluções que temos na empresa CONTOSO. Ela será nossa referência para sabermos quais lojas estão com taxa de devolução superior. Para isso, podemos calcular a soma da coluna QUANTIDADE DEVOLVIDA sobre a soma da coluna QUANTIDADE VENDIDA, conforme apresentado no print ao lado.

Vamos aproveitar para usar o `format` que já conhecemos de módulos anteriores. ☺



Agora que temos nossa taxa de todo o dataframe, vamos calcular de uma loja específica.

Como temos uma coluna IDLOJA, vamos usá-la para o filtro. Além disso, vamos criar uma variável específica para armazenamento dos dados dessa loja VENDAS\_LOJASCONSOTOEUROPEONLINE.

Nova variável para armazenamento das informações filtradas

```
vendas_lojacontosoeuropeonline = vendas_df[vendas_df['ID Loja'] == 306]
display(vendas_lojacontosoeuropeonline)
qtde_vendida = vendas_lojacontosoeuropeonline['Quantidade Vendida'].sum()
qtde_devolvida = vendas_lojacontosoeuropeonline['Quantidade Devolvida'].sum()
print('{:.2%}'.format(qtde_devolvida / qtde_vendida))
```

Filtrando pelo ID 306 o df  
vendas\_df

| Numero da Venda | Data da Venda | Data do Envio | ID Canal   | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida | Quantidade Devolvida | Nome do Produto | Nome da Loja                                      | E-mail do Cliente                                       |
|-----------------|---------------|---------------|------------|---------|------------|-------------|------------|--------------------|----------------------|-----------------|---|---|
| 1               | 880458        | 23/11/2017    | 23/11/2017 | 0       | 306        | 235         | 10         | 6825               | 8                    | 0               | Litware Home Theater System 7.1 Channel M710 B... | Loja Contoso Europe Online rbrumfieldmy@ameblo.jp       |
| 17              | 614980        | 18/08/2017    | 18/08/2017 | 0       | 306        | 1621        | 9          | 21344              | 4                    | 0               | Contoso DVD Movies E100 Yellow                    | Loja Contoso Europe Online makayla3@adventure-works.com |
| 18              | 786402        | 21/10/2017    | 25/10/2017 | 4       | 306        | 226         | 10         | 21344              | 8                    | 0               | Litware Home Theater System 2.1 Channel E210 B... | Loja Contoso Europe Online makayla3@adventure-works.com |

Diminuição do número de linhas do df

42771 rows ← 13 columns

1.33%

Taxa de devolução desta loja

Alcançamos o resultado, mas vamos entender melhor a linha onde filtramos para que a sintaxe faça um pouco mais de sentido.

Primeiro, vamos pegar a parte interna código:

```
vendas_df['ID Loja'] == 306
```

Se executarmos apenas esse código no Python, vamos ter a tabela ao lado como resultado.

Perceba que o que estamos fazendo aqui é olhar linha a linha se o ID LOJA é == 306.

É como se fizéssemos um for e um IF, linha a linha checando se ID Loja == 306.

No próximo slide vamos voltar para nosso código original e entender como essa tabela influencia o resto do código.

```
vendas_lojacontoseuropeonline = vendas_df[vendas_df['ID Loja'] == 306]
display(vendas_lojacontoseuropeonline)
```

| Numero da Venda | Data da Venda | Data do Envio | ID Canal | ID Loja | ID Produto | ID Promocao | ID Cliente |
|-----------------|---------------|---------------|----------|---------|------------|-------------|------------|
| 1 880458        | 23/11/2017    | 23/11/2017    | 0        | 306     | 235        | 10          | 6825       |

Como a linha de índice 1 da coluna ID Loja é == 306, nosso resultado é TRUE

```
loja306 = vendas_df['ID Loja'] == 306
display(loja306)
```

|        |       |
|--------|-------|
| 0      | False |
| 1      | True  |
| 2      | False |
| 3      | False |
| 4      | False |
| ...    |       |
| 980637 | False |
| 980638 | False |
| 980639 | False |
| 980640 | False |
| 980641 | False |

Name: ID Loja, Length: 980642, dtype: bool

Voltando par ao código completo, temos que vendas\_df['ID Loja'] == 306 nos retorna um df que indica se essa condição é atendida ou não.

Para nos auxiliar, vamos guardar essa tabela na df loja306

Ao usarmos vendas\_df [loja306] o que estamos fazendo é aplicando essas tabela a vendas\_df indicando quem fica e quem sai.

Assim, o Python, nos retornará a tabela ao lado já filtrada com apenas os valores que atendem a condição ['ID Loja']==306.

```
#vendas_lojacontosoeuropeonline = vendas_df[vendas_df['ID Loja'] == 306]
loja306 = vendas_df['ID Loja'] == 306
vendas_lojacontosoeuropeonline = vendas_df[loja306]
display(vendas_lojacontosoeuropeonline)
```

| Numero da Venda | Data da Venda | Data do Envio | ID Canal   | ID Loja | ID Produto | ID Promocao | ID Cliente | ID Quantidade Vendida | Quantidade Devolvida | Nome do Produto | Nome da Loja                                      | E-mail do Cliente                                       |
|-----------------|---------------|---------------|------------|---------|------------|-------------|------------|-----------------------|----------------------|-----------------|---|---|
| 1               | 880458        | 23/11/2017    | 23/11/2017 | 0       | 306        | 235         | 10         | 6825                  | 8                    | 0               | Litware Home Theater System 7.1 Channel M710 B... | Loja Contoso Europe Online rbrumfieldmy@ameblo.jp       |
| 17              | 614980        | 18/08/2017    | 18/08/2017 | 0       | 306        | 1621        | 9          | 21344                 | 4                    | 0               | Contoso DVD Movies E100 Yellow                    | Loja Contoso Europe Online makayla3@adventure-works.com |
| 18              | 786402        | 21/10/2017    | 25/10/2017 | 4       | 306        | 226         | 10         | 21344                 | 8                    | 0               | Litware Home Theater System 2.1 Channel E210 B... | Loja Contoso Europe Online makayla3@adventure-works.com |

Podemos também ter situações onde apenas 1 critério não é suficiente para nossa análise.

Vamos dizer que além da loja, também quero saber quais vendas não tiveram devolução, ou seja 'Quantidade Devolvida' == 0.

Aqui usaremos a mesma lógica que usávamos no IF quando tínhamos mais de 1 condição.

Em geral, usávamos AND ou OR dependendo do que queríamos. Aqui não será diferente. Como queremos uma loja específica E Quantidade devolvida=0, usaremos o comparador lógico AND (ou &).

Podemos perceber que o número de linhas antes de 42mil foi reduzido a 35 mil. Isso ocorre pois além da condição loja ==306, temos também devoluções == 0.

Gera uma tabela indicando quando a condição foi atendida (conforme apresentado anteriormente nos slides anteriores)

```
df_loja306semdev = vendas_df[(vendas_df['ID Loja'] == 306) & (vendas_df['Quantidade Devolvida'] == 0)]
display(df_loja306semdev)
```

Compara as 2 tabelas geradas gerando uma nova tabela onde as 2 condições são atendidas

|        |         |                     |
|--------|---------|---------------------|
|        | 0       | False               |
|        | 1       | True                |
|        | 2       | False               |
|        | 3       | False               |
| 980641 |         | False               |
|        | Length: | 980642, dtype: bool |

| Numero da Venda | Data da Venda | Data do Envio | ID Canal | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida | Quantidade Devolvida | Nome do Produto                                   | Nome da Loja                       |
|-----------------|---------------|---------------|----------|---------|------------|-------------|------------|--------------------|----------------------|---|------------------------------------|
| 1 880458        | 23/11/2017    | 23/11/2017    | 0        | 306     | 235        | 10          | 6825       | 8                  | 0                    | Litware Home Theater System 7.1 Channel M710 B... | Loja Contoso Europe Online rbrumfi |
| 17 614980       | 18/08/2017    | 18/08/2017    | 0        | 306     | 1621       | 9           | 21344      | 4                  | 0                    | Contoso DVD Movies E100 Yellow                    | Loja Contoso Europe Online mak     |
| 18 786402       | 21/10/2017    | 25/10/2017    | 4        | 306     | 226        | 10          | 21344      | 8                  | 0                    | Litware Home Theater System 2.1 Channel E210 B... | Loja Contoso Europe Online mak     |

35029 rows x 13 columns

Voltando para nosso dataframe `vendas_df` original, vamos entender como adicionar e modificar colunas do nosso dataframe.

Como podemos ver no print ao lado, apesar de parecem formatadas como datas, ao importar os dados das nossas bases .csv o Python as classificou com TYPE OBJECT e não como DATETIME que é a classificação esperada quando se trabalha com datas.

No próximo slide, vamos entender como realizar esse processo.

|   | Numero da Venda | Data da Venda | Data do Envio | ID Canal | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida | Quantidade Devolvida | Nome do Produto                                   | Nome da Loja               |
|---|-----------------|---------------|---------------|----------|---------|------------|-------------|------------|--------------------|----------------------|---|----------------------------|
| 0 | 1               | 01/01/2017    | 02/01/2017    | 1        | 86      | 981        | 2           | 6825       | 9                  | 1                    | A. Datum Advanced Digital Camera M300 Pink        | Loja Contoso Austin        |
| 1 | 880458          | 23/11/2017    | 23/11/2017    | 0        | 306     | 235        | 10          | 6825       | 8                  | 0                    | Litware Home Theater System 7.1 Channel M710 B... | Loja Contoso Europe Online |

`vendas_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 980642 entries, 0 to 980641
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Numero da Venda    980642 non-null   int64  
 1   Data da Venda      980642 non-null   object 
 2   Data do Envio      980642 non-null   object 
 3   ID Canal           980642 non-null   int64  
 4   ID Loja            980642 non-null   int64  
 5   ID Produto          980642 non-null   int64  
 6   ID Promocao         980642 non-null   int64  
 7   ID Cliente          980642 non-null   int64  
 8   Quantidade Vendida  980642 non-null   int64  
 9   Quantidade Devolvida 980642 non-null   int64  
 10  Nome do Produto     980642 non-null   object 
 11  Nome da Loja        980642 non-null   object 
 12  E-mail do Cliente   980642 non-null   object 
dtypes: int64(8), object(5)
memory usage: 104.7+ MB
```

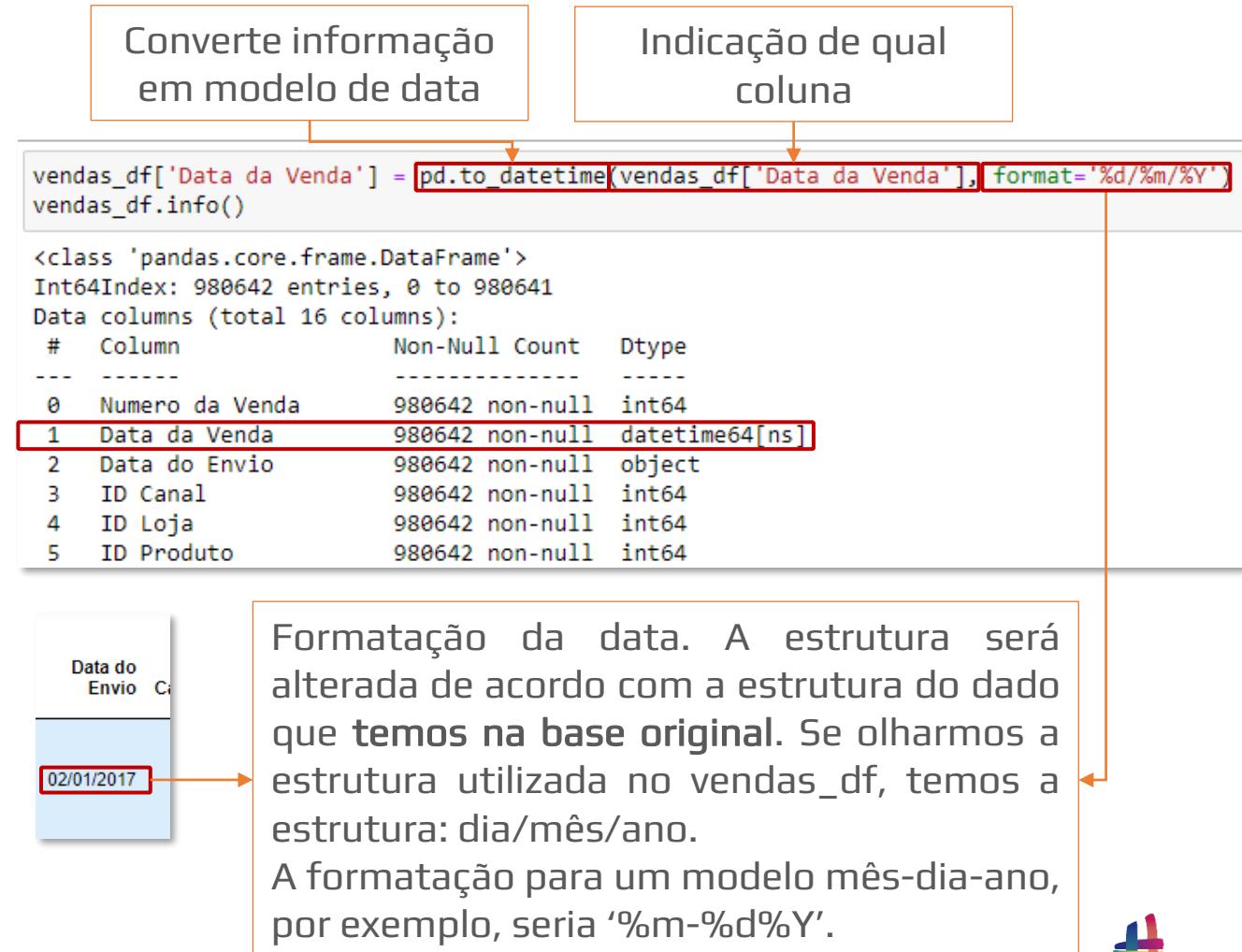
Colunas de data com tipo OBJECT.

Para realizarmos essa transformação do tipo da coluna de objeto para datetime, iremos usar o método:

`pd.to_datetime()`

Este método se utiliza de 2 argumentos. O primeiro a coluna que será modificada e o segundo o formato que será utilizado na transformação.

Perceba que ao realizarmos essa transformação, perceba que a coluna data da Venda que antes estava caracterizada como **object** agora está caracterizada com o **datetime64[ns]**. Essa transformação, apesar de não alterar “esteticamente” o que temos na tabela, nos permite gerar informações que envolvem datas de forma mais simples. E isso, é o que veremos no próximo slide.



Tendo uma coluna formatada como `datetime`, podemos usar métodos específicos deste tipo. Isso nos permite coletar informações de forma simples e direta.

Veja o exemplo ao lado. Aqui utilizamos 2 novos conceitos juntos:

- 1) Adicionar colunas a um dataframe;
- 2) Métodos `.dt` específicos para tipo `datetime`;

Para o primeiro conceito perceba que apenas colocamos o nome da coluna que queremos criar dentro do [ ] de `vendas_df`. A lógica utilizada aqui é a seguinte:

- A coluna existe em `vendas_df`?
  - Se sim, faremos a ação na coluna existente;
  - Se não, criaremos uma nova coluna que será adicionada após da última coluna existente.

Para o segundo conceito, usaremos a estrutura `.dt.XXX` para extraírmos do dado a informação desejada.

```
vendas_df['Data da Venda'] = pd.to_datetime(vendas_df['Data da Venda'], format='%d/%m/%Y')
vendas_df['Ano da Venda'] = vendas_df['Data da Venda'].dt.year
vendas_df['Mes da Venda'] = vendas_df['Data da Venda'].dt.month
vendas_df['Dia da Venda'] = vendas_df['Data da Venda'].dt.day
display(vendas_df)
```

Novas colunas criadas

| Numero da Venda | Data da Venda | Data do Envio | ID Canal | ID Loja | ID Produto | Ano da Venda | Mes da Venda | Dia da Venda |
|-----------------|---------------|---------------|----------|---------|------------|--------------|--------------|--------------|
| 1               | 2017-01-01    | 02/01/2017    | 1        | 86      | 981        | 2017         | 1            | 1            |
| 880458          | 2017-11-23    | 23/11/2017    | 0        | 306     | 235        | 2017         | 11           | 23           |
| 191019          | 2017-03-20    | 21/03/2017    | 1        | 172     | 376        | 2017         | 3            | 20           |
| 18610           | 2017-01-08    | 10/01/2017    | 2        | 200     | 448        | 2017         | 1            | 8            |
| 287704          | 2017-04-23    | 26/04/2017    | 3        | 76      | 280        | 2017         | 4            | 23           |

`.dt.year` -> extrai o ano da coluna 'Data da Venda';

`.dt.month` -> extrai o mês da coluna 'Data da Venda';

`.dt.day` -> extrai o dia da coluna 'Data da Venda';

Utilizando nosso vendas\_df vamos entender como acessar e modificar os valores deste dataframe.

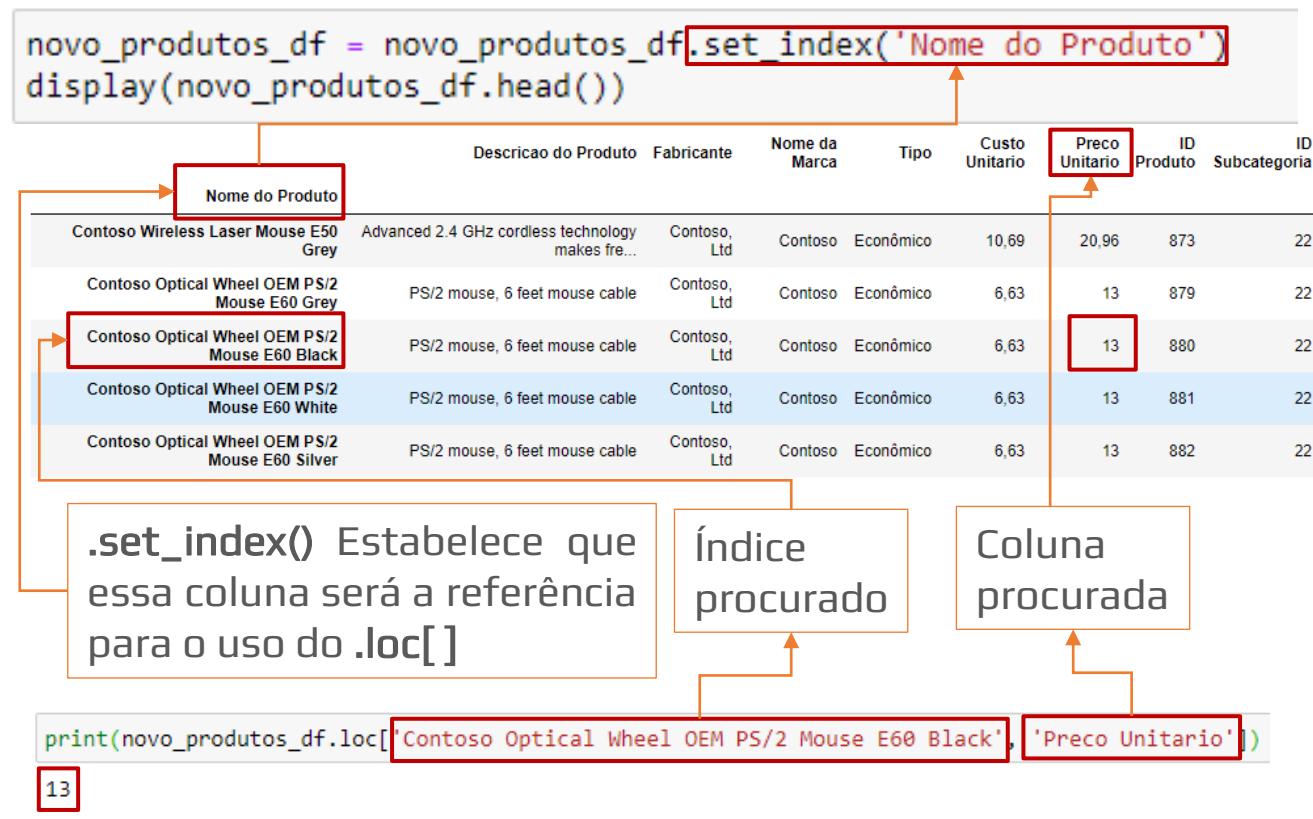
Para acessar esses dados usaremos dois métodos do pandas chamados:

.loc[]  
.iloc[]

Essencialmente eles possuem a mesma função, mas a forma de indicação do dado é distinta.

O [loc](#) pode se utilizar de strings para a “busca”, enquanto o [iloc](#), apenas se utiliza de valores de posição.

O imagem ao lado nos mostra o uso do .LOC. Perceba, que antes de utilizarmos este método, nós utilizamos outro método [.set\\_index\(\)](#) que nos permite indicar qual coluna será utilizada como Índice.



O mesmo processo é utilizado no caso do .ILOC [ ].

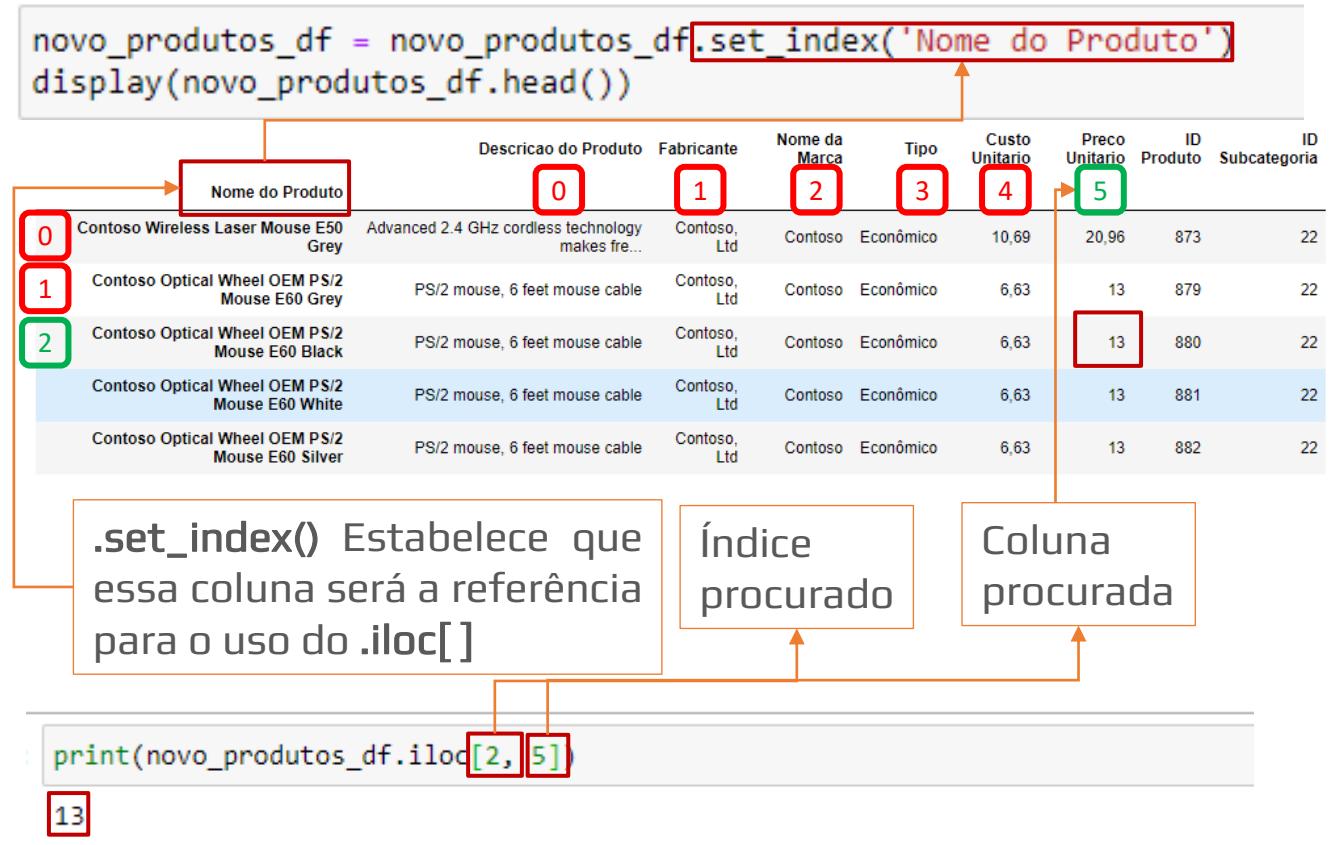
No entanto, nesse caso, não usaremos o “Nome do Produto” como ÍNDICE e sim seu ÍNDICE no Dataframe.



## ATENÇÃO !

Lembre-se que assim como nas listas, os índices se iniciam no valor 0(ZERO). Portanto, se queremos o produto da 3<sup>a</sup> linha usaremos o índice 2.

No caso da coluna usaremos o mesmo princípio, mas vamos considerar que a coluna com ÍNDICE 0 será a seguinte a coluna definida como INDEX pelo método set\_Index. Como Nome do produto é a coluna escolhida, “Descrição do produto” será 0 e “Preço Unitário” a posição 5. Conforme apresentado na imagem ao lado.



Sabendo dessas duas formas vamos entender como modificar um valor do nosso df.

Vamos dizer que o que nos interessa é alterar o preço do produto ID873 ( Contoso Wireless Laser Mouse E50 Grey) para 23.

Na figura ao lado faremos essa alteração utilizando o `.loc[ ]`, visto que sabemos qual o nome exato do produto, mas não sabemos qual a sua posição exata.

| Nome do Produto                                | Descricao do Produto                              | Fabricante   | Nome da Marca | Tipo      | Custo Unitario | Preco Unitario | ID Produto | ID Subcategoria |
|--|---|--------------|---------------|-----------|----------------|----------------|------------|-----------------|
| Contoso Wireless Laser Mouse E50 Grey          | Advanced 2.4 GHz cordless technology makes fre... | Contoso, Ltd | Contoso       | Econômico | 10,69          | 20.96          | 873        | 22              |
| Contoso Optical Wheel OEM PS/2 Mouse E60 Grey  | PS/2 mouse, 6 feet mouse cable                    | Contoso, Ltd | Contoso       | Econômico | 6,63           | 13             | 879        | 22              |
| Contoso Optical Wheel OEM PS/2 Mouse E60 Black | PS/2 mouse, 6 feet mouse cable                    | Contoso, Ltd | Contoso       | Econômico | 6,63           | 13             | 880        | 22              |

### DEPOIS

```
#novo_produtos_df.loc['Contoso Wireless Laser Mouse E50 Grey', 'Preco Unitario'] = 23
novo_produtos_df.loc[novo_produtos_df['ID Produto'] == 873, 'Preco Unitario'] = 23
display(novo_produtos_df.head())
```

Indicação de qual valor será alterado

| Nome do Produto                                | Descricao do Produto                              | Fabricante   | Nome da Marca | Tipo      | Custo Unitario | Preco Unitario | ID Produto | ID Subcategoria |
|--|---|--------------|---------------|-----------|----------------|----------------|------------|-----------------|
| Contoso Wireless Laser Mouse E50 Grey          | Advanced 2.4 GHz cordless technology makes fre... | Contoso, Ltd | Contoso       | Econômico | 10,69          | 23             | 873        | 22              |
| Contoso Optical Wheel OEM PS/2 Mouse E60 Grey  | PS/2 mouse, 6 feet mouse cable                    | Contoso, Ltd | Contoso       | Econômico | 6,63           | 13             | 879        | 22              |
| Contoso Optical Wheel OEM PS/2 Mouse E60 Black | PS/2 mouse, 6 feet mouse cable                    | Contoso, Ltd | Contoso       | Econômico | 6,63           | 13             | 880        | 22              |

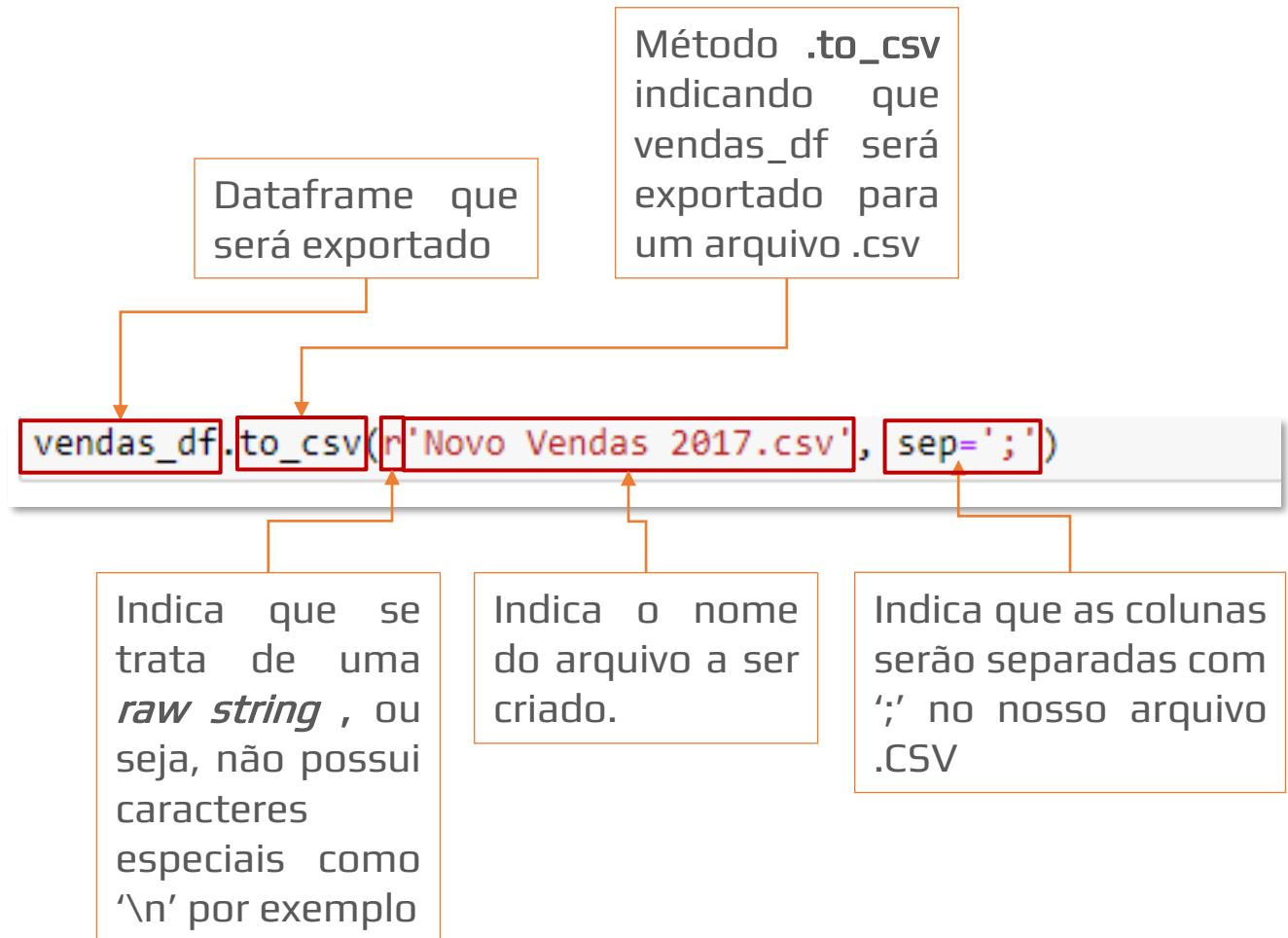
Já aprendemos como buscar arquivos .CSV do nosso computador, trata-los no PANDAS, mas é possível criar um arquivo .CSV com os dados modificados ?

Como quase tudo no Python, SIM, é possível!

Para isso, usaremos o método abaixo:

.to\_csv().

Assim como apresentado ao lado, é bem simples realizar essa exportação. Nesse exemplo específico, estamos criando este arquivo .CSV na mesma pasta onde temos o nosso arquivo do Jupyter.



Vamos imaginar agora que temos um dicionário e que temos interesse em transformá-lo em um arquivo .CSV.

Um caminho possível é transformá-lo em um dataframe e posteriormente através do método `.to_csv` realizarmos a exportação. Para a transformarmos um dicionário em dataframe, usaremos o método `.from_dict()`[doc](#).

Vamos dar uma olhada no código abaixo:

```
vendas_produtos = {'iphone': [558147, 951642], 'galaxy': [712350, 244295], 'ipad': [573823, 26964], 'tv': [405252, 787604], 'máqu
```

Transforma um dicionário em dataframe.

Indica qual o dicionário será convertido

Indica o que serão as CHAVES. Neste caso serão LINHAS.

```
vendas_produtos df = pd.DataFrame.from_dict(vendas_produtos, orient='index')
```

No entanto, apesar de termos transformado nosso dicionário em um dataframe, as colunas ainda não são intuitivas. Sendo apresentadas com os valores 0 e 1.

Vamos trata-las usando um método já conhecido, o `.rename()` e aí sim vamos exportá-lo para um arquivo .CSV.

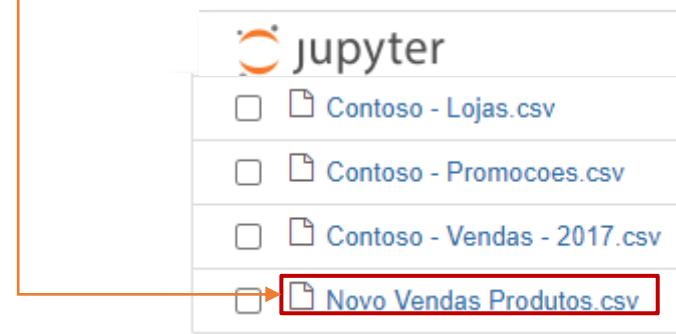
|                 | 0      | 1      |
|-----------------|--------|--------|
| iphone          | 558147 | 951642 |
| galaxy          | 712350 | 244295 |
| ipad            | 573823 | 26964  |
| tv              | 405252 | 787604 |
| máquina de café | 718654 | 867660 |
| kindle          | 531580 | 78830  |
| geladeira       | 973139 | 710331 |
| adegas          | 892292 | 646016 |
| notebook dell   | 422760 | 694913 |
| notebook hp     | 154753 | 539704 |
| notebook asus   | 887061 | 324831 |

Colunas não intuitivas

Valores do dicionários foram distribuídos em 2 colunas

Chaves do dicionário transformadas em linhas do df.

```
vendas_produtos_df = pd.DataFrame.from_dict(vendas_produtos, orient='index')
vendas_produtos_df = vendas_produtos_df.rename(columns={0: 'Vendas 2019', 1: 'Vendas 2020'})
vendas_produtos_df.to_csv(r'Novo Vendas Produtos.csv', sep=',', encoding='latin1')
```



Saindo do nosso próprio PC, vamos aprender como interagir com arquivos .csv existentes na internet.

O primeiro caso é apresentado ao lado. Nesse caso, temos um link que se parece como um link qualquer de internet, mas na realidade ele não abre uma página, mas sim baixa um arquivo. Perceba por exemplo o fim do link onde temos `=download`.

Nesse caso, usaremos o mesmo princípio utilizado anteriormente com o `read_csv()`. A única diferença, que ao invés de utilizarmos um caminho do nosso computador, usaremos um caminho “externo”.

- Criei um arquivo csv e disponibilizei o link para download no Drive: [https://drive.google.com/uc?authuser=0&id=1UzlPy6CZQeAzDXhfc\\_2sHEyK\\_Jb50vJs&export=download](https://drive.google.com/uc?authuser=0&id=1UzlPy6CZQeAzDXhfc_2sHEyK_Jb50vJs&export=download)

Link não associado a uma página, mas sim a um download

```
: import pandas as pd  
  
url = 'https://drive.google.com/uc?authuser=0&id=1UzlPy6CZQeAzDXhfc_2sHEyK_Jb50vJs&export=download'  
cotacao_df = pd.read_csv(url)  
display(cotacao_df)
```

|   | Date         | Price  | Open   | High   | Low    | Change % |
|---|--------------|--------|--------|--------|--------|----------|
| 0 | Jan 01, 2020 | 4.0195 | 4.0195 | 4.0195 | 4.0195 | 0.00%    |
| 1 | Dec 31, 2019 | 4.0195 | 4.0201 | 4.0201 | 4.0195 | 0.00%    |
| 2 | Dec 30, 2019 | 4.0195 | 4.0484 | 4.0484 | 4.0084 | -0.65%   |
| 3 | Dec 27, 2019 | 4.0460 | 4.0568 | 4.0614 | 4.0339 | -0.25%   |
| 4 | Dec 26, 2019 | 4.0560 | 4.0827 | 4.0827 | 4.0453 | -0.68%   |

O segundo caso, o nosso link de download, não é tão direto quanto o anterior.

Vamos usar o link abaixo como exemplo:

[http://portalweb.cooxupe.com.br:8080/portal/precohistoricocafe\\_2.jsp?d-3496238-e=2&6578706f7274=1](http://portalweb.cooxupe.com.br:8080/portal/precohistoricocafe_2.jsp?d-3496238-e=2&6578706f7274=1)

Se usarmos este link teremos um HTTPERROR que essencialmente é um erro que nos informa que não conseguimos estabelecer uma conexão com a página desejada.

Por isso, vamos precisar tratar esse link antes de importá-lo.

Para isso, usaremos novas bibliotecas a IO e REQUEST.

```
import pandas as pd

url = 'http://portalweb.cooxupe.com.br:8080/portal/precohistoricocafe_2.jsp?d-3496238-e=2&6578706f7274=1'
cafe_df = pd.read_csv(url)
display(cafe_df)

-----
HTTPError                                     Traceback (most recent call last)
<ipython-input-3-5e546a3aaca4> in <module>
      2
      3 url = 'http://portalweb.cooxupe.com.br:8080/portal/precohistoricocafe_2.jsp?d-3496238-e=2&6578706f7274=1'
----> 4 cafe_df = pd.read_csv(url)
      5 display(cafe_df)

~\anaconda3\lib\site-packages\pandas\io\parsers.py in parser_f(filepath_or_buffer, sep, delimiter, header, names,
       ecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace,
       pfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_form
```

Com o código ao lado (não vale tanto a pena decorá-lo) e sim entende-lo. Basicamente, sempre que tiver interesse nessa operação poderá usar 90% deste código só alterando o que for particular.

```
: import pandas as pd
import requests
import io
url = 'http://portalweb.cooxupe.com.br:8080/portal/precohistoricocafe_2.jsp?d-3496238-e=2&6578706f7274=1'
conteudo_url = requests.get(url).content
arquivo = io.StringIO(conteudo_url.decode('latin1'))
cafe_df = pd.read_csv(arquivo, sep=r'\t', engine='python')
display(cafe_df)
```

Auxilia na requisição de informação de páginas WEB ([doc](#))

Auxilia na “tradução” das informações da página para o formato desejado ([doc](#))

Busca (.get) o conteúdo(.content) do link url

StringIO decodifica(.decode) o conteúdo gerado na linha anterior

Busca os dados decodificados da linha anterior e cria um arquivo .csv com separador '\t'

Uma dúvida comum principalmente quando estamos trabalhando com bases muito grande é: “Tá demorando, será que travou?” O Pandas nos permite criar uma barra de progresso para acalmarmos nossa ansiedade.

Para criarmos a barra de progresso usaremos a biblioteca `tqdm` ([doc](#)) conforme apresentado ao lado.

Para nosso exemplo, o tamanho da barra de progresso será o tamanho proporcional ao tamanho da coluna ID LOJA do nosso dataframe `vendas_df`.

Além do `tqdm` também usaremos o método `.update` para atualizar esta barra de acordo com seu progresso.

### Importando biblioteca TQDM.

```
from tqdm import tqdm  
  
pbar = tqdm(total=len(vendas_df['ID Loja']), position=0, leave=True)  
  
for i, id_loja in enumerate(vendas_df['ID Loja']):  
    pbar.update()  
    if id_loja == 222:  
        vendas_df.loc[i, 'Quantidade Devolvida'] += 1
```

```
display(vendas_df)
```



69%

| 672295/980642 [00:04<00:02, 149834.34it/s]

### Informações de progresso

```
pbar = tqdm(total=len(vendas_df['ID Loja']), position=0, leave=True)
```

Define o tamanho da barra de progresso. Nesse caso o mesmo tamanho da quantidade de linhas da coluna `ID_Loja`

Argumentos que definem como a barra de progresso será apresentada. Nesse caso em uma única linha

Módulo 19

# Integração Python com Arquivos txt

No módulo anterior vimos Pandas que nos permite importar e trabalhar com dados em txt. No entanto, podemos trabalhar com arquivos txt usando apenas o python sem o pandas.

Para isso, usaremos alguns métodos:

## .open()

Este método ([doc](#)) nos permitirá abrir um arquivo e assim ler, editar, etc.

Dentro do método open temos alguns argumentos possíveis:

- 'r' – Abre o arquivo para leitura;
- 'w' – Abre o arquivo para escrita ;
- 'x' – Abre o arquivo apenas para criação. Se o arquivo existir, o código terá um erro como resultado;
- 'a' – Abre o arquivo para escrita mas continua o texto existente no arquivo.

```
Arquivo Editar Formatar Exibir Ajuda  
Lista de Clientes Hashtag  
Clientes Totais 500  
  
Email,*origemurl  
"fulano1151@gmail.com,origemurl:nan;109574838689;hashtag_yt_org_planilha_zUVEC13ySaA;nan;hashtag_yt_org_miniform  
fulano1173@gmail.com,origemurl:nan  
"fulano1169@gmail.com,origemurl:hashtag_yt_org_planilha_f3lg7JxrzB0;hashtag_yt_org_planilha_m5q0KjgVCgY;hashtag  
fulano1565@gmail.com,origemurl:nan  
fulano1197@gmail.com,origemurl:nan  
fulano1202@gmail.com,origemurl:nan  
"fulano1157@gmail.com,origemurl:nan;respondeai_div_org_semanaexcel;23844712306020127;nan;hashtag_site_org;nan;ha  
  
Método open abrirá o arquivo 'Alunos.txt' para Leitura ('r')  
  
arquivo = open('Alunos.txt', 'r')  
  
linhas = arquivo.readlines()  
print (linhas)  
  
Lê as linhas do arquivo txt colocando-os em uma lista,  
onde cada linha será um item desta lista.  
  
\t -> Espaço  
\n-> ENTER (Pular linha)  
, -> Separador de itens da lista criada pelo readlines()
```

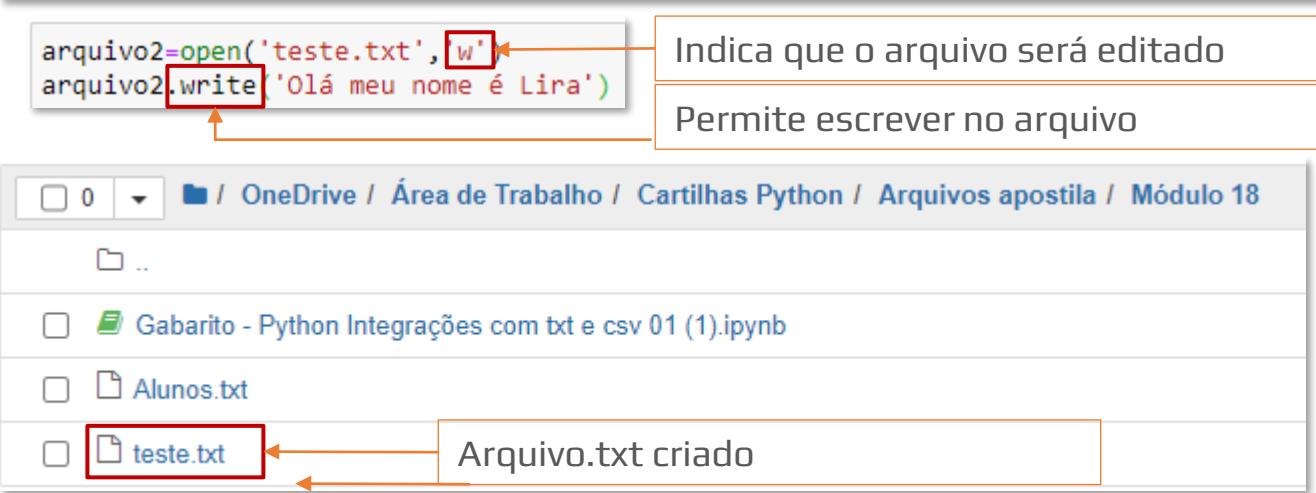
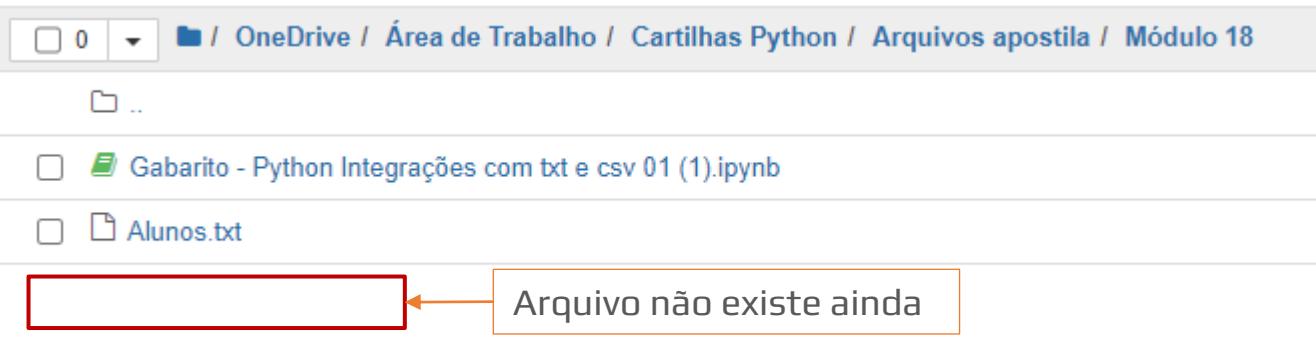
Para editarmos arquivos usaremos o `open()` mas ao invés de usarmos o argumento ‘r’, vamos utilizar o argumento ‘w’. Ele juntamente do método `.write()` nos permite escrever no arquivo.

No nosso exemplo, note que estamos editando um arquivo que não existe por enquanto. Nesse caso, ao invés de editar um novo arquivo será criado.

Ao executarmos o código, podemos perceber que um arquivo teste.txt foi criado.

Ao abrirmos o arquivo nada está escrito. Por que?

O `open` e o `write` trabalham “ocultamente” é como se seu texto não tivesse sido salvado. Para isso vamos precisar usar mais uma função, o `close()`



jupyter teste.txt 2 minutos atrás

File Edit View Language

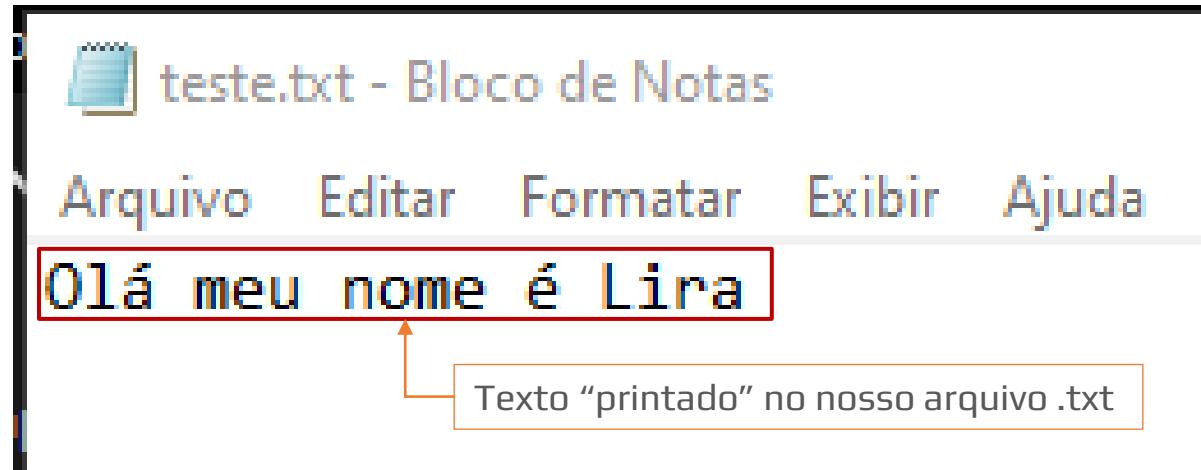
1

Como apenas executamos `open` sem o `close`, nosso arquivo ainda se encontra vazio.

O `close()` nos permitirá salvar o arquivo e deixa-lo disponível para visualização fora do Python.

É importante, sempre que se utilize o `open` que o mesmo seja acompanhado de um `close` ao fim do bloco de código.

```
arquivo2=open('teste.txt','w')
arquivo2.write('Olá meu nome é Lira')
arquivo2.close() ← Fechando o arquivo
```



Outra forma de executarmos a tarefa de abrir e fechar um arquivo .txt é utilizando o WITH.

Ele sem a necessidade de indicar o fechamento(close) do arquivo.

Para isso, usaremos a estrutura apresentada ao lado.

Iremos criar um arquivo chamado “Resumo 2.txt” perceba que o mesmo ainda não existe na pasta. Após a execução do código, o mesmo é gerado.

Se você se lembra bem quando apresentamos o método open, caso não fosse utilizado o close, o arquivo não era “salvo”.

Já com o método with, esse processo é executado.

Cria/edita um arquivo a partir de uma variável

```
with open('Resumo 2.txt', 'w') as arquivo3:  
    arquivo3.write (' Olá, meu nome é Lira\nOutra linha\n')  
    arquivo3.write ('Terceira Linha')
```

Variável arquivo 3 para tratamento do arquivo

Arquivo.txt criado

Arquivo.txt - Bloco de Notas

Módulo 20

# Integração Python - Arquivos e Pastas do Computador

## Módulo 20 – Integração Python - Arquivos e Pastas do Computador

### – Python para Navegar no seu Computador - pathlib e shutil (1/5)

307

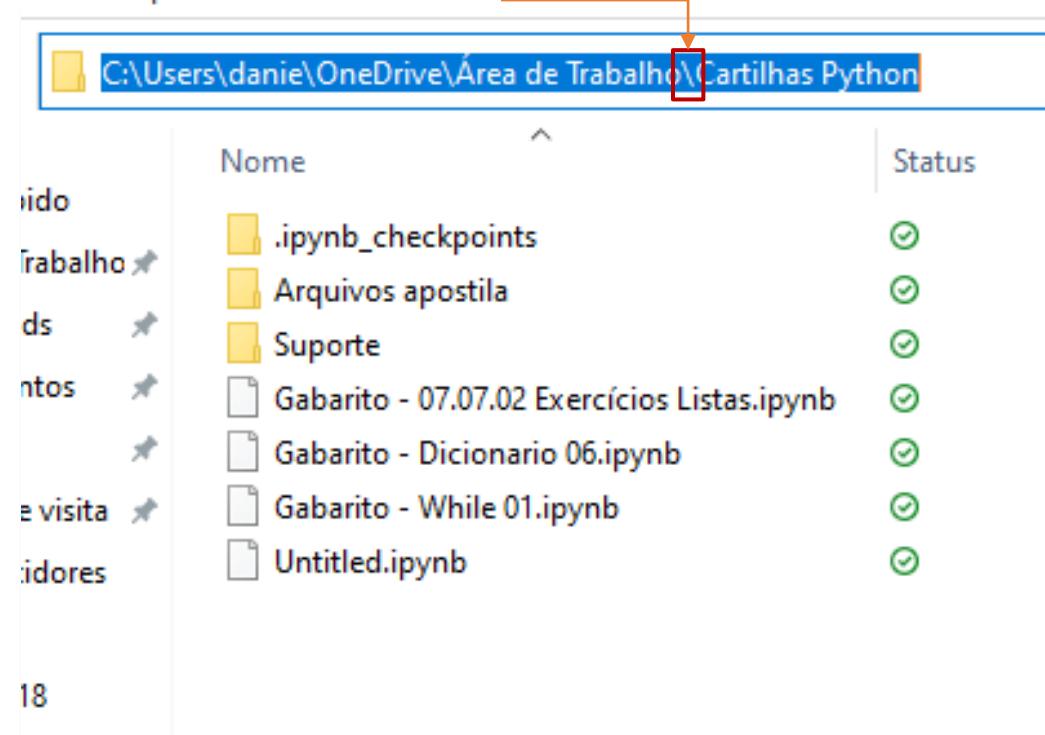
Neste módulo do curso, vamos aprender como usar o Python para executar tarefas com os arquivos do computador.

Para executar essas operações temos 2 módulos possíveis:

- `os` ([doc](#)) ;
- `pathlib` ([doc](#)) ;

Antes de entendermos melhor sobre esses módulos é importante estar atento a um ponto. Nessa apostila, usaremos códigos que se referem ao WINDOWS, pois é o sistema operacional mais utilizado no mercado de trabalho. Caso, seu computador ou do seu trabalho use LINUX ou Apple e você decida usar o módulo `os`, fique atento a pequenos detalhes como a **orientação de uma barra** ou comando que podem influenciar no seu código ☺

Atenção a detalhes como a orientação das barras



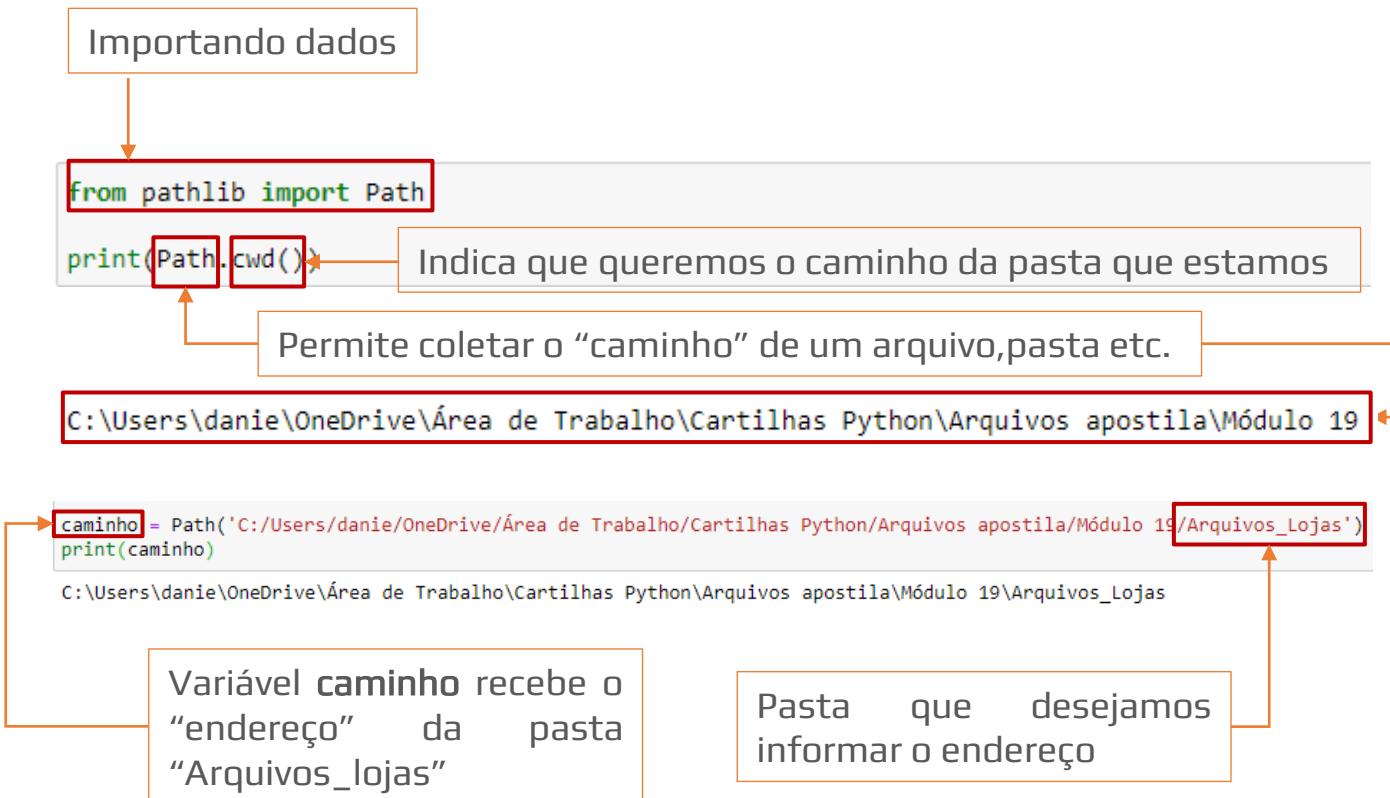
Usaremos agora o **pathlib** que essencialmente desempenha a mesma função do módulo **os**, mas é uma biblioteca mais recente e permite trabalhar com diferentes sistemas operacionais sem se preocupar com isso 😊.

Assim como todos as bibliotecas iniciamos seu uso importando-a no início do código.

Caso a importação não esteja ocorrendo lembre-se do **pip install** para fazer a instalação de pacotes, ou o **pip freeze** para consultar quais programas estão instalados no seu Python.

Após a importação, usaremos o método **.cwd()** nos retornara a pasta que estamos.

É bem comum que este caminho, seja armazenado em variáveis como apresentado no segundo exemplo com a variável **caminho** que recebe a localização da pasta “Arquivos\_Lojas”.



Agora que sabemos o endereço da nossa pasta e o armazenamos na variável **caminho**, vamos acessar seu conteúdo.

Como nessa pasta, temos dezenas de arquivos, vamos utilizar um outro método chamado **iterdir()**.

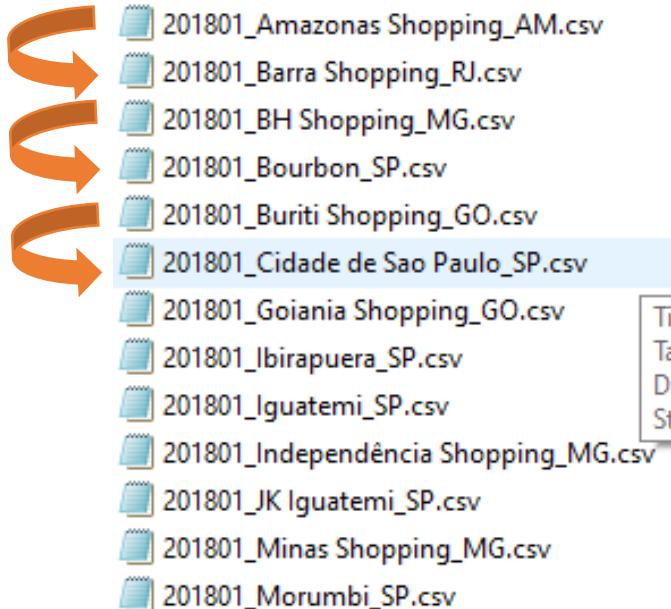
Sua função é funcionar como uma espécie de For dentro da pasta iterando todos os arquivos existentes e gerando uma lista desses arquivos.

O resultado deste método é um objeto que posteriormente usaremos para o tratamento dos dados.

```
caminho = Path('C:/Users/danie/OneDrive/Área de Trabalho')
arquivos = caminho.iterdir()
print(arquivos)
```

<generator object Path.iterdir at 0x00000276785C7190>

Assim como o método **.zip()** que vimos módulos atrás, o **iterdir()** também gera um objeto como resultado.



A partir do objeto gerado pelo `iterdir`, podemos acessar essa lista utilizando o FOR.

Assim como apresentado abaixo:

```
caminho = Path('C:/Users/danie/OneDrive/Área de Trabalho/Cartilhas Python/Arquivos apostila/Módulo 19/Arquivos_Lojas')
arquivos = caminho.iterdir()
for arquivo in arquivos:
    print(arquivo)
```

Acessando cada item da lista criada através do método `.iterdir()`

```
C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Arquivos apostila\Módulo 19\Arquivos_Lojas\201801_Amazonas Shopping_AM.csv
C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Arquivos apostila\Módulo 19\Arquivos_Lojas\201801_Barra Shopping_RJ.csv
C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Arquivos apostila\Módulo 19\Arquivos_Lojas\201801_BH Shopping_MG.csv
```

No entanto, não é tão visual esse caminho com diversas pastas em sequência. Podemos ter o mesmo resultado usando um “caminho relativo”. Nesse caso, ao invés de fornecermos todo o endereço desde do “C:/.....” podemos apenas informar a nossa pasta de interesse. Como no exemplo abaixo:

```
caminho = Path('Arquivos_Lojas')
```

Utilizando o caminho relativo, a visualização é bem melhor e apesar de não afetar em nada o resultado final, facilita a compreensão durante a elaboração do código.

```
from pathlib import Path  
  
caminho = Path('Arquivos Lojas')  
arquivos = caminho.iterdir()  
for arquivo in arquivos:  
    print(arquivo)
```

Arquivos\_Lojas\201801\_Amazonas Shopping\_AM.csv  
Arquivos\_Lojas\201801\_Barra Shopping\_RJ.csv  
Arquivos\_Lojas\201801\_BH Shopping\_MG.csv  
Arquivos\_Lojas\201801\_Bourbon\_SP.csv  
Arquivos\_Lojas\201801\_Buriti Shopping\_GO.csv  
Arquivos\_Lojas\201801\_Cidade de Sao Paulo\_SP.csv  
Arquivos\_Lojas\201801\_Goiania Shopping\_GO.csv  
Arquivos\_Lojas\201801\_Ibirapuera\_SP.csv

Apenas indicação da pasta desejada

Caminho relativo. Considerando a partir da pasta "Arquivos\_lojas"

Usando o `interdir()`, possível percorrer todos os arquivos de uma pasta.

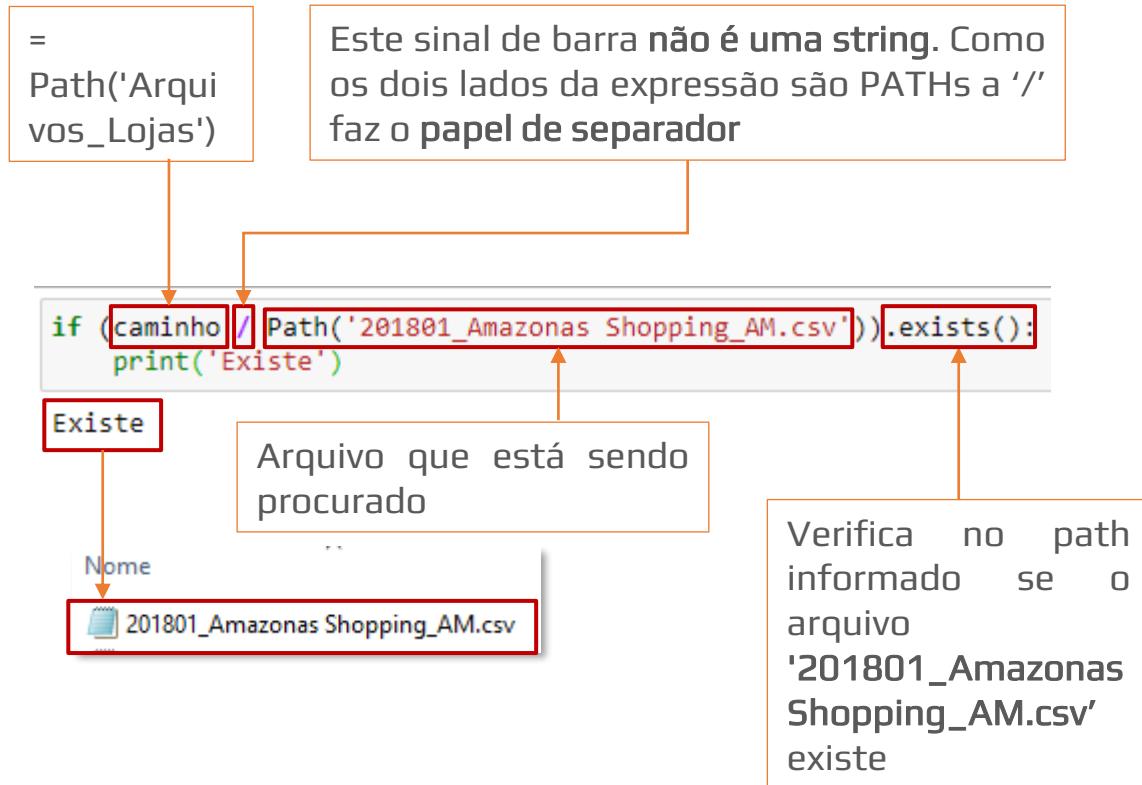
Algumas vezes, não é necessário analisar todas as pastas e sim uma específica.

No exemplo ao lado, criamos uma condição que verifica se uma pasta específica existe. Essa verificação no python é feita através do método:

`.exists()`.

Perceba que além do `exists` introduzimos uma nova forma de informar um `Path`.

Como já havia sido definido a variável `caminho`, podemos uni-la ao nome do arquivo desejado utilizando a estrutura `caminho / nome do arquivo`.



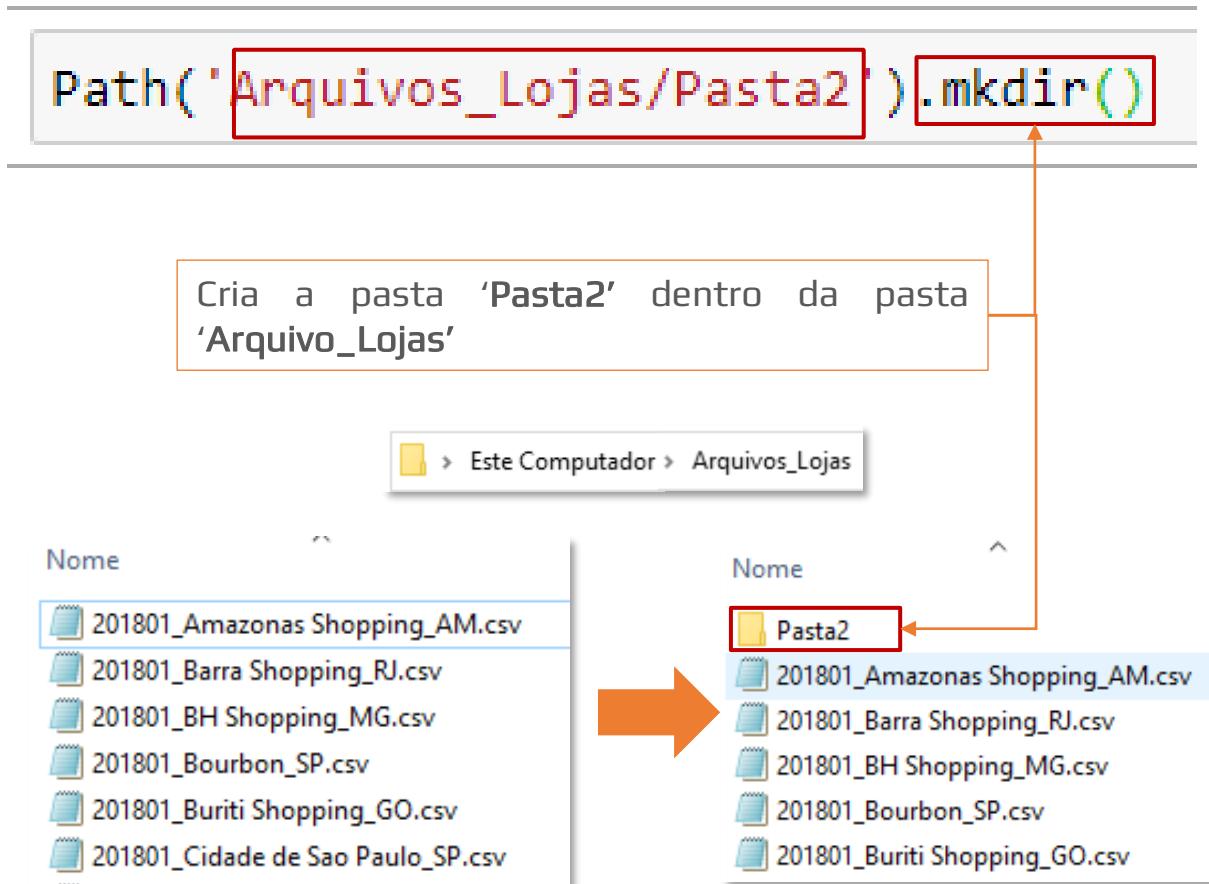
Outra operação que podemos realizar é criar pastas através do Python.

Para isso, usaremos o método abaixo:

.mkdir()

Usando o PATH, podemos fornecer o local e o nome da pasta a ser criada.

Perceba na imagem ao lado, que ao olharmos a pasta `arquivos_lojas` após a execução do código, temos uma nova pasta criada a partir do `mkdir()`.



Vamos considerar agora que queremos copiar o arquivo 'Arquivos\_Lojas/201801\_Amazonas Shopping\_AM.csv' e enviá-lo para a pasta que criamos anteriormente.

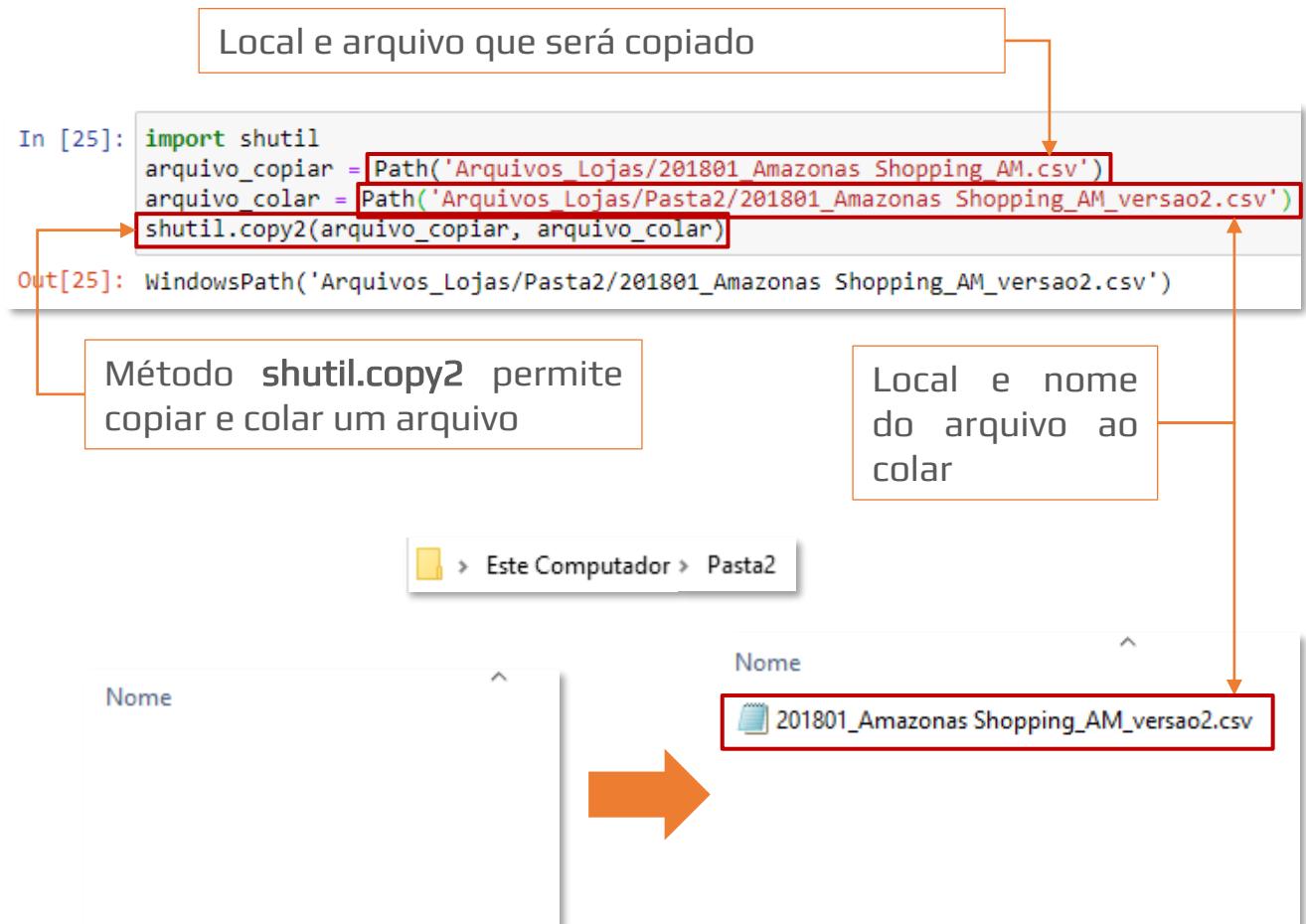
Apesar de ser possível realizar essa operação usando o `pathlib`, iremos usar outra biblioteca:

`shutil` ([doc](#))

O método que usaremos para realizar a operação será o `copy2`([doc](#)), onde:

- O primeiro argumento será o arquivo que deve ser copiado;
- O segundo argumento é o local onde o arquivo será colado e com qual nome.

Perceba que no exemplo ao lado utilizamos 2 variáveis auxiliares para facilitar o desenvolvimento do código e entendimento.



## Módulo 20 – Integração Python - Arquivos e Pastas do Computador

### – Criando pastas e movimentando arquivos (3/3)

315

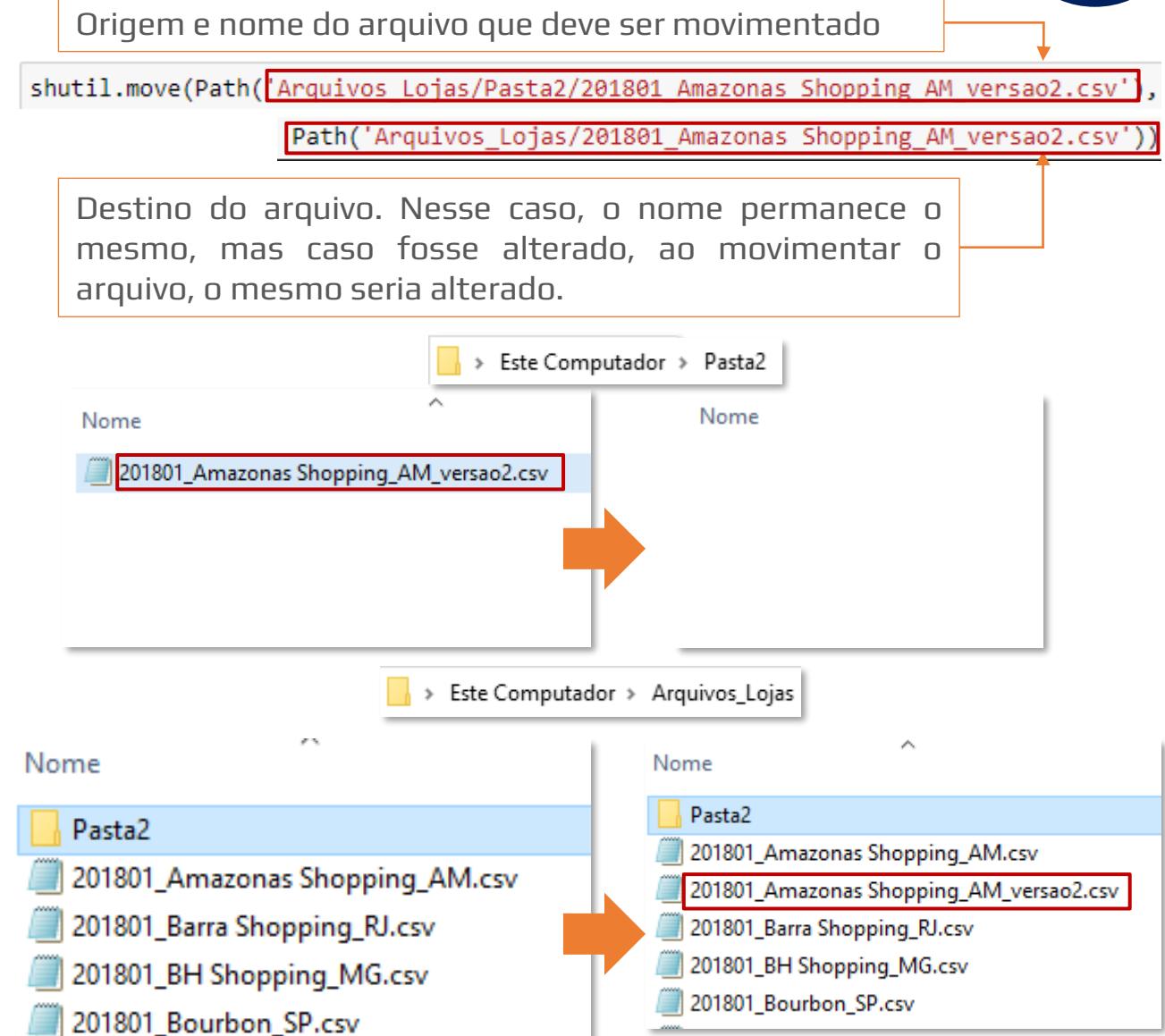
Outra operação muito comum, é movimentar um arquivo entre pastas. Para isso, usaremos mais um método da biblioteca shutil:

.move()

A estrutura é muito parecida com a anterior utilizada no shutil.copy2:

- Primeiro argumento será definido qual arquivo será movimentado;
- Segundo argumento para onde o arquivo será movimentado.

Diferentemente do copy2, podemos perceber que ao usarmos o método .move(), retiramos o arquivo da 'Pasta2' e o movimentamos para a pasta 'Arquivos\_Lojas'



Módulo 21

# Integração Python – E-mail

No curso vamos aprender como integrar o Python com 2 dos principais gerenciadores de e-mail do mercado:

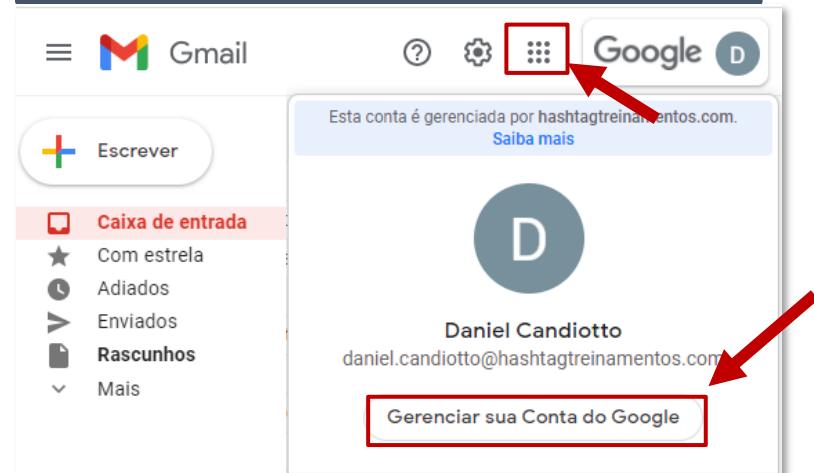
- Gmail;
- Outlook;

Antes de entendermos essas integrações vamos precisar realizar alguns setups prévios para garantir que nosso código irá funcionar.

Iniciando pelo Gmail, vamos precisar configurar nosso e-mail para que seja possível enviar e-mails a partir de “robôs” por definição padrão, o Google bloqueia esse tipo de permissão pois pode gerar uma falha de segurança.

Para realizar esse “desbloqueio”, siga os passos ao lado.

1 Acesse o gmail e clique em “Gerenciar sua Conta do Google” dentro do campo indicado



2 Na aba ‘Segurança’, clique na opção ‘Ativar acesso’



Feito este setup inicial, vamos iniciar nosso código.

Novamente vamos usar uma das bibliotecas disponíveis no Python:

yagmail([doc](#))

Como sempre vamos importar esta biblioteca. Caso a importação não ocorra, você precisará instalar esta biblioteca através do `pip install` dentro do Prompt do Anaconda.

```
import yagmail
```

Anaconda Prompt (anaconda3) - `pip install yagmail`

(base) C:\Users\danie>`pip install yagmail`

Caso você nunca tenha utilizado o yagmail é provável que seja necessária a sua instalação. Para isso, busque por **Anaconda prompt** no menu iniciar e após carregada a janela use o comando `pip install yagmail` e aguarde a instalação

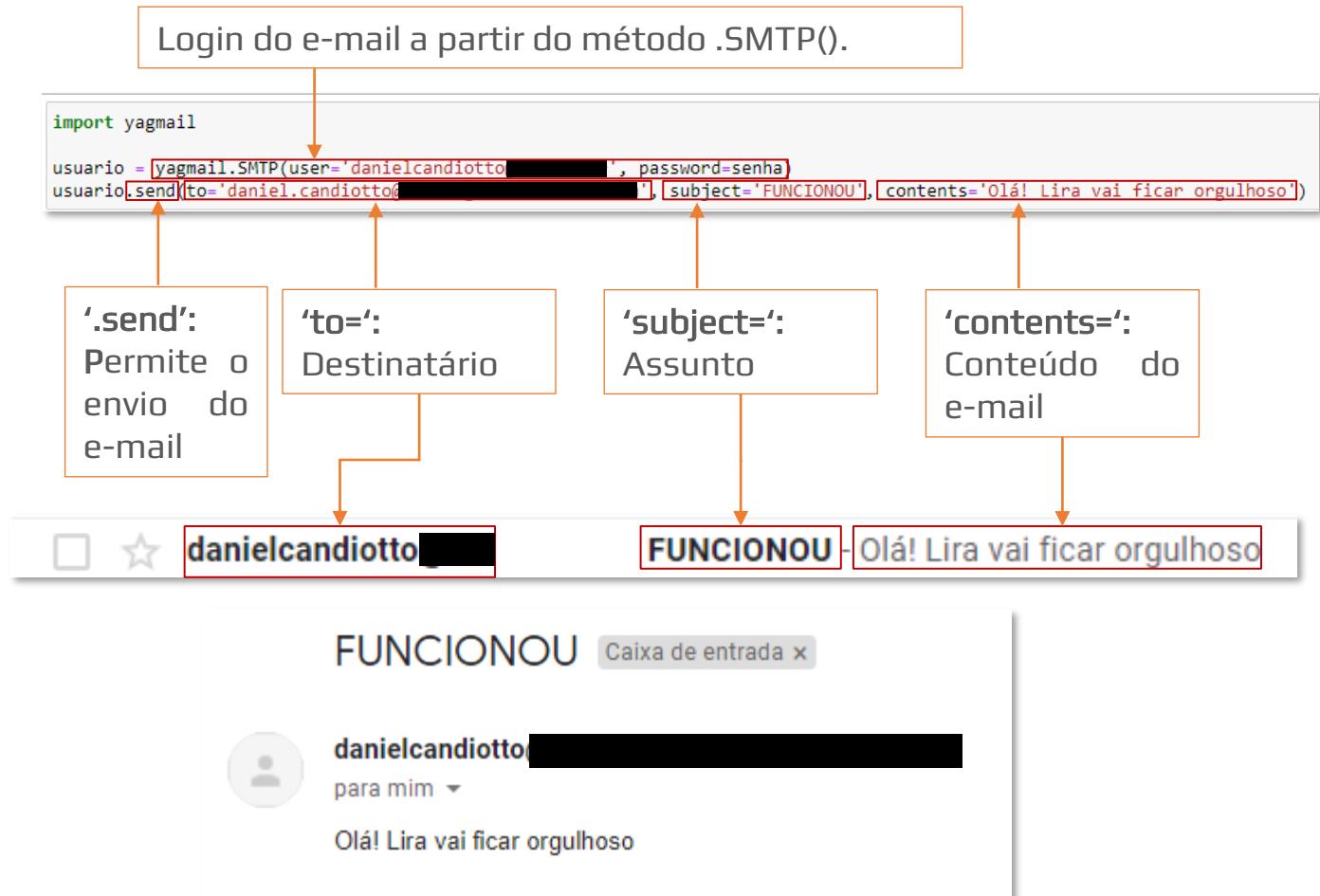
Vamos para o código em si.

O primeiro método da biblioteca `yagmail` que vamos utilizar será o `.SMTP()` que nos permitirá a fazer a identificação do usuário de e-mail (basicamente um login).

O segundo será o método que utilizaremos é o `.send()` que nos permitirá enviar os e-mails.

Vamos dar uma olhada no código ao lado para entender o que entendermos melhor como esses métodos funcionam.

Por questões de segurança, os endereços de e-mail e senha foram ocultados.



Vamos tentar agora além de uma mensagem enviar também um anexo no e-mail.

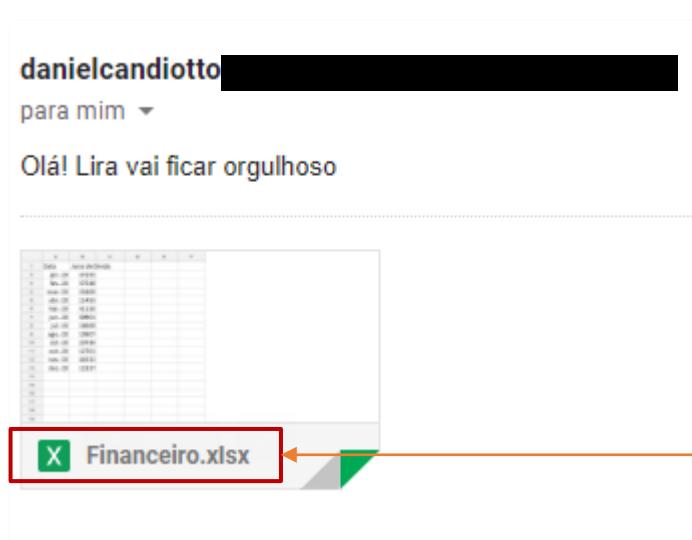
Olhando o exemplo anterior, podemos perceber que temos dentro do método `.send()` alguns argumentos **keywords**. Além dos utilizados anteriormente, podemos ver na documentação ([doc](#)) que temos outros argumentos além dos utilizados.

Um deles é o **attachments** que nos permite anexar documentos ao nosso e-mail.

O exemplo ao lado apresenta o que muda no código.

```
import yagmail

usuario = yagmail.SMTP(user='danielcandiotti@...', password=senha)
usuario.send(to='daniel.candiotti@...',  
            subject='FUNCIONOU',  
            contents='Olá! Lira vai ficar orgulhoso',  
            attachments='Financeiro.xlsx')
```



'**attachments**':  
permite o envio de anexos. Perceba que o código é essencialmente o mesmo do anterior acrescido desta keyword

Vamos agora para algumas variações, onde temos mais de um destinatário, ou mais de um anexo por exemplo.

Como talvez você possa imaginar, ao invés de colocarmos apenas uma string, usaremos uma **lista de strings**, como apresentado ao lado.

```
import yagmail  
  
usuario = yagmail.SMTP(user='danielcandiottotest', password=senha)  
usuario.send(to=['daniel.candiottotest', 'danielcandiottotest'],  
            subject='FUNCIONOU',  
            contents='Olá! Lira vai ficar orgulhoso',  
            attachments=['Financeiro.xlsx','Logística.xlsx'])
```

danielcandiottotest <danielcandiottotest>

para mim, danielcandiotto

Olá! Lira vai ficar orgulhoso

2 anexos

| Dia | Preço de Venda |
|-----|----------------|
| 1   | R\$ 100,00     |
| 2   | R\$ 120,00     |
| 3   | R\$ 140,00     |
| 4   | R\$ 160,00     |
| 5   | R\$ 180,00     |
| 6   | R\$ 200,00     |
| 7   | R\$ 220,00     |
| 8   | R\$ 240,00     |
| 9   | R\$ 260,00     |
| 10  | R\$ 280,00     |
| 11  | R\$ 300,00     |
| 12  | R\$ 320,00     |
| 13  | R\$ 340,00     |
| 14  | R\$ 360,00     |
| 15  | R\$ 380,00     |
| 16  | R\$ 400,00     |
| 17  | R\$ 420,00     |
| 18  | R\$ 440,00     |
| 19  | R\$ 460,00     |
| 20  | R\$ 480,00     |
| 21  | R\$ 500,00     |
| 22  | R\$ 520,00     |
| 23  | R\$ 540,00     |
| 24  | R\$ 560,00     |
| 25  | R\$ 580,00     |
| 26  | R\$ 600,00     |
| 27  | R\$ 620,00     |
| 28  | R\$ 640,00     |
| 29  | R\$ 660,00     |
| 30  | R\$ 680,00     |
| 31  | R\$ 700,00     |

X Financeiro.xlsx

| Data | Análise Infográficos |
|------|----------------------|
| 1    | 100,00               |
| 2    | 120,00               |
| 3    | 140,00               |
| 4    | 160,00               |
| 5    | 180,00               |
| 6    | 200,00               |
| 7    | 220,00               |
| 8    | 240,00               |
| 9    | 260,00               |
| 10   | 280,00               |
| 11   | 300,00               |
| 12   | 320,00               |
| 13   | 340,00               |
| 14   | 360,00               |
| 15   | 380,00               |
| 16   | 400,00               |
| 17   | 420,00               |
| 18   | 440,00               |
| 19   | 460,00               |
| 20   | 480,00               |
| 21   | 500,00               |
| 22   | 520,00               |
| 23   | 540,00               |
| 24   | 560,00               |
| 25   | 580,00               |
| 26   | 600,00               |
| 27   | 620,00               |
| 28   | 640,00               |
| 29   | 660,00               |
| 30   | 680,00               |
| 31   | 700,00               |

X Logística.xlsx

Usando as listas, podemos anexar mais de um arquivo e aumentar o número de destinatários.

Vamos entender melhor como enviar um e-mail de forma mais organizada utilizando o Python.

Existem 2 formas:

- 1) O texto como uma lista de frases, onde cada item da lista é uma frase/palavra ,etc;
- 2) Uso das aspas triplas "" e escrita como um e-mail normal.

Perceba que nos exemplos abaixo apesar de métodos diferentes, o resultado é o mesmo:

```
corpo_email = [  
    'Fala Lira, tranquilo?',  
    'Envio esse e-mail para te passar o relatório do ano passado',  
    'Att,'.  
    'Daniel'  
]  
corpo_email = '\n'.join(corpo_email)  
  
usuario.send(to='daniel.candiotto[REDACTED]', subject='Meu segundo Email no Python', contents = corpo_email)  
  
corpo_email = """  
    Fala Lira, tranquilo?  
    Envio esse e-mail para te passar o relatório do ano passado  
    Att,  
    Daniel  
"""  
  
usuario.send(to='daniel.candiotto[REDACTED]', subject='Meu segundo Email no Python', contents = corpo_email)
```

**CASO 1:** Nesse caso, cada uma das frases é um item da lista `corpo_email`. Perceba que são separadas por vírgulas e delimitadas pelos [ ].

**CASO 2:** Aspas Triplas ("""") delimitando o início e fim do conteúdo do texto.

danielcandiotto[REDACTED] >  
para mim ▾  
  
Fala Lira, tranquilo?  
Envio esse e-mail para te passar o relatório do ano passado  
Att,  
Daniel

Ao invés de usar o Python, vamos utilizar o HTML que é uma outra linguagem que é utilizada nas estruturação de sites por exemplo.

Não é o objetivo aqui explicar a fundo o HTML, mas é importante saber que podemos usá-lo caso necessário.

The diagram illustrates the mapping of an HTML string to an email message. The HTML code is:

```
corpo_texto = """Fala Lira tranquilo?  
Envio esse e-mail para te passar o relatório do ano passado  


Att,  
João

"""
```

Annotations explain the HTML elements:

- <b></b>** indica que o texto entre estes marcadores(Lira) está em negrito
- <p></p>** indica que o texto entre estes marcadores(Att,) é um paragrafo

The resulting email message is shown on the right:

danielcandiotto [REDACTED]  
para mim ▾

Fala Lira tranquilo?

Envio esse e-mail para te passar o relatório do ano passado

Att,

João

Com o Outlook, vamos começar pelo processo de importação, assim como fizemos com o Gmail.

A biblioteca que usaremos, é o `win32com.client`. Geralmente, usamos a versão reduzida desse nome usando apenas `win32`.

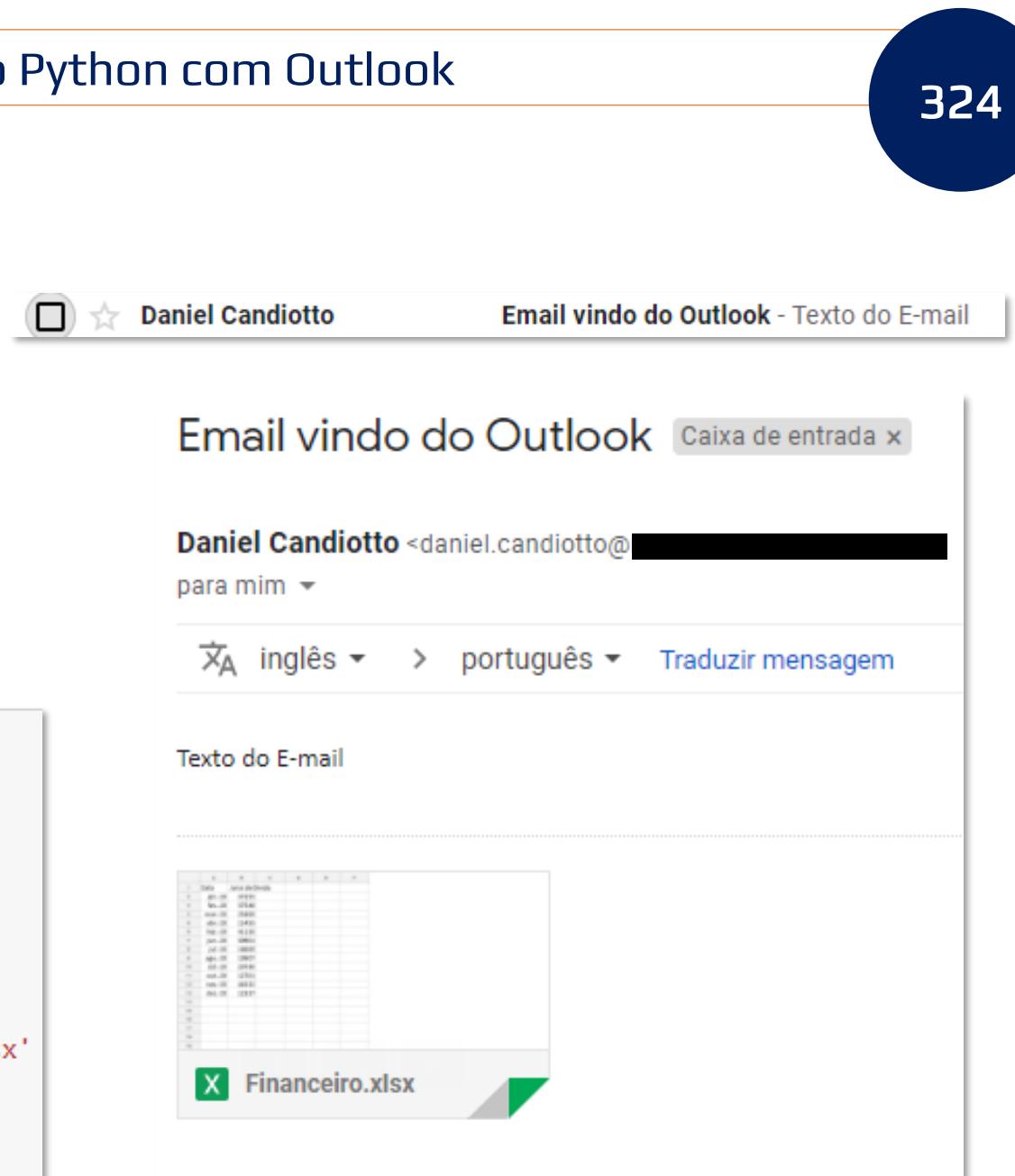
Importante frisar que é necessário possuir o Outlook instalado para que seja possível realizar essa operação.

```
import win32com.client as win32
outlook = win32.Dispatch('outlook.application')

mail = outlook.CreateItem(0)
mail.To = 'daniel.candiotto@hashtagtreinamentos.com'
mail.Subject = 'Email vindo do Outlook'
mail.Body = 'Texto do E-mail'

attachment = r'C:\Users\danie\OneDrive\Área de Trabalho\Financeiro.xlsx'
mail.Attachments.Add(attachment)

mail.Send()
```



Módulo 22

# Integração Python - SQL

Primeiro é importante lembrar que este é um módulo de integração de Python com SQL e não um curso completo de SQL.

Dito isso, vamos para o primeiro passo: A instalação.

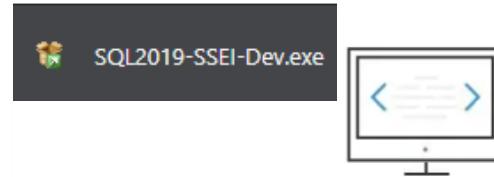
Para isso vamos baixar o SQL Server e a base “Contoso” disponibilizada pela Microsoft.

Para download do SQL Server [clique aqui](#);

Para download da base ‘Contoso’ [clique aqui](#)

Feitos os downloads, vamos iniciar a instalação dos SQL Server no próximo slide.

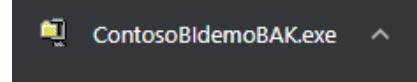
Experimente o SQL Server na infraestrutura local ou na nuvem



Desenvolvedor

O SQL Server 2019 Developer é uma edição gratuita completa, licenciada para uso como banco de dados de desenvolvimento e teste em um ambiente de não produção.

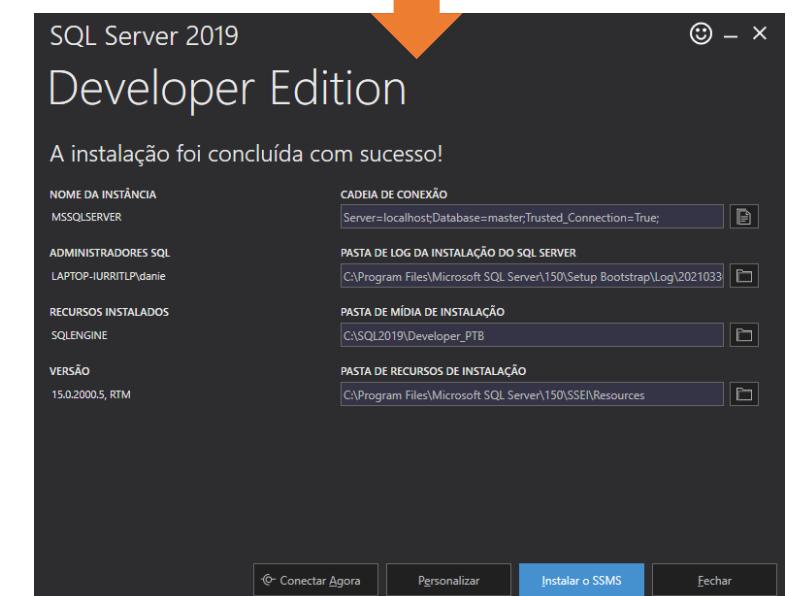
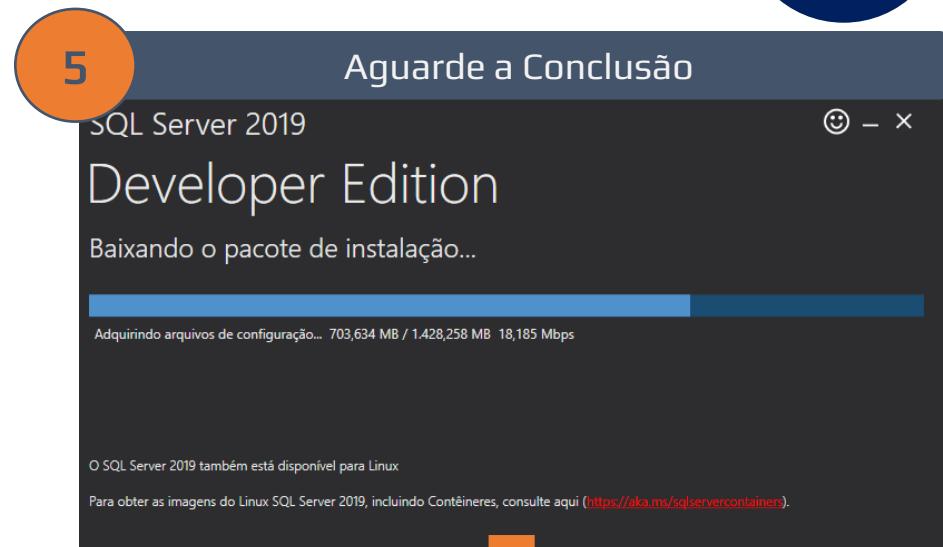
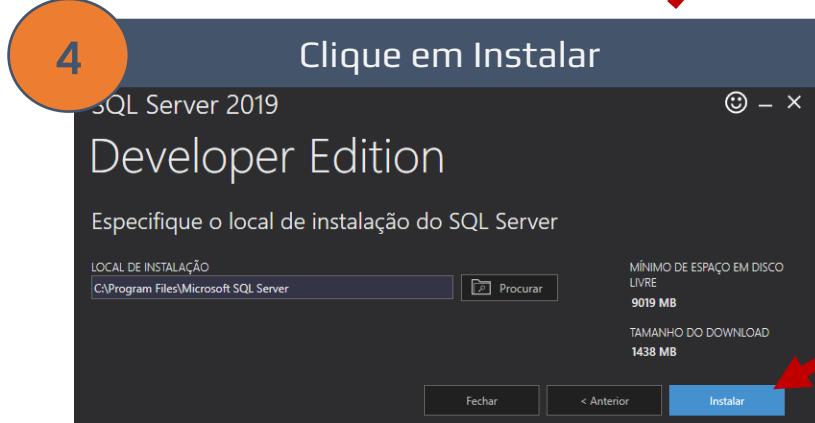
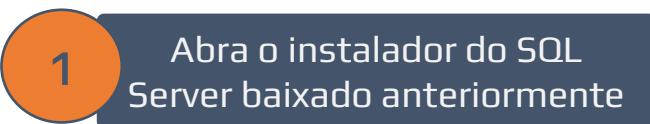
[Fazer download agora >](#)



Microsoft Contoso BI Demo Dataset for Retail Industry

*Important! Selecting a language below will dynamically change the complete page content to that language.*

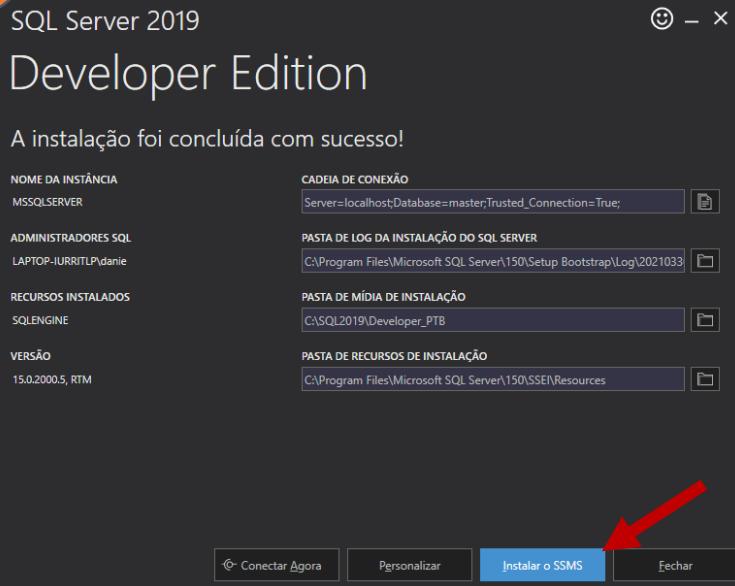
Select Language:



## Módulo 22 – Integração Python – SQL – Integrar Python com o SQL – Configurações (2/2)

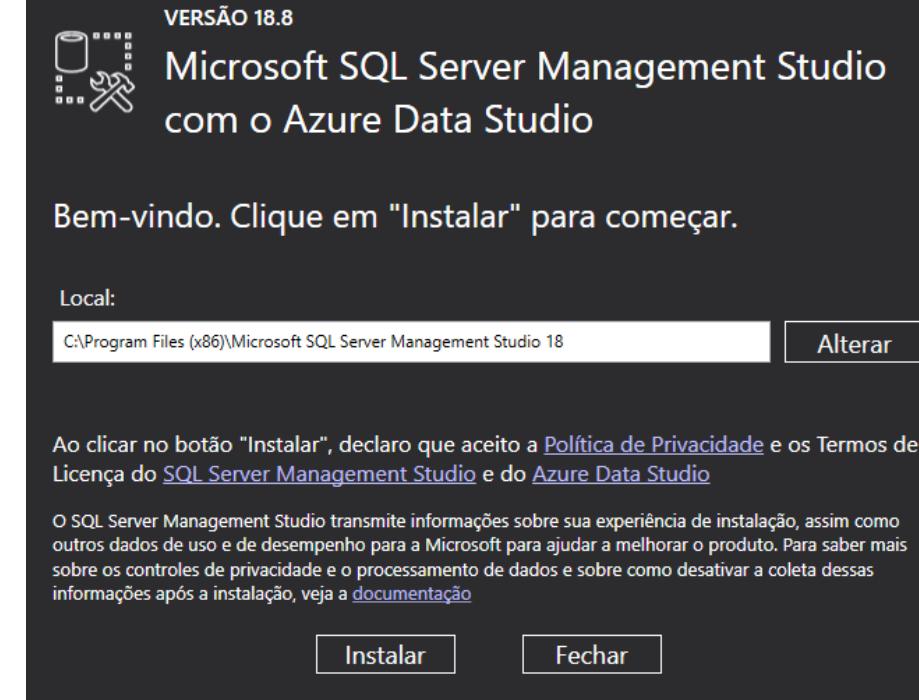
6

### Instale o SSMS



8

### Escolha a opção básico

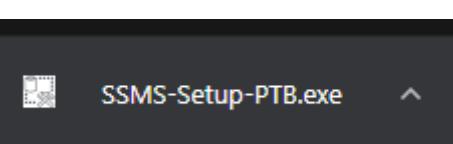


7

### Baixar o SSMS

## Baixar o SSMS

[Baixar o SQL Server Management Studio \(SSMS\)](#) ↗



9

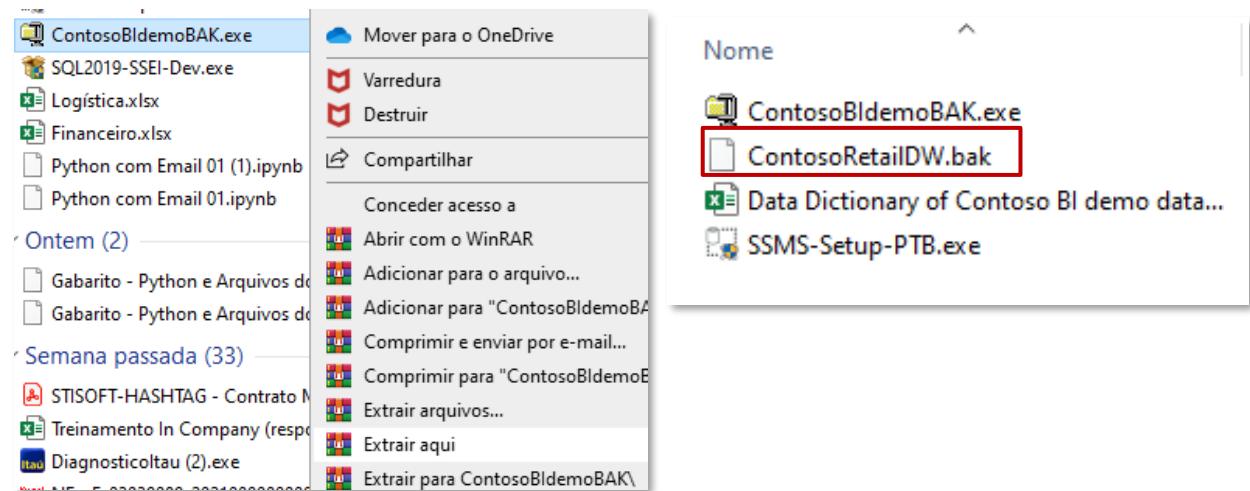
### Reinic peace seu computador

Após as configurações iniciais vamos iniciar importando nossos dados para o SQL Server.

Extraia os arquivos utilizando o Winrar ou 7zip, conforme apresentado na imagem ao lado.

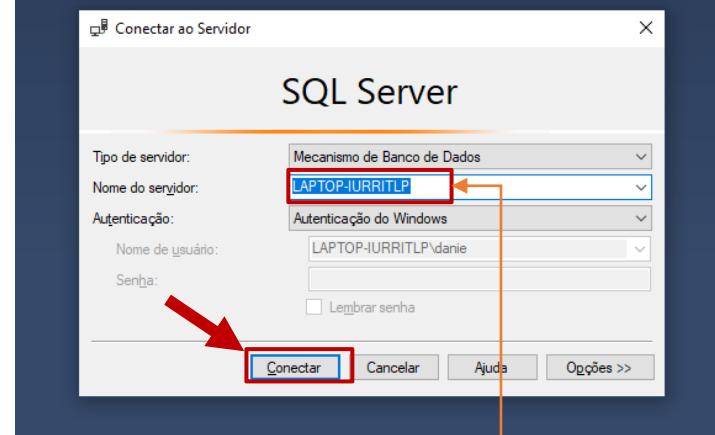
Após a extração dos arquivos, vamos importa-los com a ajuda do SSMS instalado anteriormente na etapa de configuração. Para inicializa-lo basta procurar por SSMS no menu iniciar.

Após a abertura do SSMS vamos iniciar o processo de importação nos próximos slides.



1

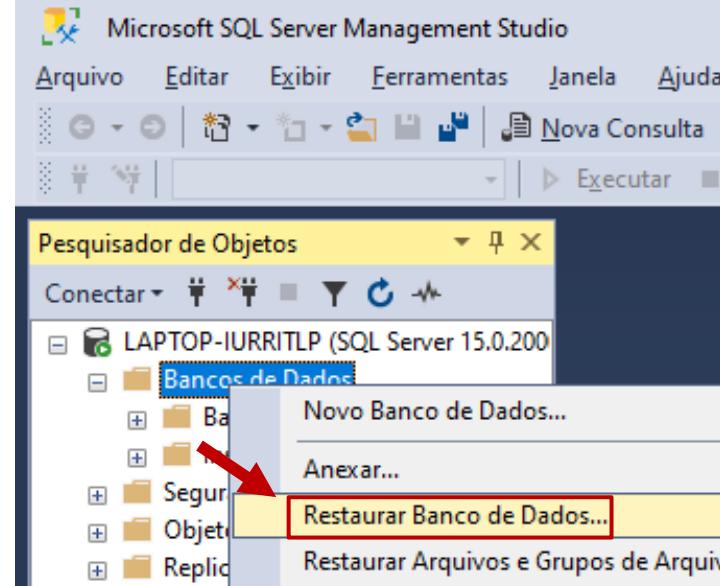
Abra o instalador do SQL Server baixado anteriormente



Nome do SEU computador. Não será o mesmo desta imagem

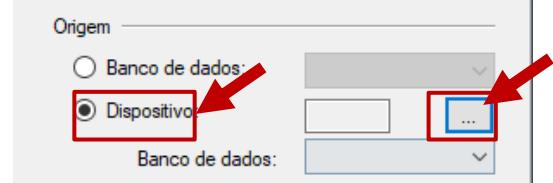
2

Abra o instalador do SQL Server baixado anteriormente



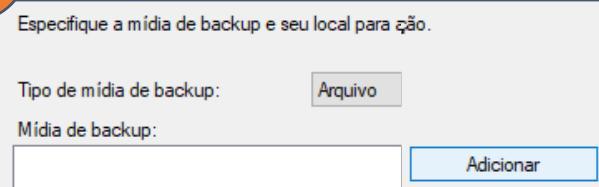
3

Selezione Dispositivo



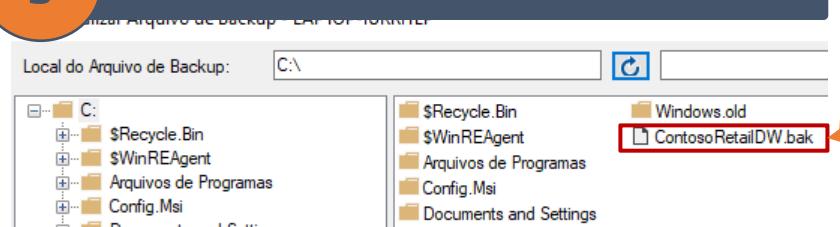
4

Clique em Adicionar



5

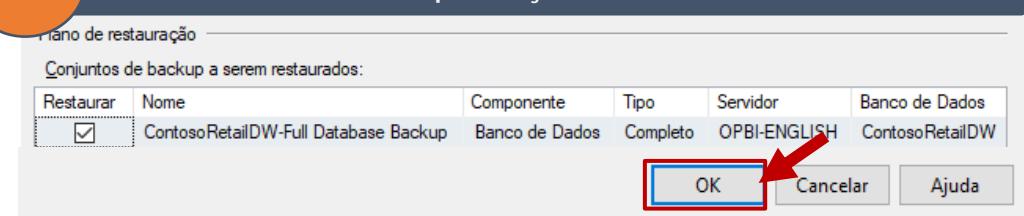
Selecione a base de dados



Para esse exemplo, a base de dados foi colocada diretamente no C:\ do computador

6

Confirme a importação da base de dados



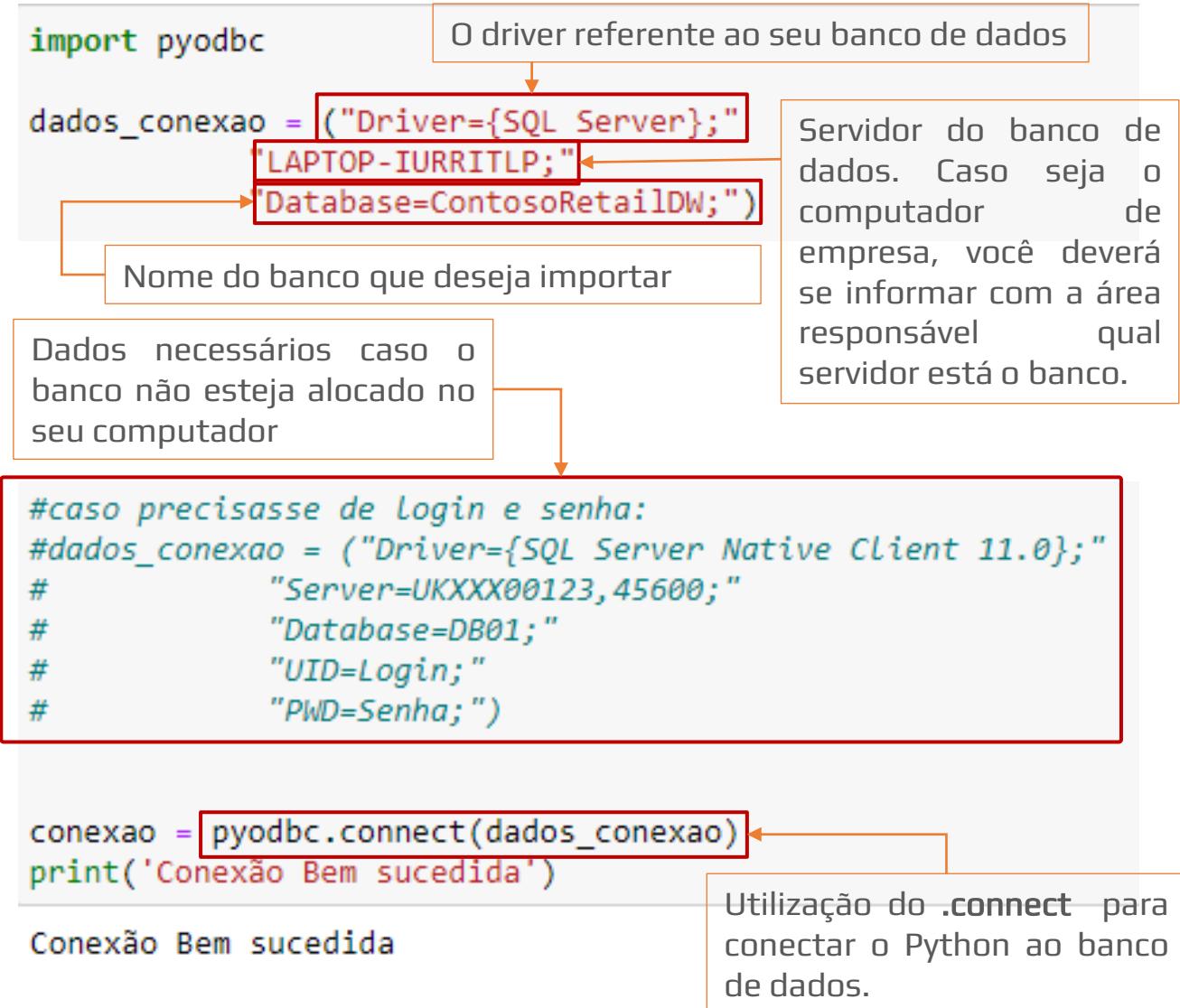
Vamos finalmente para o Python iniciar nossa integração.

Para isso, usaremos a biblioteca **pyodbc**([doc](#)).

Se você nunca utilizou essa biblioteca, é bem provável que ela ainda não esteja instalada. Portanto, assim como já fizemos com bibliotecas anteriores, vamos utilizar o **pip install** para realizar essa instalação no prompt de comando do Anaconda.

**pip install pyodbc**

Após instalação vamos, como sempre, importar nossa biblioteca e a primeira informação que precisamos inserir no nosso código serão os **dados de conexão** conforme apresentado ao lado. Esses dados permitirão a conexão entre o Python e nosso banco de dados.



Após a conexão estabelecida, precisamos criar um intermediário entre o Python e o SQL. Esse intermediário chamaremos de cursor.

O pyodbc possui um método que nos permite criar o cursor bem facilmente.

.cursor()

Criado nosso cursor, vamos começar a criar nosso código trabalhará no banco de dados.

Para realizar essas operações, temos 2 opções:

- Opção 1: Utilizar métodos do cursor;
- Opção 2: Utilizar o Pandas.

Visto que já estamos voando no pandas a essa altura do campeonato, vamos aprofundar na opção 2.

Funciona como um intermediário entre o SQL Server e o Python

cursor = conexao.cursor()

Método .cursor da biblioteca pyodbc

```
#opção 1: apenas executar um comando no banco de dados  
# cursor.execute("SELECT * FROM BaseDeDados.Tabela")  
# conexao.commit()
```

Executa as ações definidas nos parênteses no Banco de Dados

“SALVA” a situação em que o banco de dados se encontra.

Código em SQL. Basicamente está buscando todos os dados da tabela base de dados.

Vamos agora para a Opção 2, utilizando o Pandas para consulta dos dados.

Nessa opção, reduzimos a necessidade de conhecimentos em SQL e podemos focar no Pandas, que é algo que já temos maior domínio nessa altura do campeonato.

Essencialmente, o que faremos é Ler o Banco de dados em SQL através do Pandas e importar essas informações (assim como fizemos com os arquivos em .csv e excel.)

No exemplo ao lado, temos um código de importação. Perceba que a estrutura `pd.read_` é a mesma já utilizada anteriormente, só mudando o formato para SQL(`pd.read_sql(doc)`).

O código em vermelho, está em SQL mas apesar de ser um pouco estranho em um primeiro momento, não é tão complexo de ser entendido (deixamos mastigado ali para você 😊).

`pd.read_sql` realiza uma consulta a base de dados indicada

```
#opção 2: puxar uma consulta para o banco de dados com o pandas
import pandas as pd
produtos_df = pd.read_sql('SELECT * FROM ContosoRetailDW.dbo.DimProduct', conexao)
display(produtos_df)
```

| ProductKey | ProductLabel | ProductName | ProductDescription                  | ProductSubcategoryKey                             | Manufacturer | Bra          |
|------------|--------------|-------------|-------------------------------------|---|--------------|--------------|
| 0          | 1            | 0101001     | Contoso 512MB MP3 Player E51 Silver | 512MB USB driver plays MP3 and WMA                | 1            | Contoso, Ltd |
| 1          | 2            | 0101002     | Contoso 512MB MP3 Player E51 Blue   | 512MB USB driver plays MP3 and WMA                | 1            | Contoso, Ltd |
| 2          | 3            | 0101003     | Contoso 1G MP3 Player E100 White    | 1GB flash memory and USB driver plays MP3 and WMA | 1            | Contoso, Ltd |

`SELECT->` Indica uma consulta a uma tabela;  
`*` -> Indica que TODOS os dados devem ser extraídos;  
`ContosoRetailDW.dbo.DimProduct` -> Base de dados + Tabela que será consultada  
`conexao` -> variável criada anteriormente

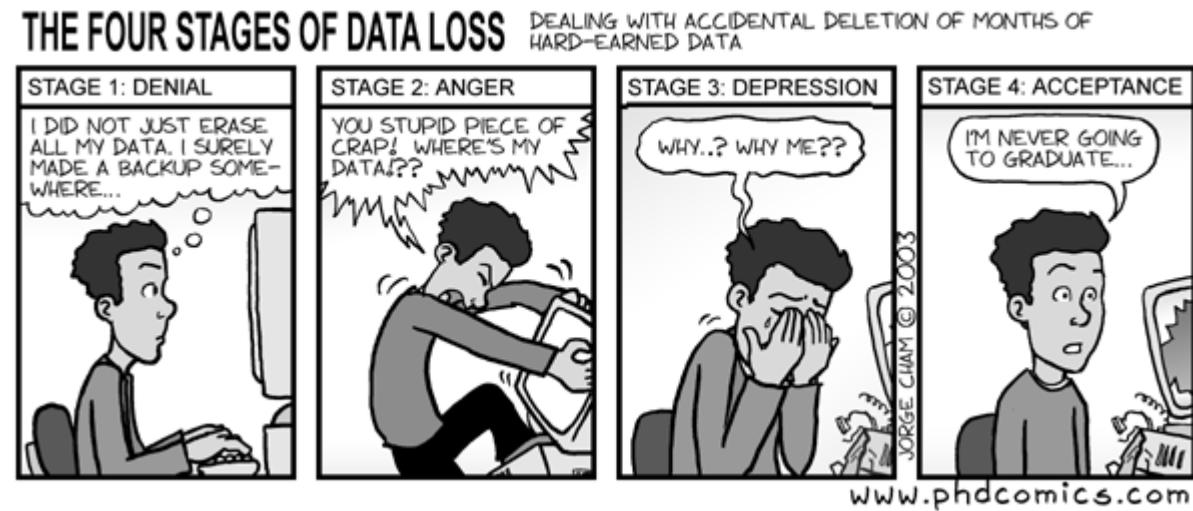
Quando estamos trabalhando com banco de dados temos que ter alguns cuidados.

Alterações em banco de dados podem significar perda de informações se não feitas com atenção.

Sempre que estamos trabalhando com consultas(**SELECT**) a banco de dados, podemos ficar um pouco mais tranquilos pois não estamos de fato alterando a fonte.

No entanto, em alguns casos, quando usamos **UPDATE** ou **CREATE** por exemplo, podemos alterar, acrescentar ou até mesmo apagar informações e se esse banco de dados não possuir um backup, iremos PERDER estes dados.

Esse slide não tem como objetivo te assustar, mas sim, te chamar atenção para tomar sempre muito cuidado pois cada vez mais DADOS são o que movem as empresas!



Módulo 23

# Integração Python - Web (Introdução Web- Scraping com Selenium)

Neste módulo, vamos aprender uma das ferramentas mais interessantes para automatizar processos que envolvam sites da internet.

Existem algumas bibliotecas que desempenham a mesma função ou similar, mas o **SELENIUM** ([doc](#)) é uma das bibliotecas mais conhecidas e utilizadas.

E em alguns casos, você verá o termo **webscrapping** essencialmente o que significa e o que faremos com o Selenium é “ler” uma página Web e conseguir extrair informações, navegar entre as páginas, etc.

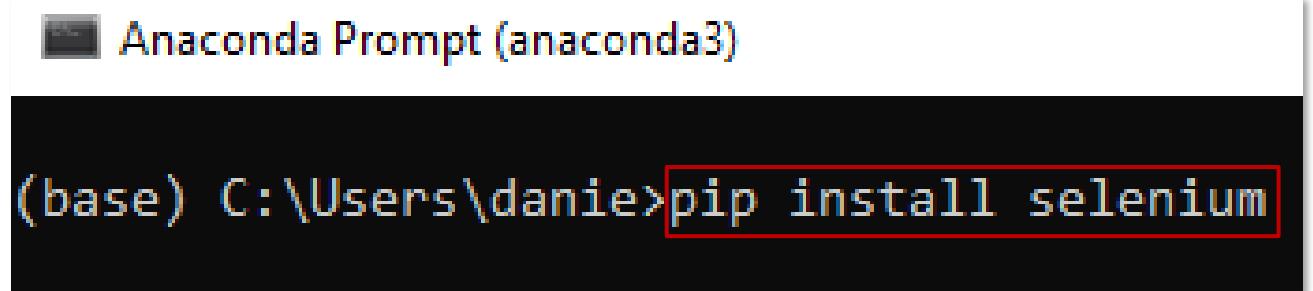
Assim como a maioria das bibliotecas que vimos até agora vamos iniciar importando essa biblioteca. Como este não é um pacote padrão do Jupyter, será necessário instalá-lo usando no prompt do anaconda:

```
pip install selenium
```



The screenshot shows a presentation slide with the following content:

- Selenium with Python**
- Author:** Baiju Muthukadan
- License:** This document is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
- Note:** (This note is partially visible at the bottom of the slide.)



The terminal window shows the command:

```
(base) C:\Users\danie>pip install selenium
```

Ao importarmos o Selenium vamos importar mais especificamente o webdriver. Ele nos permitirá navegar pelas páginas Web.

Cada navegador possui o seu webdriver e pode ser baixado nos links apresentados ao lado.

No curso usaremos o Google Chrome, que é um dos mais utilizados no mercado.

|                 |   |
|-----------------|---|
| <b>Chrome:</b>  | <a href="https://sites.google.com/a/chromium.org/chromedriver/downloads">https://sites.google.com/a/chromium.org/chromedriver/downloads</a>               |
| <b>Edge:</b>    | <a href="https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/">https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/</a> |
| <b>Firefox:</b> | <a href="https://github.com/mozilla/geckodriver/releases">https://github.com/mozilla/geckodriver/releases</a>   |
| <b>Safari:</b>  | <a href="https://webkit.org/blog/6900/webdriver-support-in-safari-10/">https://webkit.org/blog/6900/webdriver-support-in-safari-10/</a>                   |

Antes de realizarmos a importação, vamos fazer o download e “instalação” do chromedriver:

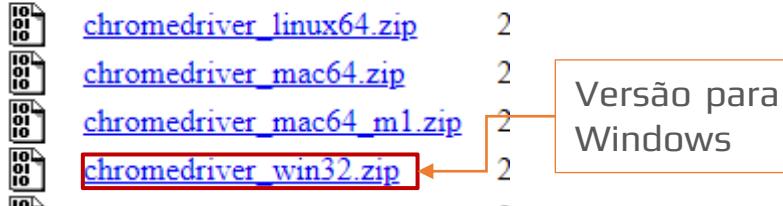
- 1 Faça o download da versão mais recente do Chromedriver;

Downloads

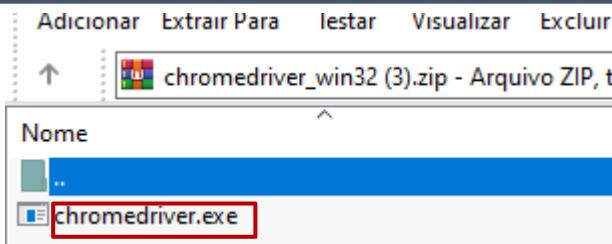
**Current Releases**

- If you are using Chrome version 90, please download [ChromeDriver 90.0.4430.24](#)

- 2 Seleione a opção que está de acordo com seu sistema operacional;



- 3 Extraia o arquivo chromedriver.exe existente dentro do arquivo compactado;



- 4 Cole o arquivo Chromedriver.exe na mesma pasta do executável do Python (python.exe)

|   |                  |
|---|------------------|
| Este Computador > Windows (C:) > Usuários > danie > anaconda3 |                  |
| chromedriver.exe  | 01/12/2020 14:33 |
| concr140.dll  | 19/11/2018 15:57 |
| cwp.py  | 13/03/2019 17:00 |
| LICENSE_PYTHON.txt  | 13/05/2020 14:31 |
| msvcp140.dll  | 19/11/2018 15:57 |
| msvcp140_1.dll  | 19/11/2018 15:57 |
| msvcp140_2.dll  | 19/11/2018 15:57 |
| python.exe  | 02/07/2020 13:31 |

Geralmente o arquivo python.exe está na pasta Usuários>nome do seu usuário>anaconda3

Finalmente vamos para a importação!😊

Assim como feito com outras bibliotecas vamos usar:

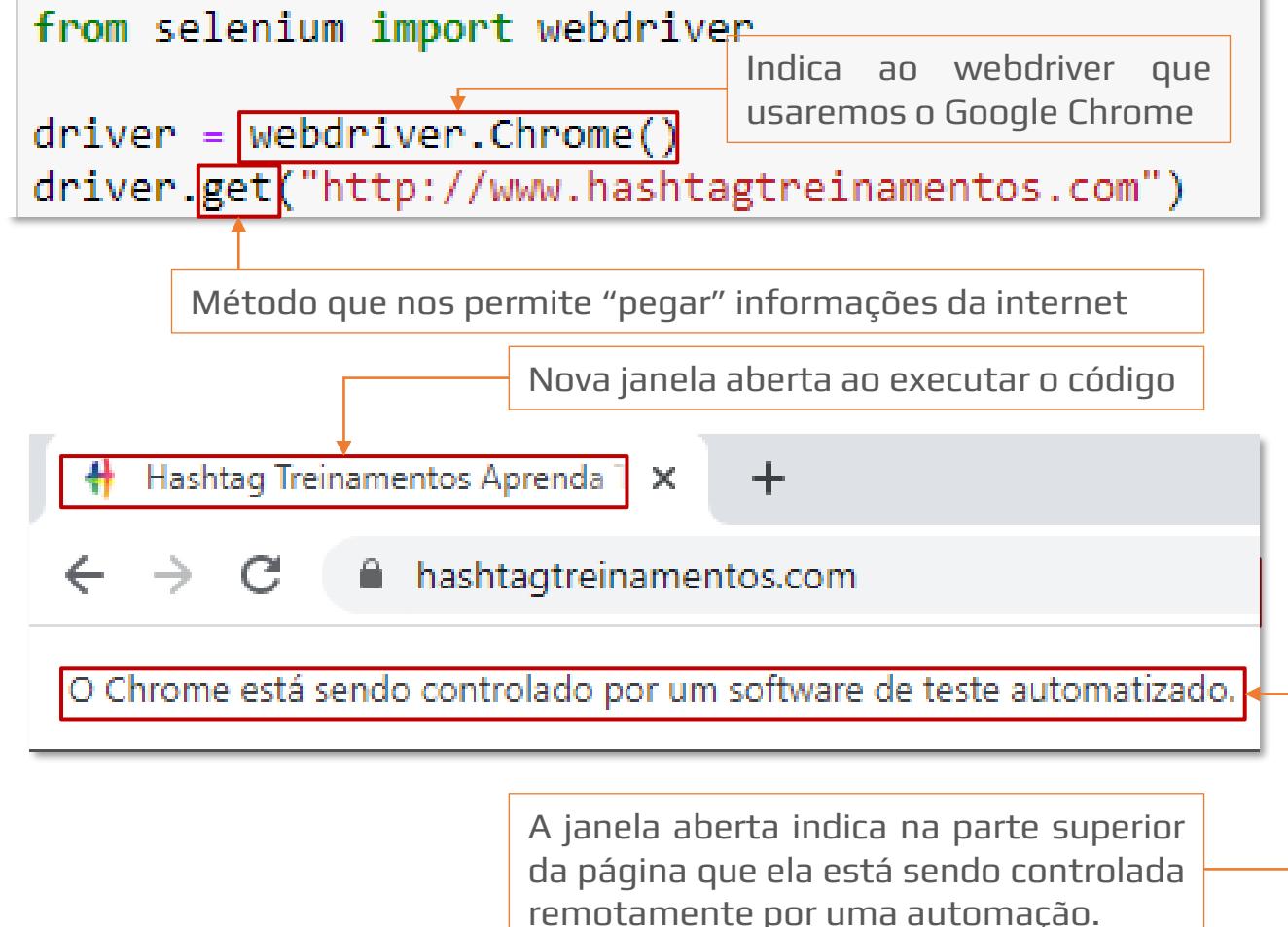
```
from selenium import webdriver
```

Após a importação do webdriver, usaremos um método chamado `.get` que essencialmente possui a função de pegar informações da internet.

No nosso exemplo, associaremos o webdriver com o `get` para podermos “ler” a página da Hashtag Treinamentos:

[www.hashtagtreinamentos.com](http://www.hashtagtreinamentos.com)

Perceba que ao rodarmos o código uma NOVA janela do Chrome será aberta e que nela existe uma mensagem que a página está sendo controlada por uma automação.



Existe uma forma de rodarmos o Selenium sem que a janela do navegador seja aberta.

Durante a fase de desenvolvimento, é interessante visualizarmos o que está acontecendo, mas a partir do momento que tudo está funcionando, não é mais necessário que uma página seja aberta.

Para isso, precisamos adicionar ao nosso código anterior alguns passos de setup:

**.ChromeOptions ()** -> Configurações padrões do uso do navegador pelo Selenium;

**.add\_argument('headless')** -> Adiciona um argumento as configurações ChromeOptions. Nesse caso, a configuração adicional é o headless que nos permitirá executar o código sem a abertura de uma página adicional;

**.Chrome(options=chrome\_options)** -> Essencialmente o mesmo código anterior, mas dessa vez, indicamos que opções de “abertura” seguem as configurações armazenadas na variável chrome\_options)

Armazenando as configurações padrões na variável chrome\_options

```
from selenium import webdriver  
chrome_options = webdriver.ChromeOptions()
```

```
chrome_options.add_argument('headless')
```

```
driver = webdriver.Chrome(options=chrome_options)
```

```
driver.get("http://www.hashtagtreinamentos.com")  
print(driver.title)
```

Hashtag Treinamentos Aprenda TUDO de Excel, VBA e Power BI Aqui!

Add\_arguments adiciona uma configuração as configurações armazenadas na variável chrome\_options

A variável chrome\_options armazena uma configuração que ao ser utilizada junto do keyword 'options =' indica como será executado o webdriver.

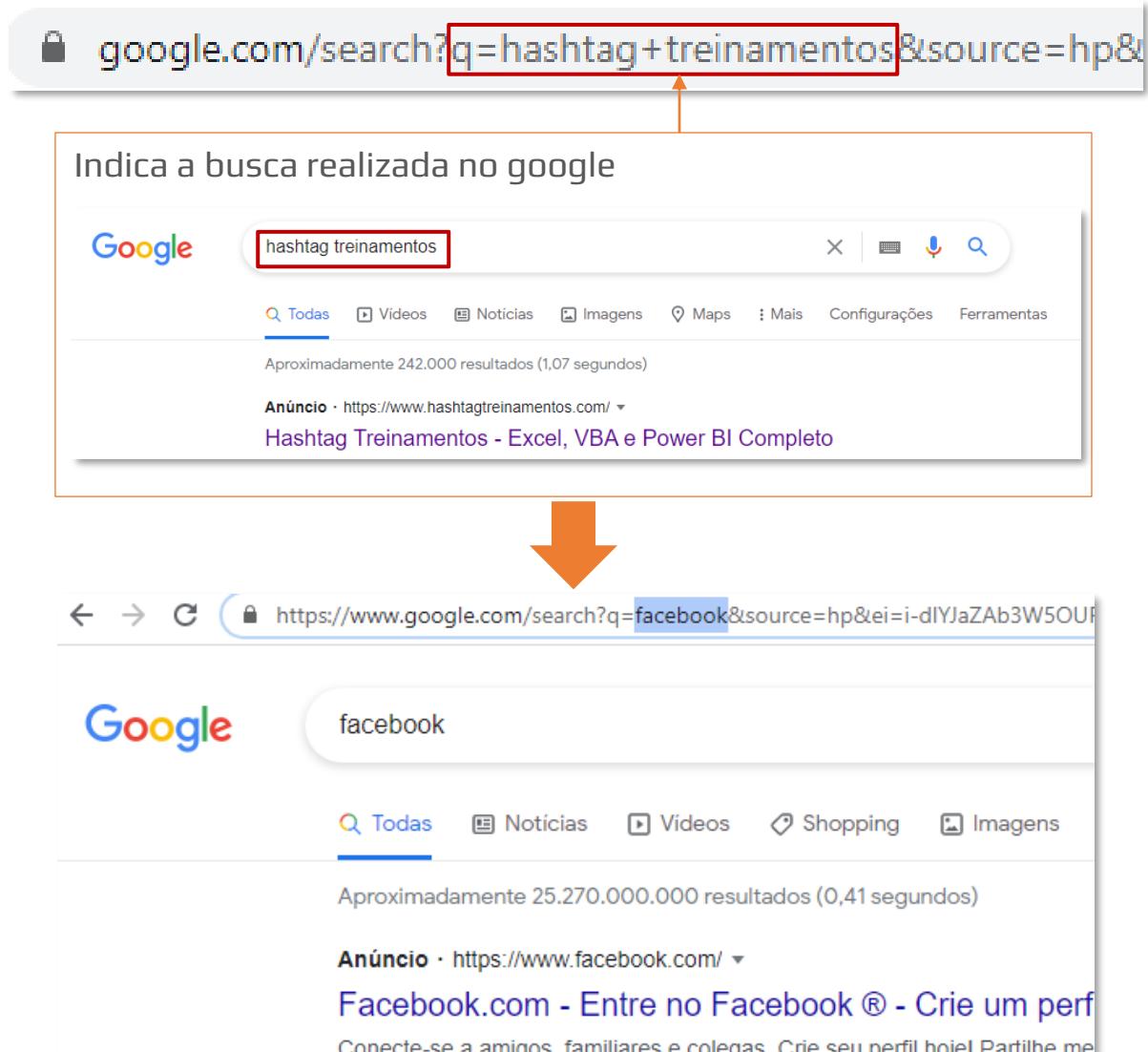
Antes de entendermos sobre como criar um código no Python é necessário entender um pouco de como as páginas WEB funcionam.

Possivelmente você já se perguntou o que significam todos aqueles códigos aleatórios na url de um site.

Veja no exemplo ao lado, ao buscarmos no google por hashtag treinamentos, esse foi o resultado da url. O que parecem números aleatórios na realidade são partes de um conteúdo que está vinculado a página.

Por exemplo no Google, `q=hashtag+treinamentos` indica que uma pesquisa com as palavras chaves “hashtag”, e “treinamentos”

Exemplos como esses são chamados de parâmetros da url. Através deles conseguimos mudar o conteúdo da nossa página conforme apresentado ao lado usando a palavra ‘facebook’.



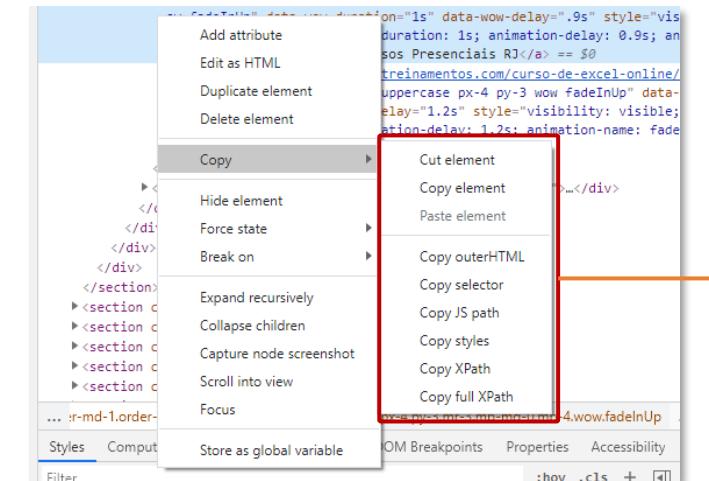
Além da URL temos outra parte muito importante na construção dos sites que é o HTML. Não é nosso objetivo aqui entendermos tudo sobre o HTML, mas conseguimos através da ferramenta INSPECIONAR do Google Chrome.

A partir dela, podemos identificar qual o código por trás das estruturas visíveis na página. Isso será muito útil quando entrarmos nos códigos do Selenium pois usaremos este código para acessarmos essas estruturas 😊

The screenshot shows the homepage of the Hashtag Treinamentos website. The header features the logo 'hashtag' and navigation links for 'HOME', 'CURSO EXCEL ONLINE', and 'CURSO POWER BI ONLINE'. The main content area has a large heading: 'Torne-se uma referência na empresa onde você trabalha ou vai trabalhar com a Hashtag Treinamentos'. Below this, there's a sub-headline: 'Aprenda tudo de Excel, VBA e Power BI'. At the bottom, there are two buttons: 'CURSOS PRESENCIAIS RJ' (highlighted with a red border) and 'CURSOS ONLINE'.

This screenshot shows the browser's developer tools with the 'Elements' tab selected. It displays the HTML structure of the page. A specific button element is highlighted with a red box and labeled with its href attribute: [Cursos Presenciais RJ](https://www.hashtagtreinamentos.com/cursos-excel-rj/). The surrounding HTML includes various CSS classes and JavaScript event handlers like 'data-wow-duration' and 'data-wow-delay'.

### Métodos de cópia do HTML



Código HTML que representa o botão laranja “Cursos Presenciais RJ”

Apesar desse módulo se chamar “Métodos úteis do Selenium” vamos iniciar falando de uma biblioteca que não é do Selenium, mas que funciona muito bem com ele.

Com certeza em algum momento você já acessou uma página na web e ela demorou a carregar. Quando usamos o Selenium, o Python está fingindo ser um humano usando a internet.

Na vida real se uma página você espera, pois não tem onde clicar. No Python ele não sabe que não carregou então tenta clicar em algo que não carregou...

Como resolver isso? Fazendo o Python esperar! Vamos fazer isso usando o método `.sleep()` da biblioteca `time` ([doc](#)).

Com esse método vamos poder definir um tempo entre a execução das linhas de código que permitirão o carregamento da página. Veja o exemplo ao lado.

```
from selenium import webdriver  
import time  
  
driver = webdriver.Chrome()  
driver.get('https://hashtagtreinamentos.com')  
time.sleep(3)
```

Após a abertura do site através do `driver.get`, o Python aguardará 3seg para executar a próxima linha de código.

O Python esperou os 3 seg, a página carregou, o que vamos fazer?

Vamos tentar criar um código que nos permita clicar no botão laranja “CURSOS PRESENCIAIS RJ”. Para isso, precisamos encontrar algo que seja único nesse botão para poder ajudar o Python a acha-lo. Existem diversas formas de identificar os itens do HTML, algumas mais específicas e outras não tanto assim...

O Selenium possui métodos distintos para cada um desses “mecanismos” de busca conforme apresentado abaixo ([fonte](#)):

- find\_element\_by\_id
- find\_element\_by\_name
- find\_element\_by\_xpath
- find\_element\_by\_link\_text
- find\_element\_by\_partial\_link\_text
- find\_element\_by\_tag\_name
- find\_element\_by\_class\_name
- find\_element\_by\_css\_selector



Como saber qual usar ou o que são esses termos?

- find\_element\_by\_id
- find\_element\_by\_name
- find\_element\_by\_xpath
- find\_element\_by\_link\_text
- find\_element\_by\_partial\_link\_text
- find\_element\_by\_tag\_name
- find\_element\_by\_class\_name
- find\_element\_by\_css\_selector

Todos eles estão vinculados a termos de HTML. Sem entrar em muitos detalhes, qual usar pode variar. O mais importante, é que você busque algo que diferencie o que você busca do resto do site.

Em geral, um dos melhores caminhos é utilizar o ID.

Para descobrir o ID, usaremos o INSPECIONAR que já falamos anteriormente.

## Torne-se uma re na empresa ond trabalha ou vai t com a Hashtag Treinamentos

Aprenda tudo de Excel, BI

CURSOS PRESENCIAIS RJ

CURSOS

```
<section class="hero text-md-left text-center">
  <div class="container-fluid">
    <div class="slider">
      <div class="slide">
        <div class="row align-items-center">
          <div class="col-md-6 order-md-1 order-2">
            <h1 class="wow fadeInUp" data-wow-duration="1s" data-wow-delay="0s" style="visibility: visible; animation-duration: 1s; animation-delay: 0s; animation-name: fadeInUp;">...</h1>
            <h3 class="mb-5 wow fadeInUp" data-wow-duration="1s" data-wow-delay=".6s" style="visibility: visible; animation-duration: 1s; animation-delay: 0.6s; animation-name: fadeInUp;">Aprenda tudo de Excel, VBA e Power BI</h3>
            ...
            <a href="https://www.hashtagtreinamentos.com/cursos-excel-rj/" class="btn btn-laranja text-uppercase px-4 py-3 mr-3 mb-md-0 mb-4 wow fadeInUp" data-wow-duration="1s" data-wow-delay=".9s" style="visibility: visible; animation-duration: 1s; animation-delay: 0.9s; animation-name: fadeInUp;">Cursos Presenciais RJ</a> == $0
            <a href="https://www.hashtagtreinamentos.com/curso-de-excel-online/" class="btn btn-cinza text-uppercase px-4 py-3 wow fadeInUp" data-wow-duration="1s" data-wow-delay="1.2s" style="visibility: visible; animation-duration: 1s; animation-delay: 1.2s; animation-name: fadeInUp;">Cursos Online</a>
          </div>
        </div>
      </div>
    </div>
  </div>
</section>
```

Styles   Computed   Layout   Event Listeners   DOM Breakpoints   Properties   Accessibility

O inspecionar, não conseguimos encontrar o ID... Vamos precisar buscar outro método de busca!

Como não foi possível achar o ID, vamos usar um método que aparentemente é bem complexo, mas pode ser bem útil:

`find_element_by_xpath`

Para utilizarmos esse método vamos precisar copiar o XPATH do botão que desejamos localizar.

Para isso:

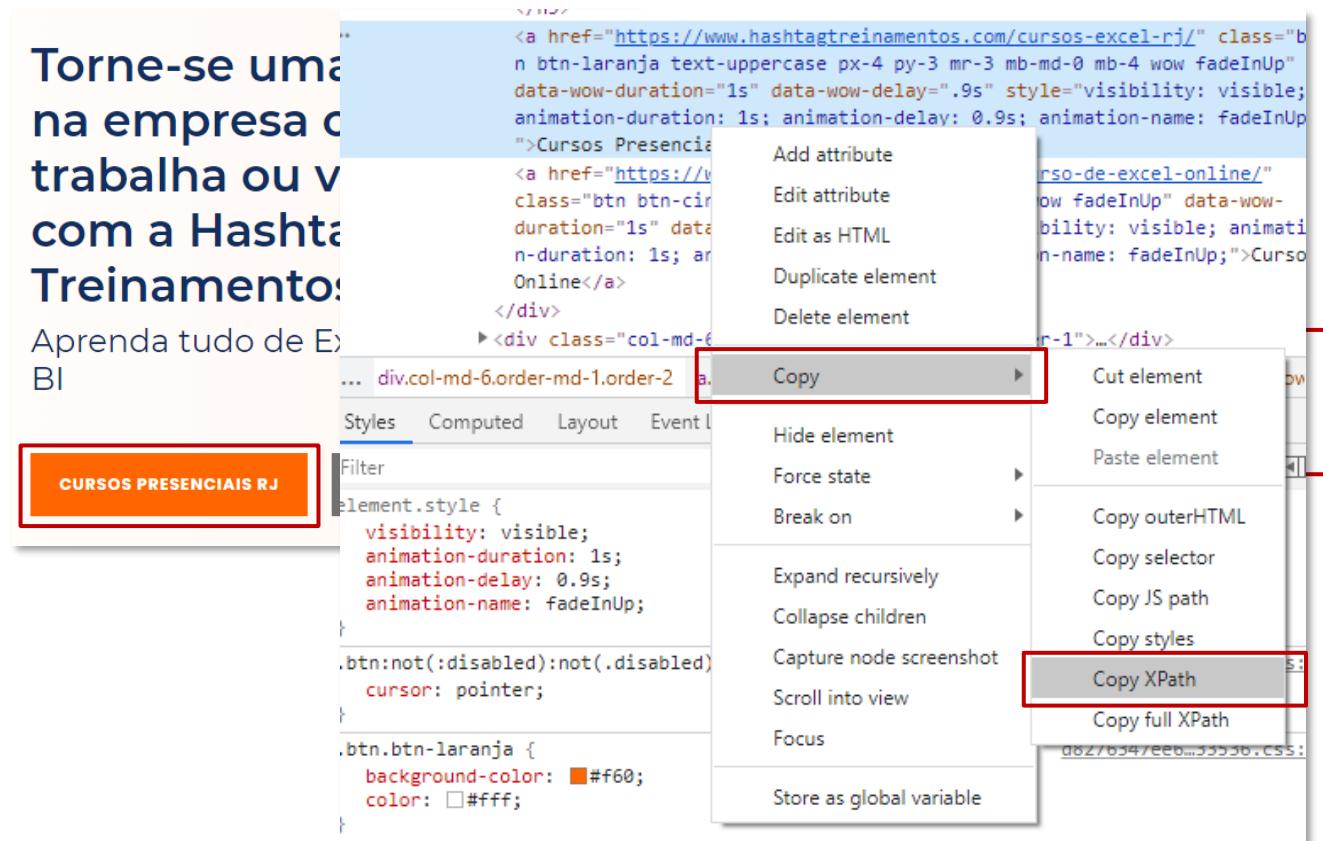
- 1) Clique com o botão direito do mouse sobre o Código HTML;
- 2) Escolha a opção Copy;
- 3) Selecione a opção Copy Xpath;

Ao realizar as etapas acima uma estrutura HTML estará armazenada na memória do seu computador.

Vá até o Jupyter e em uma célula utilize CTRL+V.

Um Código não muito intuitivo como apresentado abaixo aparecerá para você:

`/html/body/section[1]/div/div/div/div[1]/a[1]`



`/html/body/section[1]/div/div/div/div[1]/a[1]`

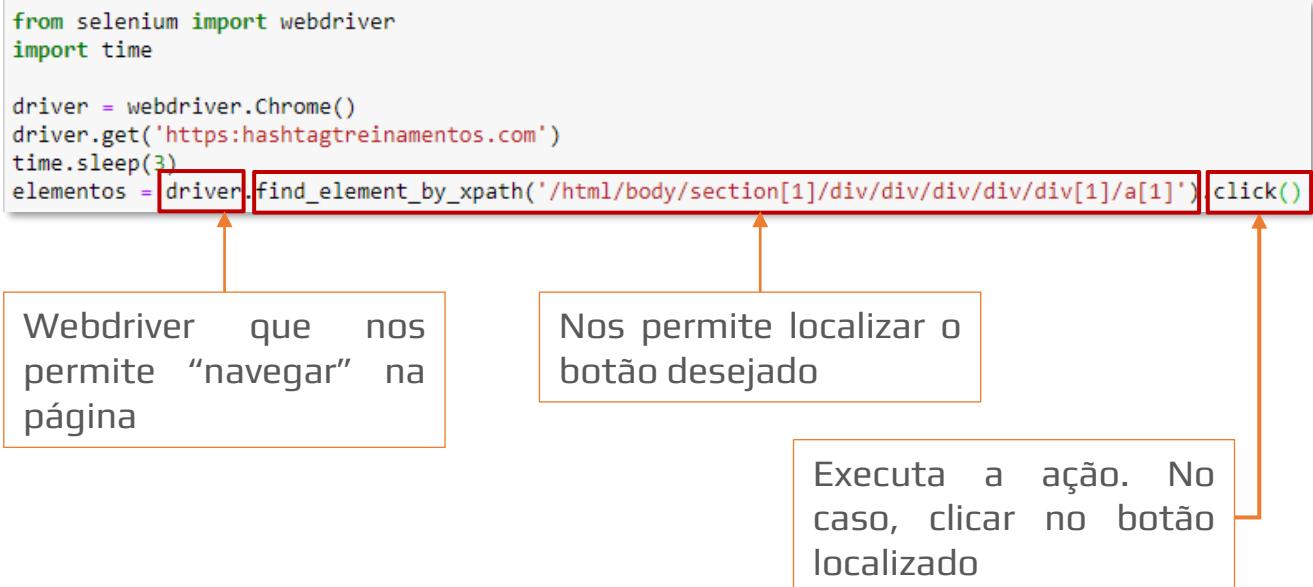
Em posse desse código, vamos utilizar nosso método:

`find_element_by_xpath`

No entanto, apenas localizar o elemento não é suficiente. Falta informarmos uma ação a ser feita com o botão encontrado. Como nosso objetivo, é acessar a página, vamos clicar no botão. Para isso, usaremos o método abaixo:

`.click()`

Perceba que no código apresentado ao lado, utilizamos o webdriver para localizar o botão e após a localização concluída, clicar.

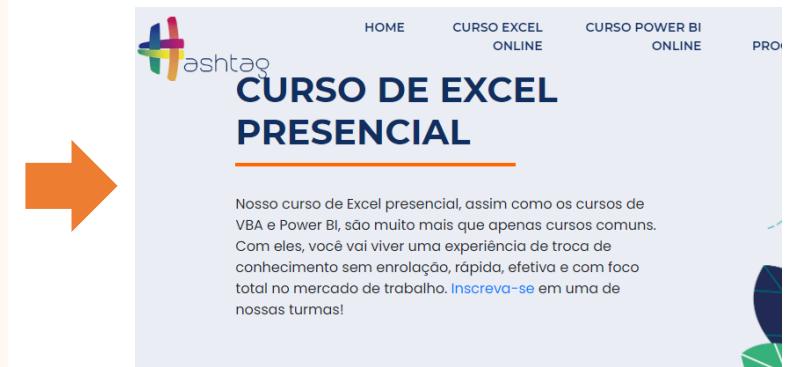


Torne-se uma referência na empresa onde você trabalha ou vai trabalhar com a Hashtag Treinamentos

Aprenda tudo de Excel, VBA e Power BI

CURSOS PRESENCIAIS RJ

CURSOS ONLINE



Outro caso comum é preencher formulários. Um campo de login e senha por exemplo.

Ainda no site da Hashtag, vamos preencher o formulário ao lado.

Assim como no caso anterior, o primeiro passo é entender como localizar esse elemento do site para podermos acessá-lo via Selenium.

Poderíamos usar o XPATH novamente como o método de localização, mas vamos usar outro método. Como temos name="fullname", vamos utilizar o método `find_element_by_name`.

Após a localização do elemento usaremos o método abaixo para a entrada de informações :

`.sendkeys()`

## CADASTRE-SE PARA RECEBER O MINICURSO BÁSICO DE 16 AULAS

The screenshot shows a registration form with two input fields. The first field is labeled "Seu primeiro nome\*" and contains the placeholder "Seu primeiro nome". The second field is labeled "Seu melhor e-mail\*" and also has a placeholder. Below the fields is an orange "Enviar" button.

Usando o inspecionar podemos encontrar o HTML do campo Seu primeiro nome

The screenshot shows the browser's developer tools with the element inspector open. It highlights the HTML code for the "Seu primeiro nome" input field. The code includes the label "Seu primeiro nome", the input element with name="fullname", and its placeholder "Seu primeiro nome". An arrow points from the text "campo Seu primeiro nome" in the previous slide to this highlighted code.

```
<div class="_form_element _x41432449 _inline-style">
  <label class="_form-label">Seu primeiro nome</label>
  <div class="_field-wrapper">
    <input type="text" name="fullname" placeholder="Seu primeiro nome" required data-name="fullname"> == $0
```

```
from selenium import webdriver
import time
```

```
driver = webdriver.Chrome()
driver.get('https://hashtagtreinamentos.com')
time.sleep(3)
```

```
elementos = driver.find_elements_by_name('fullname')
elementos[0].send_keys('Lira')
elementos = driver.find_elements_by_name('email')
elementos[0].send_keys('joaoprlira@gmail.com')
driver.find_element_by_id('_form_173_submit').click()
```

## CADASTRE-SE PARA

The screenshot shows the registration form again, but now the "Seu primeiro nome" input field contains the value "Lira".

Módulo 24

# Integração Python com APIs e JSON

Antes de qualquer coisa... O que é uma API??

API é um conjunto de códigos que uma plataforma, um site, etc fornece para que outros códigos possam usá-los e garantir esse relacionamento de forma segura e precisa.

Nem todas plataformas, oferecem serviço de API, mas existem muitas disponíveis para uso.

Veja o exemplo ao lado. Pegamos aqui a API do Youtube ([link](#)). Aqui, temos códigos já disponibilizados pelo próprio youtube que nos auxiliam a executar esses comandos de forma segura.

Ao invés de desenvolvermos um código novo, simplesmente utilizamos a API.

The screenshot shows the YouTube Data API developer page. On the left, there's a sidebar with links for Index, Apps Script, Go, Java, JavaScript, .NET, PHP, Python, Python on App Engine, and Ruby. The main content area is titled "Python" and lists various API methods:

- Recuperar meus envios (`playlistItems.list`)
- Criar uma playlist (`playlists.insert`)
- Pesquisar por palavra-chave (`search.list`)
- Adicionar inscrição de canal (`subscriptions.insert`)
- Envios a retomar (`videos.insert`)
- Postar um boletim de canal (`activities.insert`)
- Adicionar um vídeo em destaque (`channels.update`)
- Recuperar meus envios (`playlistItems.list`)
- Criar uma playlist (`playlists.insert`)
- Pesquisar por palavra-chave (`search.list`)
- Pesquisar por tópico (`search.list`)
- Adicionar inscrição de canal (`subscriptions.insert`)
- Enviar uma imagem em miniatura do vídeo (`thumbnails.set`)
- Enviar um vídeo (`videos.insert`)
- Avaliar um vídeo (gostei) (`videos.rate`)
- Atualizar um vídeo (`videos.update`)

Beleza, entendemos o que é uma API, vamos usar uma na prática.

Para esse exemplo, vamos usar uma API gratuita que nos fornecerá a cotação de moedas (dólar, euro, etc).

<https://docs.awesomeapi.com.br/api-de-moedas>

Nesse caso, não temos a necessidade de um login na API, então, podemos ir direto para a documentação da API e entender quais são as etapas a serem seguidas para conseguirmos a informação das cotações.

Conforme apresentado ao lado esses dados serão requisitados através do método GET e serão buscados através do link indicado.

Essa comunicação pode ser feita em diversas “línguas” uma das mais comuns e quando estamos trabalhando com Python é a **JSON**.

**O QUE?** - Informa que usaremos o método get para requisitar os dados da API

**GET** Retorna todas as moedas (atualizado a cada 30 segundos)

<https://economia.awesomeapi.com.br/json/all>

Retorna a ultima ocorrência de todas as moedas.

**COMO?** - Formato que será utilizada na “comunicação” entre a API e seu programa.

Nesse caso, no formato **JSON**.

**ONDE?** - Endereço que vamos requisitar as informações

Vamos para o nosso código! Como já é rotina, vamos começar importando as bibliotecas serão necessárias.

Quando estamos falando de APIs e Formato JSON, vamos utilizar 2 bibliotecas novas:

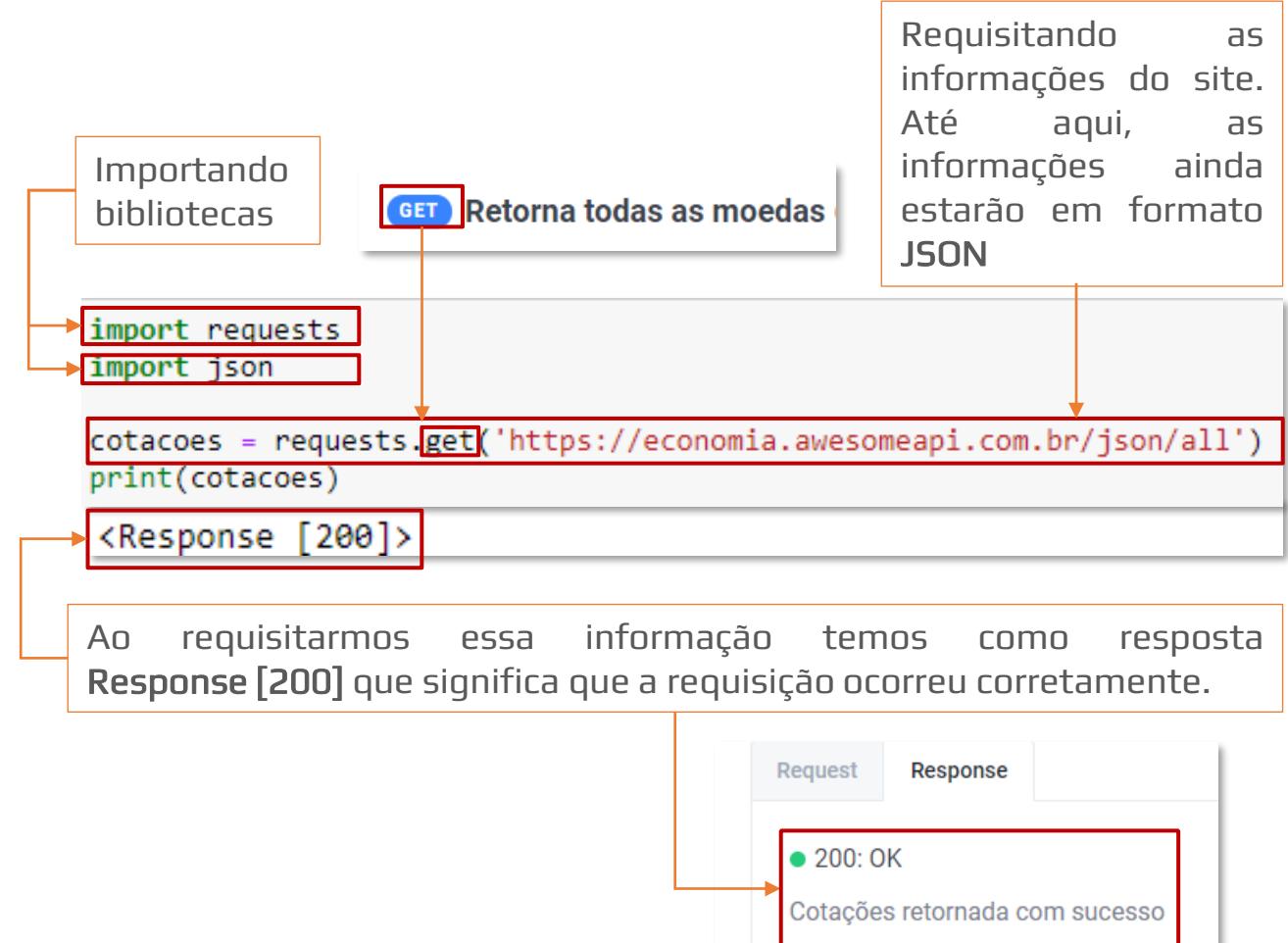
`requests(doc)`      `json(doc)`

Primeiro vamos entender a `requests`. Conforme vimos na documentação da API usaremos o `GET` para requisitar os dados.

No entanto, quando printamos o resultado armazenado na variável `cotacoes` temos:

`<Response [200]>`

Sempre que requisitamos dados temos uma resposta. Essa resposta possui um código 200 indica que a requisição foi bem sucedida.



Sabemos que a requisição está correta, agora precisamos acessar essa informação.

Para isso usaremos o **JSON**. Utilizando essa biblioteca poderemos acessar os dados e convertê-los em um dicionário Python.

Usando a biblioteca **json** conseguimos transformar as informações em um dicionário PYTHON

```
import requests
import json

cotacoes = requests.get('https://economia.awesomeapi.com.br/json/all')
cotacoes_dic = cotacoes.json() ←
print(cotacoes_dic)

{'USD': {'code': 'USD', 'codein': 'BRL', 'name': 'Dólar Comercial', 'high': '5.4585', 'low': '5.3662', 'varBid': '-0.0245', 'pctChange': '-0.45', 'bid': '5.4331', 'ask': '5.4343', 'timestamp': '1605540818', 'create_date': '2020-11-16 12:33:39'}, 'USDT': {'code': 'USD', 'codein': 'BRLT', 'name': 'Dólar Turismo', 'high': '5.61', 'low': '5.5', 'varBid': '-0.035', 'pctChange': '-0.62', 'bid': '5.41', 'ask': '5.74', 'timestamp': '1605540240', 'create_date': '2020-11-16 12:24:43'}, 'CAD': {'code': 'CAD', 'codein': 'BRL', 'name': 'Dólar Canadense', 'high': '4.1729', 'low': '4.1014', 'varBid': '0.0088', 'pctChange': '0.02', 'bid': '4.1538', 'ask': '4.1557', 'timestamp': '1605540813', 'create_date': '2020-11-16 12:33:36'}, 'EUR': {'code': 'EUR', 'codein': 'BRL', 'name': 'Euro', 'high': '6.4775', 'low': '6.3551', 'varBid': '-0.0233', 'pctChange': '-0.36', 'bid': '6.4307', 'ask': '6.4342', 'timestamp': '1605540814', 'create_date': '2020-11-16 12:33:37'}, 'GBP': {'code': 'GBP', 'codein': 'BRL', 'name': 'Libra Esterlina', 'high': '7.2275', 'low': '7.0782', 'varBid': '-0.0289', 'pctChange': '-0.4', 'bid': '7.1637', 'ask': '7.169', 'timestamp': '1605540813', 'create_date': '2020-11-16 12:33:36'}, 'ARS': {'code': 'ARS', 'codein': 'BRL', 'name': 'Peso Argentino', 'high': '0.0684', 'low': '0.0673', 'varBid': '-0.0004', 'pctChange': '-0.66', 'bid': '0.0679', 'ask': '0.068', 'timestamp': '1605540818', 'create_date': '2020-11-16 12:33:39'}, 'BTC': {'code': 'BTC', 'codein': 'BRL', 'name': 'Bitcoin', 'high': '89500', 'low': '87251', 'varBid': '2129.6', 'pctChange': '2.45', 'bid': '89210', 'ask': '89206', 'timestamp': '1605540819', 'create_date': '2020-11-16 12:33:39'}}
```

Como vimos no exemplo anterior, o resultado da nossa requisição é um dicionário.

Se olharmos esse dicionário com um pouco mais detalhe, vamos perceber que cada moeda é uma chave deste dicionário.

Se nosso objetivo é conseguir a cotação do dólar(USD), podemos acessar nosso dicionário.

Como não conhecemos essa API, antes de irmos para o código, vamos dar uma olhada em sua documentação para entendermos as nomenclaturas utilizadas pela API.



## ATENÇÃO !

Nessa API específica, os valores estarão sempre sendo atualizados, portanto, o valor indicado aqui provavelmente será diferente do valor conseguido por você ☺

### Códigos das moedas

- USD (Dólar Americano)
- CAD (Dólar Canadense)
- AUD (Dólar Australiano)
- EUR (Euro)
- GBP (Libra Esterlina)
- ~~• ARS (Peso Argentino)~~

Usando a documentação da API podemos saber o nome da moeda e qual a chave que devemos utilizar para acessar a cotação

### Legendas

| key       | Label                   |
|-----------|-------------------------|
| bid       | Compra                  |
| ask       | Venda                   |
| varBid    | Variação                |
| potChange | Porcentagem de Variação |
| high      | Máximo                  |
| low       | Mínimo                  |

```
print('Dólar: {}'.format(cotacoes_dic['USD']['bid']))
print('Euro: {}'.format(cotacoes_dic['EUR']['bid']))
print('Bitcoin: {}'.format(cotacoes_dic['BTC']['bid']))
```

Dólar: 5.4331  
 Euro: 6.4307  
 Bitcoin: 89210

Aqui temos um dicionário dentro de um dicionário. Para o primeiro, identificamos a MOEDA('BTC') depois, identificamos a INFORMAÇÃO daquela moeda que queremos('bid')

Vamos dizer que temos interesse em pegar não só a cotação de hoje, mas dos últimos 30 dias?

Como fazer? O primeiro passo, é ler a documentação da API e tentar entender se essa API nos oferece essa opção.

Nesse caso, temos essa opção, o que facilita nosso trabalho. Caso não houvesse, teríamos que programar com os dados disponíveis.

MAS, como temos vamos entender como acessar esses dados no print ao lado.

List comprehension, inserindo o valor da cotação na lista\_cotações\_dólar como float por se tratarem de números

**GET** Retorna o fechamento dos últimos dias

[https://economia.awesomeapi.com.br/json/daily/:moeda,:numero\\_dias](https://economia.awesomeapi.com.br/json/daily/:moeda,:numero_dias)

<https://economia.awesomeapi.com.br/json/daily/USD-BRL/15>

Request Response

Path Parameters

moeda string Código da moeda Ex: USD-BRL

numero\_dias integer Números de dias 30/60/90/.../360 dias

Indica que **moeda** deverá ser substituído pelo código da moeda;  
Indica que **numero de dias** será substituído por um número inteiro (integer)

```
cotacoes_dolar30d = requests.get('https://economia.awesomeapi.com.br/json/daily/USD-BRL/30')
cotacoes_dolar_dic = cotacoes_dolar30d.json()
lista_cotacoes_dolar = [float(item['bid']) for item in cotacoes_dolar_dic]
print(lista_cotacoes_dolar)
```

Aqui estamos pegando os dados de conversão ('USD-BRL') dos últimos ('30') dias

For percorre todo o dicionário disponibilizado pela API

```
[5.7125, 5.7729, 5.6211, 5.5214, 5.5067, 5.49, 5.5602, 5.5832, 5.6244, 5.6156, 5.5518, 5.5357, 5.672, 5.8025, 5.8755, 5.6908, 5.6693, 5.6193, 5.6761, 5.6418, 5.5976, 5.5308, 5.4409, 5.4659, 5.3823, 5.427, 5.4105, 5.3695, 5.3708, 5.3665]
```

Nosso objetivo agora será pegar as informações do BitCoin de Jan/20 a Out/20 e plotarmos um gráfico com essas informações.

Como fizemos anteriormente, vamos verificar na documentação da nossa API se é possível requisitar essas informações diretamente.

Como podemos ver ao lado é possível requisitar e é isso que iremos fazer.

Perceba que o link de requisição possui um número maior de parâmetros, mas todos eles estão listados abaixo na própria API

Basta substituir no link as informações conforme as orientações da API. E é isso que iremos fazer no próximo slide.

**GET** Retorna cotações sequenciais de um período específico

`https://economia.awesomeapi.com.br/:moeda?:quantidade?`

`start_date=20200301&end_date=20200330`

`https://economia.awesomeapi.com.br/USD-BRL/10?start_date=20200201&end_date=20200229`

**Request** **Response**

**Path Parameters**

|                     |        |                              |
|---------------------|--------|------------------------------|
| <code>:moeda</code> | string | Código da moeda ex.: USD-BRL |
| REQUIRED            |        |                              |

|                          |        |                               |
|--------------------------|--------|-------------------------------|
| <code>:quantidade</code> | string | Numero de dados para retornar |
| OPTIONAL                 |        |                               |

**Query Parameters**

|                         |        |  |
|-------------------------|--------|--|
| <code>start_date</code> | string | Data de inicio dos resultados no formato YYYYMMDD<br>ex.: 20200201 |
| OPTIONAL                |        |  |

|                       |        |  |
|-----------------------|--------|--|
| <code>end_date</code> | string | Data limite dos resultados no formato YYYYMMDD ex:<br>20200229 |
| OPTIONAL              |        |  |

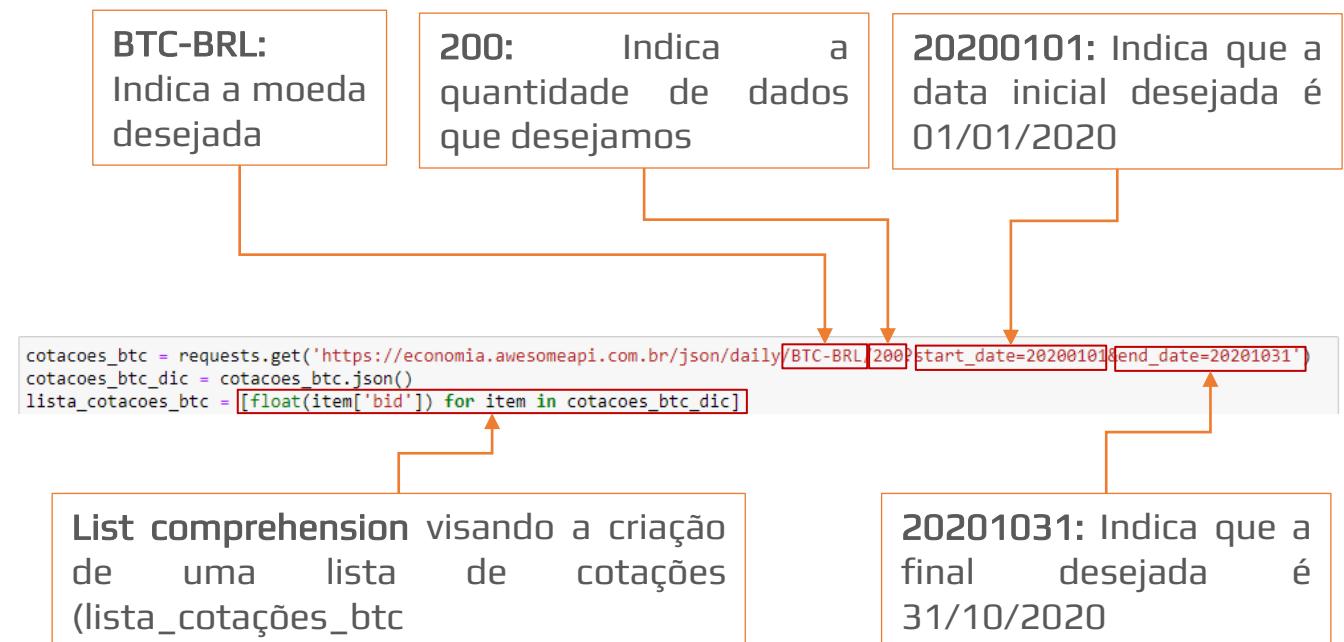
Nosso objetivo agora será pegar as informações do BitCoin de Jan/20 a Out/20 e plotarmos um gráfico com essas informações.

Como fizemos anteriormente, vamos verificar na documentação da nossa API se é possível requisitar essas informações diretamente.

Como podemos ver ao lado é possível requisitar e é isso que iremos fazer.

Perceba que o link de requisição possui um número maior de parâmetros, mas todos eles estão listados abaixo na própria API

Basta substituir no link as informações conforme as orientações da API. E é isso que iremos fazer no próximo slide.



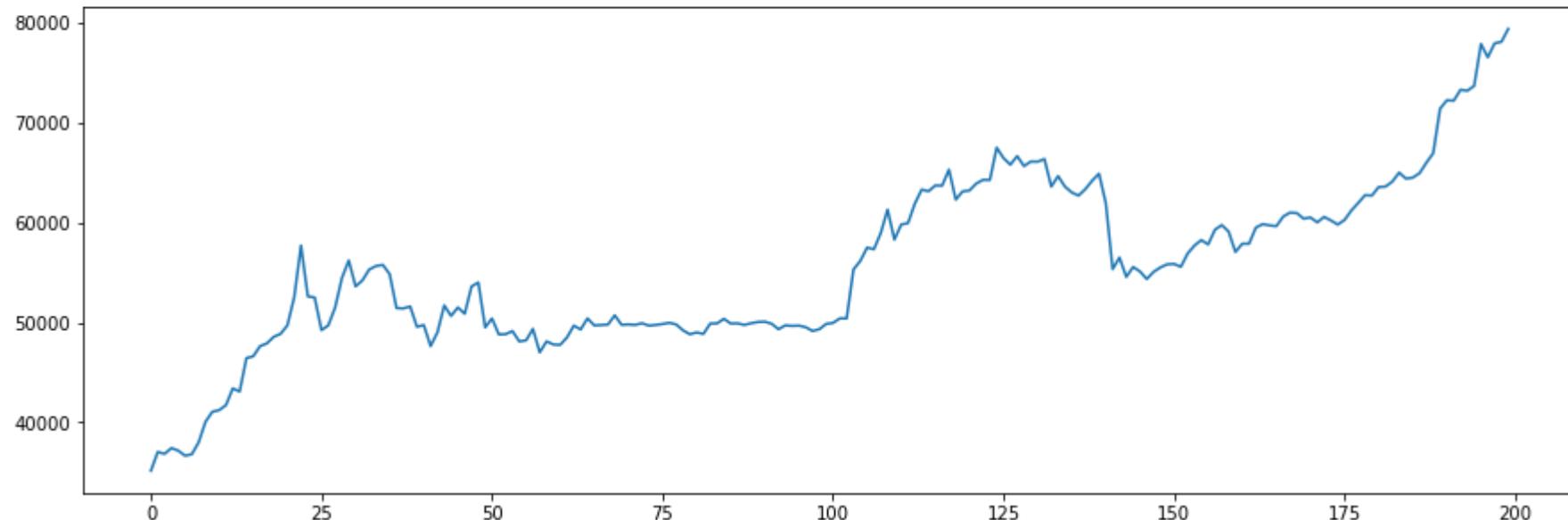
As cotações são extraídas da API da mais atual para a mais antiga. Podemos “corrigir” essa informação usando o método `.reverse()` e ter a informação da mais antiga para a mais nova. Como apresentado abaixo:

```
cotacoes_btc = requests.get('https://economia.awesomeapi.com.br/json/daily/BTC-BRL/200?start_date=20200101&end_date=20201031')
cotacoes_btc_dic = cotacoes_btc.json()
lista_cotacoes_btc = [float(item['bid'])] for item in cotacoes_btc_dic]
lista_cotacoes_btc.reverse()
print(lista_cotacoes_btc)
print(len(lista_cotacoes_btc))

[35150.0, 37025.0, 36836.7, 37415.0, 37158.0, 36652.1, 36800.0, 38000.1, 40050.0, 41050.0, 41212.0, 41715.0, 43392.0, 43070.0,
46424.7, 46610.0, 47626.0, 47900.0, 48553.0, 48852.0, 49700.0, 52500.0, 57700.0, 52609.1, 52499.0, 49225.0, 49700.0, 51488.8, 5
4450.0, 56208.0, 53609.0, 54207.4, 55300.0, 55651.1, 55760.8, 54831.8, 51440.0, 51400.0, 51600.0, 49560.0, 49735.0, 47625.1, 49
000.2, 51700.3, 50650.0, 51500.0, 50865.1, 53600.0, 54001.0, 49500.4, 50400.6, 48800.4, 48820.6, 49133.5, 48096.8, 48201.0, 493
50.0, 47000.0, 48100.1, 47788.9, 47750.0, 48501.4, 49679.0, 49290.6, 50405.0, 49700.0, 49750.0, 49782.0, 50710.0, 49759.1, 498
0.0, 49755.0, 49915.1, 49678.2, 49746.8, 49840.0, 49960.0, 49800.0, 49208.8, 48815.5, 48993.1, 48850.0, 49890.0, 49900.1, 5036
0.1, 49890.0, 49905.9, 49750.0, 49920.0, 50041.3, 50073.4, 49868.0, 49320.0, 49702.1, 49651.8, 49680.0, 49516.5, 49143.0, 4932
1.0, 49850.0, 49951.1, 50400.2, 50402.0, 55300.0, 56161.0, 57490.0, 57330.0, 58950.0, 61298.1, 58296.3, 59800.0, 59951.0, 6190
2.2, 63302.0, 63140.0, 63715.0, 63700.0, 65303.8, 62300.0, 63120.0, 63210.3, 63905.0, 64280.0, 64270.3, 67501.0, 66453.3, 6580
0.0, 66655.0, 65657.4, 66112.0, 66100.0, 66360.9, 63606.0, 64655.2, 63599.9, 63016.2, 62700.0, 63350.9, 64200.0, 64900.0, 6193
9.0, 55336.6, 56500.0, 54555.7, 55550.3, 55102.2, 54348.3, 55059.1, 55511.0, 55821.2, 55856.7, 55566.0, 56908.9, 57700.0, 5825
0.0, 57807.2, 59285.0, 59750.0, 59100.0, 57050.6, 57880.5, 57884.0, 59501.1, 59845.0, 59720.5, 59638.4, 60600.1, 60995.0, 6095
0.1, 60390.7, 60512.5, 60018.1, 60569.2, 60210.3, 59801.0, 60265.0, 61211.5, 61960.0, 62747.2, 62701.0, 63558.3, 63601.7, 6410
0.6, 65000.0, 64400.0, 64500.0, 64950.1, 66000.3, 66952.4, 71401.0, 72251.2, 72202.4, 73300.0, 73200.0, 73685.5, 77888.1, 7655
0.1, 77950.2, 78101.0, 79401.1]
200
```

Em posse dessa lista, podemos utilizá-la na elaboração de um gráfico mostrando a evolução da cotação do bitcoin para o período solicitado.

```
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(15, 5))  
plt.plot(lista_cotacoes_btc)  
plt.show()
```



Vamos usar agora uma API que necessita de um cadastro anterior.

O cadastro é bem direto e pode ser feito diretamente pelo [site do Twilio](#)

Feito o cadastro, vamos dar uma olhada na [documentação da API](#) para entendermos como vamos usá-la para enviar SMS via Python

Lendo a API, vemos que 2 informações são necessárias e ainda não as possuímos.

Vamos então descobrir onde identifica-las no próximo slide.

```
Python Helper Library SMS Test
```

PYTHON

```
1 from twilio.rest import Client
2
3 # Your Account SID from twilio.com/console
4 account_sid = "AC81ab470d064db528e5dc32901289a685"
5 # Your Auth Token from twilio.com/console
6 auth_token = "your_auth_token"
7
8 client = Client(account_sid, auth_token)
9
10 message = client.messages.create(
11     to="+15558675309",
12     from_="+15017250604",
13     body="Hello from Python!")
14
15 print(message.sid)
```

Importando a biblioteca

Precisamos encontrar o nosso account\_sid

Precisamos encontrar o nosso token de autenticação

Na aba Dashboard do Twilio, podemos encontrar o account SID e o Auth Token.

Por motivos de segurança censuramos parte do auth number do pobre coitado que está fazendo essa apostila. Sem a censura já sabemos que muitos SMS irão magicamente ser enviados ☺.

Em posse dessas informações, vamos para o nosso código desenvolver o código conforme indicado na documentação da API do Twilio.

```
1 from twilio.rest import Client  
2  
3 # Your Account SID from twilio.com/console  
4 account_sid = "AC81ab470d064db528e5dc32901289a685"  
5 # Your Auth Token from twilio.com/console  
6 auth_token = "your_auth_token"  
7  
8 client = Client(account_sid, auth_token)
```

The screenshot shows the Twilio dashboard with the title "My first Twilio account Dashboard". It has two main sections: "ACCOUNT SID" and "AUTH TOKEN".  
- In the "ACCOUNT SID" section, the value is "AC81ab470d064db528e5dc32901289a685", with "CENSURADO" in red next to it.  
- In the "AUTH TOKEN" section, the value is "08065522ec923e", with "CENSURADO" in red next to it.  
Callout boxes with arrows point to each section:  
- The first callout box points to the "ACCOUNT SID" section and contains the text: "Precisamos encontrar o nosso account\_sid".  
- The second callout box points to the "AUTH TOKEN" section and contains the text: "Precisamos encontrar o nosso token de autenticação".

```
from twilio.rest import Client  
  
account_sid = 'AC81ab470d064db528e5dc32901289a685'  
token = '08065522ec923e'  
  
client = Client(account_sid, token)
```

Outra informação que precisamos coletar no site do Twilio é o número de telefone que será utilizado para envio dos SMSs.

Para isso, basta clicar na opção **Trial number** e **confirmar**. O Twilio lhe dará um número como apresentado ao lado.

```
from twilio.rest import Client

# Your Account SID from twilio.com/console
account_sid = "AC81ab470d064db528e5dc32901289a685"
# Your Auth Token from twilio.com/console
auth_token = "your_auth_token"

client = Client(account_sid, auth_token)

message = client.messages.create(
    to="+15558675309",
    from_="+15017250604",
```

Project Info

|               |                   |
|---------------|-------------------|
| TRIAL BALANCE | \$14.50           |
| TRIAL NUMBER  | +186452 CENSURADO |

Número criado pelo Twilio para envio dos SMSs

```
from twilio.rest import Client

account_sid = 'AC81ab470d[REDACTED]a685'
token = '08065522ec92[REDACTED]3e'

client = Client(account_sid, token)

remetente = '+186452[REDACTED]'
destino = '+552199[REDACTED]
```

Por fim é só escrevermos nossa mensagem e enviar.

Antes de rodar o código, verifique se a biblioteca do twilio está instalada. Caso não esteja, assim como já fizemos outras vezes basta usar o:

```
pip install twilio
```

```
from twilio.rest import Client

account_sid = 'AC81a' CENSURADO '85'
token = '08065' CENSURADO '3e'

client = Client(account_sid, token)

remetente = '+1864' CENSURADO '
destino = '+552' CENSURADO '

message = client.messages.create(
    to=destino,
    from_=remetente,
    body="Coe, é o Lira aqui!")

print(message.sid)
```



Módulo 25

# Integração Python para Finanças

Atenção!! Este módulo não é sobre finanças ou mercado financeiro.

Nosso objetivo aqui é entender como interagir via Python com ferramentas, bases comumente utilizadas no mercado financeiro.

Essencialmente, o que veremos aqui são métodos ou bibliotecas já conhecidas aplicadas a essa realidade.

Dito isso, vamos começar ☺ !

Como sempre vamos iniciar pela importação de bibliotecas que nos permitirão analisar dados de ativos financeiros.

As importações estão destacadas ao lado.

```
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import pandas_datareader.data as web
```

Além do nosso bom e velho Pandas, vamos importar também pandas\_datareader que nos permite ler dados da web ([doc](#))

Apelido  
normalmente  
utilizado

Para essa integração, usaremos o [Yahoo Finance](#).

Por isso, vai ser necessário conhecer a nomenclatura utilizada por esse site.

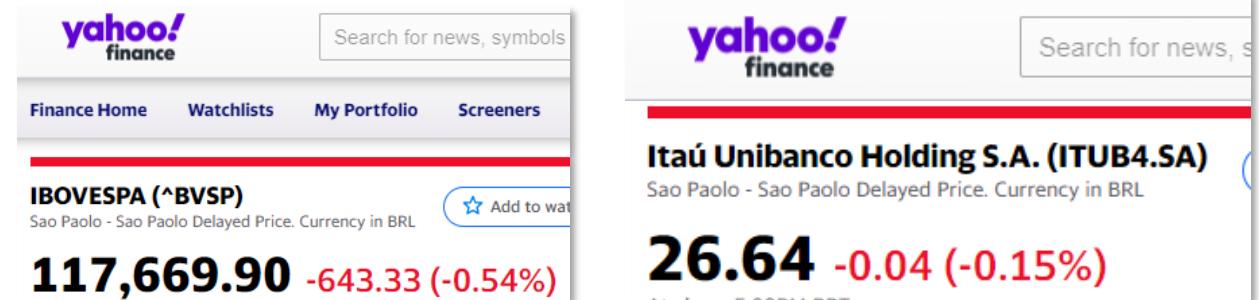
Por exemplo, se queremos saber as cotações da IBOVESPA no Yahoo Finance iremos usar o código ^BVSP.

Outro exemplo, são em casos de ações brasileiras como a ITUB4(nome original usado na Bovespa). No Yahoo Finance, utilizaremos ITUB4.SA.

Sabendo dessas informações, podemos usar nosso primeiro método do pandas\_datareader:

.DataReader()

Esse método nos permite buscar os dados de cotação da web do período solicitado conforme apresentado ao lado.



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader.data as web
cotacao_ibov = web.DataReader('^BVSP', data_source='yahoo', start='2020-1-1', end='2020-11-10')
display(cotacao_ibov)
```

Código do ativo  
data\_source: indica que será puxada do yahoo  
start: início do período desejado;  
end: fim do período desejado

| Date       | High     | Low      | Open     | Close    | Volume    | Adj Close |
|------------|----------|----------|----------|----------|-----------|-----------|
| 2020-01-02 | 118573.0 | 115649.0 | 115652.0 | 118573.0 | 5162700.0 | 118573.0  |
| 2020-01-03 | 118792.0 | 117341.0 | 118564.0 | 117707.0 | 6834500.0 | 117707.0  |
| 2020-01-06 | 117707.0 | 116269.0 | 117707.0 | 116878.0 | 6570000.0 | 116878.0  |

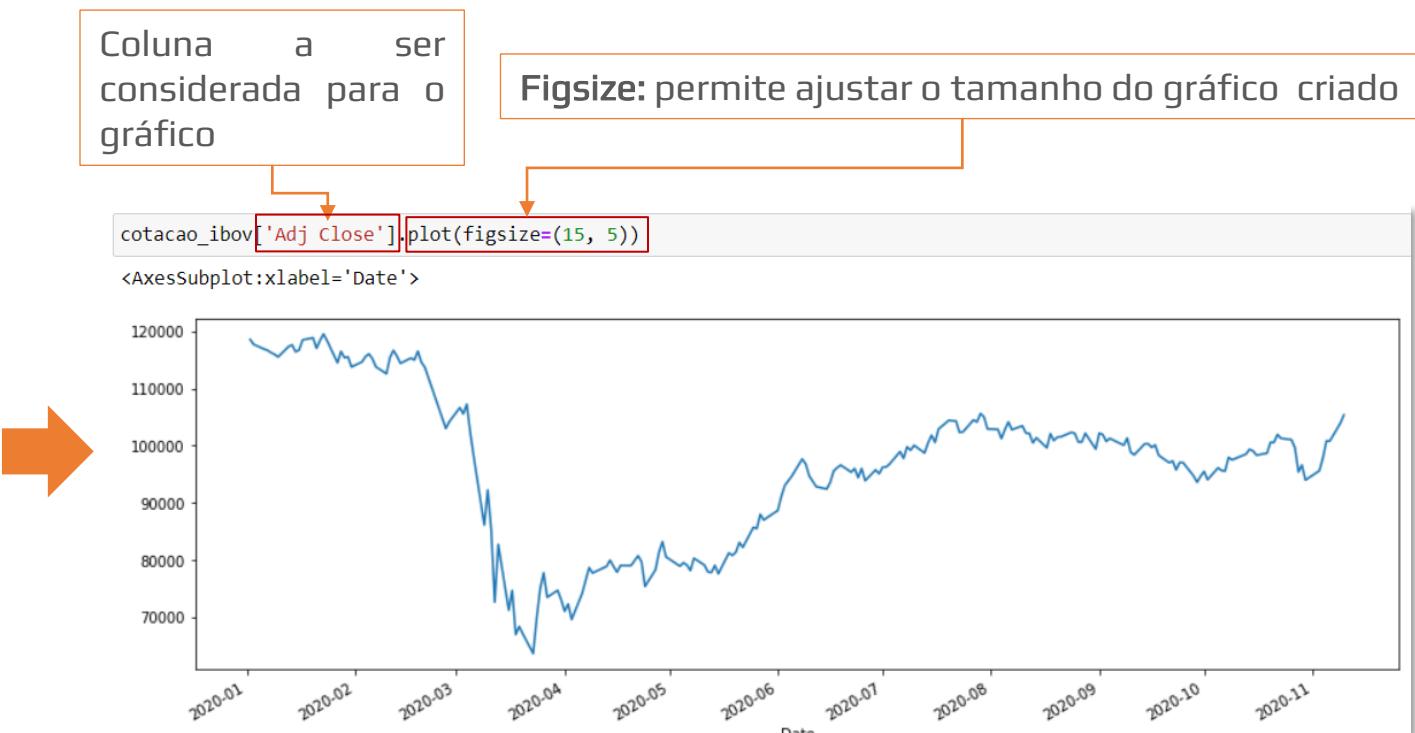
Dados Coletados do Yahoo Finance

Feito essa importação dos dados, podemos trata-los de acordo com nosso interesse. Para isso, usaremos as outras bibliotecas que foram importadas.

No exemplo ao lado fizemos um gráfico usando o **matplotlib** a partir dos dados da coluna 'Adj Close'.

| Date       | High     | Low      | Open     | Close    | Volume     | Adj Close |
|------------|----------|----------|----------|----------|------------|-----------|
| 2020-01-02 | 118573.0 | 115649.0 | 115652.0 | 118573.0 | 5162700.0  | 118573.0  |
| 2020-01-03 | 118792.0 | 117341.0 | 118564.0 | 117707.0 | 6834500.0  | 117707.0  |
| 2020-01-06 | 117707.0 | 116269.0 | 117707.0 | 116878.0 | 6570000.0  | 116878.0  |
| 2020-01-07 | 117076.0 | 115965.0 | 116872.0 | 116662.0 | 4854100.0  | 116662.0  |
| 2020-01-08 | 117335.0 | 115693.0 | 116667.0 | 116247.0 | 5910500.0  | 116247.0  |
| ...        | ...      | ...      | ...      | ...      | ...        | ...       |
| 2020-11-04 | 98296.0  | 95987.0  | 95992.0  | 97811.0  | 10704600.0 | 97811.0   |
| 2020-11-05 | 100922.0 | 97872.0  | 97873.0  | 100774.0 | 10455300.0 | 100774.0  |
| 2020-11-06 | 100928.0 | 99837.0  | 100751.0 | 100799.0 | 8382800.0  | 100799.0  |
| 2020-11-09 | 105147.0 | 100954.0 | 100954.0 | 103913.0 | 17410500.0 | 103913.0  |
| 2020-11-10 | 105758.0 | 103453.0 | 103516.0 | 105351.0 | 16665200.0 | 105351.0  |

214 rows × 6 columns



Tendo as informações extraídas no Yahoo Finance, podemos agora realizar diversas análises.

Vamos começar por uma muito comum e bem simples. O cálculo do retorno da Bovespa para o período definido anteriormente.

Como queremos o retorno do período, vamos calcular o ganho percentual entre a cotação do último dia do período e o primeiro dia do período. O cálculo do retorno será dado por:

$$\text{Retorno} = \frac{\text{Última cotação do período}}{\text{Primeira cotação do período}} - 1$$

Para acessarmos a **última cotação** usaremos a posição **[-1]** da coluna **['Adj Close']**, já para a **primeira cotação** do período usaremos a posição **[0]**.

| Date       | High     | Low      | Open     | Close    | Volume     | Adj Close |
|------------|----------|----------|----------|----------|------------|-----------|
| 2020-01-02 | 118573.0 | 115649.0 | 115652.0 | 118573.0 | 5162700.0  | 118573.0  |
| 2020-01-03 | 118792.0 | 117341.0 | 118564.0 | 117707.0 | 6834500.0  | 117707.0  |
| 2020-01-06 | 117707.0 | 116269.0 | 117707.0 | 116878.0 | 6570000.0  | 116878.0  |
| 2020-11-06 | 100928.0 | 99837.0  | 100751.0 | 100799.0 | 8382800.0  | 100799.0  |
| 2020-11-09 | 105147.0 | 100954.0 | 100954.0 | 103913.0 | 17410500.0 | 103913.0  |
| 2020-11-10 | 105758.0 | 103453.0 | 103516.0 | 105351.0 | 16665200.0 | 105351.0  |

**Posição [0] - Primeira posição**

**Posição [-1] - Última posição**

```
retorno_ibov = cotacao_ibov['Adj Close'][-1] / cotacao_ibov['Adj Close'][0] - 1
print('Retorno de {:.2%}'.format(retorno_ibov))
```

Retorno de -11.15%

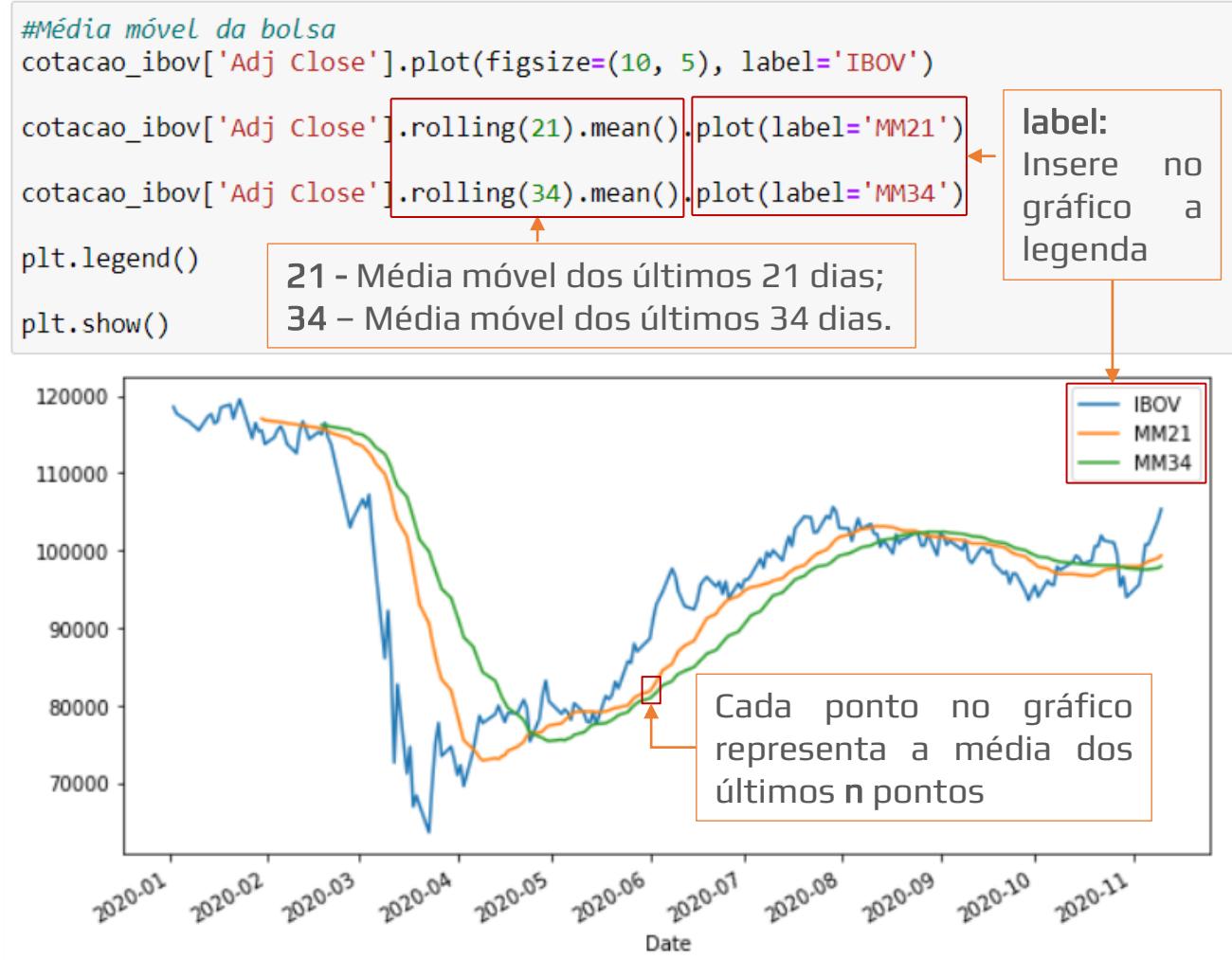
Calculado o retorno, vamos para outra análise muito comum pelos investidores de bolsa de valores. A Média Móvel.

Para isso, usaremos o Pandas. Ao invés de definirmos uma fórmula, usaremos dois métodos já existentes dentro dessa biblioteca:

### rolling().mean()

Ela nos permitirá definir as médias apenas informando o número de pontos que desejamos. Nesse caso, como cada ponto representa um dia, ao definirmos 21 no método rolling, estamos definindo 21 dias.

Veja o exemplo ao lado, além do Pandas também utilizamos o Matplotlib para plotarmos no gráfico as médias móveis definidas por nós.



Para esse exemplo vamos usar uma carteira de investimentos “Carteira do Lira” (obviamente não é a real 😊)

Nosso objetivo é criar um dataframe de todos os ativos da nossa carteira com as cotações do período indicado.

Usaremos um pouco do conhecimento adquirido anteriormente nesse módulo, mas vamos precisar adicionar alguns outros elementos.

Para que o entendimento, fique mais fácil vamos dividir a explicação em X partes:

**Passo 1 :** Importar as bibliotecas e os dados da carteira contidos no arquivo “Carteira.xlsx”;

**Passo 2 :** Criar um dataframe vazio que receberá os ativos e as cotações “puxadas” do Yahoo Finance;

**Passo 3 :** Definir linha de código de preenchimento de apenas 1 dos ativos no Dataframe;

**Passo 4:** Usando o FOR, criar um código que preencha todos os ativos no dataframe criado.

|            | BOVA11     | SMAL11     | MGLU3     | BBDC4     | ITUB4     | ENEV3     | MOVI3     | BPAC11    | GNDI3     | NTCO3     | BCRI11     | VILG11     | KNRI11     |
|------------|------------|------------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|------------|------------|------------|
| Date       |            |            |           |           |           |           |           |           |           |           |            |            |            |
| 2020-01-02 | 114.239998 | 139.500000 | 12.010077 | 37.109173 | 37.184467 | 43.860001 | 19.449736 | 74.693604 | 71.413658 | 38.330002 | 126.190002 | 149.000000 | 199.600006 |
| 2020-01-03 | 113.800003 | 140.899994 | 11.902953 | 37.126244 | 36.793362 | 44.270000 | 19.966278 | 74.910362 | 72.749046 | 41.000000 | 127.699997 | 147.000000 | 199.600006 |
| 2020-01-06 | 112.589996 | 139.100006 | 11.912691 | 36.463108 | 36.245815 | 45.400002 | 20.040001 | 74.801979 | 69.749390 | 40.130001 | 126.699997 | 148.559998 | 197.979996 |
| 2020-01-07 | 112.239998 | 139.399994 | 11.878607 | 35.829655 | 35.404934 | 45.060001 | 20.000000 | 75.100418 | 67.367607 | 40.779999 | 126.870003 | 143.830002 | 198.750000 |
| 2020-01-08 | 111.949997 | 138.199997 | 12.243802 | 35.275387 | 34.828045 | 44.849998 | 20.100000 | 74.434845 | 67.766235 | 40.990002 | 124.570000 | 139.470001 | 187.500000 |
| ...        | ...        | ...        | ...       | ...       | ...       | ...       | ...       | ...       | ...       | ...       | ...        | ...        | ...        |
| 2020-11-04 | 94.419998  | 112.459999 | 26.190001 | 20.680000 | 25.000000 | 57.599998 | 18.540001 | 75.139999 | 71.790001 | 47.669998 | 107.500000 | 126.970001 | 158.380005 |
| 2020-11-05 | 96.699997  | 115.580002 | 27.450001 | 21.000000 | 25.590000 | 58.709999 | 19.620001 | 78.459999 | 72.720001 | 50.279999 | 107.500000 | 128.380005 | 158.179993 |
| 2020-11-06 | 97.190002  | 117.510002 | 27.000000 | 20.900000 | 25.670000 | 59.700001 | 20.290001 | 79.320000 | 72.820000 | 50.419998 | 107.489998 | 129.009995 | 158.449997 |
| 2020-11-09 | 99.769997  | 120.110001 | 26.450001 | 22.940001 | 27.629999 | 57.580002 | 19.290001 | 78.739998 | 72.099998 | 49.150002 | 107.500000 | 128.210007 | 158.149994 |
| 2020-11-10 | 101.449997 | 120.510002 | 25.219999 | 24.420000 | 28.879999 | 57.169998 | 19.620001 | 76.290001 | 69.360001 | 48.450001 | 107.500000 | 128.520004 | 159.500000 |

210 rows × 14 columns

Passo 1 : Importar as bibliotecas e os dados da carteira contidos no arquivo “Carteira.xlsx”;

Assim como nos exemplos anteriores deste módulo, usaremos o pandas.reader para a busca de informações no site do Yahoo Finance e o pandas para o tratamento dos dados dentro de um dataframe.

Além disso, usaremos posteriormente o numpy e matplotlib para elaboração de gráficos.

Como já vimos no módulo sobre PANDAS, para importarmos as informações de um Excel, usaremos o método .read\_excel().

O exemplo ao lado, apresenta o código em Python para a conclusão do PASSO 1.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pandas_datareader.data as web

carteira = pd.read_excel('Carteira.xlsx')
display(carteira)
```

Informações  
do arquivo  
importadas  
excel e  
armazenados  
no  
dataframe 'carteira'.

|    | Ativos | Tipo | Qtde |
|----|--------|------|------|
| 0  | BOVA11 | ETF  | 100  |
| 1  | SMAL11 | ETF  | 100  |
| 2  | MGLU3  | Ação | 1000 |
| 3  | BBDC4  | Ação | 100  |
| 4  | ITUB4  | Ação | 100  |
| 5  | ENEV3  | Ação | 300  |
| 6  | MOVI3  | Ação | 100  |
| 7  | BPAC11 | Ação | 100  |
| 8  | GNDI3  | Ação | 100  |
| 9  | NTCO3  | Ação | 100  |
| 10 | BCRI11 | FII  | 100  |
| 11 | VILG11 | FII  | 100  |
| 12 | KNRI11 | FII  | 100  |
| 13 | XPLG11 | FII  | 100  |

Passo 2 : Criar um dataframe vazio que receberá os ativos e as cotações “puxadas” do Yahoo Finance;

Esta é a linha menos intuitiva de todas, então vamos avaliá-la por partes.

Se usarmos essencialmente o mesmo código utilizado anteriormente, conseguimos buscar as cotações, mas além disso, também obtemos uma série de informações que não nos interessa por enquanto (high, low, open, close, volume).

De todas as informações apresentadas ao lado, apenas a última coluna nos interessa e além disso, pensando que vamos ter vários ativos no nosso dataframe, será importante renomeá-lo de acordo com o nome do ativo. Nesse caso ‘BOVA11.SA’.

Vamos entender como fazer essa etapa na próxima página ☺.

```
cotacoes_carteira = pd.DataFrame()  
  
cotacoes_carteira = web.DataReader('BOVA11.SA', data_source='yahoo', start='2020-01-01', end='2020-11-10')  
  
display(cotacoes_carteira)
```

| Date       | High       | Low        | Open       | Close      | Volume     | Adj Close  |
|------------|------------|------------|------------|------------|------------|------------|
| 2020-01-02 | 114.239998 | 112.129997 | 112.449997 | 114.239998 | 5684380.0  | 114.239998 |
| 2020-01-03 | 114.500000 | 112.800003 | 112.930000 | 113.800003 | 6602450.0  | 113.800003 |
| 2020-01-06 | 113.449997 | 112.019997 | 113.000000 | 112.589996 | 6771940.0  | 112.589996 |
| 2020-01-07 | 112.900002 | 111.589996 | 112.900002 | 112.239998 | 6096900.0  | 112.239998 |
| 2020-01-08 | 113.099998 | 111.400002 | 112.650002 | 111.949997 | 6472610.0  | 111.949997 |
| ...        | ...        | ...        | ...        | ...        | ...        | ...        |
| 2020-11-04 | 94.629997  | 92.849998  | 93.510002  | 94.160004  | 12844442.0 | 94.160004  |
| 2020-11-05 | 97.169998  | 95.379997  | 95.769997  | 96.949997  | 12692233.0 | 96.949997  |
| 2020-11-06 | 97.339996  | 96.089996  | 96.089996  | 97.190002  | 8765724.0  | 97.190002  |
| 2020-11-09 | 101.150002 | 99.269997  | 100.389999 | 99.500000  | 19613783.0 | 99.500000  |
| 2020-11-10 | 101.820000 | 99.730003  | 99.930000  | 101.000000 | 13893583.0 | 101.000000 |

214 rows × 6 columns

Passo 3 : Definir linha de código de preenchimento de apenas 1 dos ativos no Dataframe;

Para facilitar a visualização desta etapa, criamos uma variável 'extracao' que armazenará o dataframe apresentado anteriormente.

Como vimos, queremos que no nosso dataframe `cotacoes_carteira` exista uma coluna com o nome do ativo ['BOVA11.SA'] mas que apenas a coluna ['Adj Close'] seja utilizada.

O exemplo ao lado, apresenta como realizar esta etapa.



## ATENÇÃO !

Por fins didáticos, dividimos essa linha em 2 etapas, mas seria possível realizar essa etapa diretamente sem a necessidade da criação da variável `extracao`.

```
cotacoes_carteira = pd.DataFrame()  
  
extracao = web.DataReader('BOVA11.SA', data_source='yahoo', start='2020-01-01', end='2020-11-10')  
cotacoes_carteira['BOVA11.SA'] = extracao['Adj Close']  
display(cotacoes_carteira)
```

**BOVA11.SA**

| Date       | BOVA11.SA  |
|------------|------------|
| 2020-01-02 | 114.239998 |
| 2020-01-03 | 113.800003 |
| 2020-01-06 | 112.589996 |
| 2020-01-07 | 112.239998 |
| 2020-01-08 | 111.949997 |
| ...        | ...        |
| 2020-11-04 | 94.160004  |
| 2020-11-05 | 96.949997  |
| 2020-11-06 | 97.190002  |
| 2020-11-09 | 99.500000  |
| 2020-11-10 | 101.000000 |

214 rows × 1 columns

Retira apenas a coluna ['Adj Close'] do dataframe `extracao` criado.

Cria a coluna [BOVA11.SA] que receberá os valores das cotações.

Passo 4: Usando o FOR, criar um código que preencha todos os ativos no dataframe criado.

Como temos vários ativos, podemos utilizar o FOR para percorrer todos esses ativos adicionando seus dados ao nosso dataframe `cotações_carteira`.

```
cotações_carteira = pd.DataFrame()

for ativo in carteira['Ativos']:
    extracão = web.DataReader('{}.SA'.format(ativo), data_source='yahoo', start='2020-01-01', end='2020-11-10')
    cotações_carteira[ativo] = extracão['Adj Close']

display(cotações_carteira)
```

For permite percorrer toda lista de ativos existente no dataframe `carteira`

display(`cotações_carteira`)

Ao invés de utilizar o nome do ativo, é utilizado a variável `ativo` que possui o nome correspondente a aquela iteração.

| Date       | BOVA11     | SMAL11     | MGLU3     | BBDC4     | ITUB4     | ENEV3   | MOVI3     | BPAC11    | GNDI3     | NTCO3     | BCRI11     |      |
|------------|------------|------------|-----------|-----------|-----------|---------|-----------|-----------|-----------|-----------|------------|------|
| 2020-01-02 | 114.239998 | 139.500000 | 12.038459 | 33.042130 | 36.671249 | 10.9650 | 19.307058 | 74.173401 | 71.413658 | 38.330002 | 126.190002 | 149. |
| 2020-01-03 | 113.800003 | 140.899994 | 11.931082 | 33.058857 | 36.285545 | 11.0675 | 19.819813 | 74.388657 | 72.749046 | 41.000000 | 127.699997 | 147. |
| 2020-01-06 | 112.589996 | 139.100006 | 11.940844 | 32.468361 | 35.745567 | 11.3500 | 19.892994 | 74.281029 | 69.749390 | 40.130001 | 126.699997 | 148. |
| 2020-01-07 | 112.239998 | 139.399994 | 11.906678 | 31.904306 | 34.916294 | 11.2650 | 19.853287 | 74.577377 | 67.367607 | 40.779999 | 126.870003 | 143. |
| 2020-01-08 | 111.949997 | 138.199997 | 12.272737 | 31.410761 | 34.347374 | 11.2125 | 19.952553 | 73.916443 | 67.766235 | 40.990002 | 124.570000 | 139. |

Ativo existente na carteira e todas as cotações do período extraídas do Yahoo Finance.

Assim como vimos no módulo do Pandas, é sempre importante analisarmos a qualidade dos dados antes de iniciar a análise dos dados.

Um método já conhecido por nós que nos permite ter uma visão geral do dados é o método `.info()` conforme apresentado ao lado.

Podemos perceber que aparentemente todos os dados foram trazidos corretamente, com exceção do ativo XPLG11. Nesse caso temos apenas 176 dados não nulos enquanto os outros ativos, 210.

Possivelmente algum problema ocorreu na própria base do Yahoo Finance. Nesse caso, não temos muito o que fazer além de tratarmos esses dados faltantes usando alguma premissa.

```
cotacoes_carteira.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 210 entries, 2020-01-02 to 2020-11-10
Data columns (total 14 columns):
 #   Column   Non-Null Count Dtype  
--- 
 0   BOVA11   210 non-null   float64 
 1   SMAL11   210 non-null   float64 
 2   MGLU3    210 non-null   float64 
 3   BBDC4    210 non-null   float64 
 4   ITUB4    210 non-null   float64 
 5   ENEV3    210 non-null   float64 
 6   MOVI3    210 non-null   float64 
 7   BPAC11   210 non-null   float64 
 8   GNDI3    210 non-null   float64 
 9   NTC03   210 non-null   float64 
 10  BCRI11   210 non-null   float64 
 11  VILG11   210 non-null   float64 
 12  KNRI11   210 non-null   float64 
 13  XPLG11   176 non-null   float64 
dtypes: float64(14)
memory usage: 24.6 KB
```

Valores faltantes no ativo XPLG11. Era esperado 210 assim como os outros ativos.

Para o ajuste destes dados faltantes, poderíamos usar algumas premissas. Segue abaixo a listagem de algumas mais comuns :

- Excluir dados faltantes;
- Utilizar a média de todos ou uma dos valores;
- Replicar o dado anterior existente mais próximo.

Não existe uma abordagem 100% correta, qualquer uma que venha a ser adotada será uma premissa utilizada e qual utilizar irá variar de projeto para projeto.

No nosso caso, usaremos o valor existente imediatamente anterior ao dado faltante. Ou seja, O valor da cotação do dia anterior ao do dado faltante.

Para isso, usaremos o método `.ffill()` que nos permite preencher os espaços vazios com os dados existentes na linha anterior.

```
cotacoes_carteira = cotacoes_carteira.ffill()
cotacoes_carteira.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 210 entries, 2020-01-02 to 2020-11-10
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   BOVA11   210 non-null    float64
 1   SMAL11   210 non-null    float64
 2   MGLU3    210 non-null    float64
 3   BBDC4    210 non-null    float64
 4   ITUB4    210 non-null    float64
 5   ENEV3    210 non-null    float64
 6   MOVI3    210 non-null    float64
 7   BPAC11   210 non-null    float64
 8   GNDI3    210 non-null    float64
 9   NTCO3    210 non-null    float64
 10  BCRI11   210 non-null    float64
 11  VILG11   210 non-null    float64
 12  KNRI11   210 non-null    float64
 13  XPLG11   210 non-null    float64
dtypes: float64(14)
memory usage: 24.6 KB
```

Apesar de não serem dados reais, a premissa nos permite igualar a quantidade de dados existentes.

Para conseguirmos analisar nossos ativos em um mesmo gráfico é interessante normalizarmos os valores.

O que isso significa? Vamos considerar o primeiro valor conhecido da nossa base de ativos como a linha de base. Todas as variações (positivas ou negativas) serão referencias a essa linha de base. Isso nos permitirá avaliar a evolução de todos os ativos em um mesmo gráfico independentemente do seus valores serem, em valores absolutos, muito distintos.

Como normalizar? Considerando nosso caso, vamos criar um novo dataframe que será:

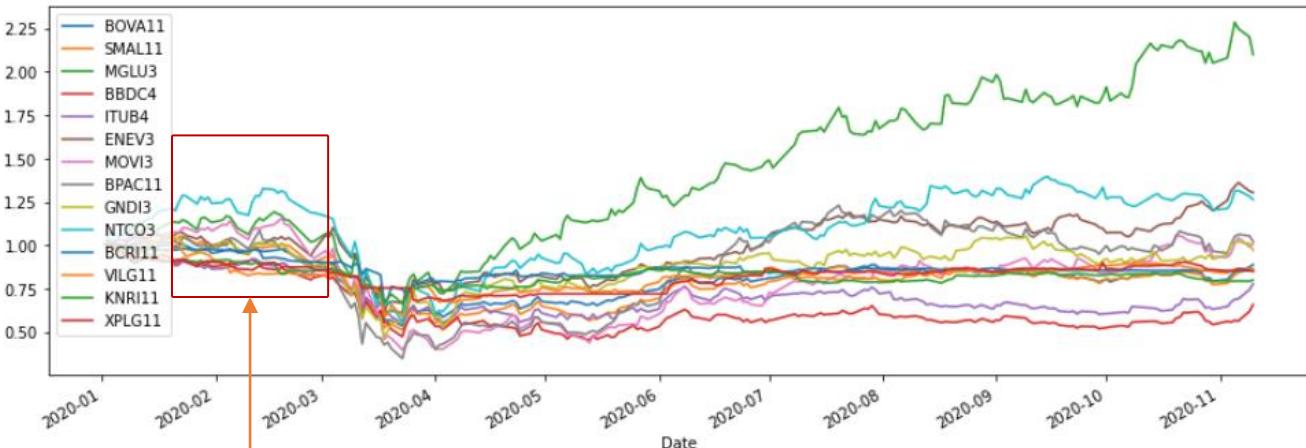
*Valor da cotação do dia*

*Valor da cotação do 1º dia da base de dados*

Chamaremos esse novo dataframe de **carteira\_norm** conforme apresentado ao lado.

```
carteira_norm = cotacoes_carteira / cotacoes_carteira.iloc[0]  
carteira_norm.plot(figsize=(15, 5))  
plt.legend(loc='upper left')
```

Cotação do 1º dia. `Iloc[0]` significa que estamos pegando a primeira linha do nosso dataframe



Os dados normalizados nos permitem ver a evolução em % dos ativos. Perceba que todos valores se iniciam com valor no eixo y = 1.

Sem a normalização a diferença entre valores absolutos dificultaria muito a análise conjunta em um único gráfico

A primeira parte é a mesma que já fizemos anteriormente. Puxar do Yahoo Finance usando o `pandas.reader`(definido como `web` nesse exemplo) para realizar a importação dos dados.

Nosso objetivo é comparar a rentabilidade entre a Bovespa e nossa carteira. Já aprendemos como calcular essa rentabilidade anteriormente, mas consideramos que todos os ativos possuíam a mesma quantidade, o que dificilmente é realidade em um caso real.

Portanto, vamos precisar incorporar a quantidade dos ativos a nossa análise.

Para isso, vamos criar um novo dataframe chamado `valor_investido` conforme apresentado ao lado, onde a quantidade do ativo apresentada no dataframe `carteira` é multiplicada pelo valor da cotação do ativo no dataframe `cotacoes_carteira`.

Vamos entender um pouco melhor este código na próxima página

| Date       | High     | Low      | Open     | Close    | Volume     | Adj Close |
|------------|----------|----------|----------|----------|------------|-----------|
| 2020-01-02 | 118573.0 | 115649.0 | 115652.0 | 118573.0 | 5162700.0  | 118573.0  |
| 2020-01-03 | 118792.0 | 117341.0 | 118564.0 | 117707.0 | 6834500.0  | 117707.0  |
| 2020-01-06 | 117707.0 | 116269.0 | 117707.0 | 116878.0 | 6570000.0  | 116878.0  |
| 2020-01-07 | 117076.0 | 115965.0 | 116872.0 | 116662.0 | 4854100.0  | 116662.0  |
| 2020-01-08 | 117335.0 | 115693.0 | 116667.0 | 116247.0 | 5910500.0  | 116247.0  |
| ...        | ...      | ...      | ...      | ...      | ...        | ...       |
| 2020-11-04 | 98296.0  | 95987.0  | 95992.0  | 97811.0  | 10704600.0 | 97811.0   |
| 2020-11-05 | 100922.0 | 97872.0  | 97873.0  | 100774.0 | 10455300.0 | 100774.0  |

Vamos utilizar a mesma estrutura de for utilizada anteriormente para “preencher” nosso novo dataframe `valor_investido`. A parte que é menos intuitiva nesse caso é como buscar as informações do nosso dataframe `carteira`. Como podemos ver no extrato apresentado abaixo, a orientação do nosso dataframe é diferente. Os ativos não são colunas e sim parte dos dados que são apresentados em linhas.

Assim, usaremos o `.loc` para acessar os dados. A informação desejada, SEMPRE, estará na coluna `Qtde`, e o que nos permite chegar no valor certo da coluna e o nome do ativo. Assim, usaremos a estrutura abaixo para acessar os dados.

```
valor_investido = pd.DataFrame()

for ativo in carteira['Ativos']:
    valor_investido[ativo] = cotacoes_carteira[ativo] * carteira.loc[carteira['Ativos']==ativo, 'Qtde'].values[0]
display(valor_investido)
```

`carteira`: nome do dataframe;  
`carteira['Ativos']==ativo`: o valor da coluna ‘Ativos’ do df `carteira` que for IGUAL ao valor de ativo nesta iteração do for;  
`‘Qtde’`: Coluna que desejamos  
`.values[0]` = indica que queremos o valor do df conforme as condições dos itens anteriores.

|   | Ativos | Tipo | Qtde |
|---|--------|------|------|
| 0 | BOVA11 | ETF  | 100  |
| 1 | SMAL11 | ETF  | 100  |
| 2 | MGLU3  | Ação | 1000 |
| 3 | BBDC4  | Ação | 100  |

BOVA11      SMAL11      MGLU3      BBDC4      ITUB4      ENEV3      MOVI3      BPAC11      GNDI3      NTCO3      BCRI11

Date

2020-01-02 11423.999786 13950.000000 12010.077477 3710.917282 3718.446732 13158.000183 1944.973564 7469.360352 7141.365814 3833.000183 12619.000244

↑  
Valores de cotação multiplicados pela quantidade do ativo

Agora que temos o valor investido por carteira, vamos descobrir o valor total da carteira.

Para isso, vamos criar uma coluna de **TOTAL** no nosso dataframe `valor_investido`.

Vamos calculá-la utilizando o método `.sum()`. No entanto, esse método por padrão soma colunas e não linhas... Vamos precisar fornecer um argumento para especial para nosso método que nos permita somar as linhas que representarão o valor total da carteira/dia.

Esse argumento será `axis=1`.

```
valor_investido['Total'] = valor_investido.sum(axis=1)
```

Criação de uma coluna ['Total'] no dataframe `valor_investido`

`axis=1` permite somar linhas ao invés de colunas (`axis=0`) que é a configuração padrão do método `sum()`.

Criada nossa coluna Total, podemos replicar os conhecimentos das páginas anteriores realizando as etapas abaixo:

- 1) Normalizar valores para plotagem dos gráficos;
- 2) Plotar gráfico acrescentando as labels necessárias para identificação;
- 3) Utilizando a fórmula abaixo, calcular o retorno:

$$\text{Retorno} = \frac{\text{Última cotação do período}}{\text{Primeira cotação do período}} - 1$$

Após a realização das etapas acima teremos o gráfico apresentado ao lado assim como os retornos indicados na parte inferior.

Podemos perceber que os gráficos parecem relacionados e vamos aproveitar isso para aprender mais um método do PANDAS, o .corr().

```
valor_investido['Total'] = valor_investido.sum(axis=1)

valor_investido_norm = valor_investido / valor_investido.iloc[0]
cotacao_ibov_norm = cotacao_ibov / cotacao_ibov.iloc[0]

valor_investido_norm['Total'].plot(figsize=(15, 5), label='Carteira')
cotacao_ibov_norm['Adj Close'].plot(label='IBOV')
plt.legend()
plt.show()
```



```
retorno_carteira = valor_investido['Total'][-1] / valor_investido['Total'][0] - 1
retorno_ibov = cotacao_ibov['Adj Close'][-1] / cotacao_ibov['Adj Close'][0] - 1
print('Retorno da Carteira: {:.2%}'.format(retorno_carteira))
print('Retorno IBOV: {:.2%}'.format(retorno_ibov))
```

Retorno da Carteira: 1.93%  
Retorno IBOV: -11.15%

Retorno da carteira e Bovespa para o período estudado.

Os valores de correlação, variam sempre entre -1 e 1.

Valores próximos de 1 indicam que existe uma relação forte e diretamente proporcional, ou seja, se um índice sobe o outro também sobe.

Valores próximos de -1 indicam que existe uma relação forte mas inversamente proporcional, ou seja, se um índice sobe o outro desce.

Na imagem ao lado apresentamos a correlação alcançada no nosso exemplo utilizando o método `.corr()`.

Como já era esperado pela nossa análise gráfica, podemos ver que esta correlação é bastante forte e de proporcionalidade positiva.

Compara a coluna 'Adj Close' com a variação o valor de fechamento da carteira calculado na coluna 'Total'

```
correlacao = valor_investido['Total'].corr(cotacao_ibov['Adj Close'])  
print(correlacao)  
0.8879301020686045
```

# Módulo 26

# Ambientes Virtuais

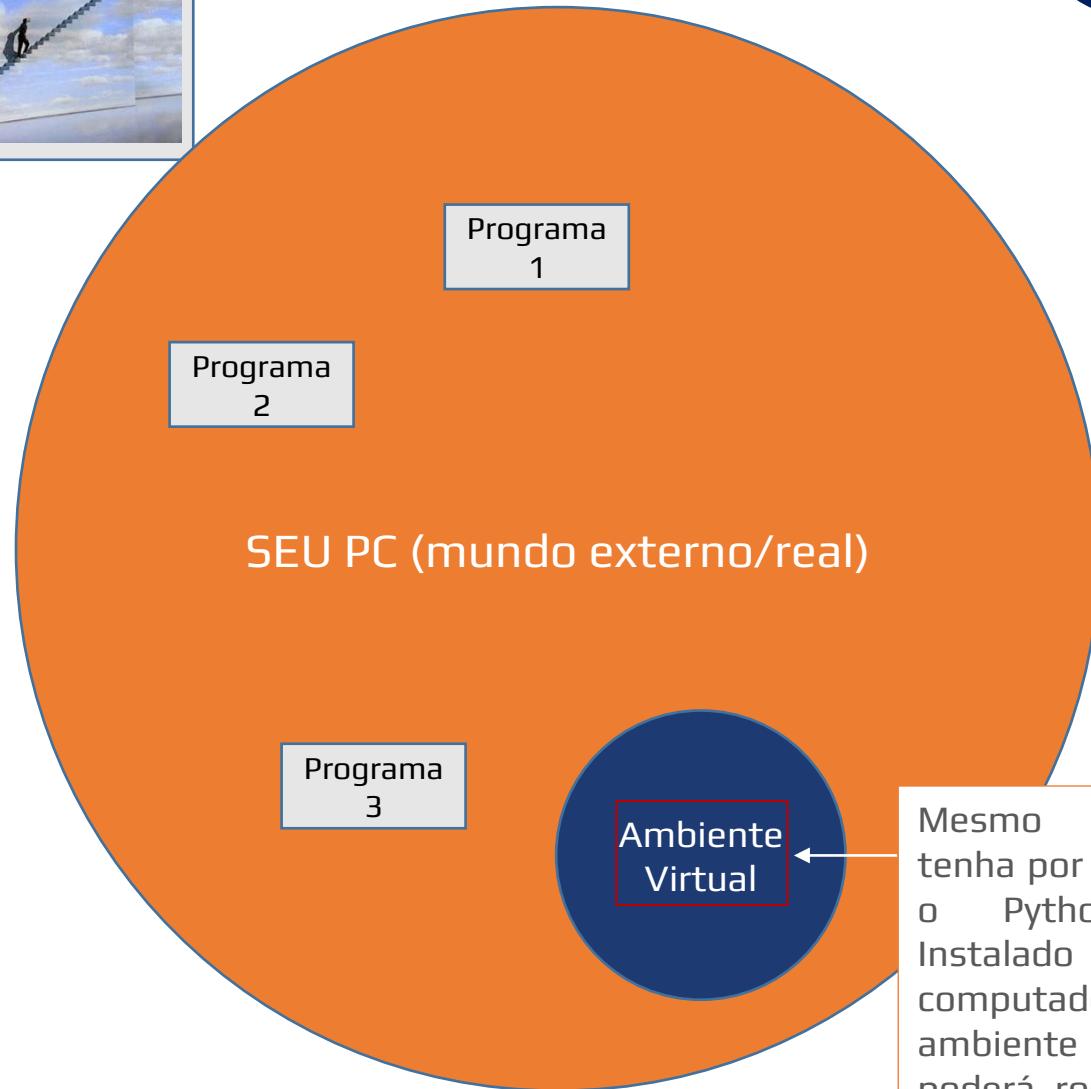
Caso você goste de filmes/séries(**SPOILER ALERT**), imagine um ambiente virtual como o mundo do [Show de Truman](#) ou a cidade criada pela Wanda de [WandaVision](#).



O que acontece nesses casos é que são criadas bolhas dentro do mundo real. Lá, as regras são outras, as condutas são outras sem a interferência do que está fora da bolha (mundo real).

O que isso significa em termos de programação e Python?

O ambiente virtual quando criado, não recebe por definição os programas instalados do seu computador(mundo externo), dentro desse ambiente, você poderá instalar outros programas, bibliotecas, até mesmo versões mais antigas do Python que só existirão dentro daquele ambiente.



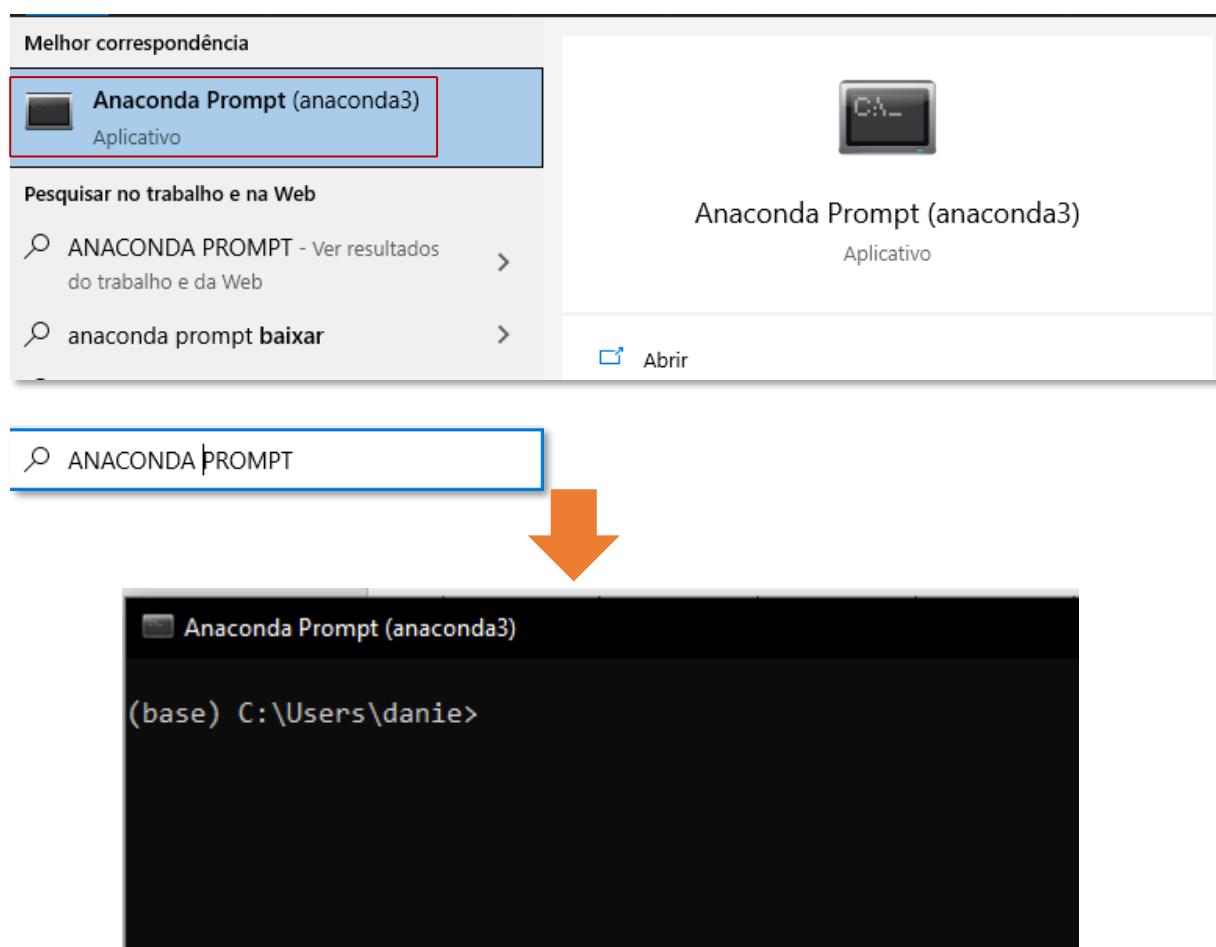
Mesmo que se tenha por exemplo o Python 3.8 Instalado no seu computador, seu ambiente Virtual poderá rodar uma versão anterior.

Imagino que esteja pensando... OK... E daí? Para que serve, de que se alimenta e no fim das contas como que eu consigo usar esse tal de Ambiente Virtual?

Tudo se inicia no **PROMPT DE COMANDO** talvez você não se lembre, mas já acessamos o prompt outras vezes durante o curso... É aquela tela preta cheia de texto que as vezes usamos para instalar uma biblioteca através do pip lembra?

Se não lembra, não tem problema, vamos entender primeiro como abrir o prompt e depois como navegar até conseguirmos criar nosso ambiente virtual.

Para acessar o prompt basta escrever no seu menu iniciar **ANACONDA PROMPT** e clicar no ícone indicado ao lado.



Vamos entender o que estamos vendo no Prompt.

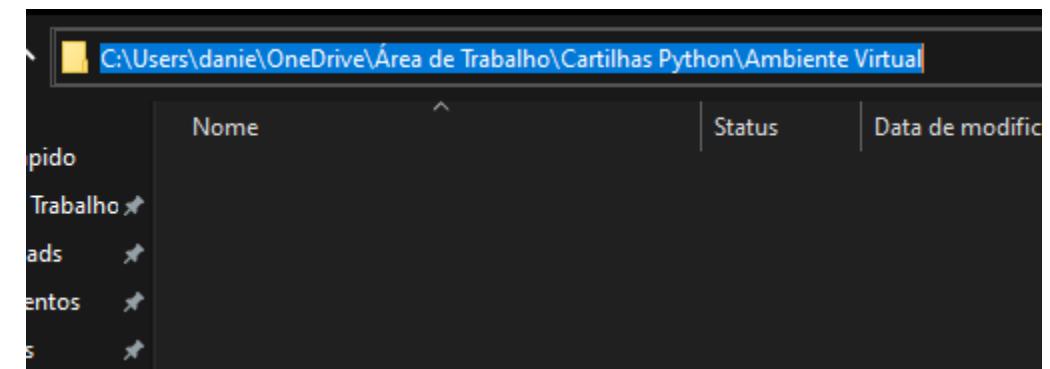
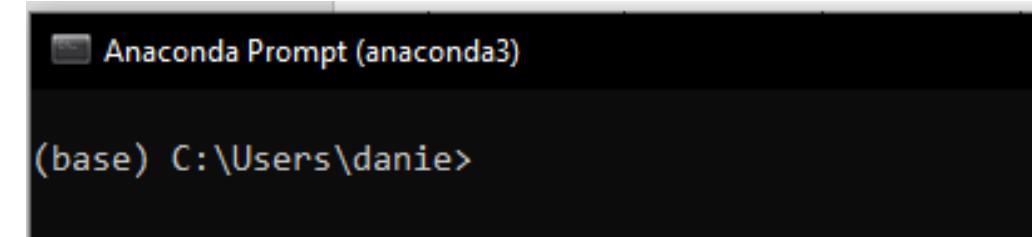
(base) -> Indica que estamos trabalhando no ambiente base do Anaconda que está sendo aplicado a todo o Anaconda;

C:\Users\danie> -> Indica em que pasta do nosso PC estamos.

Agora vamos definir onde gostaríamos de criar nosso ambiente virtual (nossa “bolha”). No nosso caso, vamos criar dentro de uma pasta do nosso curso.

No caso do rapaz que escreve essa apostila, esse ambiente está dentro do caminho assinalado na imagem ao lado. NO ENTANTO, VOCÊ PODERÁ ESCOLHER O LOCAL!

Escolhido o lugar, vamos precisar navegar através do nosso prompt até a pasta escolhida. Para isso, vamos precisar aprender como navegar no prompt. Essa navegação é feita exclusivamente pelo TECLADO através de comandos. É isso que vamos ver na próxima página



**DIR:**

- Esse comando nos permite acessar todos os arquivos existentes na pasta onde ele é acionado. Portanto, ao usar `dir` estamos listando tudo que existe na pasta **Cartilhas Python**.

**CD (LS para MAC e LINUX):**

- Esse comando nos permite abrir pastas do nosso computador. As pastas são identificadas pelo `<DIR>`. Como é de nosso interesse entrar na pasta **Ambiente Virtual**, basta usarmos o comando `cd Ambiente Virtual`.
- O comando `cd..` Permitirá voltar para a pasta mãe da pasta em que se está. Ou seja, se estivermos dentro de **Ambiente Virtual** e usarmos `cd ..`, iremos **VOLTAR** para a pasta Cartilhas Python

```
(base) C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python>dir
0 volume na unidade C é Windows
0 Número de Série do Volume é 7234-2414

Pasta de C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python

11/04/2021 16:01    <DIR>      .
11/04/2021 16:01    <DIR>      ..
24/03/2021 13:34    <DIR>      .ipynb_checkpoints
11/04/2021 16:01    <DIR>      Ambiente Virtual
04/04/2021 23:17    <DIR>      Arquivos apostila
23/03/2021 22:48          3.945 Gabarito - 07.07.02 Exercícios Listas.ipynb
23/03/2021 22:41          2.539 Gabarito - Dicionario 06.ipynb
23/03/2021 22:09          2.939 Gabarito - While 01.ipynb
24/03/2021 13:48    <DIR>      Suporte
25/03/2021 01:56          795 Untitled.ipynb
                           4 arquivo(s)   10.218 bytes
                           6 pasta(s)  845.870.264.320 bytes disponíveis
```

```
Área de Trabalho\Cartilhas Python>cd "Ambiente Virtual"
Área de Trabalho\Cartilhas Python\Ambiente Virtual>cd..
Área de Trabalho\Cartilhas Python>cd "Ambiente Virtual"
Área de Trabalho\Cartilhas Python\Ambiente Virtual>
```

Abrindo a pasta Ambiente Virtual

Volta para a pasta acima de Ambiente Virtual (Cartilhas Python)

Abrindo novamente a pasta Ambiente Virtual

Indica que a pasta 'Ambiente Virtual' está ativa e aguardando comando

Vamos agora para a criação do nosso ambiente virtual.

Aqui, vamos focar nos comandos mais usados e logo, mais importantes, mas caso queira se aprofundar sobre o tema, é só clicar aqui na [documentação](#) ☺

Para criação, do nossos ambiente virtual, vamos voltar para o nosso prompt de comando dentro da pasta Ambiente Virtual e usar o comando:

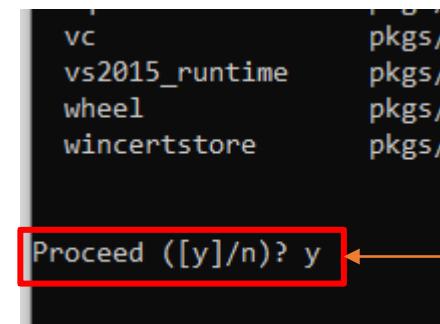
**conda create - n**

No nossos caso, vamos instalar o Python 3.6 não é a versão mais atualizada do Python. Por que faríamos isso? Alguns programas (Ex: ArcGis) por uma questão de atualização, muitas vezes não conseguem ser executados para as versões mais recentes do Python.

Instalando versões mais antigas conseguimos realizar integrações para esses casos.

```
has Python\Ambiente Virtual>conda create -n novoambiente python=3.6
```

conda create: comando de criação de um ambiente virtual;  
-n: indica que daremos um nome personalizado a nosso ambiente virtual;  
novoambiente: nome do nosso ambiente virtual;  
python=3.6: definição de qual versão do Python usaremos dentro desse ambiente virtual.



Após o prompt definir o que será instalado, será perguntado se você deseja seguir ou não. Escreva Y no seu teclado e aperte ENTER

Agora temos nosso ambiente virtual instalado, ,mas como saber se funcionou e já estamos dentro dele?

Se olharmos no nosso prompt ainda estaremos com **(base)** isso significa que ainda não estamos dentro do nosso ambiente.

Para “entrarmos” no ambiente virtual precisamos antes ativá-lo. Para isso, usaremos o comando:

**conda activate novoambiente**

(base) indica que o ambiente virtual por mais que já tenha sido criado, não foi ativado.

Ativando nosso ambiente virtual

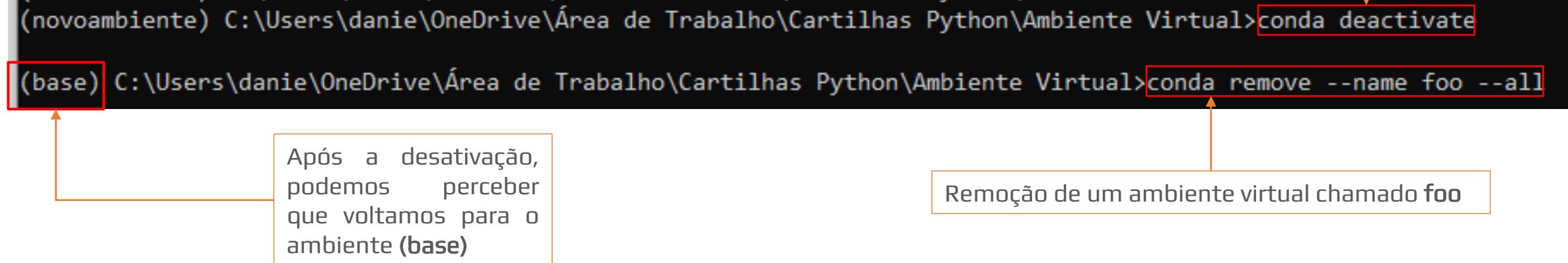
```
(base) C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Ambiente Virtual>conda activate novoambiente  
(novoambiente) C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Ambiente Virtual>python -V  
Python 3.6.13 :: Anaconda, Inc.
```

(novoambiente) indica que estamos agora dentro do ambiente virtual criado.

Ao usarmos o comando **python - V** temos como retorno a versão do python instalada nesse ambiente. Podemos ver, que conforme solicitado durante a criação do ambiente a versão do Python é a 3.6.

Vamos aproveitar para conhecermos outros 2 comandos que são opostos aos que acabamos de aprender:

- **conda deactivate** : desativa o ambiente virtual;
- **conda remove -- name XXX -- all** : deleta o ambiente virtual criado



Lembram que falamos antes que o ambiente virtual funciona como uma espécie de “Bolha” onde as regras do “mundo externo” não influenciam o ambiente virtual?

Vamos agora entender o que isso significava.

Primeiro vamos abrir o Jupyter Notebook usando o ambiente **(base)**(o “mundo externo”) e depois tentando fazer a mesma ação dentro do ambiente virtual **(novoambiente)**

Por que isso acontece? Ao criarmos o ambiente apenas instalamos o Python, não instalamos outros programas como o Jupyter... Mesmo que ele existe no mundo externo, ele não acessa esse nosso ambiente virtual. Portanto, precisamos instalar esses programas antes de abri-los 😊

The screenshot shows a terminal window with several sections:

- Top Section:** Shows the command `(base) C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Ambiente Virtual>jupyter notebook`. A red box highlights the prefix `(base)`. An orange arrow points down to a callout box containing the text: "A partir da (base) conseguimos acessar o Jupyter Notebook".
- Second Section:** Shows the output of the command: "The notebook list is empty."
- Third Section:** Shows the command `(novoambiente) C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Ambiente Virtual>jupyter notebook`. A red box highlights the prefix `(novoambiente)`. An orange arrow points up to a callout box containing the text: "No entanto, não é possível dentro do (novoambiente) pois o Jupyter ainda não está instalado." Below this, another red box highlights the command `jupyter notebook`.
- Fourth Section:** Shows the command `(novoambiente) C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Ambiente Virtual>pip install jupyter`. A red box highlights the command `pip install jupyter`. An orange arrow points down to a callout box containing the text: "The notebook list is empty."
- Bottom Section:** Shows the output of the command: "The notebook list is empty."

Módulo 27

# Integração Python com ArcGIS

Antes de tudo... o que é o ArcGIS? Essa é uma ferramenta que cada vez mais vem sendo usada no mercado para localização, mapas, rotas e mais uma série de informações. Se não conhece e deseja conhecer, é só [clicar aqui](#).

Algumas ressalvas importantes sobre o ArcGIS:

- Não é um software 100% gratuito;
- Até o momento da elaboração dessa apostila, o sistema não suporta versão do Python superior a 3.6;
- Vamos precisar de um ambiente virtual para utilização do Python.

Assim como aprendemos no módulo anterior, vamos começar criando um ambiente virtual dentro de uma pasta Python e ArcGIS.

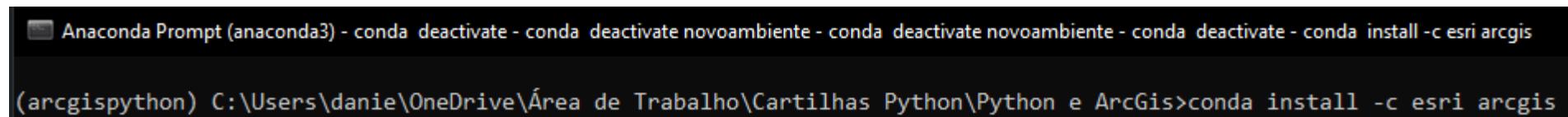
```
(arcgispython) C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Python e ArcGis>python -V  
Python 3.6.13 :: Anaconda, Inc.
```

```
(arcgispython) C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Python e ArcGis>pip install jupyter notebook
```

Vamos agora instalar dentro desse mesmo ambiente virtual a Instalação da API do ArcGIS.

Usando as informações fornecidas pela documentação da API, vamos usar o código abaixo para realizar a instalação (demora um pouquinho ☺):

### Conda install – c esri arcgis



```
Anaconda Prompt (anaconda3) - conda deactivate - conda deactivate novoambiente - conda deactivate novoambiente - conda deactivate - conda install -c esri arcgis  
(arcgispython) C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Python e ArcGis>conda install -c esri arcgis
```

Feita a instalação do ArcGis, vamos abrir nosso Jupyter e acessa a nossa pasta Python e ArcGis.

**ATENÇÃO:** Abra via prompt do Anaconda dentro do nosso ambiente virtual criado.



Vamos agora para a Integração em si.

Com Jupyter aberto DENTRO do seu ambiente virtual, vamos começar a comunicar nosso código com a API do ArcGIS.

Essencialmente o que faremos aqui é uma parte do que pode ser aprendido por meio das documentações do próprio ArcGis indicadas abaixo:

- <https://developers.arcgis.com/python/>
- <https://developers.arcgis.com/python/guide/install-and-set-up/>

Como sempre, vamos iniciar importando a biblioteca que nos permitirá trabalhar com o ArcGIS. Nesse caso usaremos :

```
from arcgis.gis import GIS
```

Além disso, vamos acessar essa biblioteca com um usuário genérico.

Importando biblioteca que nos permitirá trabalhar com ArcGIS

```
from arcgis.gis import GIS
```

```
gis = GIS()
```

Ao não fornecermos nenhum parâmetro, usaremos um usuário genérico

```
user = gis.users.get('rxsingh')  
display(user)
```

Consultando o usuário genérico que estamos usando



[Rohit Singh](#)

**Bio:**

**First Name:** Rohit

**Last Name:** Singh

**Username:** rxsingh

**Joined:** July 02, 2014

Um baita de um bigode

Agora “logados” como nosso usuário bigodudo vamos para um código um pouco mais útil.

Usando a biblioteca do ArcGIS, podemos por exemplo exibir um mapa interativo.

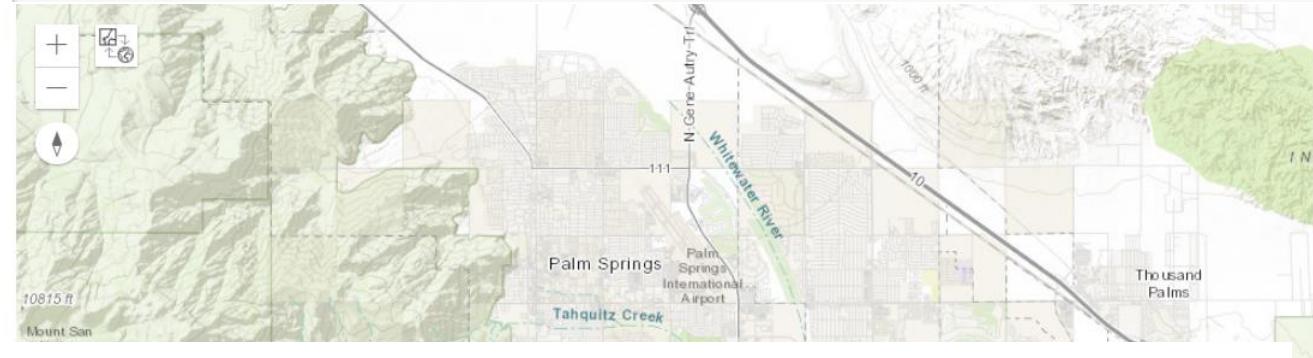
Utilizando o método `.add_layer()` conseguimos acrescentar por exemplo marcações de trilhas existentes da região.

Lembrando que não estamos focando em como usar o ArcGIS e sim na integração do Python com essa ferramenta.

Em resumo, a premissa será a mesma das demais integrações que utilizamos:

- 1) Encontramos a biblioteca que mais nos oferece recursos para a integração;
- 2) Estudamos a biblioteca para conhecer os métodos disponíveis;
- 3) Usamos a biblioteca juntamente com os conhecimentos do Python.

```
map = gis.map("Palm Springs, CA", zoomlevel=12)  
display(map)
```



```
map.add_layer(items[1].layers[0])
```



Seguiremos novamente mais uma documentação do ArcGIS mas agora usaremos um usuário específico ao invés de um usuário genérico.

Para isso, usaremos a estrutura abaixo:

```
gis = GIS('link', 'usuario', 'senha')
```

Novamente vamos buscar trilhas e como podemos ver, ao rodar o código temos a exibição das trilhas marcadas no mapa do ArcGIS.

```
from arcgis.gis import GIS  
  
print("ArcGIS Online Org account")  
gis = GIS("https://www.arcgis.com", "arcgis_python", "P@ssword123")  
print("Logged in as " + str(gis.properties.user.username))
```

```
ArcGIS Online Org account  
Logged in as arcgis_python
```

```
from arcgis.mapping import WebMap  
la_parks_trails = WebMap(webmap)  
la_parks_trails
```



Módulo 28

# Integração Python com Power BI

Existem 2 formas de integrar o Python com Power BI.

Uma delas é usar todo nosso conhecimento adquirido em tratamento de dados para criar uma automação que nos permita criar base de dados que servirão como base para o Power BI. O que não seria diretamente uma integração e sim uma opção por usar o PowerBI para gráficos etc ao invés de usar outras bibliotecas.

A outra opção, é usarmos o Python via Power BI. Para isso, você precisará ter o Power BI instalado no seu computador. Aqui embaixo vamos deixar o link de instalação e um vídeo que explica como instalar 😊.

Link para download:

<https://powerbi.microsoft.com/pt-br/downloads/>

Link de instalação com o MESTRE ALON:

<https://www.youtube.com/watch?v=Y01Tn1z2grE>



Assim como fizemos com o ArcGIS, iremos precisar de um ambiente virtual dedicado a essa integração pois existem(pelo menos no momento da elaboração dessa apostila) alguns *bugs* entre a versão mais recente do Python e o Power Bi.

Então, assim como fizemos no [Módulo 26](#) iremos criar uma pasta chamada Python e Power BI onde criaremos um ambiente virtual específico com a versão 3.6 do Python.

Além disso, vamos instalar as bibliotecas que usaremos nessa integração:

- Jupyter Notebook;
- Pandas;
- Matplotlib.

Feito essas importações vamos indicar para o Power BI que queremos usar esse ambiente virtual como ambiente de integração com o Power BI. Em geral, seus ambientes virtuais estarão no caminho abaixo:

C:\Users\SEU USUÁRIO\anaconda3\envs\

```
Python\Python e Power BI>conda create -n pythonpowerbi python=3.6
```

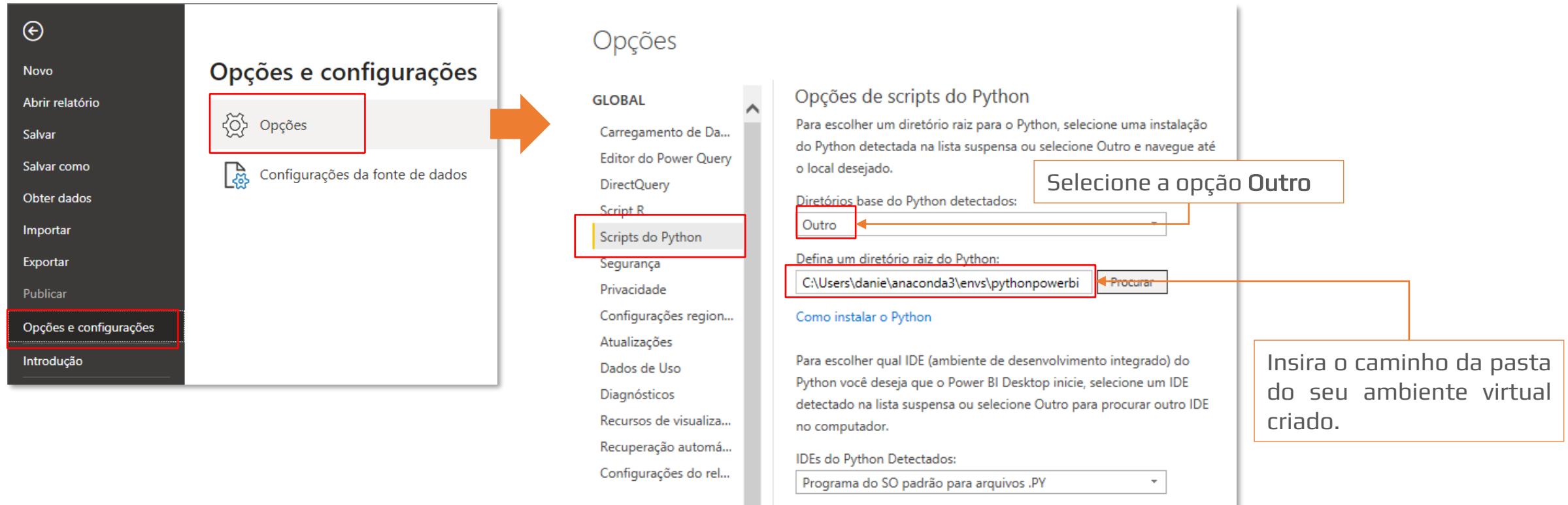
```
(pythonpowerbi)\Python e Power BI>pip install jupyter notebook
```

```
(pythonpowerbi)\Python e Power BI>pip install pandas
```

```
(pythonpowerbi)\Python e Power BI>pip install matplotlib
```



Iremos indicar o ambiente virtual que desejamos dentro na parte **Opções e configurações** dentro da aba **arquivo** do Power BI.



## Módulo 28 – Integração Python com Power BI – Configurações Importantes - Python e Power BI (3/4)

402

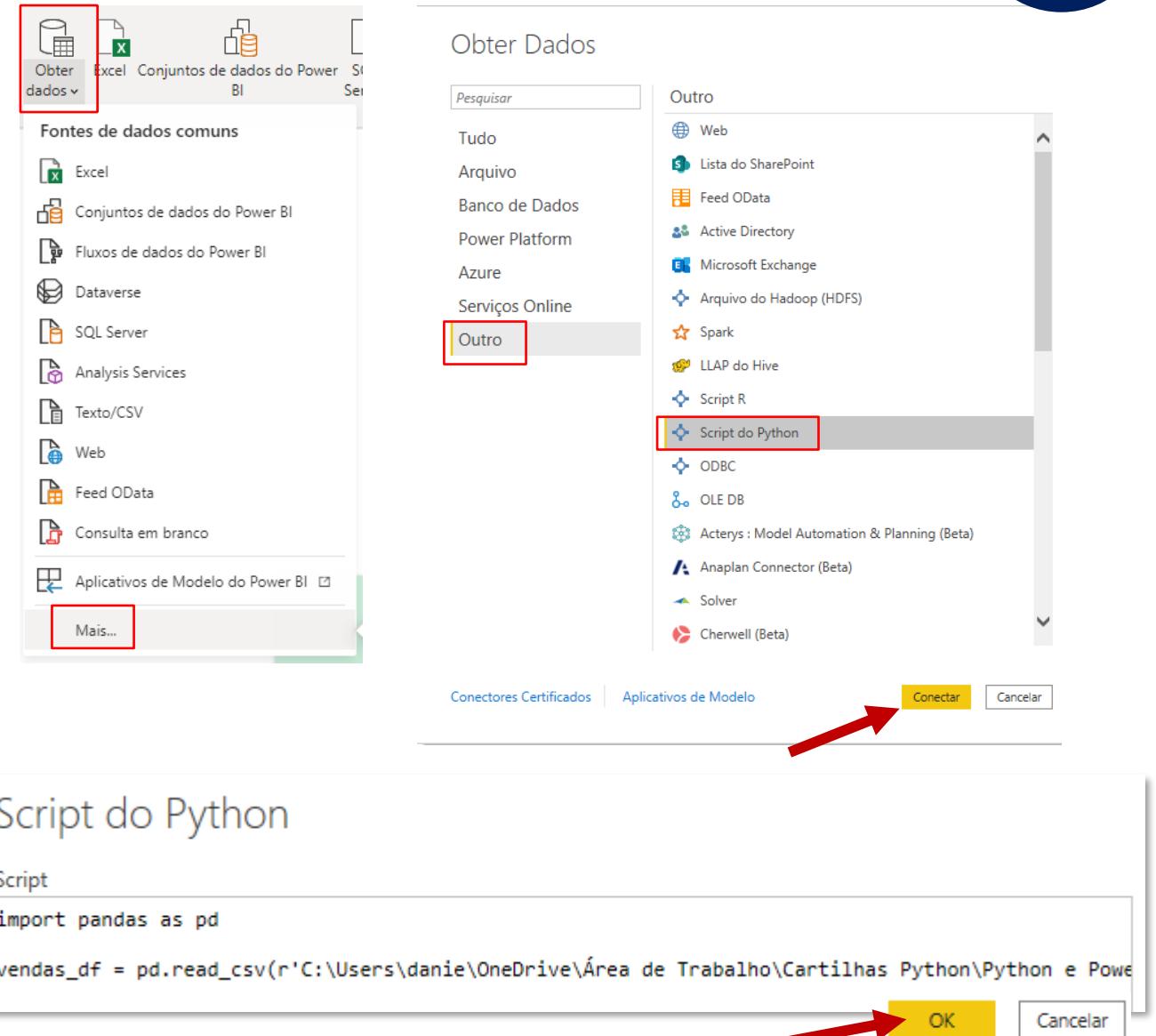
Vamos agora fazer alguns testes para saber se todas as configurações estão corretas.

Dentro do Power BI:

- 1) Clique **Obter Dados**;
- 2) Clique na opção **Mais...**;
- 3) Após a abertura da Janela procure a opção **Outro**;
- 4) Depois selecione a opção **Script do Python**;
- 5) Clique em **Conectar**;
- 6) Dentro do campo **Script** vamos escrever nosso código Python de importação do Pandas e do arquivo CSV. Dica: sempre colocar o caminho completo do arquivo.
- 7) Clique em **OK** para rodar

Se rodar, sabemos que a integração está funcionando.

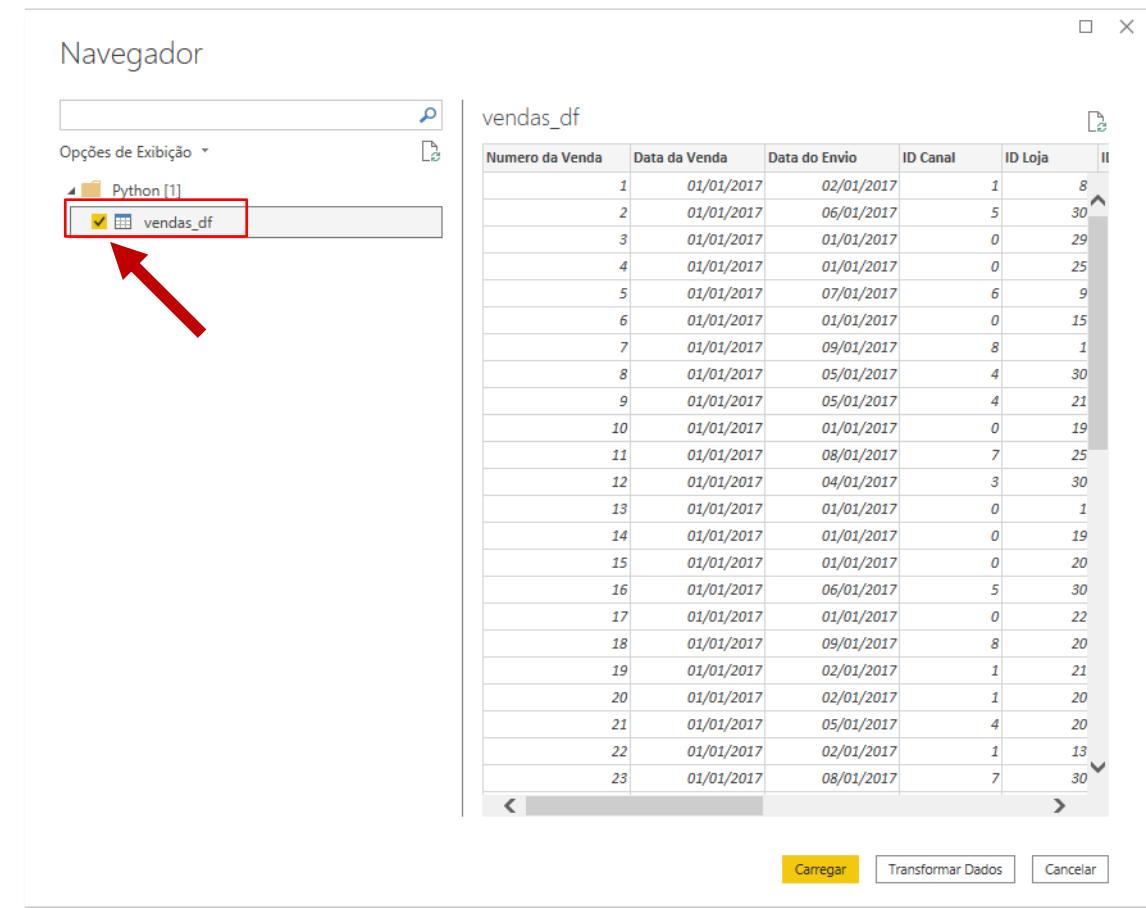
Caso não, precisamos entender o que está acontecendo lendo o erro indicado. Se tiver dúvida, lembre-se que temos o suporte na plataforma para te ajudar 😊.



Se ao rodar o script Python dentro do Power BI surgiu uma janela como a apresentada ao lado, sabemos que tanto nossa integração quanto nosso código estão funcionando.

Por default, `vendas_df` não estará marcada ( `vendas_df` ). Ao marcar o checkbox, nosso `vendas_df` aparecerá no display do ao lado direito da janela.

Feita a configuração, agora é só executar nossos códigos. E é isso que vamos ver até o fim deste módulo.



# Módulo 28 – Integração Python com Power BI – Usando o Python para gerar Base de Dados no Power BI (1/2)

404

Feita a configuração inicial, vamos criar um código de importação de dados.

**Dica1:** Teste seus códigos no Jupyter e quando tudo estiver funcionando, cole no campo de Script Python do Power BI.

**Dica2:** Geralmente damos print ou display no Jupyter, retire essas linhas de código quando colar no Power BI.

```
import pandas as pd
import os
#importando os arquivos
caminho_padrao = r'C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Python e Power BI'
vendas_df = pd.read_csv(os.path.join(caminho_padrao, r'Contoso - Vendas - 2017.csv'), sep=';')
produtos_df = pd.read_csv(os.path.join(caminho_padrao, r'Contoso - Cadastro Produtos.csv'), sep=';')
lojas_df = pd.read_csv(os.path.join(caminho_padrao, r'Contoso - Lojas.csv'), sep=';')
clientes_df = pd.read_csv(os.path.join(caminho_padrao, r'Contoso - Clientes.csv'), sep=';')

#limpando apenas as colunas que queremos
clientes_df = clientes_df[['ID Cliente', 'E-mail']]
produtos_df = produtos_df[['ID Produto', 'Nome do Produto']]
lojas_df = lojas_df[['ID Loja', 'Nome da Loja']]

#mesclando e renomeando os dataframes
vendas_df = vendas_df.merge(produtos_df, on='ID Produto')
vendas_df = vendas_df.merge(lojas_df, on='ID Loja')
vendas_df = vendas_df.merge(clientes_df, on='ID Cliente').rename(columns={'E-mail': 'E-mail do Cliente'})
display(vendas_df)
```

| Número da Venda | Data da Venda | Data do Envio | ID Canal   | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida | Quantidade Devolvida | Nome do Produto | Nome da Loja                               | E-mail do Cliente                             |
|-----------------|---------------|---------------|------------|---------|------------|-------------|------------|--------------------|----------------------|-----------------|--|---|
| 0               | 1             | 01/01/2017    | 02/01/2017 | 1       | 86         | 981         | 2          | 6825               | 9                    | 1               | A. Datum Advanced Digital Camera M300 Pink | Loja Contoso Austin<br>rbrumfieldmy@ameblo.jp |
| 1               | 880458        | 23/11/2017    | 23/11/2017 | 0       | 306        | 235         | 10         | 6825               | 8                    | 0               | Litware Home Theater System 7.1            | Loja Contoso Europe<br>rbrumfieldmy@ameblo.jp |

Script do Python

Script

```
clientes_df = pd.read_csv(os.path.join(caminho_padrao, r'Contoso - Clientes.csv'), sep=';')

#limpando apenas as colunas que queremos
clientes_df = clientes_df[['ID Cliente', 'E-mail']]
produtos_df = produtos_df[['ID Produto', 'Nome do Produto']]
lojas_df = lojas_df[['ID Loja', 'Nome da Loja']]

#mesclando e renomeando os dataframes
vendas_df = vendas_df.merge(produtos_df, on='ID Produto')
vendas_df = vendas_df.merge(lojas_df, on='ID Loja')
vendas_df = vendas_df.merge(clientes_df, on='ID Cliente').rename(columns={'E-mail': 'E-mail do Cliente'})
```

O script será executado com a seguinte instalação do Python C:\Users\danie\anaconda3\envs\pythonpowerbi.  
Para definir suas configurações e alterar qual instalação do Python você deseja executar, acesse Opções e configurações.

OK Cancelar

# Módulo 28 – Integração Python com Power BI – Usando o Python para gerar Base de Dados no Power BI (2/2)

405

Perceba que automaticamente foram importadas as 4 bases em CSV.

Se clicarmos em Transformar Dados, vamos poder ir para o Power Query que é uma ferramenta que existe no Power BI.

Navegador

Opções de Exibição ▾

- Python [4]
  - clientes\_df
  - lojas\_df
  - produtos\_df
  - vendas\_df

vendas\_df

| Numero da Venda | Data da Venda | Data do Envio | ID Canal | ID Loja |
|-----------------|---------------|---------------|----------|---------|
| 1               | 01/01/2017    | 02/01/2017    | 1        | 8       |
| 880458          | 23/11/2017    | 23/11/2017    | 0        | 30      |
| 191019          | 20/03/2017    | 21/03/2017    | 1        | 17      |
| 18610           | 08/01/2017    | 10/01/2017    | 2        | 20      |
| 287704          | 23/04/2017    | 26/04/2017    | 3        | 7       |
| 373600          | 22/05/2017    | 29/05/2017    | 7        | 3       |
| 873068          | 21/11/2017    | 22/11/2017    | 1        | 27      |
| 26350           | 12/01/2017    | 18/01/2017    | 6        | 2       |
| 180875          | 16/03/2017    | 16/03/2017    | 0        | 26      |
| 413808          | 04/06/2017    | 06/06/2017    | 2        | 18      |
| 860046          | 16/11/2017    | 16/11/2017    | 0        | 19      |
| 415344          | 04/06/2017    | 05/06/2017    | 1        | 23      |
| 811927          | 29/10/2017    | 31/10/2017    | 2        | 10      |
| 364459          | 19/05/2017    | 20/05/2017    | 1        | 2       |
| 836335          | 07/11/2017    | 08/11/2017    | 1        | 23      |
| 638297          | 27/08/2017    | 27/08/2017    | 0        | 26      |
| 372597          | 21/05/2017    | 22/05/2017    | 1        | 8       |
| 614980          | 18/08/2017    | 18/08/2017    | 0        | 30      |
| 786402          | 21/10/2017    | 25/10/2017    | 4        | 30      |
| 383897          | 25/05/2017    | 28/05/2017    | 3        | 30      |
| 645873          | 31/08/2017    | 31/08/2017    | 0        | 30      |
| 169169          | 11/03/2017    | 14/03/2017    | 3        | 2       |
| 333103          | 09/05/2017    | 09/05/2017    | 0        | 19      |

Carregar    Transformar Dados    Cancelar

POWER QUERY:

Consultas [4]

= Fonte{[Name="vendas\_df"]}[Value]

|    | A <sup>B</sup> Numro da Venda | A <sup>B</sup> Data da Venda | A <sup>B</sup> Data do Envio | A <sup>B</sup> ID Canal | A <sup>B</sup> ID Loja | A <sup>B</sup> ID Produto |
|----|-------------------------------|------------------------------|------------------------------|-------------------------|------------------------|---------------------------|
| 1  | 1                             | 01/01/2017                   | 02/01/2017                   | 1                       | 86                     | 981                       |
| 2  | 880458                        | 23/11/2017                   | 23/11/2017                   | 0                       | 306                    | 235                       |
| 3  | 191019                        | 20/03/2017                   | 21/03/2017                   | 1                       | 172                    | 376                       |
| 4  | 18610                         | 08/01/2017                   | 10/01/2017                   | 2                       | 200                    | 448                       |
| 5  | 287704                        | 23/04/2017                   | 26/04/2017                   | 3                       | 76                     | 280                       |
| 6  | 373600                        | 22/05/2017                   | 29/05/2017                   | 7                       | 33                     | 1284                      |
| 7  | 873068                        | 21/11/2017                   | 22/11/2017                   | 1                       | 275                    | 246                       |
| 8  | 26350                         | 12/01/2017                   | 18/01/2017                   | 6                       | 21                     | 568                       |
| 9  | 180875                        | 16/03/2017                   | 16/03/2017                   | 0                       | 268                    | 975                       |
| 10 | 413808                        | 04/06/2017                   | 06/06/2017                   | 2                       | 185                    | 1197                      |
| 11 | 860046                        | 16/11/2017                   | 16/11/2017                   | 0                       | 191                    | 578                       |
| 12 | 415344                        | 04/06/2017                   | 05/06/2017                   | 1                       | 231                    | 997                       |
| 13 | 811927                        | 29/10/2017                   | 31/10/2017                   | 2                       | 100                    | 431                       |
| 14 | 364459                        | 19/05/2017                   | 20/05/2017                   | 1                       | 26                     | 845                       |
| 15 | 836335                        | 07/11/2017                   | 08/11/2017                   | 1                       | 237                    | 322                       |
| 16 | 638297                        | 27/08/2017                   | 27/08/2017                   | 0                       | 262                    | 356                       |
| 17 | 372597                        | 21/05/2017                   | 22/05/2017                   | 1                       | 86                     | 981                       |
| 18 | 614980                        | 18/08/2017                   | 18/08/2017                   | 0                       | 306                    | 1621                      |
| 19 | 786402                        | 21/10/2017                   | 25/10/2017                   | 4                       | 306                    | 226                       |
| 20 | 383897                        | 25/05/2017                   | 28/05/2017                   | 3                       | 306                    | 347                       |
| 21 | 645873                        | 31/08/2017                   | 31/08/2017                   | 0                       | 309                    | 968                       |
| 22 | 169169                        | 11/03/2017                   | 14/03/2017                   | 3                       | 22                     | 437                       |
| 23 | 333103                        | 09/05/2017                   | 09/05/2017                   | 0                       | 199                    | 591                       |

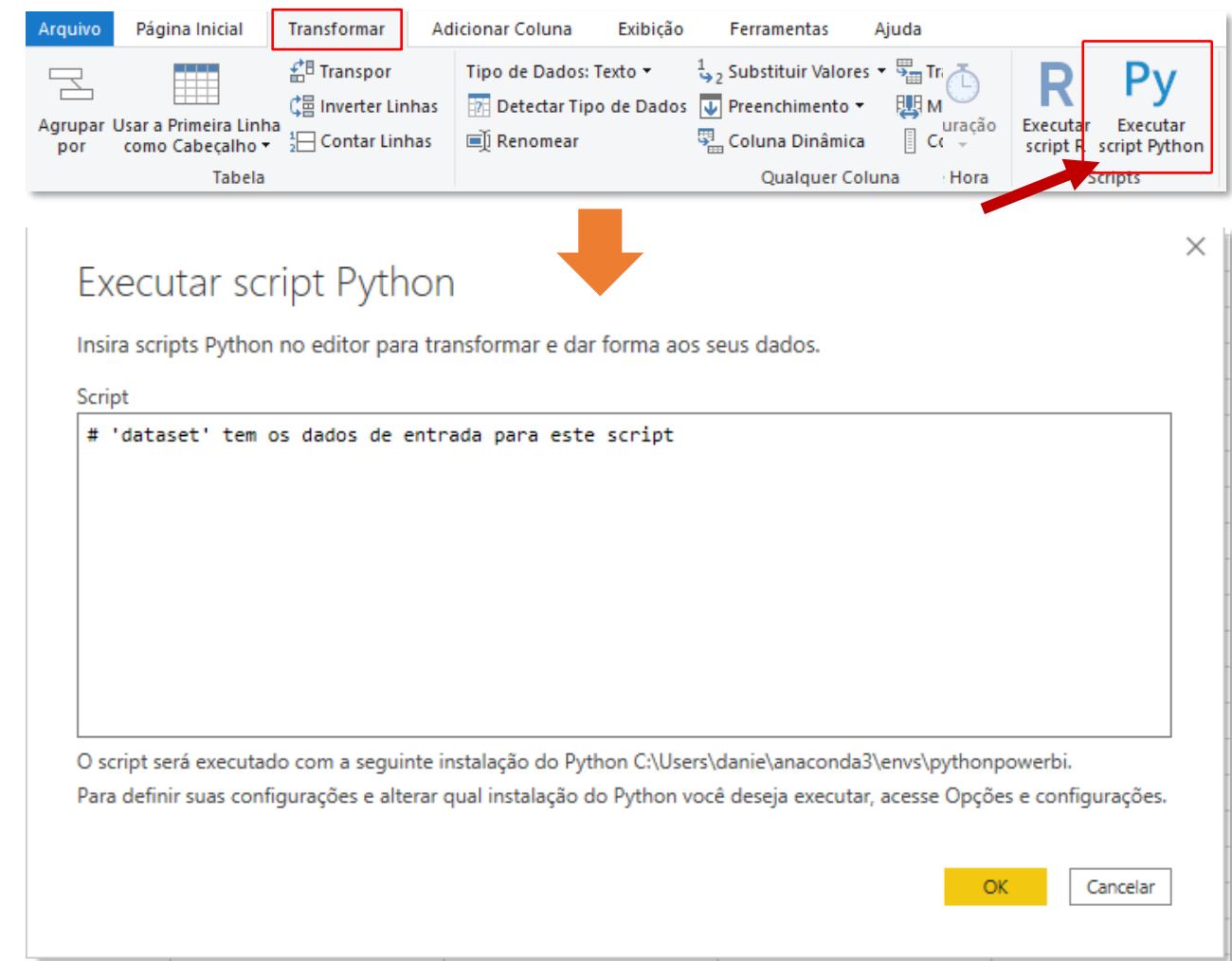


É muito comum no Power BI, antes de carregarmos os dados para elaboração de relacionamentos, gráficos, etc, fazer a edição desses dados para garantirmos que todos os dados estão corretos e formatados da forma que nos interessa.

Podemos fazer essa edição via Python! Na guia **Transformar** do Power Query clique na opção **Executar script Python**.

Nesse campo, é possível criar seus códigos que rodarão dentro do Power Query para edição dos dados.

**Dica:** Crie seus códigos no Jupyter e só depois coloque-o no script do Python.



Agora já sabemos onde colocar nosso código, vamos voltar para o Jupyter para escrever nosso código e quando tudo estiver certinho, a gente volta e coloca no campo **script Python**.

Dentro do PowerBI os dataframes são chamados de **dataset** conforme apresentado no print ao lado.

Sabendo dessa informação, vamos usar esse nome para realizar nosso código.

Nosso objetivo é filtrar apenas as lojas com ID IGUAL a 86, 306 ou 172. Ao executarmos o código e termos sucesso, podemos copiar a linha de código e colá-la dentro do nos **Script Python** do Power BI.

Executar script Python

Insira scripts Python no editor para transformar e dar forma aos seus dados.

Script

```
# 'dataset' tem os dados de entrada para este script
```

dataset = vendas\_df

tres\_lojas\_df = dataset[dataset['ID Loja'].isin([86, 306, 172])]

display(tres\_lojas\_df)

Código a ser levado para o Power Query/Power BI

| Numero da Venda | Data da Venda | Data do Envio | ID Canal   | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida | Quantidade Devolvida | Nome do Produto                            | Nome da Loja                                      | E-mail do Cliente          |                        |
|-----------------|---------------|---------------|------------|---------|------------|-------------|------------|--------------------|----------------------|--|---|----------------------------|------------------------|
| 0               | 1 01/01/2017  | 02/01/2017    | 1          | 86      | 981        | 2           | 6825       | 9                  | 1                    | A. Datum Advanced Digital Camera M300 Pink | Loja Contoso Austin                               | rbrumfieldmy@ameblo.jp     |                        |
| 1               | 880458        | 23/11/2017    | 23/11/2017 | 0       | 306        | 235         | 10         | 6825               | 8                    | 0  | Litware Home Theater System 7.1 Channel M710 B... | Loja Contoso Europe Online | rbrumfieldmy@ameblo.jp |
| 2               | 191019        | 20/03/2017    | 21/03/2017 | 1       | 172        | 376         | 2          | 6825               | 9                    | 0  | Adventure Works Laptop12 M1201 Silver             | Loja Contoso Hartford      | rbrumfieldmy@ameblo.jp |

↓

Executar script Python

Insira scripts Python no editor para transformar e dar forma aos seus dados.

Script

```
# 'dataset' tem os dados de entrada para este script
```

tres\_lojas\_df = dataset[dataset['ID Loja'].isin([86, 306, 172])]

Script

```
# 'dataset' tem os dados de entrada para este script
```

tres\_lojas\_df = dataset[dataset['ID Loja'].isin([86, 306, 172])]

## Módulo 28 – Integração Python com Power BI – Usando o Python para Editar Tabelas do Power BI (2/3)

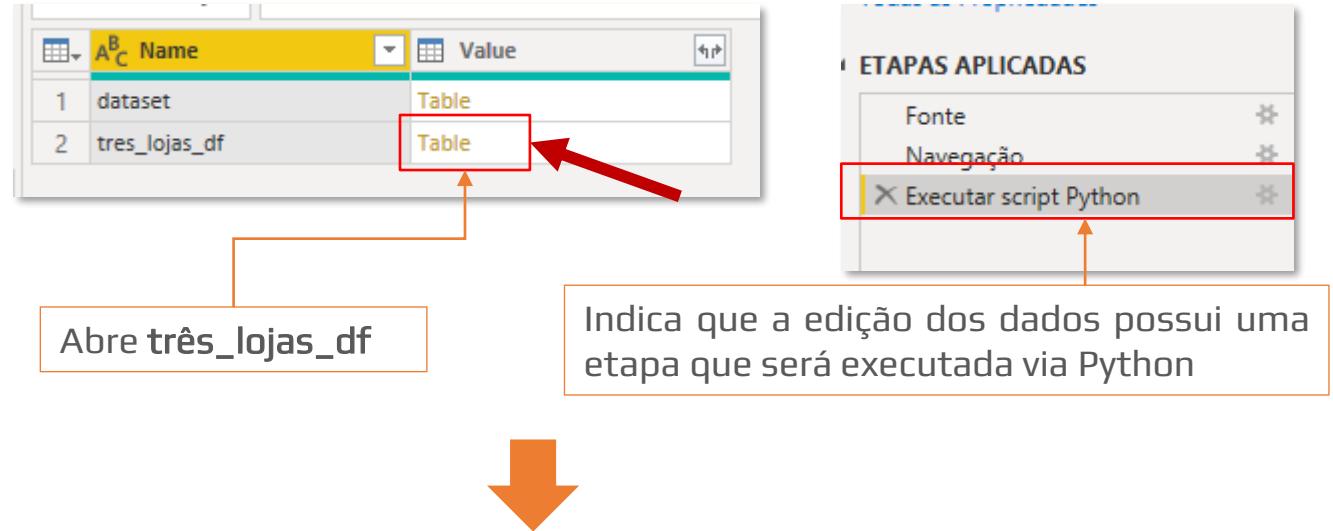
408

Após rodar o Script (pode demorar um pouquinho) vamos perceber que no campo ETAPAS APLICADAS aparecerá uma linha Executar script Python.

Essa linha indica que agora, criamos uma etapa na edição dos dados via Python.

Além disso, a tabela foi substituída por uma tabela menor com 2 linhas.

Clicando em table da linha 2 referente a três\_lojas\_df podemos abrir a tabela editada.



| ID | Numero da Venda | Data da Venda | Data do Envio | ID Canal | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida |
|----|-----------------|---------------|---------------|----------|---------|------------|-------------|------------|--------------------|
| 1  | 1               | 01/01/2017    | 02/01/2017    | 1        | 86      | 981        | 2           | 6825       |                    |
| 2  | 880458          | 23/11/2017    | 23/11/2017    | 0        | 306     | 235        | 10          | 6825       |                    |
| 3  | 191019          | 20/03/2017    | 21/03/2017    | 1        | 172     | 376        | 2           | 6825       |                    |
| 4  | 372597          | 21/05/2017    | 22/05/2017    | 1        | 86      | 981        | 1           | 21344      |                    |
| 5  | 614980          | 18/08/2017    | 18/08/2017    | 0        | 306     | 1621       | 9           | 21344      |                    |
| 6  | 786402          | 21/10/2017    | 25/10/2017    | 4        | 306     | 226        | 10          | 21344      |                    |
| 7  | 383897          | 25/05/2017    | 28/05/2017    | 3        | 306     | 347        | 1           | 21344      |                    |
| 8  | 767274          | 15/10/2017    | 21/10/2017    | 6        | 86      | 1444       | 1           | 4743       |                    |
| 9  | 328617          | 07/05/2017    | 07/05/2017    | 0        | 172     | 1339       | 1           | 4743       |                    |
| 10 | 835611          | 06/11/2017    | 08/11/2017    | 2        | 86      | 1468       | 4           | 24440      |                    |
| 11 | 559478          | 26/07/2017    | 29/07/2017    | 3        | 306     | 1382       | 1           | 24440      |                    |
| 12 | 560574          | 27/07/2017    | 01/08/2017    | 5        | 306     | 585        | 1           | 24440      |                    |
| 13 | 838173          | 07/11/2017    | 07/11/2017    | 0        | 86      | 1468       | 4           | 16915      |                    |
| 14 | 97747           | 10/02/2017    | 16/02/2017    | 6        | 86      | 1106       | 2           | 34115      |                    |
| 15 | 429560          | 09/06/2017    | 11/06/2017    | 2        | 86      | 1106       | 1           | 3979       |                    |
| 16 | 453352          | 17/06/2017    | 18/06/2017    | 1        | 86      | 1106       | 1           | 13327      |                    |
| 17 | 784425          | 20/10/2017    | 20/10/2017    | 0        | 86      | 1466       | 1           | 13327      |                    |
| 18 | 265665          | 16/04/2017    | 16/04/2017    | 0        | 86      | 1261       | 1           | 26471      |                    |
| 19 | 24931           | 11/01/2017    | 17/01/2017    | 6        | 306     | 1467       | 10          | 26471      |                    |
| 20 | 328197          | 07/05/2017    | 11/05/2017    | 4        | 306     | 1558       | 1           | 26471      |                    |

# Módulo 28 – Integração Python com Power BI – Usando o Python para Editar Tabelas do Power BI (3/3)

409

Agora é só clicar em **Fechar e Aplicar** no Power Query e nosso dataframe já estará disponível dentro do Power BI.



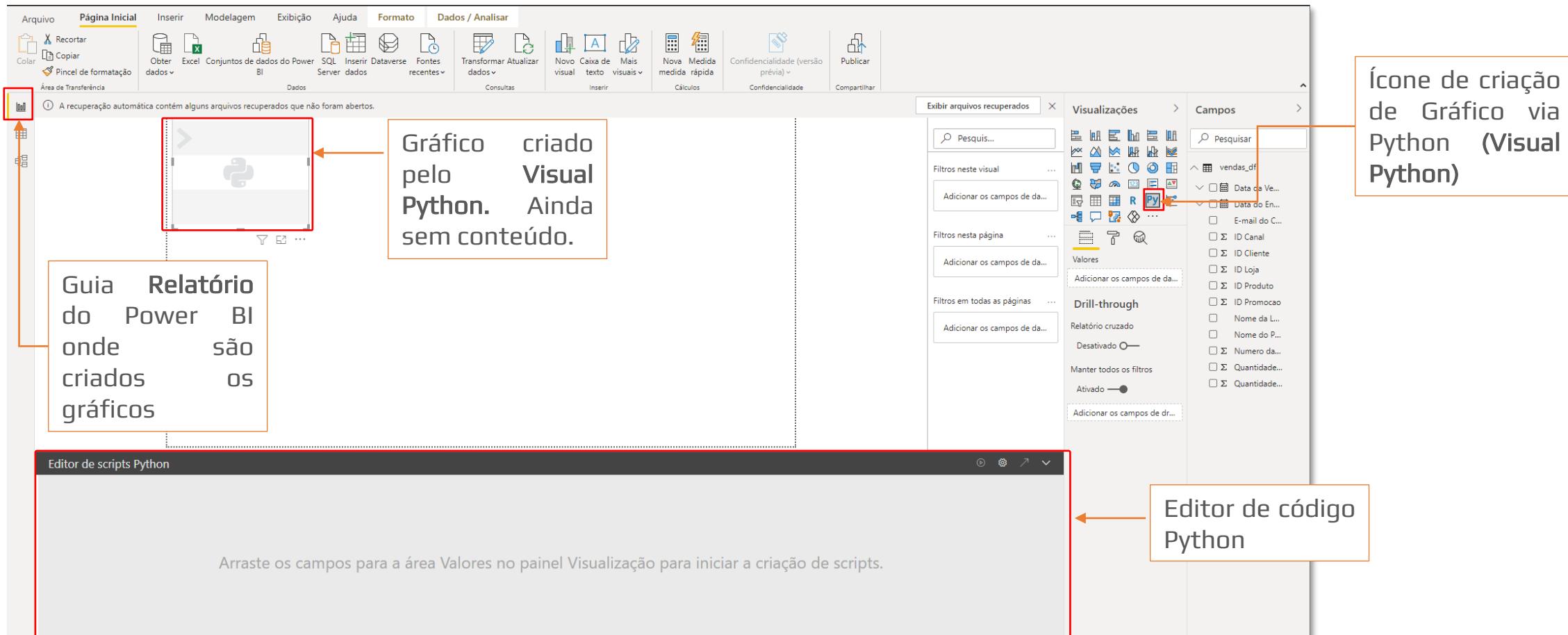
| Numero da Venda | Data da Venda                         | Data do Envio                         | ID Canal | ID Loja | ID Produto | ID Promocao | ID Cliente | Quantidade Vendida | Quantidade Devolvida | Nome do Produto  | Nome da Loja               | E-mail do Cliente     |
|-----------------|---------------------------------------|---------------------------------------|----------|---------|------------|-------------|------------|--------------------|----------------------|--|----------------------------|-----------------------|
| 880458          | quinta-feira, 23 de novembro de 2017  | quinta-feira, 23 de novembro de 2017  | 0        | 306     | 235        | 10          | 6825       | 8                  | 0                    | Litware Home Theater System 7.1 Channel M710 Brown         | Loja Contoso Europe Online | rbrumfie@contoso.com  |
| 818047          | terça-feira, 31 de outubro de 2017    | terça-feira, 31 de outubro de 2017    | 0        | 306     | 1008       | 10          | 26471      | 8                  | 0                    | A. Datum Consumer Digital Camera M300 Orange               | Loja Contoso Europe Online | jessica44@contoso.com |
| 36490           | segunda-feira, 16 de janeiro de 2017  | segunda-feira, 16 de janeiro de 2017  | 0        | 306     | 196        | 10          | 28318      | 8                  | 0                    | Litware Home Theater System 2.1 Channel E211 Black         | Loja Contoso Europe Online | arianna4@contoso.com  |
| 911713          | terça-feira, 5 de dezembro de 2017    | terça-feira, 5 de dezembro de 2017    | 0        | 306     | 1380       | 10          | 31431      | 8                  | 0                    | Contoso Bedroom Phone with AM/FM Stereo and Call Waiting   | Loja Contoso Europe Online | cody19@contoso.com    |
| 951107          | quarta-feira, 20 de dezembro de 2017  | quarta-feira, 20 de dezembro de 2017  | 0        | 306     | 566        | 10          | 31431      | 8                  | 0                    | Proseware Projector 720p DLP56 Silver                      | Loja Contoso Europe Online | cody19@contoso.com    |
| 802072          | quinta-feira, 26 de outubro de 2017   | quinta-feira, 26 de outubro de 2017   | 0        | 306     | 855        | 10          | 20064      | 8                  | 0                    | Contoso Multimedia Speakers M25 Blue                       | Loja Contoso Europe Online | jimlacke@contoso.com  |
| 58933           | quarta-feira, 25 de janeiro de 2017   | quarta-feira, 25 de janeiro de 2017   | 0        | 306     | 279        | 10          | 25296      | 8                  | 0                    | Contoso Home Theater System 2.1 Channel E1200 Brown        | Loja Contoso Europe Online | sophia17@contoso.com  |
| 732741          | quarta-feira, 4 de outubro de 2017    | quarta-feira, 4 de outubro de 2017    | 0        | 306     | 710        | 10          | 10683      | 8                  | 0                    | Proseware Scan Jet Digital Flat Bed Scanner M300 White     | Loja Contoso Europe Online | ncelloj0@contoso.com  |
| 767650          | domingo, 15 de outubro de 2017        | domingo, 15 de outubro de 2017        | 0        | 306     | 510        | 10          | 24700      | 8                  | 0                    | Adventure Works LCD19W M100 White                          | Loja Contoso Europe Online | alfredo1@contoso.com  |
| 955271          | sexta-feira, 22 de dezembro de 2017   | sexta-feira, 22 de dezembro de 2017   | 0        | 306     | 405        | 10          | 24612      | 8                  | 0                    | Proseware Laptop15 M510 Black                              | Loja Contoso Europe Online | alan29@contoso.com    |
| 944485          | segunda-feira, 18 de dezembro de 2017 | segunda-feira, 18 de dezembro de 2017 | 0        | 306     | 1192       | 10          | 30129      | 8                  | 0                    | Fabrikam Home and Vacation Moviemaker 1" 25mm M40          | Loja Contoso Europe Online | brandy1@contoso.com   |
| 864903          | sábado, 18 de novembro de 2017        | sábado, 18 de novembro de 2017        | 0        | 306     | 388        | 10          | 23763      | 8                  | 0                    | Adventure Works Laptop12 M1201 Blue                        | Loja Contoso Europe Online | kathleen@contoso.com  |
| 874760          | terça-feira, 21 de novembro de 2017   | terça-feira, 21 de novembro de 2017   | 0        | 306     | 626        | 10          | 13494      | 8                  | 0                    | WWI Projector 480p LCD12 White                             | Loja Contoso Europe Online | hgaddr@contoso.com    |
| 934010          | quinta-feira, 14 de dezembro de 2017  | quinta-feira, 14 de dezembro de 2017  | 0        | 306     | 939        | 10          | 11959      | 8                  | 0                    | SV Wireless LAN PCI Network Card Adapter E901 Black        | Loja Contoso Europe Online | cdackeq@contoso.com   |
| 961828          | domingo, 24 de dezembro de 2017       | domingo, 24 de dezembro de 2017       | 0        | 306     | 1007       | 10          | 10009      | 8                  | 0                    | A. Datum Consumer Digital Camera E100 Orange               | Loja Contoso Europe Online | wprettej@contoso.com  |
| 783258          | sexta-feira, 20 de outubro de 2017    | sexta-feira, 20 de outubro de 2017    | 0        | 306     | 880        | 10          | 24994      | 8                  | 0                    | Contoso Optical Wheel OEM PS/2 Mouse E60 Black             | Loja Contoso Europe Online | jenny17@contoso.com   |
| 957969          | sábado, 23 de dezembro de 2017        | sábado, 23 de dezembro de 2017        | 0        | 306     | 203        | 10          | 13636      | 8                  | 0                    | Litware Home Theater System 7.1 Channel M710 Black         | Loja Contoso Europe Online | cdohmer@contoso.com   |
| 61331           | quinta-feira, 26 de janeiro de 2017   | quinta-feira, 26 de janeiro de 2017   | 0        | 306     | 1201       | 10          | 13816      | 8                  | 0                    | Fabrikam Trendsetter 1/2" 3mm X300 Grey                    | Loja Contoso Europe Online | gtownen@contoso.com   |
| 750080          | segunda-feira, 9 de outubro de 2017   | segunda-feira, 9 de outubro de 2017   | 0        | 306     | 1592       | 10          | 29975      | 8                  | 0                    | SV DVD 48 DVD Storage Binder M50 Red                       | Loja Contoso Europe Online | casey0@contoso.com    |
| 824394          | quinta-feira, 2 de novembro de 2017   | quinta-feira, 2 de novembro de 2017   | 0        | 306     | 675        | 10          | 29975      | 8                  | 0                    | Proseware Laser Jet All in one X300 Grey                   | Loja Contoso Europe Online | casey0@contoso.com    |
| 1821            | domingo, 1 de janeiro de 2017         | domingo, 1 de janeiro de 2017         | 0        | 306     | 667        | 10          | 26250      | 8                  | 0                    | Proseware Office Jet Wireless All-in-One Inkjet Printer M6 | Loja Contoso Europe Online | xavier33@contoso.com  |

# Módulo 28 – Integração Python com Power BI – Usando Python para criar Gráficos no Power BI (1/5)

410

Nessa parte do módulo, iremos fazer uma integração do Python com a parte gráfica.

Usaremos a guia **Relatório** do Power BI (indicada abaixo) e o ícone **Visual Python** para criar um gráfico.



Usando a interface do próprio Power BI, vamos arrastar a coluna **Data da Venda** para o campo **Valores** de código do Python.

Automaticamente o campo de edição de código do Python criará algumas linhas de código.

The screenshot shows the Power BI interface with the Python Script Editor open. In the Fields pane on the right, the 'vendas\_df' table is expanded, showing columns like 'Data da Venda', 'Ano', 'Trimestre', 'Mês', and 'Dia'. The 'Data da Venda' column is selected and highlighted with a red box. A red arrow points from this selection to the 'Valores' section in the Fields pane, which also has a red box around it. In the Python Editor at the bottom, there is generated code:

```
1 # O código a seguir para criar um dataframe e remover as linhas duplicadas sempre é executado e age como um preâmbulo para o script:  
2  
3 # dataset = pandas.DataFrame(Ano, Trimestre, Mês, Dia)  
4 # dataset = dataset.drop_duplicates()  
5  
6 # Cole ou digite aqui seu código de script:
```

A callout box with an orange border and arrow points from the 'Valores' section in the Fields pane to the generated code in the Python Editor, explaining that the values are created based on the chosen 'Data da Venda' column as the value.

Analisando o código criado, percebemos que nosso DataFrame está considerando ao invés da Data exata da venda as informações de Ano, Trimestre, Mês e Dia.

Isso acontece pois o Power BI tende a trabalhar os dados de forma que para ELE faça mais sentido.

Para retornarmos essa informação para a Data da Venda original, basta clicarmos na seta para baixo da coluna Data da Venda e selecionar a opção Data da Venda.

Como podemos ver no exemplo ao lado, o Power BI estava considerando a opção Hierarquia de datas o que não nos atende nesse exemplo.

The screenshot shows the Power BI Python script editor. At the top, it says "Editor de scripts Python". Below that is a warning: "⚠ As linhas duplicadas serão removidas dos dados." followed by a code snippet:

```
1 # O código a seguir para criar um dataframe e remover as linhas duplicadas sempre é executado e age como um preâmbulo para o script:  
2  
3 # dataset = pandas.DataFrame(Ano, Trimestre, Mês, Dia)  
4 # dataset = dataset.drop_duplicates()  
5  
6 # Cole ou digite aqui seu código de script:
```

To the right of the editor is a "Valores" (Values) pane containing "Data da Venda", "Ano", "Trimestre", "Mês", and "Dia". A red arrow points from the "Data da Venda" item in this pane down to a dropdown menu. The dropdown menu includes "Remover campo", "Novas medidas rápidas", "Mostrar itens sem dados", and "Hierarquia de datas". The "Hierarquia de datas" option is selected, indicated by a checked checkbox. Another red arrow points from the "Hierarquia de datas" option in the dropdown menu back up to the "Data da Venda" item in the "Valores" pane.

The screenshot shows the Power BI Python script editor again. The "Editor de scripts Python" header is present. The warning "⚠ As linhas duplicadas serão removidas dos dados." is shown. The code has been modified:

```
1 # O código a seguir para criar um dataframe e remover as linhas duplicadas sempre é executado e age como um preâmbulo para o script:  
2  
3 # dataset = pandas.DataFrame(Data da Venda)  
4 # dataset = dataset.drop_duplicates()  
5  
6 # Cole ou digite aqui seu código de script:
```

A red arrow points from the "Data da Venda" part of the third line of code back up to the "Hierarquia de datas" option in the previous screenshot, indicating the automatic update. To the right of the editor, a callout box contains the text: "Após seleção, código é alterado automaticamente para Data da Venda ao invés de Ano, Trimestre, Mês, Dia".

Vamos agora adicionar outras colunas a nosso gráfico: **Numero da Venda** e **Quantidade Vendida**.

Assim como fizemos em **Data da Venda**, vamos alterar a configuração sugerida pelo Power BI para a coluna **Numero da Venda**.

Feito isso, vamos voltar para nosso Jupyter e validar o código que usaremos dentro do Power BI na criação do Gráfico (Lembrando que não é uma obrigação. O código poderá ser feito diretamente no Power BI)

Código rodando, vamos agora levar esse código para o nosso Power BI.

The screenshot shows the Power BI Python script editor interface. At the top, there's a message about removing duplicates. Below it, the script code is displayed:

```
1 # O código a seguir para criar um dataframe e remover as linhas duplicadas sempre é executado e age como um preâmbulo para o script:  
2  
3 # dataset = pandas.DataFrame('Numero da Venda', 'Data da Venda', 'Quantidade Vendida')  
4 # dataset = dataset.drop_duplicates()  
5  
6 # Cole ou digite aqui seu código de script:  
7 import matplotlib.pyplot as plt  
8  
9 tres_lojas_df.plot(x='Data da Venda', y='Quantidade Vendida', figsize=(15, 5))  
10 plt.show()
```

To the right of the code, there's a "Valores" (Values) dropdown menu with three items: "Número da Venda", "Data da Venda", and "Quantidade Vendida". A red arrow points to the "X" button next to "Número da Venda". Below the dropdown is a "Relatório cruzado" (Cross-report) section with "Drill-through" and "Soma" (Sum) buttons. A red box highlights the "Soma" button. To the right of the "Relatório cruzado" section, there are buttons for "Remover campo" (Remove field), "Renomear para este visual" (Rename to this visual), and "Mover" (Move). A red arrow points to the "Não resumir" (Do not summarize) checkbox, which is checked. At the bottom of the editor, there's a large orange downward-pointing arrow indicating the flow from the Power BI configuration back to the script editor.

Antes de rodarmos o código, precisamos tomar alguns cuidados.

- 1) Nome do Dataframe: No Jupyter estávamos usando `tres_lojas_df`, no entanto, por padrão, o PowerBI está chamando essa variável de **dataset**;
- 2) Os valores de `x= 'Data da Venda'` e `y='Quantidade Vendida'`, só funcionam pois essas duas colunas foram incluídas no gráfico( passos anteriores);
- 3) O uso do `plt.show()` é obrigatório para a execução do código.

**Editor de scripts Python**

⚠️ As linhas duplicadas serão removidas dos dados.

```
1 # O código a seguir para criar um dataframe e remover as linhas duplicadas sempre
2
3 # dataset = pandas.DataFrame(Numero da Venda, Data da Venda, Quantidade Vendida)
4 # dataset = dataset.drop_duplicates()
5
6 # Cole ou digite aqui seu código de script:
7 import matplotlib.pyplot as plt
8
9 tres_lojas_df.plot(x='Data da Venda', y='Quantidade Vendida', figsize=(15, 5))
10 plt.show()
```

Colunas foram incluídas no gráfico, por isso podemos usá-las no código

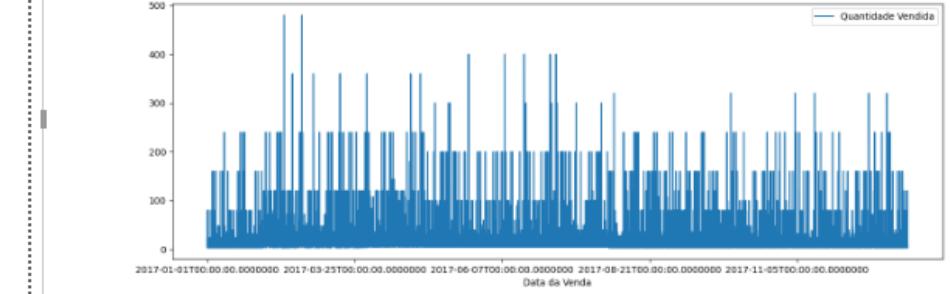
Tres\_lojas\_df precisa ser substituída por `dataset.plot`

# Cole ou digite aqui seu código de script:

```
6 # Cole ou digite aqui seu código de script:
7 import matplotlib.pyplot as plt
8
9 dataset.plot(x='Data da Venda', y='Quantidade Vendida', figsize=(15, 5))
10 plt.show()
```

Ao rodar o código, temos o nosso gráfico plotando dentro do Power BI.

Sabemos que não é dos mais bonitos, mas vamos entender como melhorar a cara dele mais para frente nesse módulo 😊.



### Editor de scripts Python

```
1 # O código a seguir para criar um dataframe e remover as linhas duplicadas sempre é executado e age
2
3 # dataset = pandas.DataFrame(Numero da Venda, Data da Venda, Quantidade Vendida)
4 # dataset = dataset.drop_duplicates()
5
6 # Cole ou digite aqui seu código de script:
7 import matplotlib.pyplot as plt
8
9 dataset.plot(x='Data da Venda', y='Quantidade Vendida', figsize=(15, 5))
10 plt.show()
```

Nessa parte do módulo, nosso objetivo é mostrar como usar gráficos mais interessantes que o usado anteriormente e aproveitar para apresentar uma nova biblioteca de gráficos chamada **Seaborn** ([doc](#)).

Vamos lembrar que estamos trabalhando em um ambiente virtual específico para a integração com o PowerBI. Portanto, ainda não temos a biblioteca do Seaborn instalada.

Usando o pip, faça a instalação dessa biblioteca.

**ATENÇÃO:** Essa instalação precisa ser feita dentro do ambiente virtual. Caso esteja com dúvidas retorne ao módulo 26 ☺ para consulta.

Como sempre vamos usar o Jupyter para validar nosso código e só depois vamos para o Power BI.

O código que usaremos é o código apresentado ao lado.

```
(pythonpowerbi) \Python e Power BI>pip install seaborn
```

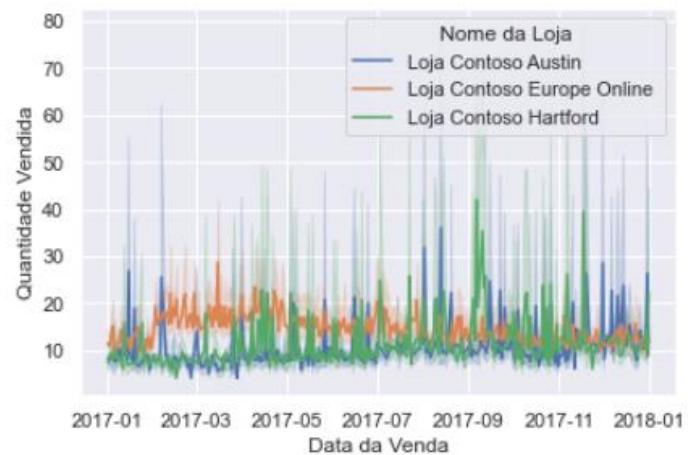
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_theme(style="darkgrid")

sns.lineplot(x="Data da Venda", y="Quantidade Vendida", hue="Nome da Loja", data=tres_lojas_df)

plt.show()
```

Define a cor de fundo do gráfico



Código rodando, vamos para o Power BI!

Vamos abrir uma nova aba na guia relatórios e criar um gráfico usando o Visual Python.

The screenshot shows the Microsoft Power BI interface in 'Report' mode. A new visual has been created, represented by a Python logo icon. The interface includes a ribbon menu at the top, a 'Visualizar' (Visualize) pane on the left, and a 'Valores' (Values) pane on the right. An 'Editor de scripts Python' (Python Script Editor) window is open at the bottom, with the message: 'Arraste os campos para a área Valores no painel Visualização para iniciar a criação de scripts.' (Drag fields to the Values panel in the Visualization pane to start creating scripts.)

Guia Relatório do Power BI onde são criados os gráficos

Gráfico criado pelo Visual Python. Ainda sem conteúdo.

Ícone de criação de Gráfico via Python (Visual Python)

Editor de código Python

Selecione as colunas de interesse ( Número da Venda, Data da Venda, Nome da Loja e Quantidade Vendida)

**ATENÇÃO:** Certifique-se que as colunas estão com a exibição correta:

- Número da Venda->Não resumir;
- Data da Venda ->Data da Venda

Feito isso, cole o código em Python que elaboramos nos Jupyter e execute o Python Script.

The screenshot shows the Power BI interface. On the left is a line plot titled "Número da Venda, Data da Venda, Nome da Loja e Quantidade Vendida". The Y-axis is "Quantidade Vendida" (Quantity Sold) from 0 to 80. The X-axis is "Data da Venda" (Date of Sale). The legend indicates three data series: "Loja Contoso Austin" (blue), "Loja Contoso Europe Online" (orange), and "Loja Contoso Hartford" (green). The plot shows high volatility with several sharp peaks. To the right is the "Campos" (Fields) pane, which lists fields from the "vendas\_df" table. Fields selected with checked checkboxes are highlighted in red: "Data da Venda", "Nome da Loja", and "Quantidade Vendida". A red arrow points from the "Nome da Loja" checkbox to the Python script editor below. The "Editor de scripts Python" (Python Script Editor) contains the following code:

```
1 # O código a seguir para criar um dataframe e remover as linhas duplicadas sempre é executado e age como o script principal
2
3 # dataset = pandas.DataFrame(Número da Venda, Data da Venda, Nome da Loja, Quantidade Vendida)
4 # dataset = dataset.drop_duplicates()
5
6 # Cole ou digite aqui seu código de script:
7 import matplotlib.pyplot as plt
8 import seaborn as sns
9
10 sns.set_theme(style="darkgrid")
11
12 sns.lineplot(x="Data da Venda", y="Quantidade Vendida", hue="Nome da Loja", data=dataset)
13 plt.show()
```

An orange callout box with the text "Ao colar o código feito no Python para o Power BI, é necessário alterar o nome do dataframe para dataset" has an arrow pointing to the line "data=dataset" in the Python code.

Gráfico criado, mas o tamanho ainda não é o ideal.

Ao invés de voltarmos para o Jupyter, vamos fazer esse tratamento diretamente no campo de Script Python do Power BI.

Editor de scripts Python

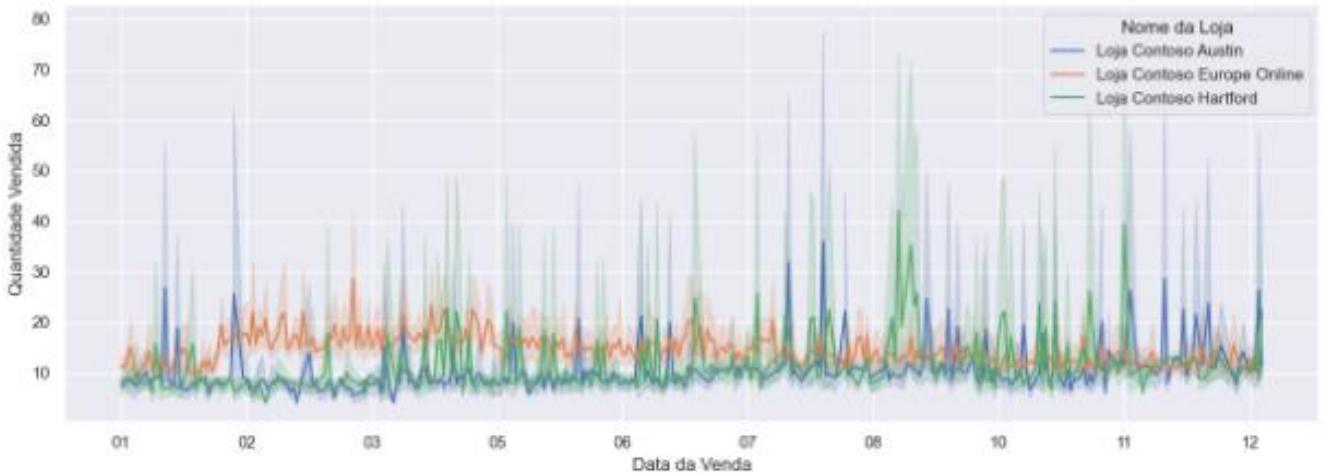
```
5  
6 # Cole ou digite aqui seu código de script:  
7 import matplotlib.pyplot as plt  
8 from matplotlib.dates import DateFormatter  
9 import seaborn as sns  
10  
11 sns.set_theme(style="darkgrid")  
12  
13 fig,ax = plt.subplots(figsize=(15,5))  
14 sns.lineplot(x="Data da Venda", y="Quantidade Vendida", hue="Nome da Loja", data=dataset,ax=ax)  
15 ax.xaxis.set_major_locator(plt.MaxNLocator(12))  
16 ax.xaxis.set_major_formatter(DateFormatter('%m'))  
17  
18 plt.show()
```

Vamos importar uma nova biblioteca para editar o gráfico.

Altera o tamanho do gráfico.

Formata os 12 dados para 12 meses

Altera o eixo X para apenas 12 dados



Módulo 29

# Transformando Python em exe

Nesse módulo, vamos aprender como transformar nossos códigos em programas **EXECUTÁVEIS** (que funcionam quando damos um duplo clique).

Até agora, sempre usamos o Jupyter para rodar nossos códigos, mas queremos fazer um arquivo que rode em qualquer computador mesmo que o Python não esteja instalado.

Para transformarmos nosso código em um arquivo executável vamos seguir os passos abaixo:

- 1) **Passo 1:** Garantir que seu código **PRECISA FUNCIONAR SEMPRE**;
- 2) **Passo 2:** Transformar seu arquivo `.ipynb` em um arquivo `.py`;
- 3) **Passo 3:** Usar uma biblioteca de conversão para transformar o arquivo `.py` em `.exe`;
- 4) **Passo 4:** Testar e adaptar o que for necessário A CADA ETAPA de conversão.



### ATENÇÃO !

Se seu PC é Linux ou MAC, esse módulo é em parte “redundante” pois por padrão esses sistemas operacionais já possuem dentro deles o Python instalado. No entanto, não deixe de aprender ☺

Antes de irmos para arquivos mais complexos, vamos pegar um código bem básico e aplicar os 4 passos citados anteriormente.

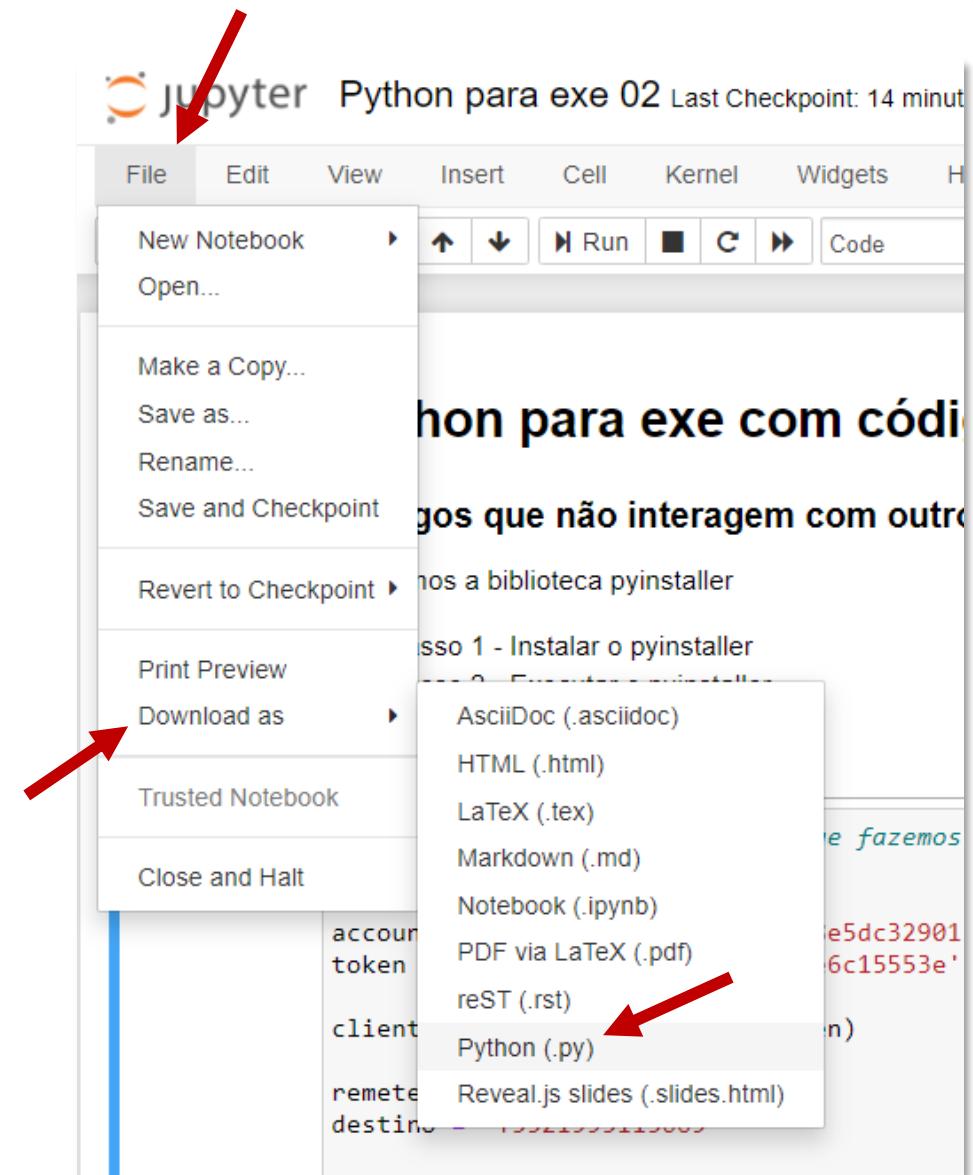
Vamos utilizar um código já conhecido por nós no módulo de APIs. Por que esse é considerando um código básico.

Essencialmente, como não estamos acessando informações dentro do nosso PC, a chance de erro é um pouco menor no caso de um executável.

No Passo 1: Garantir que seu código **PRECISA FUNCIONAR SEMPRE**; já está atendido, pois usamos este mesmo código no gabarito de módulos anteriores.

Para realizarmos o Passo 2: Transformar seu arquivo .ipynb em um arquivo .py; basta seguir os passos indicados ao lado no próprio Jupyter.

Feito isso, temos agora arquivo com extensão .py  
Teste!



Baixado o arquivo .py vamos executá-lo. Ele ainda não possui a extensão .exe, portanto, ainda precisa de um ambiente para sua execução.

Esse ambiente será o Python! E como acessar o Python fora do Jupyter?

- 1) Abra seu Anaconda Prompt;
- 2) Encontre o arquivo na pasta onde o mesmo foi baixado;
- 3) Execute-o usando o Python:

`python "Python para exe 02.py"`

- 4) Verifique se o código executou corretamente.

```
Pasta de C:\Users\danie\OneDrive\Área de Trabalho\Cartilhas Python\Arquivos apostila\Módulo 29
12/04/2021 17:18    <DIR>      .
12/04/2021 17:18    <DIR>      ..
12/04/2021 16:58    <DIR>      .ipynb_checkpoints
12/04/2021 17:13                3.072 Python para exe 02.ipynb
12/04/2021 17:14                1.676 Python para exe 02.py
                                2 arquivo(s)          4.748 bytes
                                3 pasta(s)   851.605.393.408 bytes disponíveis
```

Arquivo com extensão .py criado

Informando ao Anaconda Prompt que o comando será executado via Python

```
\Módulo 29>python "Python para exe 02.py"
SMeC4e91ebc0c14223ad320dec2bf6992c
```

Confirmação da execução

Agora vamos para o Passo 3: Usar uma biblioteca de conversão para transformar o arquivo .py em .exe;

Para isso vamos antes de tudo instalar uma biblioteca de conversão:

### pyinstaller

Como já fizemos diversas vezes, vamos instalá-la através do nosso Anaconda Prompt.

### pip install pyinstaller

Feito isso, vamos instalar nosso arquivo .py através dessa biblioteca.

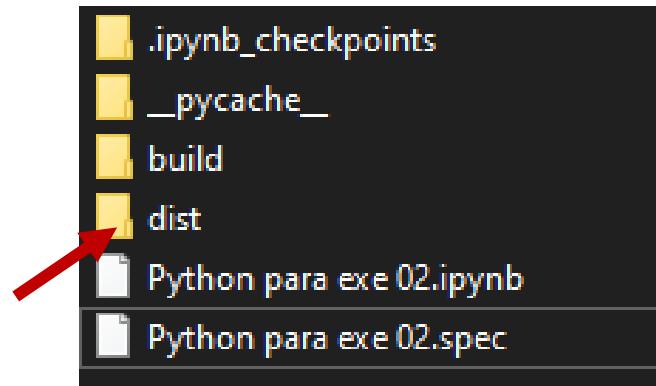
### pyinstaller -w "Python para exe 02.py"

Após o processamento o pyinstaller criou uma série de arquivos e entre eles o nosso arquivo executável 😊

Basta dar um duplo clique e agora temos um arquivo executável

```
>pip install pyinstaller
```

```
>pyinstaller -w "Python para exe 02.py"
```

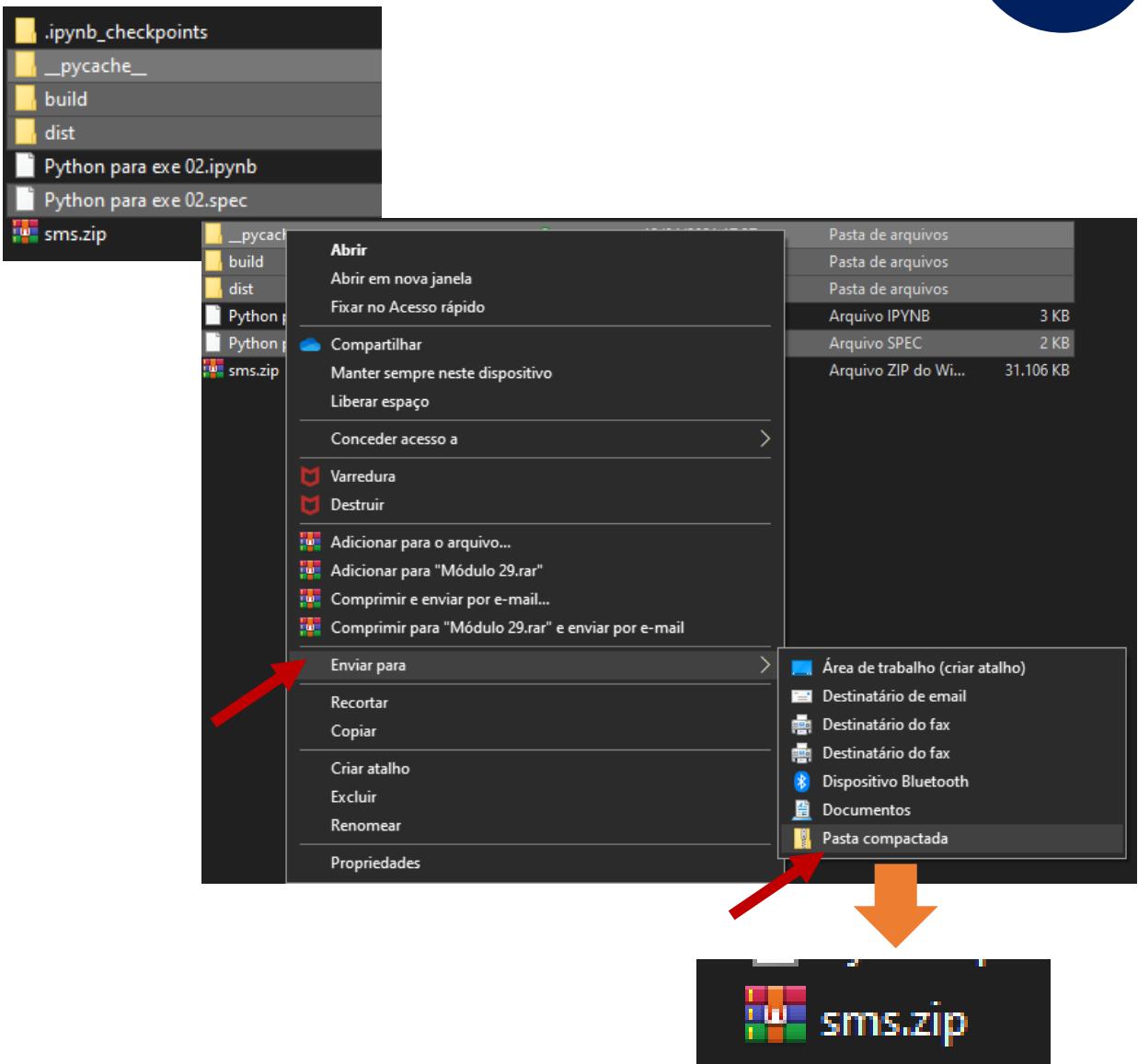


Dentro da pasta dist temos um arquivo executável (.exe)

Perceba que foram criadas várias pastas assim como vários arquivos. Para facilitar a vida do usuário podemos criar um atalho do arquivo para área de Trabalho ☺

Assim, basta clicar para executar o programa.

Caso seja do seu interesse enviar esse programa para outras pessoas, basta juntar todos os arquivos que foram criados em uma única pasta compactada conforme apresentado ao lado.

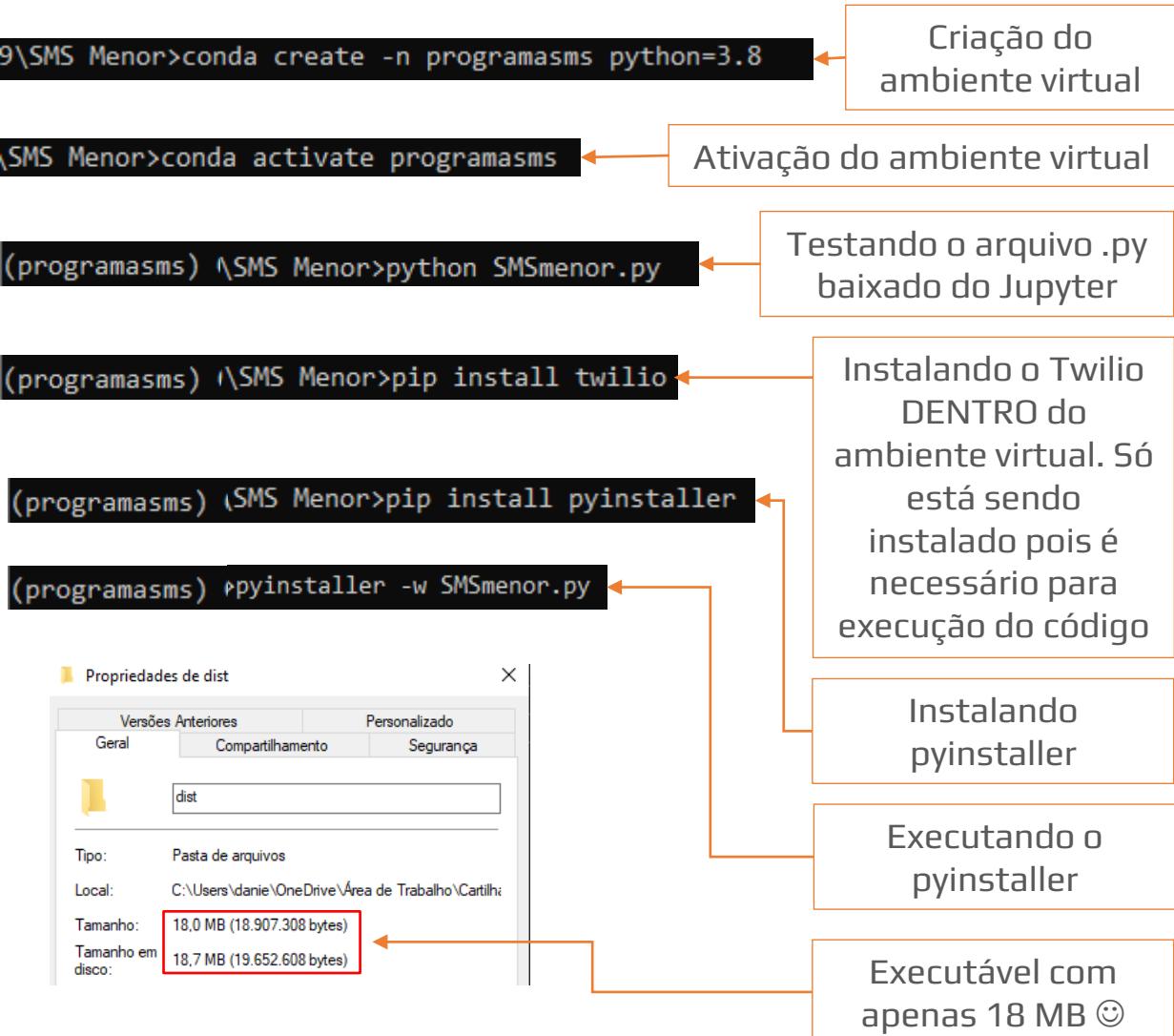


O que permitirá reduzir o tamanho do executável será criá-lo dentro de um ambiente virtual onde apenas o que é **NECESSÁRIO** para seu funcionamento estará instalado.

Para esse exemplo, iremos criar uma pasta chamada **SMS MENOR** onde dentro dela, criaremos um ambiente virtual dedicado.

Para isso, usaremos o passo a passo aprendido no módulo 26 dessa apostila.

Como nesse caso, não é uma questão de versão do Python, vamos utilizar a versão 3.8 ao criarmos esse ambiente virtual.



Antes de irmos para adaptação do código, vamos entender o problema e o código original.

Para a Hashtag realizar anúncios no Youtube, ela precisa identificar quais são os vídeos em que ela deseja anunciar.

Para isso, temos um código que busca os links desses vídeos de forma automática.

Se rodarmos esse código no computador da Hashtag sempre rodará pois ele está sendo rodado no computador que foi criado (nesse caso o pc do Alon, conforme exemplo ao lado).

Vamos dizer que agora queremos rodar esse código no computador do Lira, como fazer?

Vamos precisar criar um código genérico que funcione em qualquer computador.

```
#Ler csv  
buscas_df = pd.read_csv(r'C:\Users\alonp\Downloads\Canais Youtube.csv', encoding = 'ISO-8859-1', sep=';')  
display(buscas_df.head())
```

O fato deste código está vinculado a uma pasta específica dentro do usuário Alon, nos impede de criar um arquivo executável que possa ser usado por diversas pessoas.

Iremos precisar criar um código genérico que se aplique a qualquer usuário

Para resolver esse problema, temos 3 soluções possíveis:

## 1) Indicar apenas o nome do arquivo:

Nesse caso, é necessário que o arquivo esteja OBRIGATORIAMENTE na mesma pasta do nosso programa python.

## 2) Usar a biblioteca 'os' ou a 'pathlib':

Essas bibliotecas, permitem buscar o caminho do computador. Sendo assim, usando esse código poderíamos o local do pc antes da execução do código em si.

## 3) Criar uma interface para o usuário escolher o local do arquivo:

Usando a biblioteca TKINTER, ofereceremos ao usuário indicar onde está o arquivo que será lido.

### Caso 1 - Indicar apenas o nome do arquivo:

```
buscas_df = pd.read_csv(r'C:\Users\alonp\Downloads\Canais Youtube.csv', encoding = 'ISO-8859-1', sep=';')
display(buscas_df.head())
```

```
buscas_df = pd.read_csv(r'Canais Youtube.csv', encoding = 'ISO-8859-1', sep=';')
display(buscas_df.head())
```

### Caso 2 - Usar a biblioteca 'os' ou a 'pathlib'

```
buscas_df = pd.read_csv(r'C:\Users\alonp\Downloads\Canais Youtube.csv', encoding = 'ISO-8859-1', sep=';')
display(buscas_df.head())
```

```
import os
caminho = os.getcwd()
```

Identifica o local do arquivo antes da execução da extração

```
buscas_df = pd.read_csv(os.join(caminho,r'Canais Youtube.csv'), encoding = 'ISO-8859-1', sep=';')
display(buscas_df.head())
```

### Caso 3 -Criar uma interface para o usuário escolher o local do arquivo

```
buscas_df = pd.read_csv(r'C:\Users\alonp\Downloads\Canais Youtube.csv', encoding = 'ISO-8859-1', sep=';')
display(buscas_df.head())
```

```
from tkinter import *
import tkinter.filedialog
from tkinter import messagebox

janela = Tk()
arquivo=tkinter.filedialog.askopenfilename(title = 'Selecione o arquivo com os Canais do Youtube a serem mapeados')
janela.destroy()

buscas_df = pd.read_csv(arquivo, encoding = 'ISO-8859-1', sep=';')
display(buscas_df.head())
```

## Módulo 29 – Transformando Python em exe – Adaptando o código de Arquivos Complexos (3/4)

428

Usaremos o caso 3 daqui para frente pois nesse caso, o risco de erro é menor visto que o usuário pode escolher o local do arquivo.

Vamos entender um pouco melhor o código apresentado anteriormente.

```
from tkinter import *
import tkinter.filedialog
from tkinter import messagebox

janela = Tk()
arquivo=tkinter.filedialog.askopenfilename(title = 'Selecione o arquivo com os Canais do Youtube a serem mapeados')
janela.destroy()

buscas_df = pd.read_csv(arquivo, encoding = 'ISO-8859-1', sep=';')
display(buscas_df.head())

if canal in np.nan:
    break
hrefs.append(canal)
driver.get(canal)
myElem = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "content")))
time.sleep(2)
tab = driver.find_element(By.ID, "content")
time.sleep(2)
altura = 0
nova_altura = 1
while nova_altura > altura:
    nova_altura =
```

Através do TkInter abre uma janela para seleção do arquivo desejado pelo usuário

Cria Janela Tkinter

Destroi a janela Tkinter

The diagram illustrates the interaction between the Python code and the file selection dialog. It shows two boxes: 'Cria Janela Tkinter' pointing to the line 'janela = Tk()' and 'Destroi a janela Tkinter' pointing to the line 'janela.destroy()'. A red box highlights the line 'arquivo=tkinter.filedialog.askopenfilename(title = 'Selecione o arquivo com os Canais do Youtube a serem mapeados')'. An orange arrow points from this line to a screenshot of the Windows File Explorer dialog titled 'Selecione o arquivo com os Canais do Youtube a serem mapeados'. The dialog shows a list of files, with 'Canais Youtube.csv' selected. Another orange arrow points from the 'Nome:' field in the dialog back to the code line 'arquivo=...'. A callout box with the text 'Através do TkInter abre uma janela para seleção do arquivo desejado pelo usuário' is positioned above the dialog.

No entanto, a partir do momento que o usuário interage com o nosso código, é importante que ele também seja comunicado que o mesmo foi executado com sucesso.

Para isso, vamos usar novamente o Tkinter mas agora apresentando uma mensagem conforme indicado abaixo.

Com isso, podemos garantir que o programa rodará em qualquer computador ou qualquer usuário que venha a utilizar o programa.

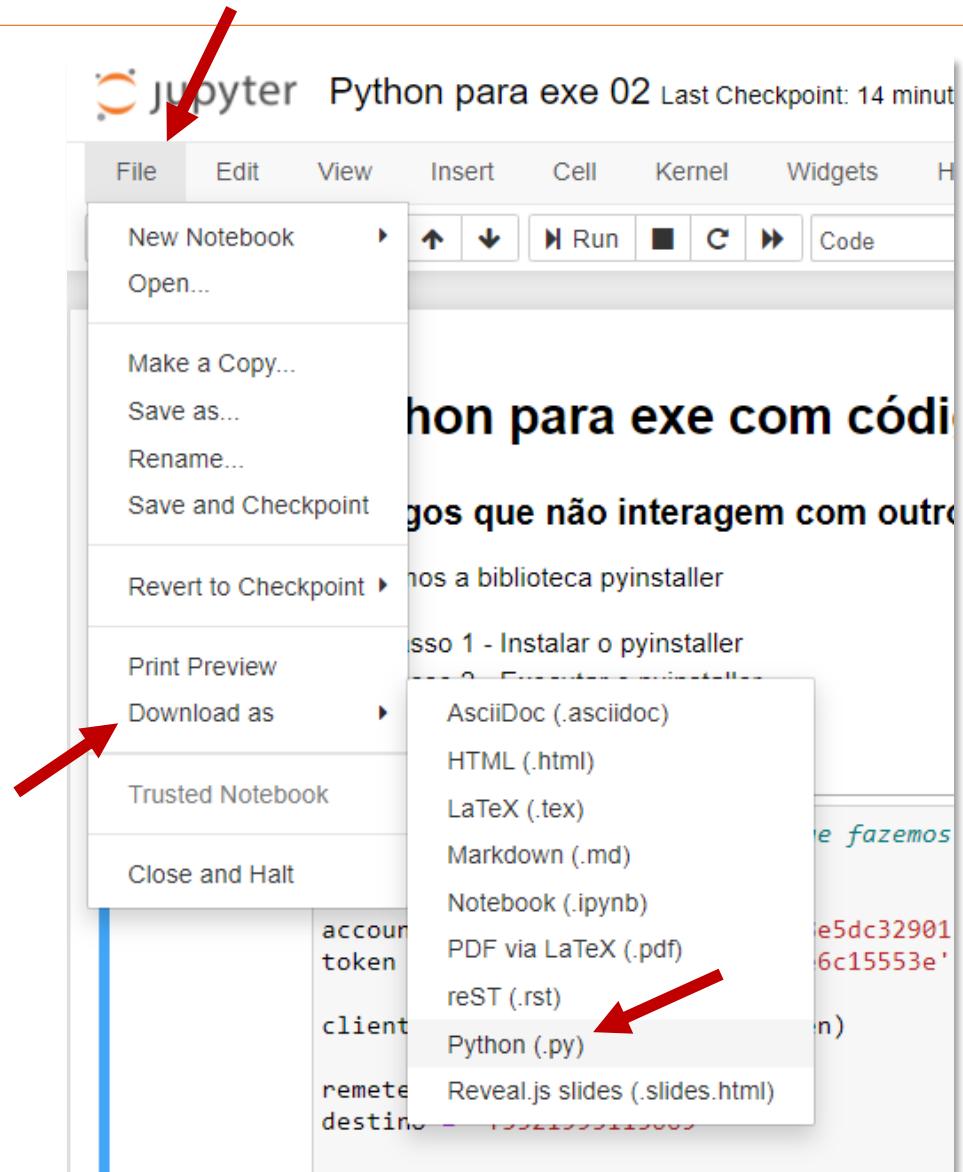
```
#salvando o resultado em um csv
hrefs_df = pd.DataFrame(href)
hrefs_df.to_csv(r'Canais Prontos.csv', sep=',', encoding='utf-8')
root= Tk()
messagebox.showinfo("Programa Finalizado com Sucesso", "Seu arquivo csv foi gerado com sucesso na pasta do Programa")
root.destroy()
```

## Módulo 29 – Transformando Python em exe – Python para exe com Arquivos Complexos (1/5)

430

Bem, agora que temos nosso programa rodando, vamos transformá-lo em um executável.

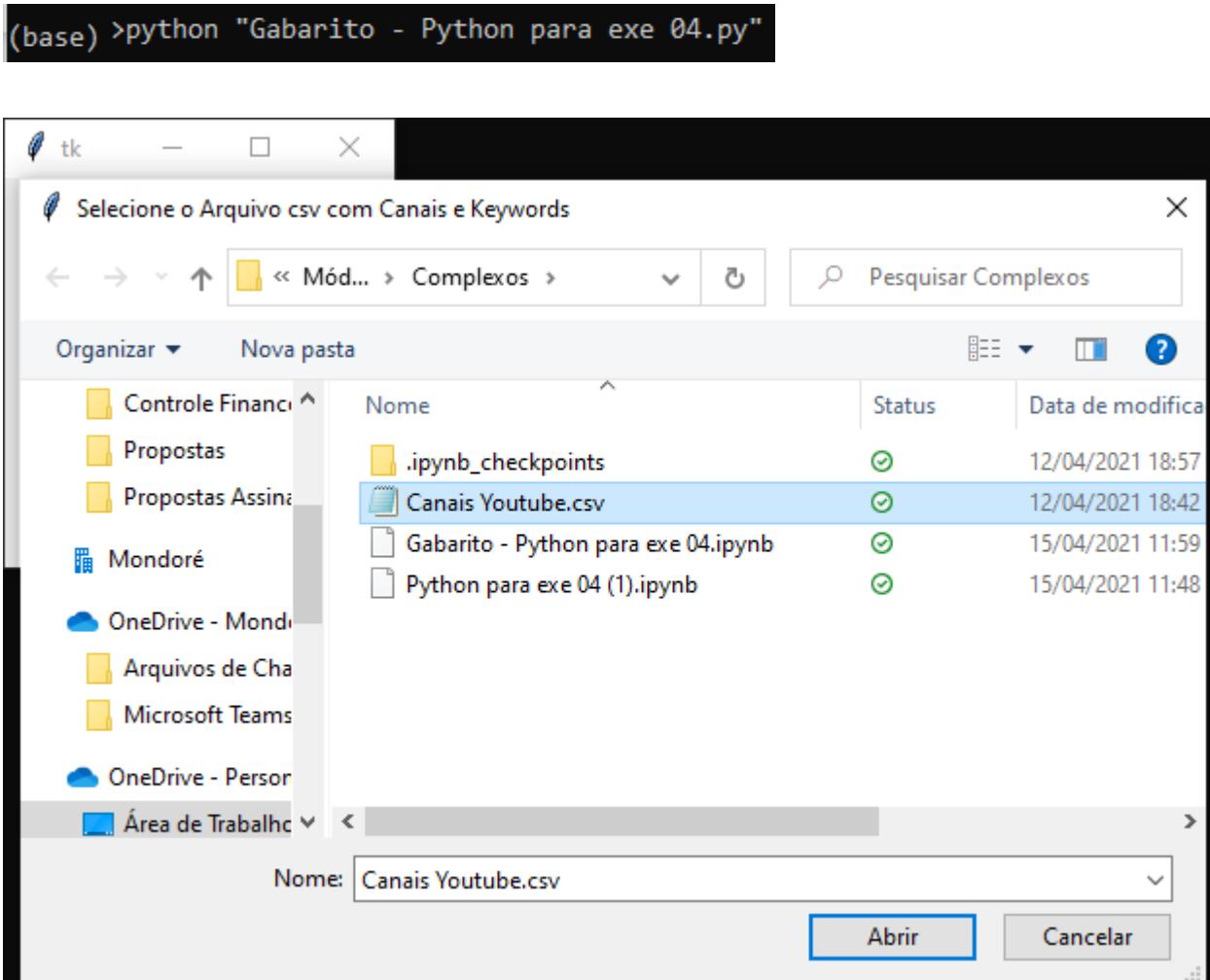
Assim como vimos anteriormente, o primeiro passo é baixar o nosso arquivo para a versão .py, conforme apresentado ao lado.



Usando o **Anaconda Prompt** execute seu arquivo.py e veja se tudo funciona bem.

Ocorrendo tudo certo, podemos ir para o próximo passo que já vimos anteriormente.

Transformar esse arquivo em um arquivo executável.



Para isso usaremos uma nova biblioteca:

**auto-py-to-exe**

Assim como fizemos em exemplos anteriores, vamos instalar essa biblioteca utilizando o comando abaixo:

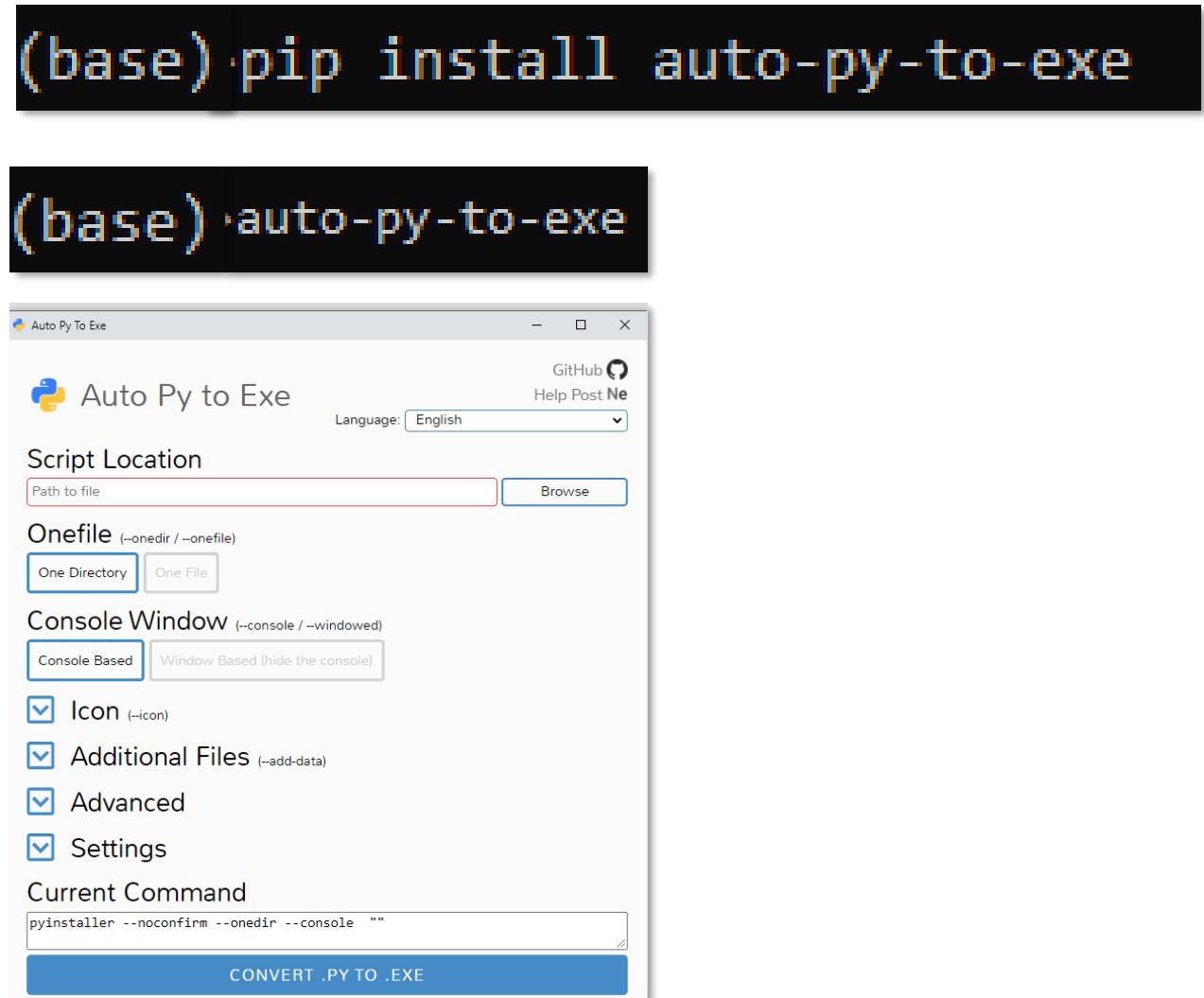
**pip install auto-py-to-exe**

Após a instalação, basta executarmos nossa biblioteca via prompt de comando do Anaconda usando:

**auto-py-to-exe**

Essa biblioteca, nos oferece uma interface gráfica para criarmos esse novo executável, conforme apresentado ao lado.

Vamos entender no próximo slide como utilizá-lo.



**Script Location:** Qual o script que você deseja transformar em executável. Para nosso caso específico usaremos "Gabarito - Python para exe 04.py"

**Onefile:** Decisão se tudo deverá ser comprimido em um arquivo único ou uma pasta com tudo;

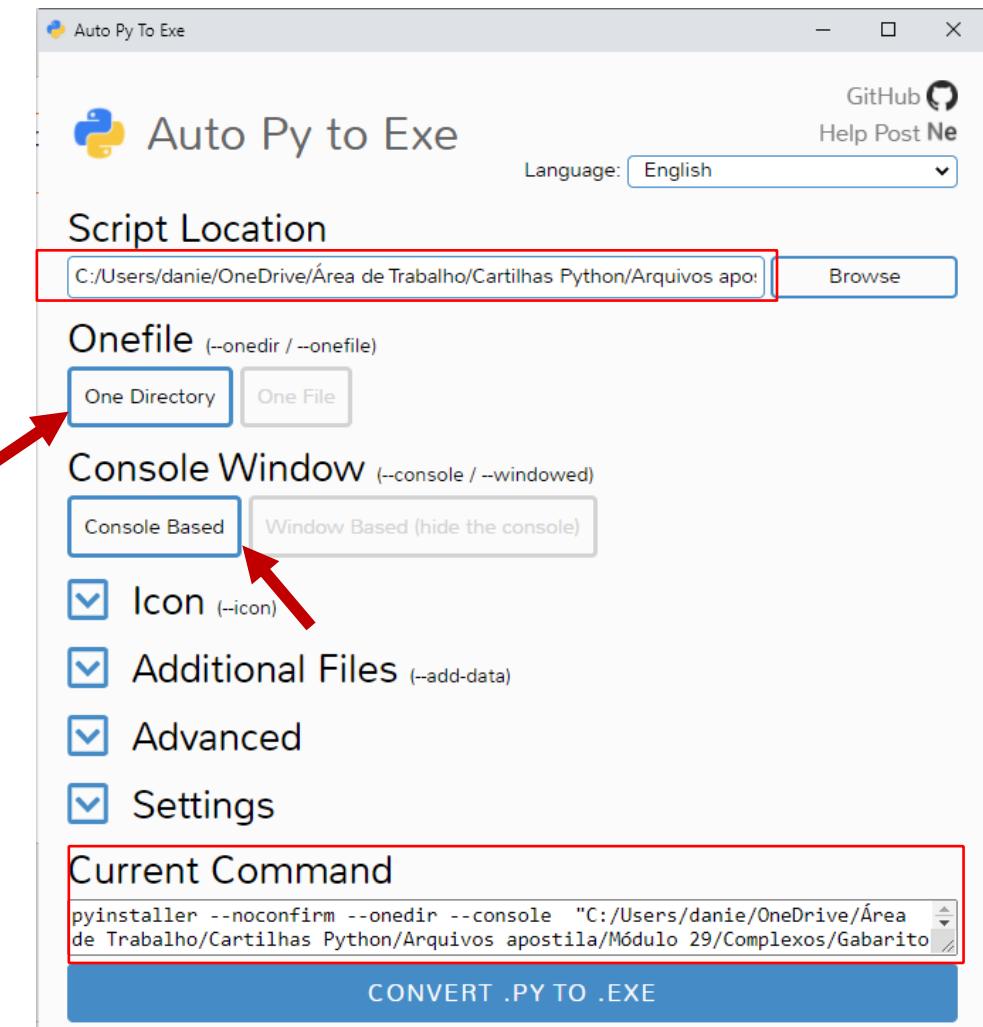
**Console Window:** Indicado usar Console Based;

**Icon:** Caso você queira usar um ícone especial para seu programa, é só importar a imagem;

**Additional Files:** Onde se insere arquivos que são necessários para execução do código. Para nosso exemplo o chromedriver.exe por exemplo;

**Advanced:** Acionar –debug todas as opções são possíveis com exceção de vazio;

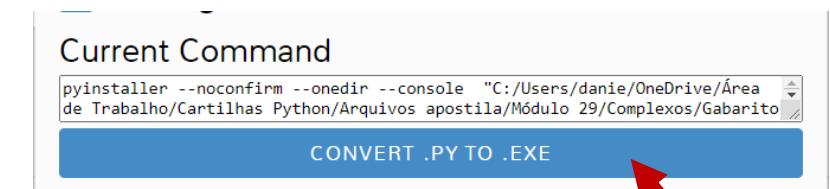
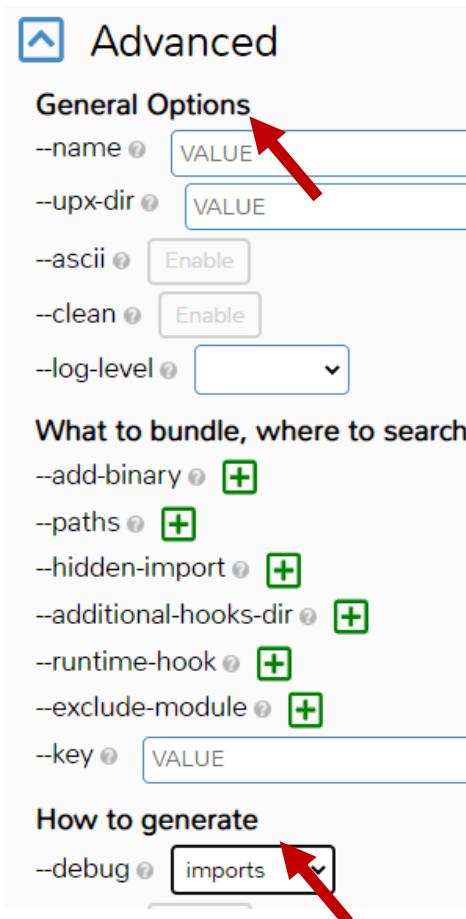
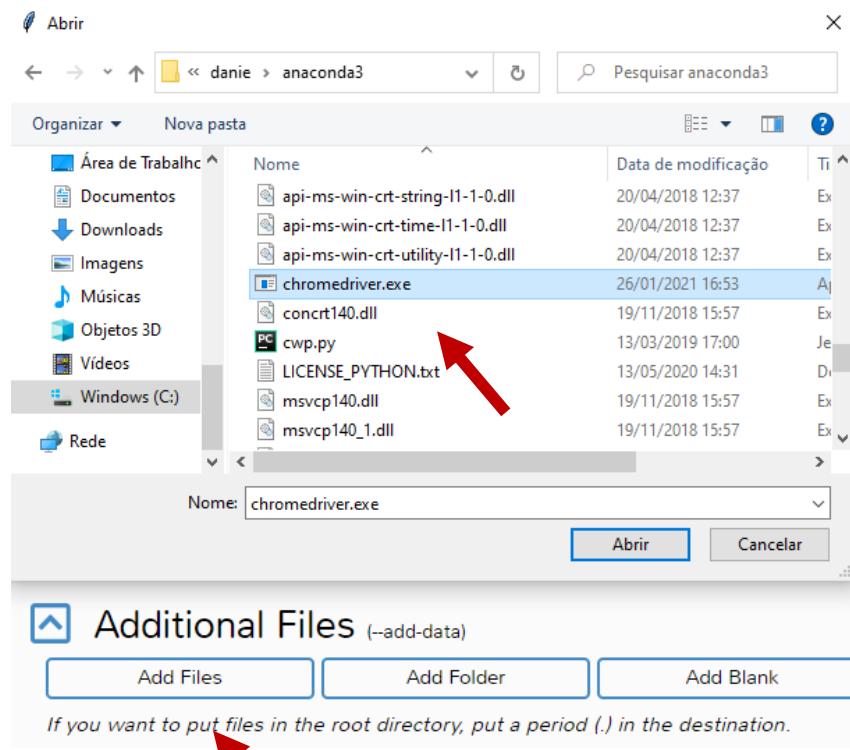
**Current Command:** Cria um código para ser executado no prompt do anaconda (perceba que se utiliza do pyinstaller



## Módulo 29 – Transformando Python em exe – Python para exe com Arquivos Complexos (5/5)

434

Aqui colocamos todas configurações necessárias. Feitas as configurações, basta cliclar em **convert.py to.exe**.



Pronto! Agora basta acessar a pasta OUTPUT criada pelo Auto-py-to-exe e executar 😊

Módulo 30

# Orientação a Objetos Completo - Classes e Métodos

Até agora, usamos o Jupyter Notebook para todos nossos códigos. Mas como muitos já devem ter pelo menos ouvido falar nessa altura do curso, existem outras IDEs no mercado que nos permitem escrever nossos códigos. Assim como fazemos com o Jupyter.

Para o Python, um dos mais “famosos” é o Pycharm ele pode ser baixado no [link](#).

Basta agora seguir os passos de instalação desse e próximos slides ☺

1

Clique em Download



2

Escolha a opção Community(gratuita)

This screenshot shows the 'Download PyCharm' section of the website. It has tabs for Windows, macOS, and Linux. Under the Windows tab, there are two options: 'Professional' and 'Community'. The 'Professional' option is described as being for both Scientific and Web Python development, with HTML, JS, and SQL support. The 'Community' option is described as being for pure Python development. Both options have a 'Download' button. A red arrow points to the 'Community' download button.

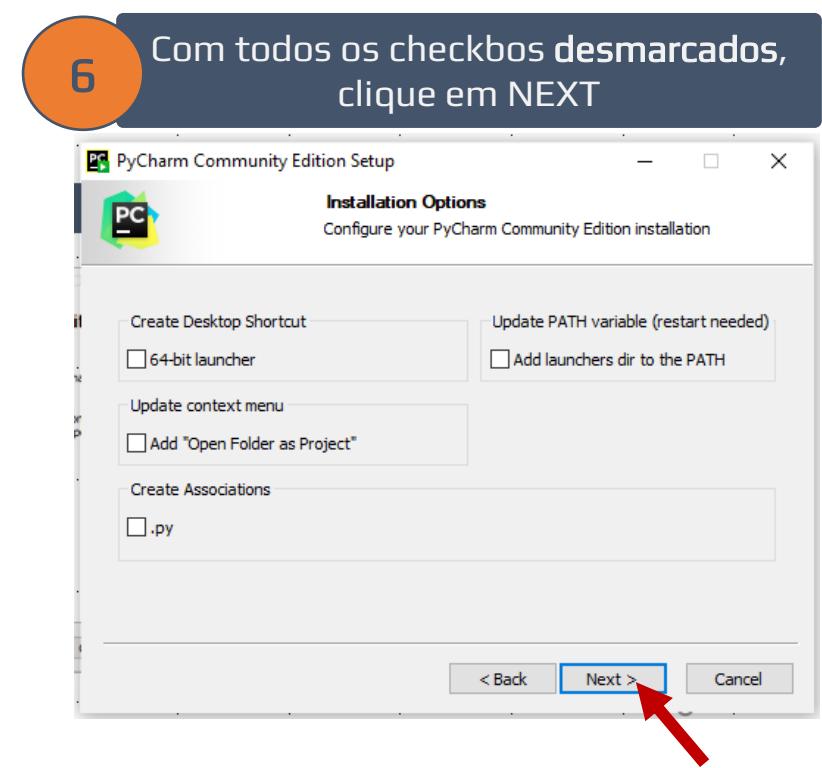
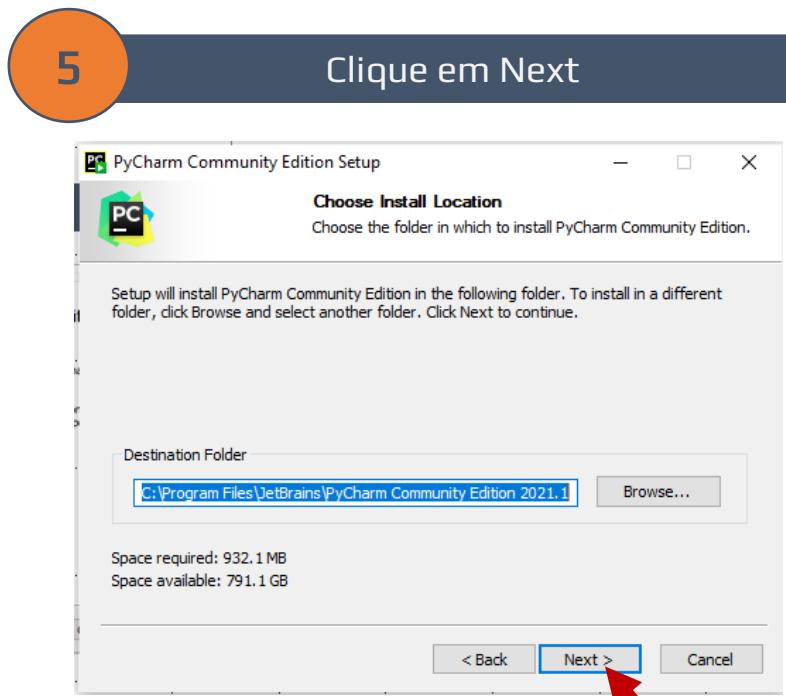
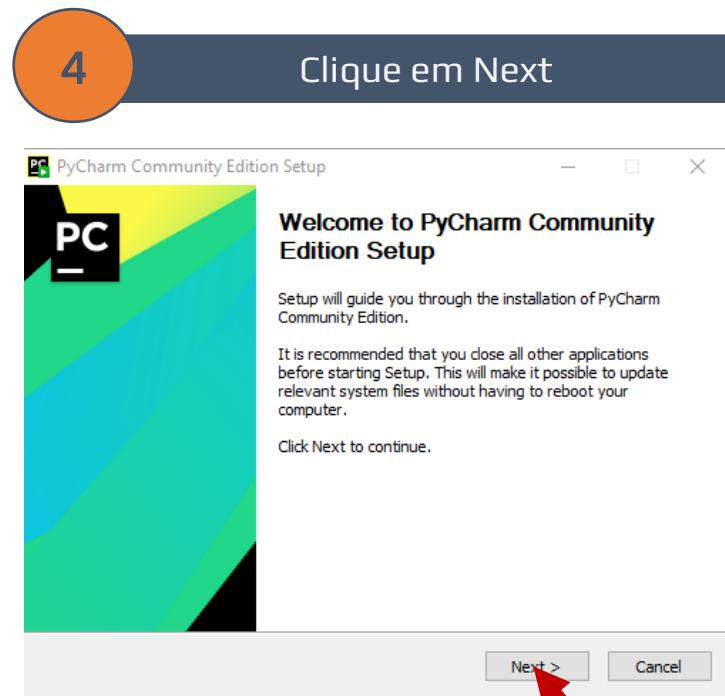
3

Aguarde o download e abra o executável

The image consists of two parts. On the left, a screenshot of a file download window titled 'pycharm-commun....exe' showing a progress bar at 166/364 MB with the text 'Um minuto resta...'. A red arrow points to the top right corner of this window. On the right, a screenshot of the 'PyCharm Community Edition Setup' window titled 'Welcome to PyCharm Community Edition Setup'. The window shows the PyCharm logo and some setup instructions.

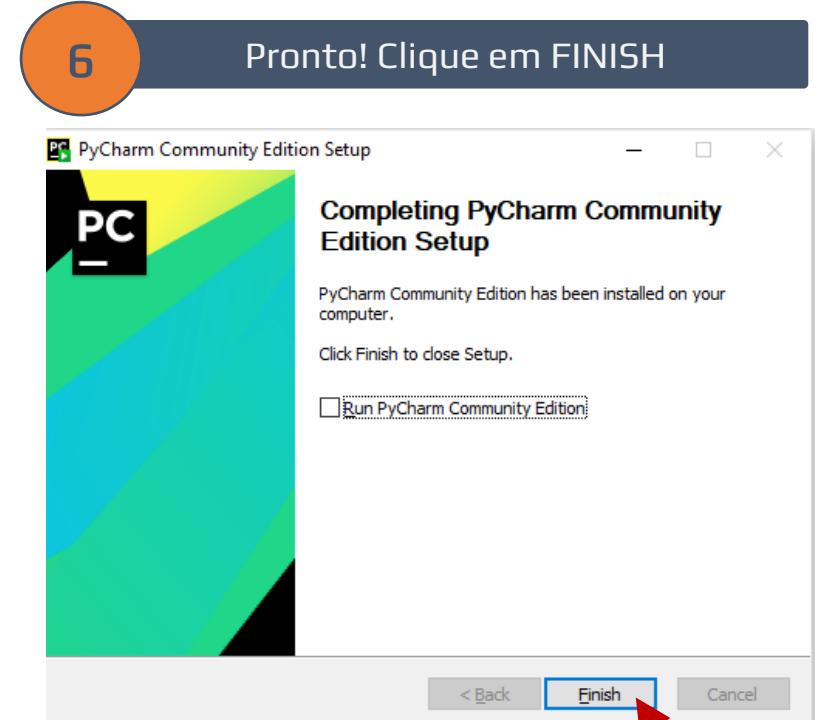
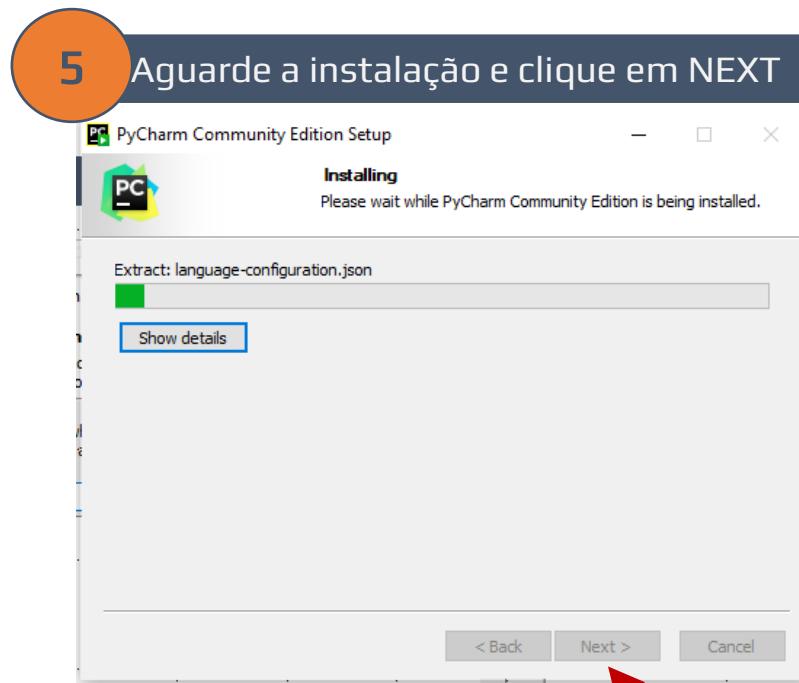
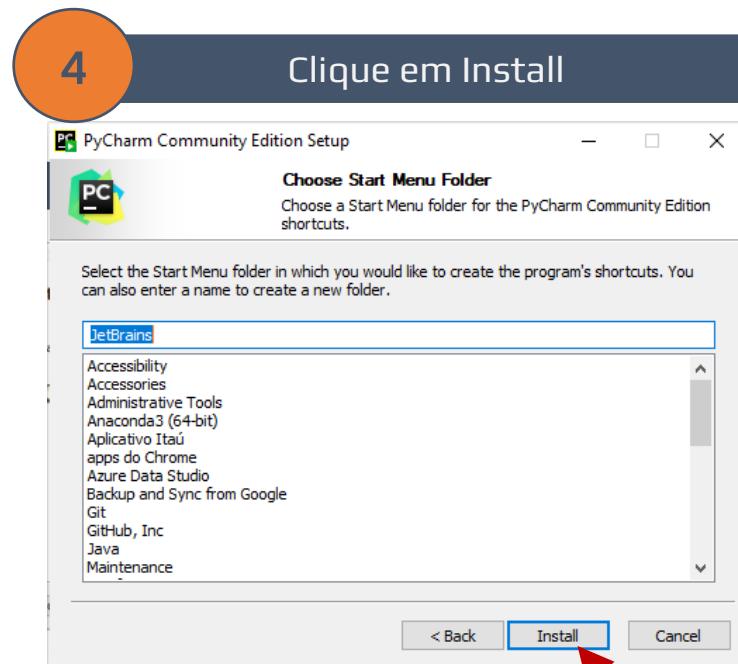
# Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Instalando o PyCharm (2/5)

437



# Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Instalando o PyCharm (3/5)

438



## Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Instalando o PyCharm (4/5)

439

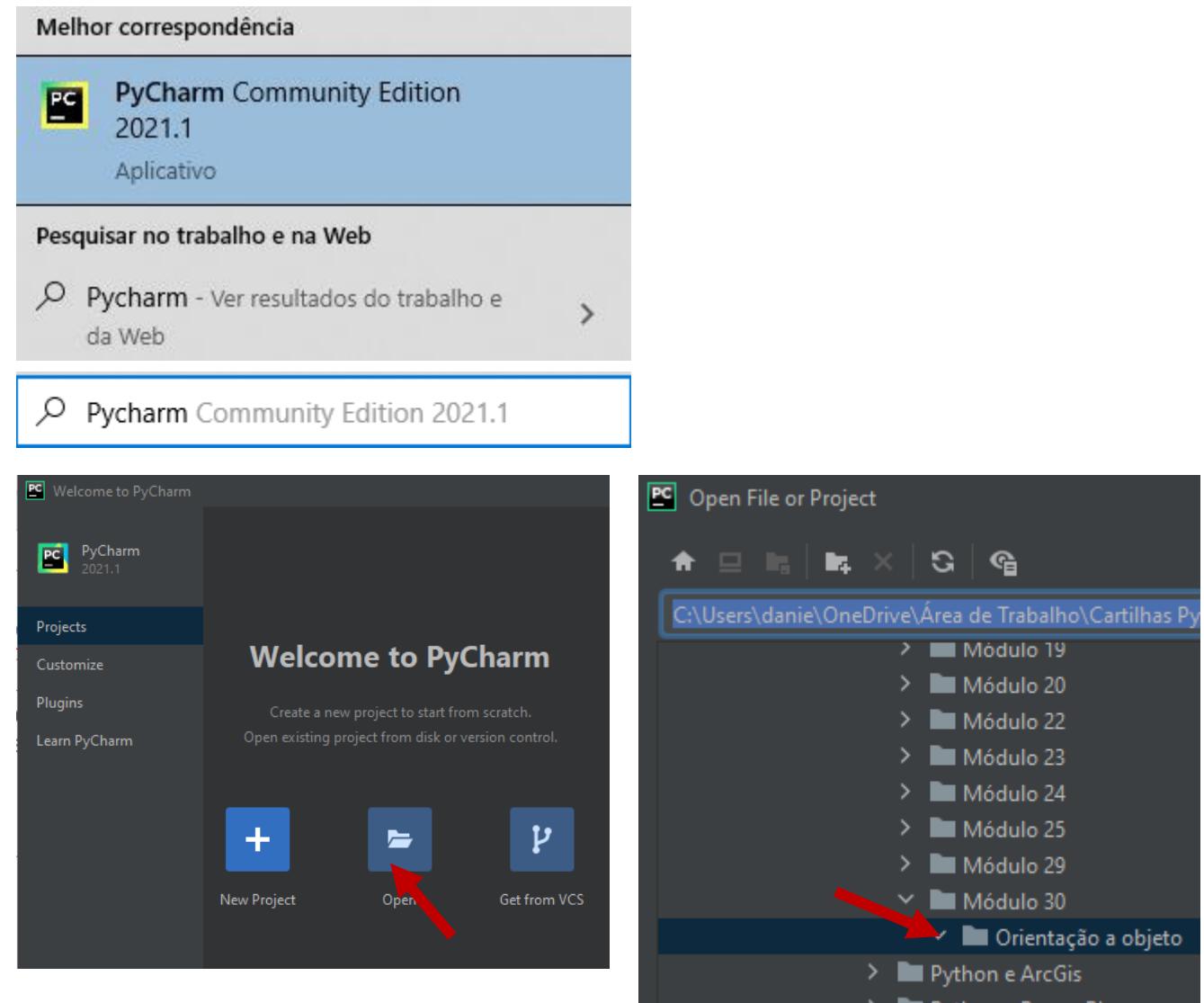
Feita a instalação, vamos conhecer o nosso Pycharm. Para acessá-lo basta digitar Pycharm no menu iniciar.

Pycharm Aberto você encontrará essa tela preta que muito provavelmente será igual ao print ao lado.

Aqui, você poderá criar um novo projeto ou abrir uma pasta existente.

No nosso caso, vamos abrir uma pasta que criamos dedicada a esse módulo chamada **Orientação a Objeto**.

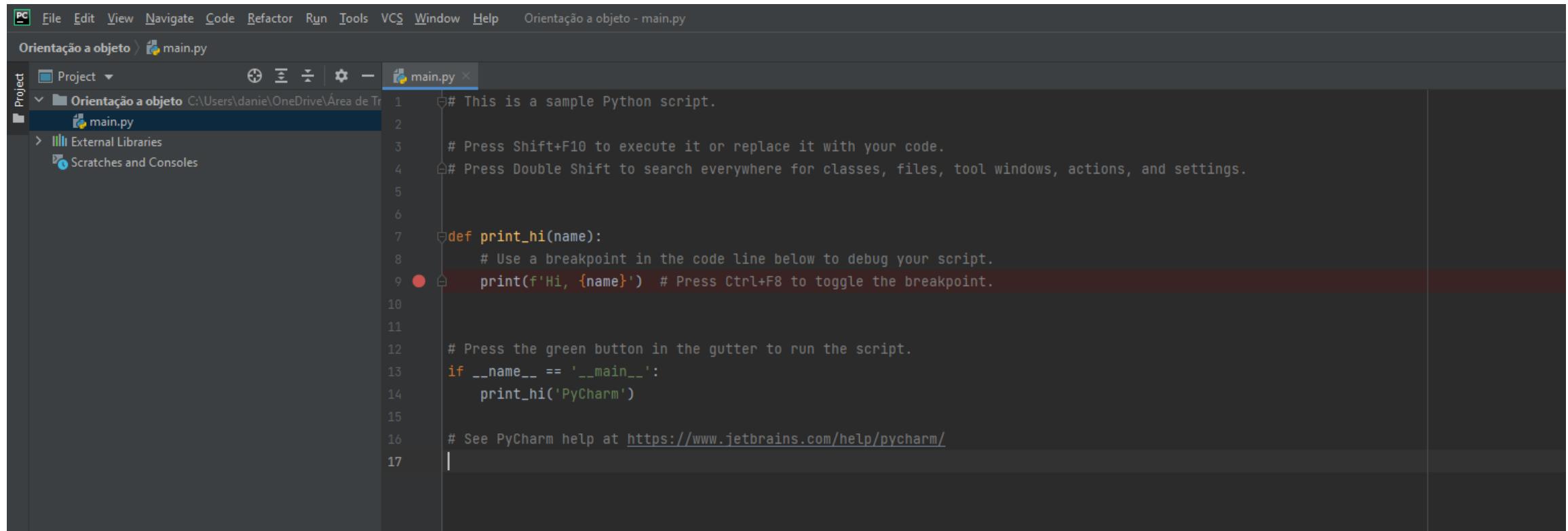
Como sugestão, use a mesma premissa para facilitar o aprendizado, MAS, sinta-se livre para fazer diferente se quiser 😊.



# Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Instalando o PyCharm (5/5)

440

Agora estamos com nosso Pycharm preparado para iniciarmos nossos códigos. Mas isso, vamos explicar melhor nos próximos slides. Só nos resta agora entender um pouco mais sobre Orientação a Objetos



```
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.

def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}') # Press Ctrl+F8 to toggle the breakpoint.

# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

Se você se lembra [lá trás](#) falamos um pouco sobre orientação a objeto mas sem nos aprofundarmos muito.

Agora chegou o momento de entendermos o que fato significa esse termo.

O grande diferencial do Python e você ouvirá esse tipo de frase por ai” é que ele é orientado a objeto”.

**Tudo no Python é um objeto!**

Como assim ?

- Lista é objeto? Sim
- Dicionário é objeto? Sim
- E as tuplas? Também...
- E... Também ☺.

Esses objetos no Python são **Classes** e essas classes possuem Métodos e atributos.

Veja o exemplo ao lado.

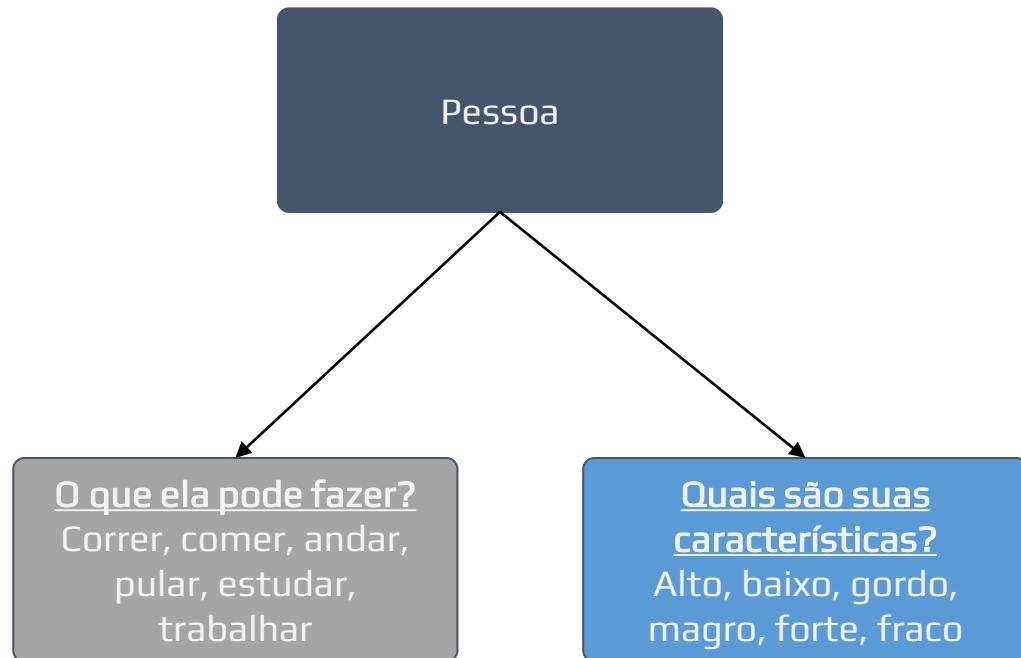
Sem nos preocuparmos com nomes, imagine uma pessoa.

O que uma pessoa pode fazer?

Quais são as características de uma pessoa?

Podemos claramente perceber que essas duas perguntas possuem respostas muito claras.

O que ela pode fazer em geral são ações desempenhadas pela pessoa, já as características simplesmente existem na pessoa. Ações como correr ou comer, podem afetar a característica de ser forte ou fraco, mas são coisas distintas.



Usando o mesmo exemplo, vamos agora fingir que essa pessoa existe dentro do Python.

Nesse caso, essa pessoa seria um **objeto ou classe**.

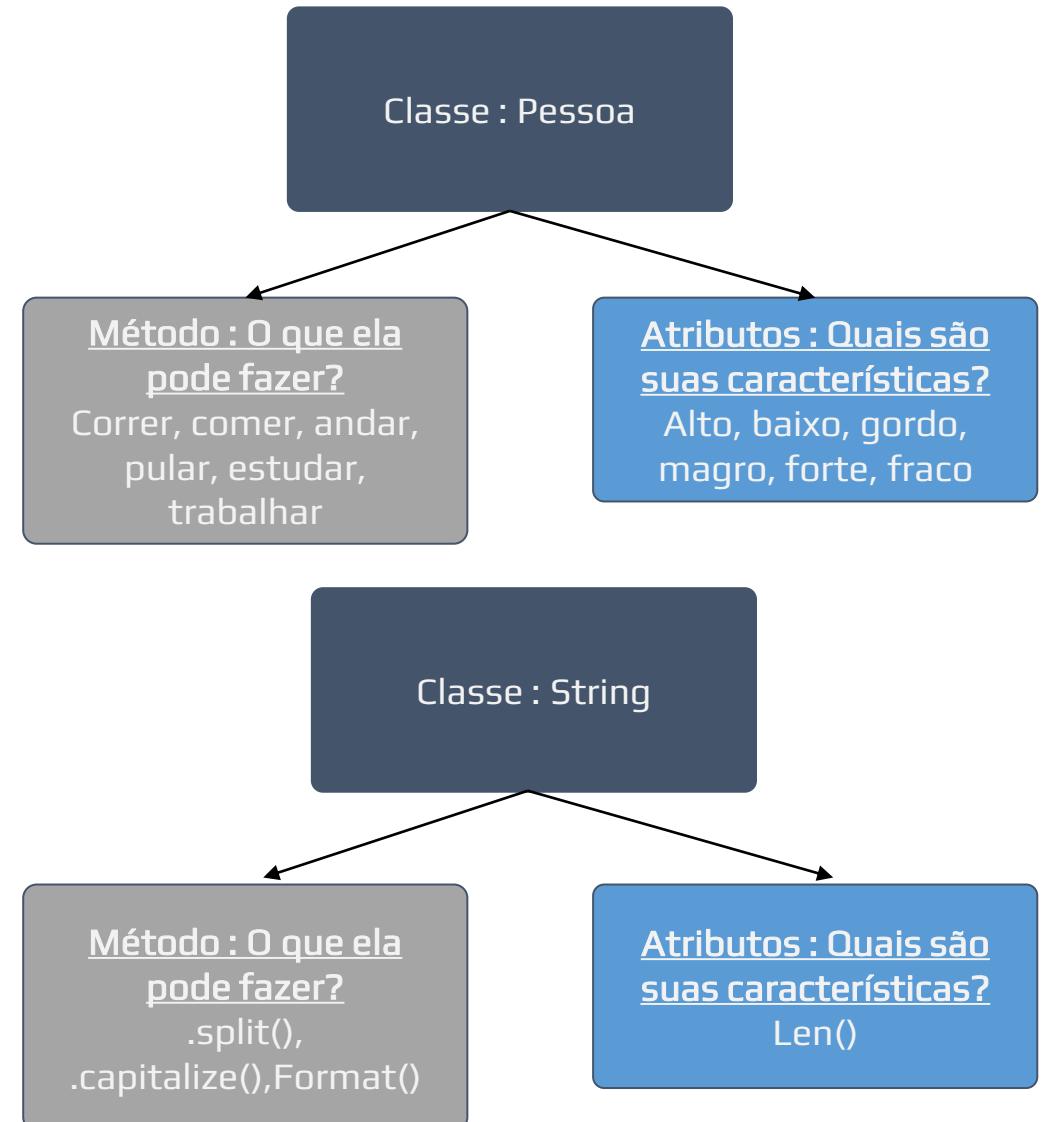
As ações, que essa Classe : ‘pessoa’ pode fazer são os **métodos**.

Já as características são os **atributos**.

Apesar de parecerem novidade, você vai perceber que já vinhamos usando esse conceito diversas vezes.

Vamos para o segundo exemplo ao lado. Ao invés de uma pessoa, vamos usar uma string.

Como podemos ver, o princípio é exatamente o mesmo 😊



Se lembra do `type()` lá no início do curso? Lá perguntarmos para o python qual era o tipo da variável e recebíamos respostas como `Int`, `float`, `string`, `dict`, etc.

Essa resposta significa qual era a classe daquela variável. Consequentemente, sabendo a classe, sabemos o que podemos fazer com aquela variável, ou seja, quais são os métodos que podemos aplicar.

Imagino que esteja pensando: “Ok, até entendi mas e daí? Como uso isso?” Vamos voltar agora para o nosso Pycharm e deixar tudo mais claro.

## Módulo 30– Orientação a Objetos Completo - Classes e Métodos – O que é uma Classe no nosso código(1/3)

445

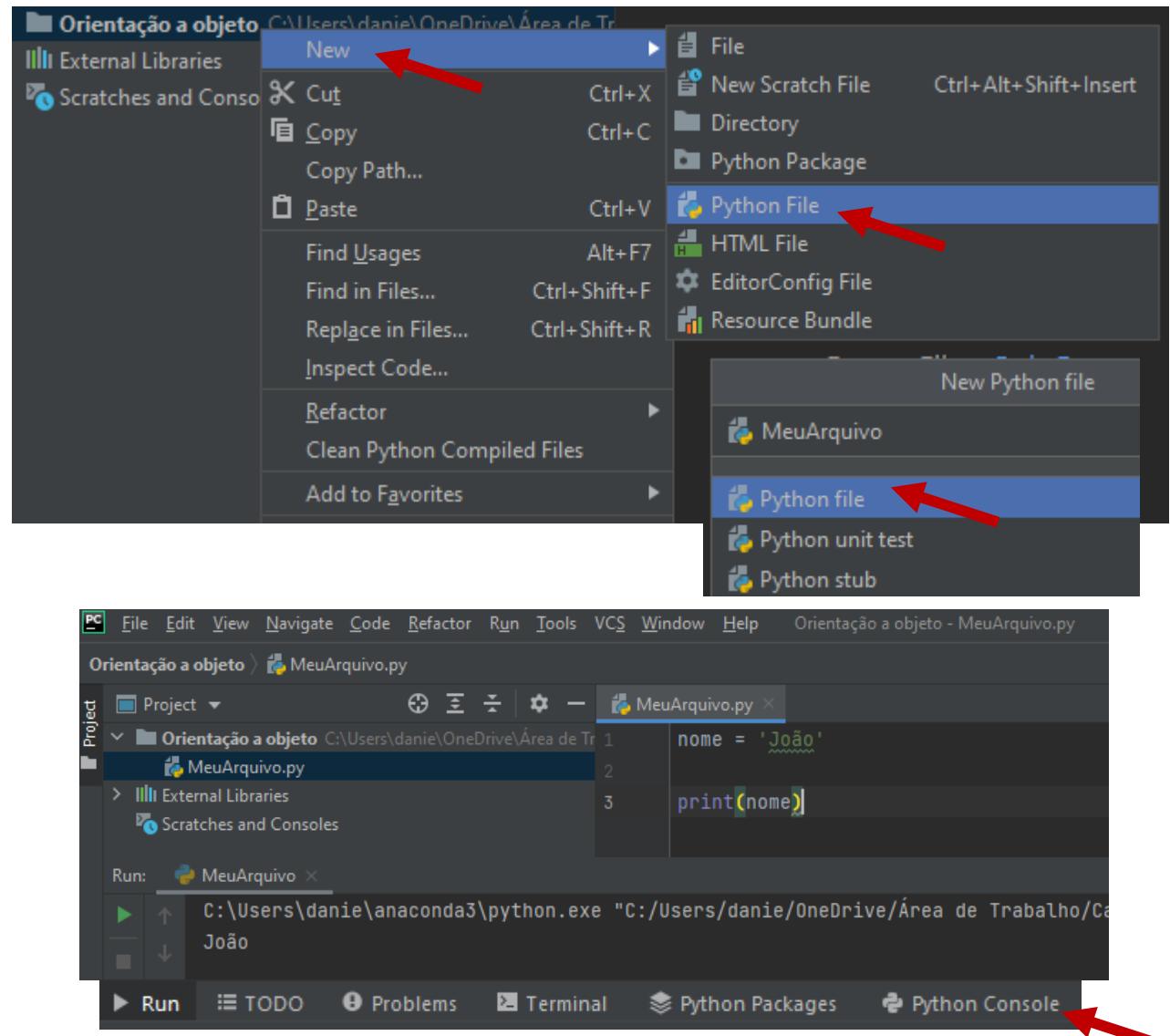
Com o nosso Pycharm aberto, vamos criar um novo arquivo .py. Perceba que aqui, não criaremos mas arquivos .inpy e sim diretamente um script python .py.

Para criar um novo arquivo basta **clicar com o botão direito do mouse na pasta Orientação a objeto**, selecionar a opção **New** e clicar em **Python file**, conforme apresentado ao lado.

Para o nosso exemplo usaremos o nome **MeuArquivo** também apresentado ao lado.

Iniciaremos com algo bem simples como um print bem simples para entendermos como isso pode ser depurado no Pycharm. Ele funcione levemente diferente do Jupyter.

No Pycharm, precisamos habilitar o **PYTHON CONSOLE** que nada mais é que essa janelinha preta na parte inferior.



Vamos usar agora um método nessa nossa variável `nome` que é uma string como já sabemos.

Ao usarmos esse método na variável `nome`, alteramos joão para João.

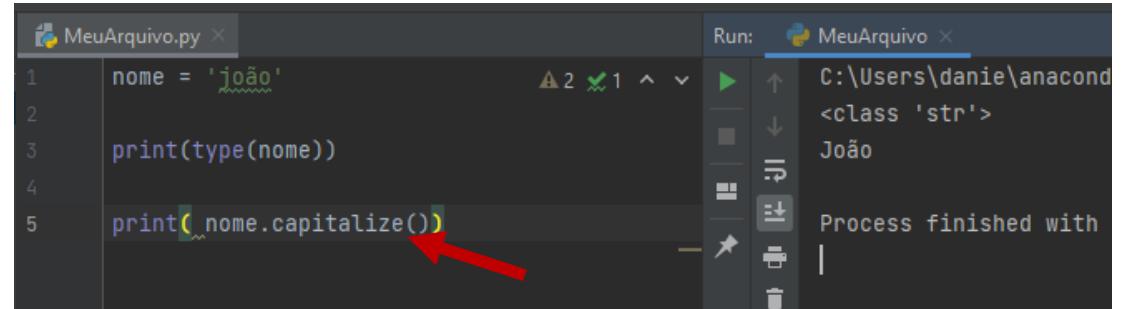
O Pycharm, nos dá uma funcionalidade muito interessante para o entendimento do que significam as classes , métodos e atributos.

Com o **CTRL** segurado, clique no método `capitalize` usado no código.

Perceba que uma nova guia `builtins.pyi` foi aberta com centenas de linhas.

Essas linhas são listas de classes e métodos que existem dentro do Python. Se olharmos na imagem ao lado, podemos ver que `capitalize` está dentro de uma class `str` e assim como fizemos no módulo das funções, está sendo definido pelo `def`. Por que?

Pois `capitalize` é um método da classe `string`. Assim com todos os demais métodos dessa imagem (`casefold`, `center`, `count`, etc...)



```
MeuArquivo.py
1 nome = 'joão'
2
3 print(type(nome))
4
5 print(nome.capitalize())
```

```
MeuArquivo
C:\Users\danie\anaconda>
<class 'str'>
João
Process finished with
```



```
builtins.pyi
class str(Sequence[str]):
    @overload
    def __new__(cls: Type[_T], o: object = ...) -> _T: ...
    @overload
    def __new__(cls: Type[_T], o: bytes, encoding: str = ...):
        def capitalize(self) -> str: ...
        def casefold(self) -> str: ...
        def center(self, __width: int, __fillchar: str = ...) -> str: ...
        def count(self, x: str, __start: Optional[int] = ..., __end: Optional[int] = None) -> int: ...
```

Class Str (string)

Método capitalize

Vamos imaginar que queremos criar uma classe TV.

Usando a estrutura anterior e as premissas abaixo nosso código ficaria parecido com a imagem ao lado.

Quais são seus atributos? (*suas características*)

- Cor = preto;
- Tamanho = 55
- Canal = 10
- Volume = 50

Quais são seus métodos? (*O que posso fazer com essa TV?*)

- Mudar canal
- Mudar volume
- Ligar
- Desligar

```
1
2
3 class TV():
4     cor = 'preta'
5     tamanho = 55
6     canal = 10
7     volume = 50
8
9     def mudar_canal(self):
10        pass
11
12     def mudar_volume(self):
13        pass
14
15     def ligar_desligar(self):
16        pass
17
18 TV.mudar_canal()
19 TV.ligar_desligar()
```

Por que isso tudo agora? E se até agora não usei classes, para que usar agora?

Até agora, fizemos códigos que resolvíamos problemas pontuais, e nosso código basicamente era feito sequencialmente (programação Estruturada).

Agora, vamos organizar essa programação para conseguirmos dividir esses códigos em programas ou aplicações de um programa, ou um sistema maior (programação orientada a objeto).

Para isso, faz muito sentido criarmos as classes que funcionarão para todo esse sistema e não só para um código único.

## Orientada a Objeto (POO)

### Classes

Classe TV:

...

Método Ligar/Desligar TV:

...

Método Mudar Canal:

...

Método Mudar Volume:

...

## Programa

```
minha_tv = TV()
minha_tv.ligar()
minha_tv.mudar_canal("Netflix")
minha_tv.mudar_volume(55)
```

## Estruturada

### Programa

```
variavel = 10
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for item in lista:
    if item == variável:
        print(f'{variável} encontrado')
print('Fim do Programa')
```

Em geral, o que acontecerá nessa forma de programar que a parte mais de inteligência dos nossos códigos, onde usávamos Ifs, for, etc, vão passar a pertencerem a métodos de classes específicas.

A ideia é criar a inteligência fora do nosso código principal e só chama-los assim como fizemos até agora chamando SUM(), SORT(), ETC.

Vamos aprofundar um pouco mais entendendo alguns conceitos importantes:

**Encapsulamento:** Proteção a mudanças indesejadas. Ao criarmos uma classe o encapsulamento nos permite definir as regras que existem dentro dela. Por exemplo, podemos definir que a TV não pode reduzir o volume se estiver desligada;

**Herança:** As instâncias do Objeto possuem as mesmas características, apesar de terem valores diferentes. Por exemplo, quando fazemos nome= 'João', a variável nome que até então não existia ao receber 'João' que é um texto se transforma em um texto. Ou seja, todos os métodos e atributos que existem para textos(strings) por herança também são aplicáveis a variável nome;

**Polimorfismo:** Um mesmo método pode ter várias “formas” em diferentes classes ( ou subclasses). Imaginem a classe ANIMAIS. Humanos são animais, gatos são animais e cachorros também ou seja, são subclasses da classe Animal. Se existisse um método “fala” desses animais teríamos 3 formas distintas mas que desempenham essencialmente a mesma função (humano-> fala “normal”, cachorro-> Latir, gato-> Miar)

Finalmente vamos escrever nosso primeiro código.

E vamos voltar para nossa classe TV. Mas agora com um pouco mais de detalhes.

Primeiro passo é criar a função `__init__(self)`. Essa função irá **definir** os atributos dessa TV. No nosso caso significa dizer que toda e qualquer TV que venha a existir “nasce” de cor preta, desligada, com tamanho 55, no canal “Netflix” e volume 10.

Se imagine como DEUS criando o conceito de TV. É isso que você está fazendo.

Se depois alguém vai trocar de canal, aumentar o volume ou pintar de branco é outra coisa.

O `__init__` essencialmente define como essa classe surge no nosso código.

```
class TV:
    def __init__(self):
        self.cor = 'preta'
        self.ligada = False
        self.tamanho = 55
        self.canal = "Netflix"
        self.volume = 10

    def mudar_canal(self):
        self.canal = "Disney+"

#programa
tv_sala = TV()
tv_quarto = TV()

tv_sala.mudar_canal()

print(tv_sala.canal)
print(tv_quarto.canal)
```

Definição dos atributos

Agora que temos nosso conceito de TV criado pela função `__init__`, vamos criar um método que nos permita mudar os canais.

Perceba que assim como o `__init__`, estamos usando o `def` para defini-la dentro da indentação da class TV.

Nossa simplesmente mudará o canal da TV do estado que ela estiver para o canal Disney+.

Ou seja:

1) se o atributo canal é Netflix e usamos o método `mudar_canal` o atributo será alterado para Disney+.

2) se o atributo canal é Disney+ e usamos o método `mudar_canal` o atributo será alterado para Disney+.

3) Se por algum motivo o atributo canal foi alterado para globo e usamos o método `mudar_canal` o atributo será alterado para Disney+

```
class TV:  
  
    def __init__(self):  
        self.cor = 'preta'  
        self.ligada = False  
        self.tamanho = 55  
        self.canal = "Netflix"  
        self.volume = 10  
  
    def mudar_canal(self):  
        self.canal = "Disney+"  
  
#programa  
tv_sala = TV()  
tv_quarto = TV()  
  
tv_sala.mudar_canal()  
  
print(tv_sala.canal)  
print(tv_quarto.canal)
```

Alterar o valor do atributo canal da TV para Disney +

Agora que temos nossa Class TV e seus métodos e atributos definidos, podemos sair da indentação e começar a criar nosso código.

Vamos dizer que temos 2 TVs distintas uma na sala e outra no quarto.

Como dissemos, se uma TV foi criada seja para onde for ela foi criada segundo as definições feitas dentro da Classe e do método Init. Ou seja, tanto a tv\_sala quanto tv\_quarto:

- são pretas;
- estão desligadas;
- possuem tamanho 55;
- estão no canal Netflix;
- volume 10.



```
class TV:  
  
    def __init__(self):  
        self.cor = 'preta'  
        self.ligada = False  
        self.tamanho = 55  
        self.canal = "Netflix"  
        self.volume = 10  
  
    def mudar_canal(self):  
        self.canal = "Disney+"  
  
#programa  
tv_sala = TV()  
tv_quarto = TV()  
  
tv_sala.mudar_canal()  
  
print(tv_sala.canal)  
print(tv_quarto.canal)
```

Agora que temos nossas TV criadas, vamos usá-las!

Nesse caso, para usá-las usaremos **mudar\_canal** que criamos **dentro da nossa classe**. Mas não basta dizer que quero mudar canal. Preciso saber em qual TV vou mudar. Por isso, precisamos usar a estrutura abaixo:

```
tv_sala.mudar_canal()
```

Perceba que nada aconteceu na tv\_quarto. Por que? Pois só aplicamos o método **mudar\_canal** a uma TV e não a todas.



```
class TV:  
  
    def __init__(self):  
        self.cor = 'preta'  
        self.ligada = False  
        self.tamanho = 55  
        self.canal = "Netflix"  
        self.volume = 10  
  
    def mudar_canal(self):  
        self.canal = "Disney+"  
  
#programa  
tv_sala = TV()  
tv_quarto = TV()  
  
tv_sala.mudar_canal()  
  
print(tv_sala.canal)  
print(tv_quarto.canal)
```

Você já deve ter se perguntado o que significa **self** que aparece no nosso código.

**Self** em inglês seria algo como “*si próprio*”.

No nosso Pycharm, sempre aqui abrirmos o () de um método vai preencher com um **self**. Por que? Todos os métodos que faremos precisarão desse argumento.

Além disso, vemos que o **self** sempre aparece na frente do nosso atributo.

**Sempre** que estivermos dentro de uma classe e queremos referenciar a um atributo daquela classe.

Ao fazermos **self.cor**, essencialmente estamos dizendo:

A cor da instância da classe TV.

Indica que o atributo canal da instância de TV terá valor “Netflix”

```
class TV:  
    def __init__(self):  
        self.cor = 'preta'  
        self.ligada = False  
        self.tamanho = 55  
        self.canal = "Netflix"  
        self.volume = 10  
  
    def mudar_canal(self):  
        self.canal = "Disney+"
```

Vamos entender um pouco melhor como criar métodos dentro da nossa classe.

Já fizemos anteriormente o método `mudar_canal` que nos permitia obviamente mudar o canal da nossa tv.

Vamos entender um pouco melhor do código.

Vimos antes que o `self` quando utilizado junto de um atributo (Ex: `self.canal`) estamos mudando o valor atribuído ao atributo canal de uma instância da classe TV.

Ou seja, criamos um controle remoto que troca o canal para Disney+.

MAS o mais interessante é que esse controle remoto por ter sido criado na classe e não dentro do programa, se tornou em um controle “universal” que posso mudar o canal de TODAS as TVs que sejam da instância TV.

```
class TV:  
  
    def __init__(self):  
        self.cor = 'preta'  
        self.ligada = False  
        self.tamanho = 55  
        self.canal = "Netflix"  
        self.volume = 10  
  
    def mudar_canal(self):  
        self.canal = "Disney+"  
  
#programa  
tv_sala = TV()  
tv_quarto = TV()  
  
tv_sala.mudar_canal()  
  
print(tv_sala.canal)  
print(tv_quarto.canal)
```

Muda o valor atribuído a canal para “Disney+”

Bem, muito interessante ter um controle universal que serve para TODAS as instâncias da classe TV, mas só tem 1 problema ai...

Esse controle por enquanto, só tem 1 botão.



```
def mudar_canal(self):  
    self.canal = "Disney+"
```

Da forma como criamos esse código, se quisermos voltar para a Netflix, não conseguimos. Vamos precisar criar algum elemento que nos permita transitar melhor pelos canais.

Esse elemento será um novo parâmetro dentro do nossos método.

Usando um novo parâmetro na nossa função. Assim como vimos anteriormente no módulo de funções, temos a possibilidade de “aumentar o número de botões” do no controle remoto.

Esse novo parâmetro deverá ser passado sempre que usarmos o método.

Usando a analogia do controle remoto, seria dizer que não basta simplesmente pegar o controle mas apertar o botão da emissora que queremos ver.

Veja o exemplo ao lado usando nosso controle universal digo ao Python “Na minha tv da sala quero **mudar\_canal** para **Globo**” , já na “ minha tv do quarto, quero mudar canal para o **Youtube**”.

1) Primeiro o Python recebe a instância:

Qual TV você quer? **tv\_sala**

2) Segundo o Python recebe o método:

O que quer fazer com a **tv\_sala**? **mudar\_canal**

3) Terceiro o Python recebe o argumento do método escolhido:

Qual o **novo\_canal**? **Globo**

```
#classes
class TV:

    def __init__(self):
        self.cor = 'preta'
        self.ligada = False
        self.tamanho = 55
        self.canal = "Netflix"
        self.volume = 10

    def mudar_canal(self,novo_canal):
        self.canal = novo_canal

#programa
tv_sala = TV()
tv_quarto = TV()

tv_sala.mudar_canal('Globo')
tv_quarto.mudar_canal('Youtube')

print(tv_sala.canal)
print(tv_quarto.canal)
```

Nossa função `__init__` também pode receber outros parâmetros além do parâmetro `self`.

A premissa que usamos era que ao criarmos nossa função `__init__` estávamos criando um **conceito de tv** que era único como se fosse não existissem variedades de cores, tamanhos, etc...

Mas na vida real não é assim... Se vamos à uma loja e pedimos ao vendedor “Quero uma TV”. Certamente ele não aparecerá com uma Tv preta de 55 polegadas pré setada para o canal Netflix e com volume 10.

Certamente, nos perguntaria. “Qual cor? “Preferência de tamanho?” e assim vai... Essas respostas serão os nossos parâmetros do método `__init__` conforme apresentado ao lado.

Perceba que usamos esses parâmetros como variáveis dentro do bloco do método `__init__`.

```
#classes
class TV:

    def __init__(self, cor, tamanho):
        self.cor = cor
        self.ligada = False
        self.tamanho = tamanho
        self.canal = "Netflix"
        self.volume = 10
```

Um ponto de atenção é que apesar de termos nomes repetidos (cor, tamanho), eles são elementos com funções totalmente diferente no nosso código.

### Os casos self.cor e self.tamanho, são atributos!

Por isso, precisam ser lidos como se fossem uma coisa única.

Aqui, estamos falando da cor da instância da classe TV.

### Os casos cor e tamanho, são variáveis!

Assim como vimos no módulo de funções, cor e tamanho são variáveis que existem somente dentro do método e que serão parâmetros necessários para que o método possa ser executado.

```
#classes
class TV:

    def __init__(self, cor, tamanho):
        self.cor = cor
        self.ligada = False
        self.tamanho = tamanho
        self.canal = "Netflix"
        self.volume = 10
```

O que muda no nosso programa?

Antes quando criávamos nossas instâncias tv\_sala ou tv\_quarto, simplesmente usávamos =TV().

Pois não existiam variáveis só existia 1 tipo de tv.

Agora, uma TV só poderá ser criada se os dados de cor e tamanho forem fornecidas.

Essa declaração, conforme apresentado ao lado, pode ser feita por keyword(variável=valor) ou por posição (apenas valor, mas na ordem correta).

```
#classes
class TV:

    def __init__(self, cor, tamanho):
        self.cor = cor
        self.ligada = False
        self.tamanho = tamanho
        self.canal = "Netflix"
        self.volume = 10

    def mudar_canal(self,novo_canal):
        self.canal = novo_canal

#programa
tv_sala = TV(cor='preta',tamanho=55)
tv_quarto = TV('branca',29)

print(tv_sala.cor)
print(tv_quarto.tamanho)
```

Declaração por Keywords

Declaração por posição

Agora vamos entender um outro conceito que é o atributo de uma Classe.

Até agora, vimos que quando usamos:

```
self.atributo = valor
```

Estamos fornecendo um valor a algum atributo de uma instância da classe TV.

No entanto, podemos também estabelecer atributos que são da classe. Ou seja, que são comuns a todas as suas instâncias.

Para isso, iremos criar um atributo dentro da indentação da classe, mas fora dos nossos métodos, conforme apresentado ao lado.

No nosso exemplo, perceba que a cor deixa de ser uma variável da instância. Ser uma tv “preta” passa a ser uma premissa da instância.

Voltando ao caso em que vamos numa loja de TVs, seria como ir em uma que só vende TV preta. Portanto o vendedor perguntaria “Qual o tamanho da TV preta de sua preferência?”

```
class TV:  
  
    cor = "preto"  
  
    def __init__(self, tamanho):  
        self.ligada = False  
        self.tamanho = tamanho  
        self.canal = "Netflix"  
        self.volume = 10  
  
    def mudar_canal(self, novo_canal):  
        self.canal = novo_canal  
  
#programa  
tv_sala = TV(cor='preta', tamanho=55)  
tv_quarto = TV('branca', 29)  
  
print(tv_sala.cor)  
print(tv_quarto.tamanho)
```

Chega de TV! Vamos para uma aplicação mais prática!

Vamos criar um “sistema de banco”. E para começar vamos criar nossa classe de CONTAS bancárias.

A partir de um novo arquivo .py, vamos criar a nossa primeira classe:

```
class ContaCorrente()
```

Dentro da nossa classe, vamos começar criando nosso método `__init__` com os seguintes parâmetros:

- `self`: padrão;
- `nome` : Nome do dono da conta;
- `cpf` : Nº do CPF do dono da conta;

Isso significa dizer que esses são os parâmetros necessários sempre que quisermos criar uma nova instância da classe Conta Corrente, ou seja, sempre quisermos criar uma nova conta corrente

```
class ContaCorrente():  
    def __init__(self, nome, cpf):  
        self.nome = nome  
        self.cpf = cpf  
        self.saldo = 0
```

Criada nossa classe e método `__init__`, podemos criar nossa primeira instância que no caso é uma nova conta corrente.

Perceba na imagem ao lado, que ao criarmos essa nova instância, precisamos fornecer os 2 argumentos definidos no nossos método `__init__`.

Nesse caso, utilizamos a declaração pelo método de posição onde:

- nome -> Lira
- cpf -> 111.222.333-45

Outro ponto importante é que `saldo` não é um parâmetro da conta e sim um atributo definido internamente do método `__init__`.

Assim, ao “printarmos” o valor do saldo da conta temos o valor definido dentro do método `__init__`, logo ZERO.

```
class ContaCorrente():  
    def __init__(self, nome, cpf):  
        self.nome = nome  
        self.cpf = cpf  
        self.saldo = 0  
  
#programa  
  
conta_lira = ContaCorrente("Lira", "111.222.333-45")  
print('Saldo da conta é', conta_lira.saldo)  
print(conta_lira.cpf)
```

```
Saldo da conta é 0  
111.222.333-45
```

Como falamos anteriormente, nosso objetivo é criar um sistema de banco. Apenas a conta corrente, não seria suficiente... Só ter a conta não tem muita funcionalidade se não existissem funções como:

- Depositar dinheiro;
- Sacar dinheiro;
- Consultar saldo;

Lembre-se agora do conceito de método e atributo. As funções acima não se encaixam bem no conceito de método?

Não é a toa, são exatamente os métodos que criaremos para a nossa classe ContaCorrente.

Vamos iniciar pelo Método **consultar\_saldo** ao lado.

```
class ContaCorrente:  
  
    def __init__(self, nome, cpf):  
        self.nome = nome  
        self.cpf = cpf  
        self.saldo = 0  
  
    def consultar_saldo(self):
```

Este método, nos permite consultar o saldo.

Para esse método, não precisamos de nenhuma informação (parâmetro) adicional. Portanto, temos como único argumento do método, **self**.

Como se trata apenas de uma consulta de saldo, essencialmente é uma função que “printa” o valor do saldo. Logo, só usaremos o método `.format` para melhorar o formato que será apresentado esse número.

Além disso, perceba que o valor utilizado dentro do `format` é `self.saldo` pois a informação que queremos será o atributo de uma conta corrente específica ou em termos de programação, de uma instância da classe `ContaCorrente` específica.

```
class ContaCorrente:

    def __init__(self, nome, cpf):
        self.nome = nome
        self.cpf = cpf
        self.saldo = 0

    def consultar_saldo(self):
        print('Seu saldo atual é de R${:,.2f}'.format(self.saldo))
        pass
```

Próximo métodos, `depositar_dinheiro` e `sacar_dinheiro`.

Iremos tratar essas duas de forma conjunta, pois essencialmente são iguais, a única diferença é que uma adiciona valor do saldo e a outra diminuir valor do saldo.

Diferente de consultar o saldo, saber que queremos depositar ou sacar dinheiro de uma conta corrente não é suficiente por si só.

Precisamos de uma variável importante QUANTO ou seja o valor a ser depositado ou a ser retirado da conta corrente.

Essa variável será o segundo parâmetro dos nossos métodos. Logo após o `self`.

```
class ContaCorrente:  
  
    def __init__(self, nome, cpf):  
        self.nome = nome  
        self.cpf = cpf  
        self.saldo = 0  
  
    def consultar_saldo(self):  
        print('Seu saldo atual é de R${:,.2f}'.format(self.saldo))  
        pass  
  
    def depositar_dinheiro(self, valor):  
        self.saldo += valor  
        pass  
  
    def sacar_dinheiro(self, valor):  
        self.saldo -= valor  
        pass
```

Variável valor como parâmetro dos métodos

## Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Criando Métodos para a nossa Classe (4/4)

467

Agora que temos nossa classe e seus métodos, vamos rodar o nosso programa testando o nosso sistema.

```
class ContaCorrente:  
  
    def __init__(self, nome, cpf):  
        self.nome = nome  
        self.cpf = cpf  
        self.saldo = 0  
  
    def consultar_saldo(self):  
        print('Seu saldo atual é de R${:,.2f}'.format(self.saldo))  
        pass  
  
    def depositar_dinheiro(self, valor):  
        self.saldo += valor  
        pass  
  
    def sacar_dinheiro(self, valor):  
        self.saldo -= valor  
        pass
```

```
#programa  
  
conta_lira = ContaCorrente("Lira", "111.222.333-45")  
conta_lira.consultar_saldo()  
  
#Depositar um dinheirinho na conta:  
conta_lira.depositar_dinheiro(10000)  
conta_lira.consultar_saldo()  
  
#Sacando um dinheirinho da conta:  
conta_lira.sacar_dinheiro(1000)  
conta_lira.consultar_saldo()  
  
Seu saldo atual é de R$0.00  
Seu saldo atual é de R$10,000.00  
Seu saldo atual é de R$9,000.00
```

Conseguimos criar um sistema que nos permite fazer operações na conta corrente. Mas, ainda temos um problema de concepção nesse sistema.

Se olharmos o código com atenção, vamos perceber que caso tente sacar um valor de R\$1.000.000 mas com um saldo de R\$10.000, irei conseguir.

Precisamos criar algumas “medidas de controle” no nosso sistema para garantir a segurança das operações.

O modo de fazer isso, não é muito diferente do que fizemos nos módulos mais antigos. Usaremos um IF para validar se temos saldo para fazer o saque.

A imagem ao lado apresenta esse incremento no código. Perceba que aqui, utilizamos o método `consultar_saldo()` dentro do método `sacar_dinheiro`.

```
class ContaCorrente:
    def __init__(self, nome, cpf):
        self.nome = nome
        self.cpf = cpf
        self.saldo = 0

    def consultar_saldo(self):
        print('Seu saldo atual é de R${:.2f}'.format(self.saldo))
        pass

    def depositar_dinheiro(self, valor):
        self.saldo += valor
        pass

    def sacar_dinheiro(self, valor):
        if self.saldo - valor < 0:
            print('Você não tem saldo suficiente para sacar esse valor')
            self.consultar_saldo()
        else:
            self.saldo -= valor
        pass
```

Teste de saldo

Uso de um método dentro de outro método.

Executando esse código, podemos perceber que agora não é mais possível realizar o saque caso não haja saldo suficiente.

```
#programa

conta_lira = ContaCorrente("Lira", "111.222.333-45")
conta_lira.consultar_saldo()

#depositar um dinheirinho na conta:
conta_lira.depositar_dinheiro(10000)
conta_lira.consultar_saldo()

#Sacando um dinheirinho da conta:
conta_lira.sacar_dinheiro(1000000)
conta_lira.consultar_saldo()
```

```
Seu saldo atual é de R$0.00
Seu saldo atual é de R$10,000.00
Você não tem saldo suficiente para sacar esse valor
Seu saldo atual é de R$10,000.00
Seu saldo atual é de R$10,000.00

Process finished with exit code 0
```

Outra forma de criar esse “controle” no nosso sistema é criando um método dedicado para o cálculo do limite, o que nos auxiliará na consulta.

No exemplo ao lado fazemos a demonstração desse caso.

Primeiro precisamos criar um atributo **limite** para a conta. A princípio, não existe um valor fixo para esse limite, em geral, esses valores são calculados pelos bancos baseados no histórico de uso do cliente.

Para facilitar, fixamos que esse limite será de -1.000. Ou seja, o cliente poderá sacar até R\$1.000,00 a mais do que o saldo em conta.

Esse valor substitui a segunda parte da expressão criada anteriormente na estrutura IF do método `sacar_dinheiro`. Ou seja, se a operação de sacar dinheiro for menor que o limite definido no método `limite_conta`, a operação será bloqueada.

```
class ContaCorrente:
    def __init__(self, nome, cpf):
        self.nome = nome
        self.cpf = cpf
        self.saldo = 0
        self.limite = None

    def consultar_saldo(self):
        print('Seu saldo atual é de R${:,.2f}'.format(self.saldo))
        pass

    def depositar_dinheiro(self, valor):
        self.saldo += valor
        pass

    def limite_conta(self):
        self.limite = -1000
        return self.limite

    def sacar_dinheiro(self, valor):
        if self.saldo - valor < self.limite_conta():
            print('Você não tem saldo suficiente para sacar esse valor')
            self.consultar_saldo()
        else:
            self.saldo -= valor
        pass
```

Criação de um novo atributo, **limite**, que será utilizado na consulta

Método que define o **limite** e retorna este valor para o método **sacar\_dinheiro**

Aqui vai uma dica/convenção que nos ajudará a criar códigos mais organizados.

Vamos comparar nossos métodos **consultar\_saldo** e **limite\_conta**. Perceba que os 2 estão endo usados dentro de outro método:

- **limite\_conta**: define o limite daquela conta;
- **consultar\_saldo**: printa o saldo da conta corrente.

Se analisarmos com atenção existem uma diferença de propósito nos dois métodos.

**limite\_conta**, essencialmente, não é um método de interação com o usuário. Simplesmente nos fornece um mecanismo de definição do limite. Já **consultar\_saldo**, possui em sua concepção fornecer informações ao usuário. Inclusive havendo uma preocupação com a formatação do seu resultado.

```
class ContaCorrente:  
  
    def __init__(self, nome, cpf):  
        self.nome = nome  
        self.cpf = cpf  
        self._saldo = 0  
        self._limite = None  
  
    def consultar_saldo(self):  
        print('Seu saldo atual é de R${:,.2f}'.format(self.saldo))  
        pass  
  
    def depositar_dinheiro(self, valor):  
        self.saldo += valor  
        pass  
  
    def limite_conta(self):  
        self._limite = -1000  
        return self._limite  
  
    def sacar_dinheiro(self, valor):  
        if self.saldo - valor < self.limite_conta():  
            print('Você não tem saldo suficiente para sacar esse valor')  
            self.consultar_saldo()  
        else:  
            self.saldo -= valor  
        pass
```

Método criado para interagir com o usuário

Método criado apenas para definição do limite da conta. Não foi criado para interação com o Usuário.

Na prática, essa diferença não irá representar nenhuma mudança no resultado final do sistema, mas em termos desenvolvimento de códigos, é comum dividir os métodos por:

- **Públicos:** São usados em algum nível no resultado para o usuário;
- **Não-Públicos:** Métodos que são executados apenas internamente dentro do código.

A notação utilizada para identificarmos essas variáveis em um código é o `_` no início do nome da variável. Portanto, no caso da nossa variável `limite_conta`, iremos chama-la de `_limite_conta`.

A figura ao lado apresenta a alteração.

```
class ContaCorrente:  
  
    def __init__(self, nome, cpf):  
        self.nome = nome  
        self.cpf = cpf  
        self.saldo = 0  
        self._limite = None  
  
    def consultar_saldo(self):  
        print('Seu saldo atual é de R${:,.2f}'.format(self.saldo))  
        pass  
  
    def depositar_dinheiro(self, valor):  
        self.saldo += valor  
        pass  
  
    def _limite_conta(self):  
        self._limite = -1000  
        return self._limite  
  
    def sacar_dinheiro(self, valor):  
        if self.saldo - valor < self._limite_conta():  
            print('Você não tem saldo suficiente para sacar esse valor')  
            self.consultar_saldo()  
        else:  
            self.saldo -= valor  
        pass
```

Métodos públicos se mantém inalterados

'\_' indica que se trata de um método não-público

Antes de seguirmos vamos fazer uma revisão de algumas convenções bem comuns no Python:

- 1) Defina suas classes iniciando com Letras Maiúsculas e caso precise de mais de uma palavra, não use espaços e sim Letras maiúsculas. Ex: ContaCorrente;
- 2) Sempre tenha uma linha de espaço entre os métodos e 2 linhas após a declaração dos métodos;
- 3) Defina métodos pequenos. Caso seu método necessite de mais funcionalidades, crie um método auxiliar;
- 4) Todos os atributos das suas instâncias precisam estar no método `__init__`;

```
class ContaCorrente:  
    Convenção 1  
  
    def __init__(self, nome, cpf):  
        self.nome = nome  
        self.cpf = cpf  
        self.saldo = 0  
        self.limite = None  
        Convenção 4  
        Convenção 2  
  
    def consultar_saldo(self):  
        print('Seu saldo atual é de R${:.2f}'.format(self.saldo))  
        pass  
  
    def depositar_dinheiro(self, valor):  
        self.saldo += valor  
        pass  
  
    def _limite_conta(self):  
        self.limite = -1000  
        return self.limite  
        Convenção 3: Método auxiliar  
        Convenção 4: Métodos curtos.  
  
    def sacar_dinheiro(self, valor):  
        if self.saldo - valor < self._limite_conta():  
            print('Você não tem saldo suficiente para sacar esse valor')  
            self.consultar_saldo()  
        else:  
            self.saldo -= valor  
        pass
```

Nossa conta por enquanto, ainda não possui, informações básicas como agência e conta.

Como são informações ligadas a instância da `ContaCorrente`, vamos criar esses atributos no nosso método `__init__`, conforme apresentado na imagem ao lado.

Simples, certo? No entanto, se tentarmos executar o nosso programa, o Python apresentará um erro. Conforme indicado ao lado.

### Por que isso acontece?

Se olharmos o erro com atenção, veremos que o Python indica a ausência de 2 argumentos de posição. Nosso programa não foi criado considerando a informação de agência e conta. Obviamente nesse caso é simples, bastaria adicionar essas informações. Mas imagine se fosse um caso real onde diversas contas já existissem...

```
class ContaCorrente:  
    def __init__(self, nome, cpf, agencia, num_conta):  
        self.nome = nome  
        self.cpf = cpf  
        self.saldo = 0  
        self.limite = None  
        self.agencia = agencia  
        self.num_conta = num_conta
```

```
#programa  
  
conta_lira = ContaCorrente("Lira", "111.222.333-45")  
conta_lira.consultar_saldo()  
  
#depositar um dinheirinho na conta:  
conta_lira.depositar_dinheiro(10000)  
conta_lira.consultar_saldo()  
  
#Sacando um dinheirinho da conta:  
conta_lira.sacar_dinheiro(1000000)  
  
print('Saldo Final:')  
conta_lira.consultar_saldo()
```

```
C:\Users\danie\anaconda3\python.exe "C:/Users/danie/OneDrive/A  
Traceback (most recent call last):  
  File "C:/Users/danie/OneDrive/Área de Trabalho/Cartilhas Pyt  
  conta_lira = ContaCorrente("Lira", "111.222.333-45")  
TypeError: __init__() missing 2 required positional arguments:  
  
Process finished with exit code 1
```

Outra funcionalidade muito importante na gestão de um sistema bancário é o histórico de transações que ocorreram naquela conta.

Para isso, iremos criar outro método que nos permite armazenar a informação, mas **sem alterar nosso programa**.

As informações importantes nesse controle de transações são:

- Data e hora;
- Qual o tipo da transação;
- Valor da transação;
- Saldo após transação.

Para isso , vamos criar uma lista para receber todos esses valores.

```
def __init__(self, nome, cpf, agencia, num_conta):  
    self.nome = nome  
    self.cpf = cpf  
    self.saldo = 0  
    self.limite = None  
    self.agencia = agencia  
    self.num_conta = num_conta  
    self.transacoes = []
```

A lista nos permitirá armazenar as informações necessárias no histórico de transações. Por padrão, toda nova conta(nova instância) terá um histórico vazio, por isso usamos []

Se olharmos as informações necessárias vamos perceber que nos falta a informação **Data e Hora**.

Portanto, vamos precisar **criar um método** que nos permita gerar essa informação.

No entanto, essa informação não será uma instância ou um método de uma instância específica.

Como o tempo é comum a qualquer conta corrente do nosso banco, vamos criar um **método estático**.

Qual a diferença na prática. Ao invés de criarmos esse método dentro da instância ou no programa, vamos cria-lo “globalmente” dentro da **nossa classe ContaCorrente** conforme a imagem ao lado.

Além disso, usaremos 2 bibliotecas para auxiliar este método.



```
from datetime import datetime
import pytz

class ContaCorrente:
    @staticmethod
    def _data_hora():
        fuso_BR = pytz.timezone('Brazil/East')
        horario_BR = datetime.now(fuso_BR)
        return horario_BR
```

Bibliotecas que nos auxiliam a trabalhar com data e horário

Comentário que nos ajuda na organização do código

Essa função retornará os dados de data e hora da transação

## Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Métodos Estáticos e Modificando Classe da Melhor Forma (3/5)

477

Agora, vamos identificar quais são os métodos que correspondem a transações da nossa conta corrente:

- Depositar\_dinheiro;
- Sacar\_dinheiro

Ou seja, nesses métodos, precisamos adicionar ao código um controle das transações.

Perceba na imagem ao lado que utilizamos mais de um atributo nesta linha de código. Inclusive o nosso método estático `_data_hora` que nos dará os dados temporais da transação.

Agora que temos nosso método `_data_hora`, podemos definir as informações das transações

```
def depositar_dinheiro(self, valor):
    self.saldo += valor
    self.transacoes.append((valor, self.saldo, ContaCorrente._data_hora()))
    pass

def sacar_dinheiro(self, valor):
    if self.saldo - valor < self._limite_conta():
        print('Você não tem saldo suficiente para sacar esse valor')
        self.consultar_saldo()
    else:
        self.saldo -= valor
        self.transacoes.append((valor, self.saldo, ContaCorrente._data_hora()))
    pass
```

Agora que temos nosso método `_data_hora`, podemos definir as informações das transações

Vamos melhorar nosso código criando um método que pertencerá a nossa instância da Conta corrente.

Esse método terá como objetivo indicar todas as transações que ocorreram naquela conta específica.

Perceba que não fizemos mudanças no programa, além de adicionar o código abaixo que nos permite printar o histórico:

```
conta_lira.consultar_historico_transacoes()
```

```
C:\Users\danie\anaconda3\python.exe "C:/Users/danie/OneDrive/Área de Trabalho/Cartilhas Python/Arquivos apostila/Módulo 30/or
Seu saldo atual é de R$0.00
Seu saldo atual é de R$10,000.00
Você não tem saldo suficiente para sacar esse valor
Seu saldo atual é de R$10,000.00
Saldo Final:
Seu saldo atual é de R$10,000.00
-----
Histórico de Transações:
(10000, 10000, datetime.datetime(2021, 4, 27, 17, 5, 39, 956550, tzinfo=<DstTzInfo 'Brazil/East' -03-1 day, 21:00:00 STD>))
```

Usando o for iremos printar todas as transações que ocorreram nesta conta

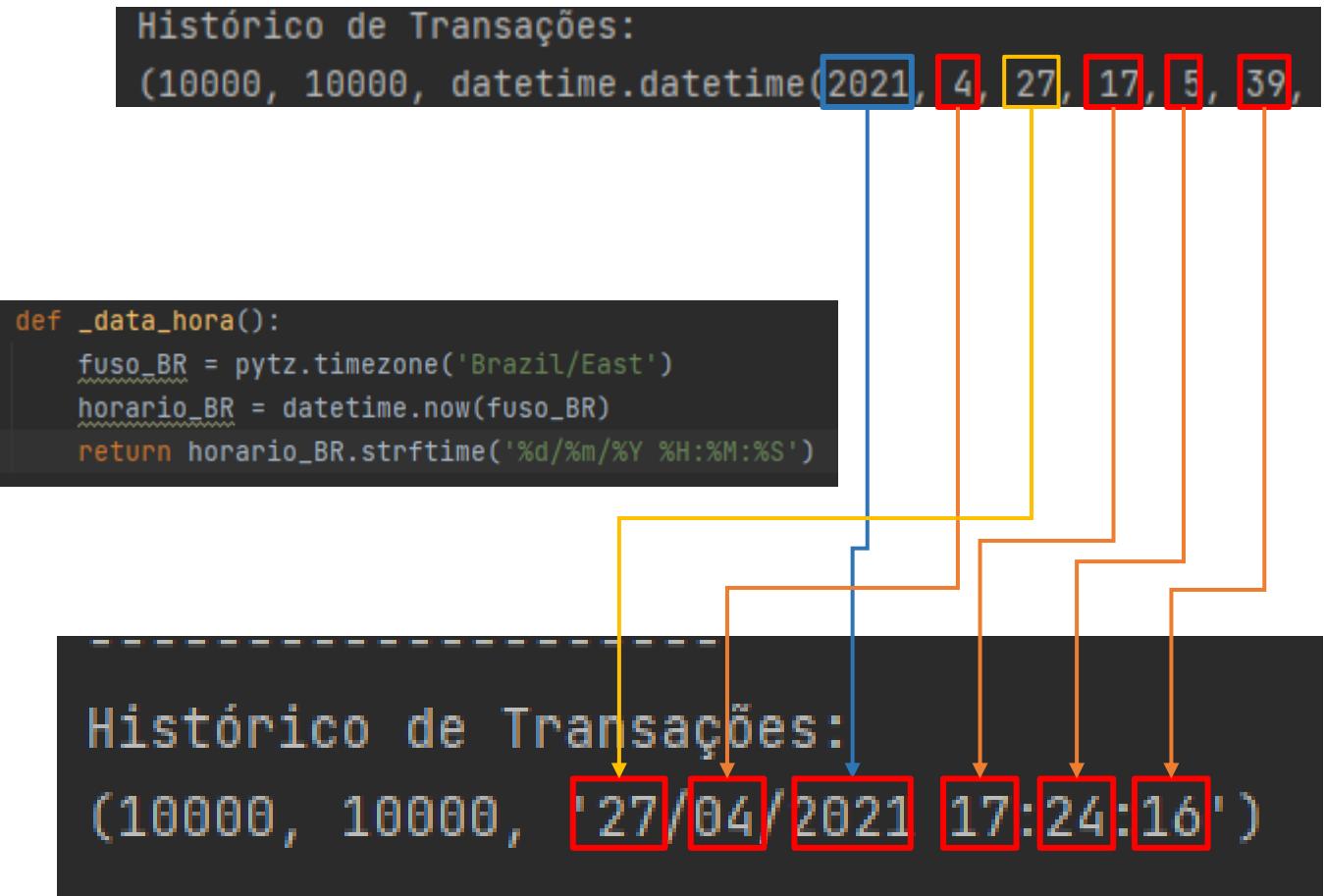
Nosso histórico apesar de correto ainda poderia apresentar uma formatação mais amigável ao usuário.

Para isso, vamos retornar ao nosso método estático `_data_hora()`, e usarmos o método `.strftime()` que nos permite formatar a data para o método geralmente utilizado:

dia/mês/ano hora:minutos:segundos

Agora temos nosso histórico de transações com uma data bem mais compreensível.

PS: A diferença dos minutos e segundos são apenas devido ao tempo entre os prints, mas se referem a mesma informação.



Já temos um sistema bancário simples para controle de uma conta corrente por exemplo.

No entanto, ainda nos falta um método que nos permita fazer transferência entre contas.

Esse método, diferentemente dos outros possui uma diferença.

Até agora os argumentos de um método sempre eram atributos da própria instância e não para outras instâncias(No nosso exemplo outras contas).

Veja no exemplo abaixo

```
def transferir(self, valor, conta_destino):
    self.saldo -= valor
    self.transacoes.append((-valor, self.saldo, ContaCorrente._data_hora()))
    conta_destino.saldo += valor
    conta_destino.transacoes.append((valor, conta_destino.saldo, ContaCorrente._data_hora()))
```

Conta destino não se trata de um atributo mas sim um **OBJETO**. Ou seja, uma instância da classe ContaCorrente

Perceba que aqui, não utilizamos `.self`, por se tratar de um objeto(instância) distinta da do método que estamos.

Calcula o Saldo da conta destino dentro do método de transferência

Vamos agora criar nossa nova conta corrente que receberá a transferência (Nova instância)

Para criarmos uma nova conta corrente (nova instância), vamos precisar revisitar o nosso método `__init__` da classe `ContaCorrente`.

Perceba que agora temos um método que consegue se relacionar com diferentes contas.

```
def __init__(self, nome, cpf, agencia, num_conta):  
    self.nome = nome  
    self.cpf = cpf  
    self.agencia = agencia  
    self.num_conta = num_conta  
  
conta_maeLira= ContaCorrente('Beth', 222.333.444-55, 5555, 656565)
```

```
def transferir(self, valor, conta_destino):  
    self.saldo -= valor  
    self.transacoes.append((-valor, self.saldo, ContaCorrente._data_hora()))  
    conta_destino.saldo += valor  
    conta_destino.transacoes.append((valor, conta_destino.saldo, ContaCorrente._data_hora()))
```

```
conta_lira.transferir(1000, conta_maeLira)
```

Instância `conta_lira` se utiliza do método `transferir` para realizar uma transferência de 1000 reais para outra instância `conta_maeLira`

Outro conceito importante é declarar corretamente se os seus atributos são Públicos ou não.

O que significa isso?

Ter um atributo privado significa que ele não deve ser alterado fora da classe a qual pertence. No nosso exemplo, temos que atributos como cpf, saldo, só deveriam ser alterados através do nosso sistema e nunca fora dele.

Para indicarmos a “privacidade” de um atributo, usaremos o sinal de \_ a frente do atributo. Isso indicará que se trata de um atributo privado.

Veja o exemplo, ao lado, perceba que apenas inserimos o \_ no atributo os demais “nomes”, são variáveis e não atributos.

```
def __init__(self, nome, cpf, agencia, num_conta):  
    self._nome = nome
```

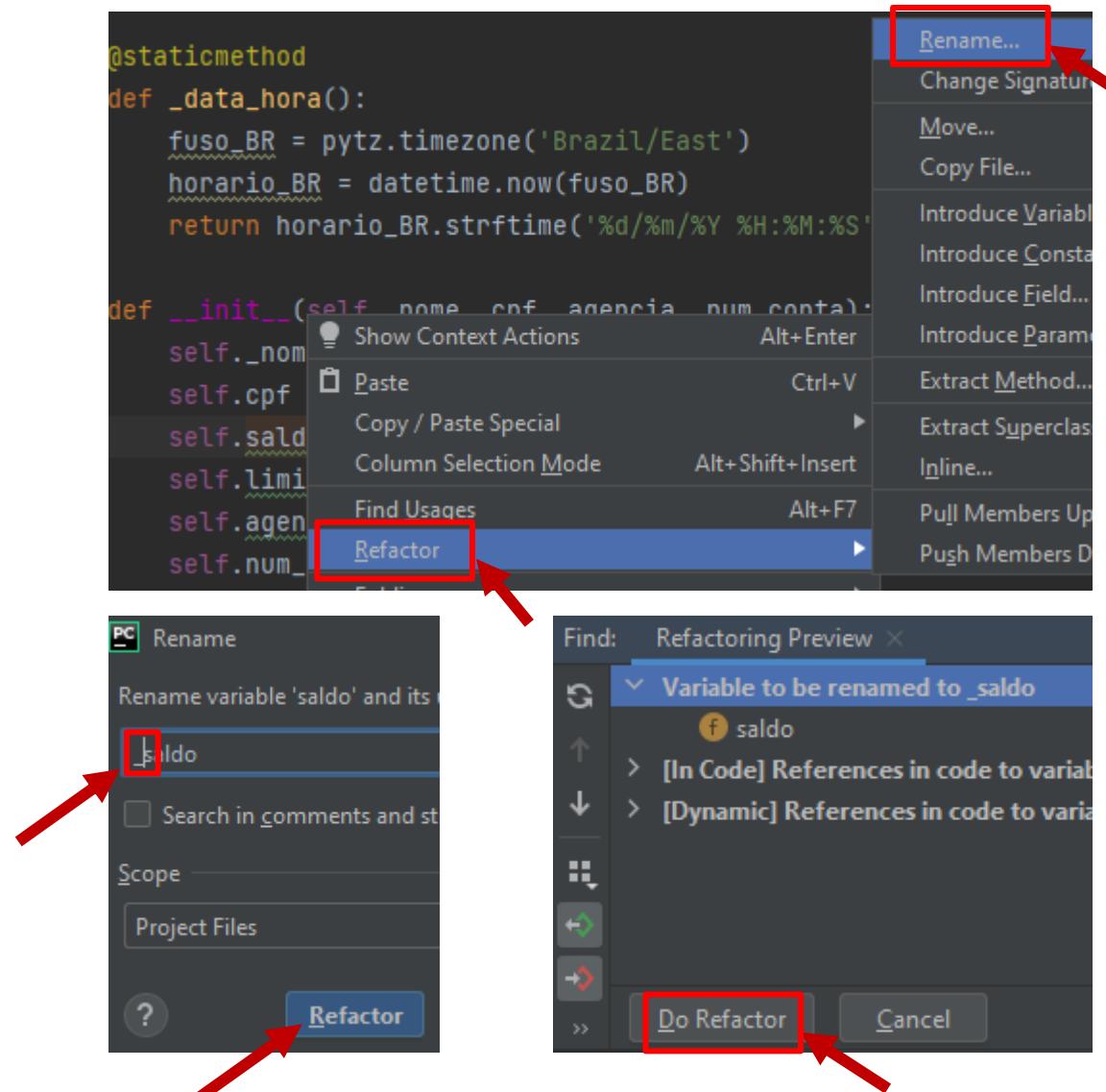


Instância **conta\_lira** se utiliza do método transferir para realizar uma transferência de 1000 reais para outra instância **conta\_maeLira**

No entanto, já escrevemos todo nosso código sem indicar essa diferença, como podemos fazer de forma mais automática alterar todas as ocorrências desse atributo?

- 1) Clique com o **botão direito** do mouse sobre o nome do atributo e selecione as opções **Refactor** e **Rename** conforme a imagem ao lado;
- 2 ) Altere o nome para `_saldo` e clique em **Refactor**;
- 3) Do **Refactor**.

Automaticamente o Pycharm irá alterar todas as ocorrências desse atributo



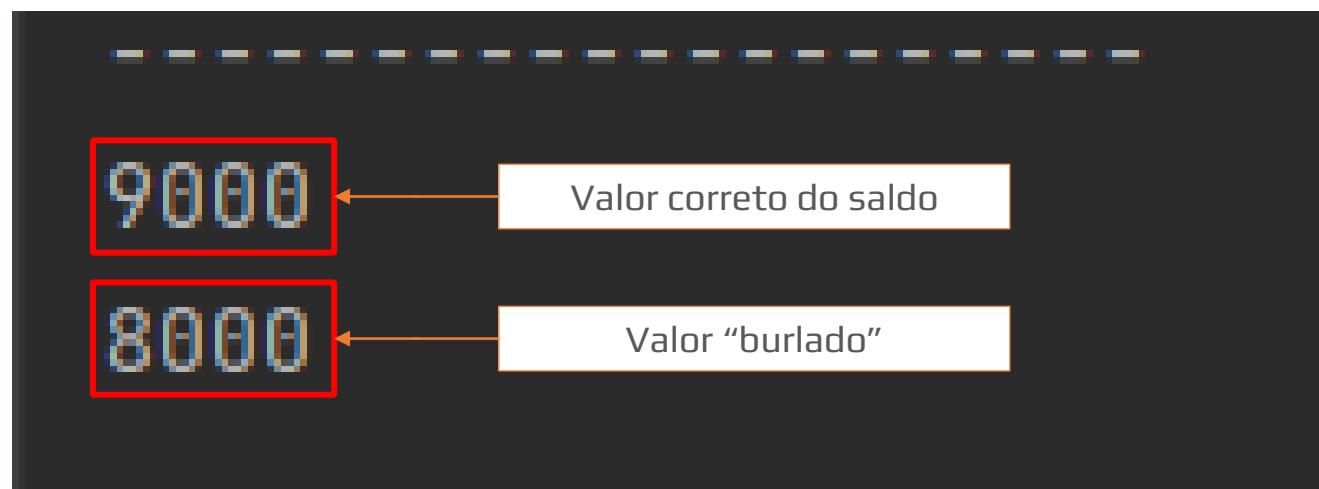
Feita a mudança, vamos tentar “burlar o sistema” mudando nosso saldo “por fora do programa”.

Perceba que para ainda é possível acessar o atributo saldo. A única diferença na prática é que agora esse atributo possui um \_ na frente.

Ou seja, essa mudança é uma convenção que não interfere em nenhum resultado prático do sistema.

No entanto, vamos ver no próximo slide um outro tipo de notação que não será apenas uma convenção mas sim uma informação dada ao Python sobre aquele atributo específico.

```
# Saldo via métodos
print(conta_lira._saldo)
# Tentando mudar o valor do saldo "por fora" do programa
conta_lira._saldo = 8000
#Novo valor após tentativa de "burlar" o sistema
print(conta_lira._saldo)
```



Ao invés de usarmos apenas 1 `_`, usaremos 2 no início do nosso atributo.

Essa informação nos permitirá informar que aquele atributo de fato só existirá dentro da classe.

Vamos refazer o exemplo anterior mas utilizando `__saldo` ao invés de `_saldo`.

Perceba que agora, o Python não consegue mais acessar esse atributo. O duplo `(__)` indica que este método é inacessível.

Consequentemente, temos um erro no nosso código conforme apresentado ao lado.

```
# Saldo via métodos
print(conta_lira.__saldo)
# Tentando mudar o valor do saldo "por fora" do programa
conta_lira.__saldo = 8000
#Novo valor após tentativa de "burlar" o sistema
print(conta_lira.__saldo)
```

```
-----
Traceback (most recent call last):
  File "C:/Users/danie/OneDrive/Área...
    print(conta_lira.__saldo)
AttributeError: 'ContaCorrente' obj...
```

Falamos anteriormente sobre convenções utilizadas na programação em Python. No entanto, algumas empresas definem suas próprias convenções e criam suas próprias documentações que permitem que todos os colaboradores daquela empresa possam utilizar aquele código e entender do que se trata.

Uma forma de documentar no próprio código os detalhes do código é utilizando o Docstring. Se você se lembra bem, já usamos esse conceito quando falamos sobre criação de funções.

Para criarmos uma Docstring, basta usar “”” conforme indicado ao lado.

Uma convenção muito utilizada no Python é a definida no documento PEP 257([link](#)).

```
class ContaCorrente:  
    """  
        Cria um objeto ContaCorrente para gerenciar as contas dos clientes.  
  
    Atributos:  
        nome (str): Nome do Cliente  
        cpf(str): CPF do Cliente. Deve ser inserido com pontos e traços (xxx.xxx.fff-xx)  
        agencia(str): Número da agencia  
        num_conta(str): Número da Conta Corrente do Cliente  
        saldo: Saldo disponível pelo Cliente  
        limite: Limite de cheque especial daquele Cliente  
        transacoes: Histórico de Transações do Cliente  
    """
```

Com o Docstring criado, podemos utilizar o comando abaixo para consultar via Python a documentação de uma classe:

`help(nome da classe)`

No nosso exemplo, basta utilizarmos:

`help(ContaCorrente)`

Informações documentadas  
por nós sobre os atributos  
da classe ContaCorrente

```
Cria um objeto ContaCorrente para gerenciar as contas dos clientes.

Atributos:
    nome (str): Nome do Cliente
    cpf(str): CPF do Cliente. Deve ser inserido com pontos e traços (XXX.XXX.XXX-XX)
    agencia(str): Número da agencia
    num_conta(str): Número da Conta Corrente do Cliente
    saldo: Saldo disponível pelo Cliente
    limite: Limite de cheque especial daquele Cliente
    transacoes: Histórico de Transações do Cliente

Methods defined here:

    __init__(self, nome, cpf, agencia, num_conta)
        Initialize self. See help(type(self)) for accurate signature.

    consultar_historico_transacoes(self)

    consultar_limite_chequeespecial(self)

    consultar_saldo(self)

    depositar_dinheiro(self, valor)

    sacar_dinheiro(self, valor)

    transferir(self, valor, conta_destino)

-----
Data descriptors defined here:

    __dict__
        dictionary for instance variables (if defined)

    __weakref__
        list of weak references to the object (if defined)
```

Agora que temos um classe de Conta Corrente vamos criar uma nova Classe que nos permita criar um outro objeto que possui relação com uma Conta Corrente.

Vamos pegar o exemplo de um cartão de crédito.

Já listamos os atributos iniciais dessa classe. Vamos ver como cria-la no Pycharm no próximo slide.

Iremos considerar que um dos atributos da classe Cartão de Crédito é uma das instâncias da classe ContaCorrente

### Classe : Conta Corrente

#### Atributos :

- Nome;
- CPF;
- Saldo;
- Limite;
- Agencia;
- Número da Conta;
- Transações

#### Métodos :

- consultar\_saldo;
- depositar\_dinheiro;
- limite\_conta;
- sacar\_dinheiro;
- consultar\_limite\_chequeespecial;
- consultar\_histórico\_transações;
- transferir

### Classe : Cartão de Crédito

#### Atributos :

- Numero;
- Titular;
- Validade;
- Código de segurança;
- Limite;
- Conta corrente;

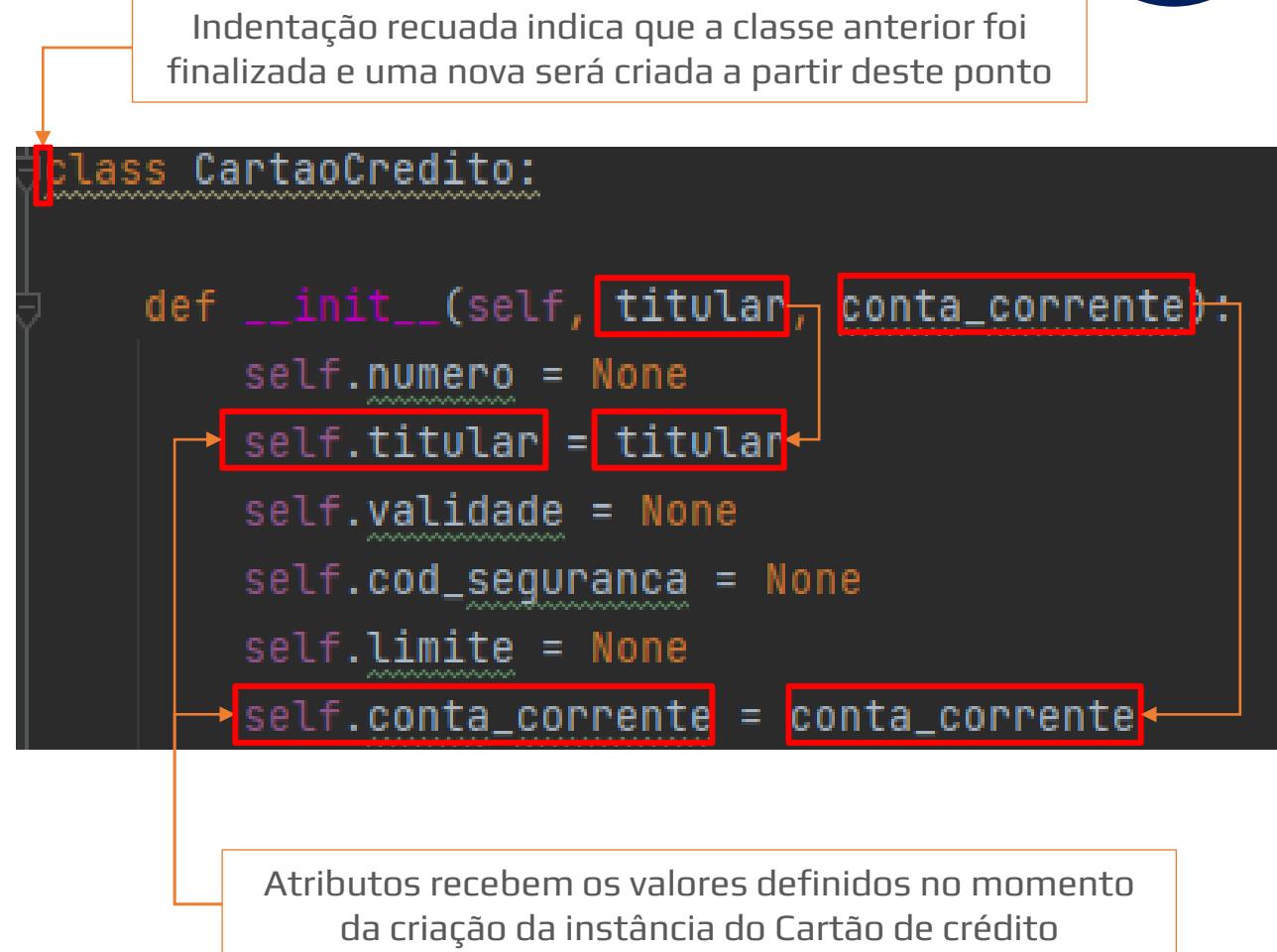
## Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Criando outra Classe e Relação entre Classes (2/6)

489

Logo ao fim da nossa classe ContaCorrente criamos a nova classe CartaoCredito. Perceba que a indentação foi recuada simbolizando que se trata de uma nova classe.

Assim como fizemos na classe corrente, iniciaremos a escrevê-la a partir do método `__init__` conforme apresentado ao lado.

Perceba os atributos `self.titular` e `self.conta_corrente` recebem seus valores a partir dos argumentos no momento da criação da instância.



Indentação recuada indica que a classe anterior foi finalizada e uma nova será criada a partir deste ponto

```
class CartaoCredito:  
  
    def __init__(self, titular, conta_corrente):  
        self.numero = None  
        self.titular = titular  
        self.validade = None  
        self.cod_seguranca = None  
        self.limite = None  
        self.conta_corrente = conta_corrente
```

Atributos recebem os valores definidos no momento da criação da instância do Cartão de crédito

Feito isso, podemos por exemplo acessar o número da conta usando as informações de um cartão de crédito.

```
conta_lira = ContaCorrente("Lira", "111.222.333-45", 1234, 34062)  
  
cartao_lira = CartaoCredito("Lira", conta_lira)  
  
print(cartao_lira.conta_corrente._num_conta)
```

Como vimos anteriormente nessa linha de código instanciamos uma nova conta corrente, **conta\_lira**

Criação de um cartão de crédito(nova instância). Perceba que no argumento **conta\_corrente**, usamos a instância **conta\_lira**. Ou seja, a instância **cartão\_lira** está relacionado a instância **conta\_lira**

O objetivo dessa linha de código é “printar” o número da conta corrente associada a um cartão de crédito:

**cartao\_lira**: Cartão desejado;  
**conta\_corrente**: acessa a conta corrente de cartão lira;  
**\_num\_conta**: acessa o atributo número da conta corrente associado ao **cartão\_lira**

34062

Número da conta Corrente

No entanto, o contrário não é possível. Não conseguimos a partir da classe Conta Corrente acessar todos os cartões atrelados aquela conta.

Para termos essa funcionalidade, vamos precisar voltar na nossa classe ContaCorrente e criar um novo atributo, conforme apresentado ao lado.

Perceba que este atributo é uma lista. Ou seja, caso existam mais de um cartão associados a uma Conta corrente, todos serão listados.

Mas assim como fizemos com o atributo `self._transacoes`, precisamos criar alguma forma que essa lista seja sempre atualizada a partir da criação de novos cartões(novas instâncias)

```
def __init__(self, nome, cpf, agencia, num_conta):  
    self._nome = nome  
    self._cpf = cpf  
    self._saldo = 0  
    self._limite = None  
    self._agencia = agencia  
    self._num_conta = num_conta  
    self._transacoes = []  
    self._cartoes = []
```

Atributo inicia como uma lista vazia. A cada instância criada a partir da classe CartaoCredito indicando esta conta corrente a lista será acrescida daquele cartão.

Como vimos no exemplo anterior, precisamos criar um mecanismo que inclua os cartões criados dentro da nossa lista.

Esse mecanismo é essencialmente, o mesmo que utilizamos no nosso método do histórico de transações.

Veja o print ao lado... Perceba que nesse caso, não utilizamos **self.** no início da linha de código.

Por que? Assim como vimos no caso das transações, o **self** referencia a própria instância. Nesse caso, não estamos falando de uma mesma instância e sim uma instância da classe ContaCorrente.

Perceba também que usamos o **self DENTRO** do método **.append**. O que isso significa?

Significa que estamos adicionando à lista não apenas um dos atributos mas sim **TODO** os atributos da desse cartão de crédito

```
class CartaoCredito:  
  
    def __init__(self, titular, conta_corrente):  
        self.numero = None  
        self.titular = titular  
        self.validade = None  
        self.cod_seguranca = None  
        self.limite = None  
        self.conta_corrente = conta_corrente  
        conta_corrente._cartoes.append(self)
```

Usamos o método **.append** para adicionar a lista **TODA** a instância a lista e não apenas o atributo **numero** por exemplo.

Com essas modificações, conseguimos acessar todos os cartões de crédito associados a uma conta corrente específica.

```
#Cria uma nova instância da classe ContaCorrente (conta_lira)
conta_lira = ContaCorrente("Lira", "111.222.333-45", 1234, 34062)

#Cria uma nova instância da classe CartaoCredito (cartao_lira)
cartao_lira = CartaoCredito("Lira", conta_lira)
#Cria um número aleatório de cartão apenas para o exemplo
cartao_lira.numero=123

#Retorna o número da conta associada ao cartão_lira
print(cartao_lira.conta_corrente._num_conta)
#Retorna a lista de cartões associadas a conta corrente conta_lira
print(conta_lira._cartoes)

#Acessando o primeiro item da lista
print(conta_lira._cartoes[0].numero)
```

```
34062
[<__main__.CartaoCredito object at 0x000002CD34AEE100>]
123
```

É criado um objeto que contém todos os cartões de crédito associados a conta. Se trata da lista self.\_cartoes que definimos anteriormente. TODOS os atributos definidos na instância CartaoCredito podem ser acessados.

Feita a relação entre as classes, vamos focar em definir melhor os atributos da nossa Classe Cartao de Credito e seus métodos.

Vamos iniciar pelo atributo validade. Se trata de um atributo que possui relação direta com data. Anteriormente, já tínhamos criado um método estático usando as bibliotecas datetime e pytz.

Vamos aproveitá-la na nossa nova classe.

**MAS ATENÇÃO!** Como vimos anteriormente, o nosso método estático foi criado para a classe ContaCorrente e deverá existir somente lá. Portanto, vamos precisar replicar o código dentro da classe CartaoCredito.

Não devemos referenciar um método estático de uma classe em outra.

```
class CartaoCredito:

    @staticmethod
    def _data_hora():
        fuso_BR = pytz.timezone('Brazil/East')
        horario_BR = datetime.now(fuso_BR)
        return horario_BR
```

Agora podemos alterar nosso método validade a partir do método estático criado.

```
class CartaoCredito:

    @staticmethod
    def _data_hora():
        fuso_BR = pytz.timezone('Brazil/East')
        horario_BR = datetime.now(fuso_BR)
        return horario_BR

    def __init__(self, titular, conta_corrente):
        self.numero = None
        self.titular = titular
        self.validade = '{}/{}/{}'.format(CartaoCredito._data_hora().month, CartaoCredito._data_hora().year + 4)
```

Usa o método `_data_hora` da classe `CartaoCredito` para coletar o mês

Usa o método `_data_hora` da classe `CartaoCredito` para coletar o ano. O valor + 4 permite estabelecer 4 anos a mais de validade em relação a criação do cartão

Outros dois atributos que vamos alterar são:

- self.numero
- self.cod\_segurança

Existem formas mais “corretas” de se realizar essa operação, mas por enquanto não é nosso objetivo.

POR ISSO, vamos usar uma biblioteca para gerar um número aleatório.

```
from random import randint
```

```
from datetime import datetime  
import pytz  
from random import randint
```

```
def __init__(self, titular, conta_corrente):  
    self.numero = randint(10000000000000, 99999999999999)  
    self.titular = titular  
    self.validade = '{}{}'.format(CartaoCredito._data_hora().month, CartaoCredito._data_hora().year)  
    self.cod_segurança = '{}{}{}'.format(randint(0, 9), randint(0, 9), randint(0, 9))
```

Vamos agora analisar todos os atributos e rodar o código uma vez para entender o que cada atributo cada código de atributo está gerando ao criarmos uma nova instância.

```
class CartaoCredito:  
  
    @staticmethod  
    def _data_hora():  
        fuso_BR = pytz.timezone('Brazil/East')  
        horario_BR = datetime.now(fuso_BR)  
        return horario_BR  
  
    def __init__(self, titular, conta_corrente):  
        self.numero = randint(10000000000000, 9999999999999999)  
        self.titular = titular  
        self.validade = '{}-{}'.format(CartaoCredito._data_hora().month, CartaoCredito._data_hora().year + 4)  
        self.cod_seguranca = '{}-{}-{}'.format(randint(0, 9), randint(0, 9), randint(0, 9))  
        self.limite = 1000  
        self.conta_corrente = conta_corrente  
        conta_corrente._cartoes.append(self)
```

```
#Cria uma nova instância da classe ContaCorrente (conta_lira)  
conta_lira = ContaCorrente("Lira", "111.222.333-45", 1234, 34062)  
  
#Cria uma nova instância da classe CartaoCredito (cartao_lira)  
cartao_lira = CartaoCredito("Lira", conta_lira)  
  
print(cartao_lira.numero,  
      cartao_lira.titular,  
      cartao_lira.validade,  
      cartao_lira.cod_seguranca,  
      cartao_lira.limite,
```

```
5762101841599675 Lira 5/2025 445 1000
```

No slide anterior, usamos um print listando todos os atributos que existiam dentro de cartao\_lira.

Nesse caso foram poucos mas imagine que fossem 100 atributos. Listá-los um a um não parece uma opção muito automática.

Existe um método que nos auxilia a executar essa operação de forma mais direta.

Esse método é o `__dict__` conforme apresentado ao lado.

Seu resultado é um dicionário o onde os atributos são as chaves.

```
print(cartao_lira.__dict__)
```

```
{'numero': 9935766831577870, 'titular': 'Lira', 'validade': '5/2025', 'cod_seguranca': '380', 'limite': 1000, '_senha': '1234', 'conta_corrente': <__main__.ContaCorrente object at 0x0000021E829EE100>}
```

# Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Separando Programa e Arquivo de Classes e Importando nossas Classes (1/4)

499

Até agora sempre usamos o mesmo arquivo tanto para as classes como para o nosso programa.

O mais comum, é que essas informações são criadas em arquivos diferentes.

Por enquanto temos o código dessa forma.

Perceba que existe uma separação clara entre a parte de definições do programa como classes, atributos, métodos etc... e os comandos que de fato “rodam” o programa. Criação de contas correntes, cartões de crédito, etc...

Vamos ver como realizar essa separação no próximo slide.

```
from datetime import datetime
import pytz
from random import randint

class ContaCorrente:
    """Cria um objeto ContaCorrente para gerenciar as contas dos clientes"""

    @staticmethod
    def _data_hora(): ...

    def __init__(self, nome, cpf, agencia, num_conta): ...

    def consultar_saldo(self): ...

    def depositar_dinheiro(self, valor): ...

    def _limite_conta(self): ...

    def sacar_dinheiro(self, valor): ...

    def consultar_limite_chequeespecial(self): ...

    def consultar_historico_transacoes(self): ...

    def transferir(self, valor, conta_destino): ...

class CartaoCredito:

    @staticmethod
    def _data_hora(): ...

    def __init__(self, titular, conta_corrente): ...

#PROGRAMA

#Cria uma nova instância da classe ContaCorrente (conta_lira)
conta_lira = ContaCorrente("Lira", "111.222.333-45", 1234, 34062)

#Cria uma nova instância da classe CartaoCredito (cartao_lira)
cartao_lira = CartaoCredito("Lira", conta_lira)
```

Criação das classes, atributos e métodos

Programa. Onde são criadas as instâncias e os códigos são de fato executados

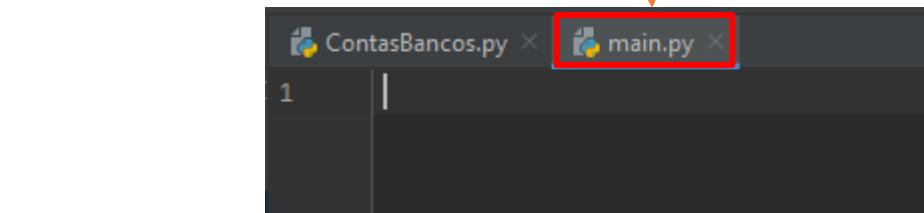
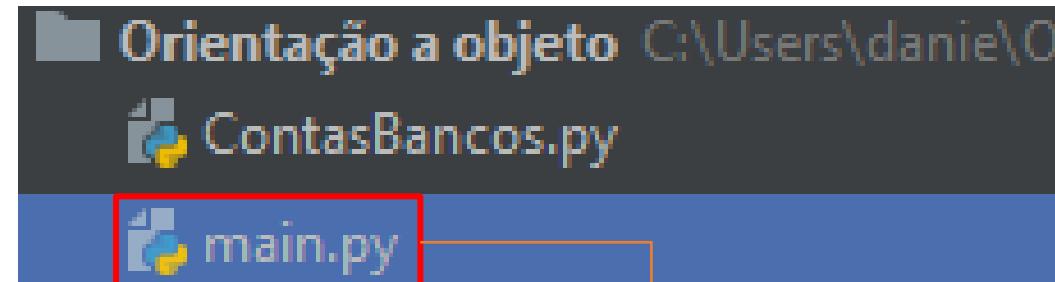
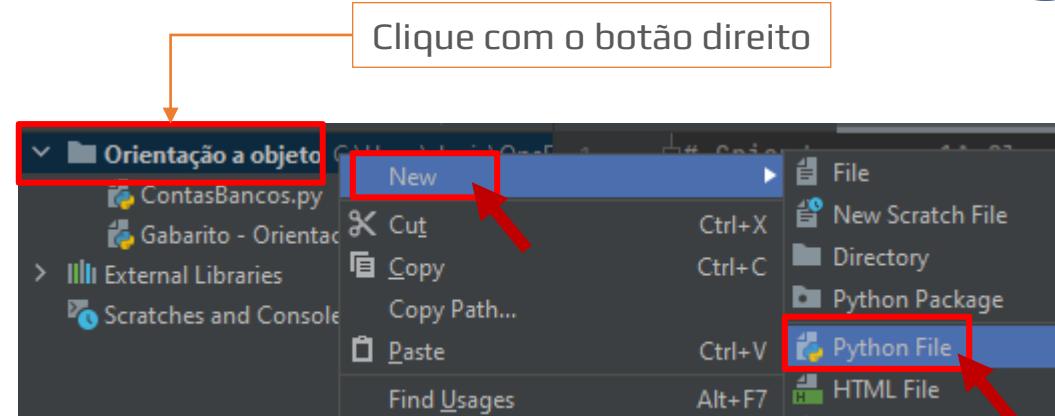
## Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Separando Programa e Arquivo de Classes e Importando nossas Classes (2/4)

Nessa separação iremos transferir a parte do **PROGRAMA** para um novo arquivo do tipo .py.

Esse arquivo terá um nome específico: main.

Ele deverá ser criado na mesma pasta do nosso arquivo ContasBancos.py que estamos usando até agora.

Para isso, basta seguir os passos indicados ao lado.



# Módulo 30– Orientação a Objetos Completo - Classes e Métodos – Separando Programa e Arquivo de Classes e Importando nossas Classes (3/4)

501

Agora que criamos nosso arquivo `main.py`, é só transferir o código do nosso programa para esse arquivo.

```
from datetime import datetime
import pytz
from random import randint

class ContaCorrente:
    """Cria um objeto ContaCorrente para gerenciar as contas dos clientes"""

    @staticmethod
    def _data_hora():
        ...

    def __init__(self, nome, cpf, agencia, num_conta):
        ...

    def consultar_saldo(self):
        ...

    def depositar_dinheiro(self, valor):
        ...

    def _limite_conta(self):
        ...

    def sacar_dinheiro(self, valor):
        ...

    def consultar_limite_chequeespecial(self):
        ...

    def consultarHistorico_transacoes(self):
        ...

    def transferir(self, valor, conta_destino):
        ...

class CartaoCredito:
    """Cartão de crédito"""

    @staticmethod
    def _data_hora():
        ...

    def __init__(self, titular, conta_corrente):
        ...

#PROGRAMA
#Cria uma nova instância da classe ContaCorrente (conta_lira)
conta_lira = ContaCorrente("Lira", "111.222.333-45", 1234, 34062)

#Cria uma nova instância da classe CartaoCredito (cartao_lira)
cartao_lira = CartaoCredito("Lira", conta_lira)
```

```
ContasBancos.py x main.py x
1
2
3
4
5
6
7
8
#PROGRAMA
#Cria uma nova instância da classe ContaCorrente (conta_lira)
conta_lira = ContaCorrente("Lira", "111.222.333-45", 1234, 34062)

#Cria uma nova instância da classe CartaoCredito (cartao_lira)
cartao_lira = CartaoCredito("Lira", conta_lira)
```

Bem, mas como o nosso arquivo main.py vai entender que as classes que ele utiliza estão no nosso arquivo ContasBancos?

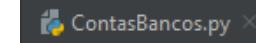
Iremos usar o mesmo princípio que usamos várias vezes até aqui, através do IMPORT.

No fundo, o que estamos fazendo agora, é importando uma biblioteca como outra qualquer, mas só que essa biblioteca foi criada por nós ☺.

No exemplo ao lado mostramos como fazer essa importação.

Parabéns! Você acabou de criar a sua primeira biblioteca ☺

Usamos o from para indicar o nome da biblioteca. No nosso caso, o nome do arquivo que estão as classes.



```
from ContasBancos import ContaCorrente, CartaoCredito

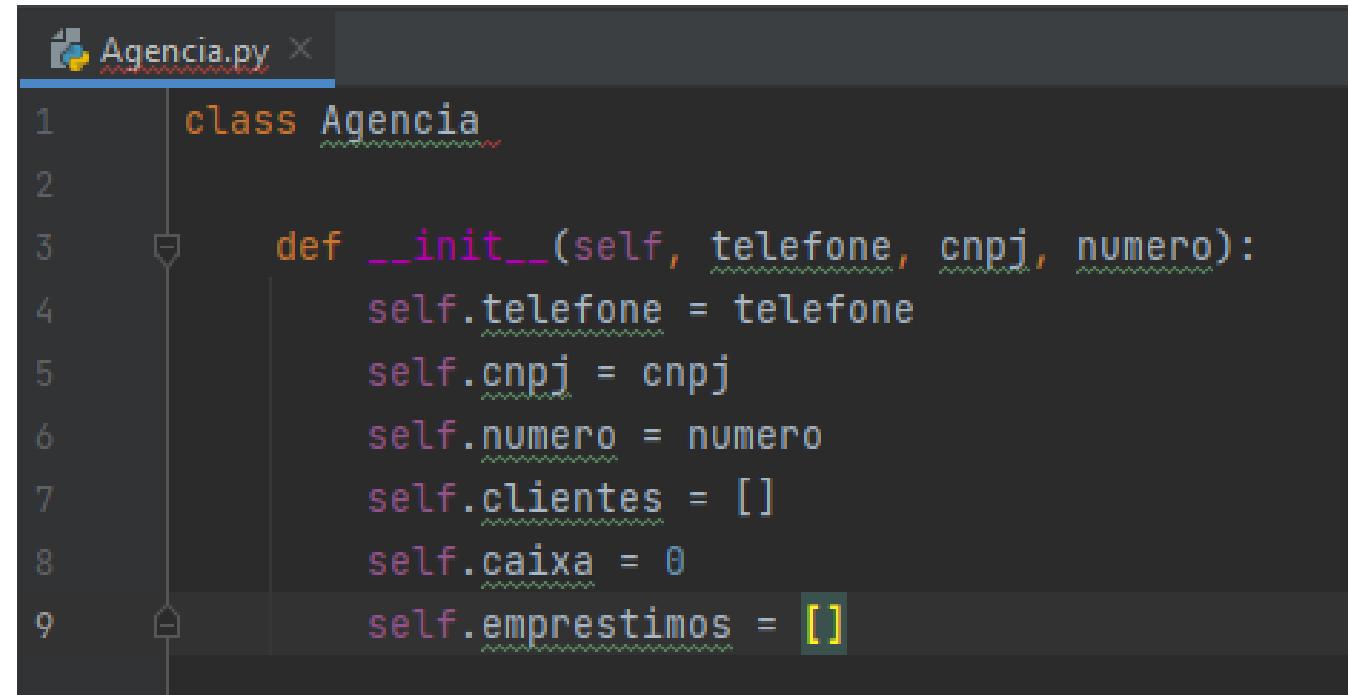
#PROGRAMA

#Cria uma nova instância da classe ContaCorrente (conta_lira)
conta_lira = ContaCorrente("Lira", "111.222.333-45", 1234, 34062)

#Cria uma nova instância da classe CartaoCredito (cartao_lira)
cartao_lira = CartaoCredito("Lira", conta_lira)
```

Vamos aprender outros conceitos importantes quando estamos falando de classes.

Para isso, vamos criar um novo arquivo Agencia.py e criar uma classe chamada Agencia conforme apresentado ao lado.

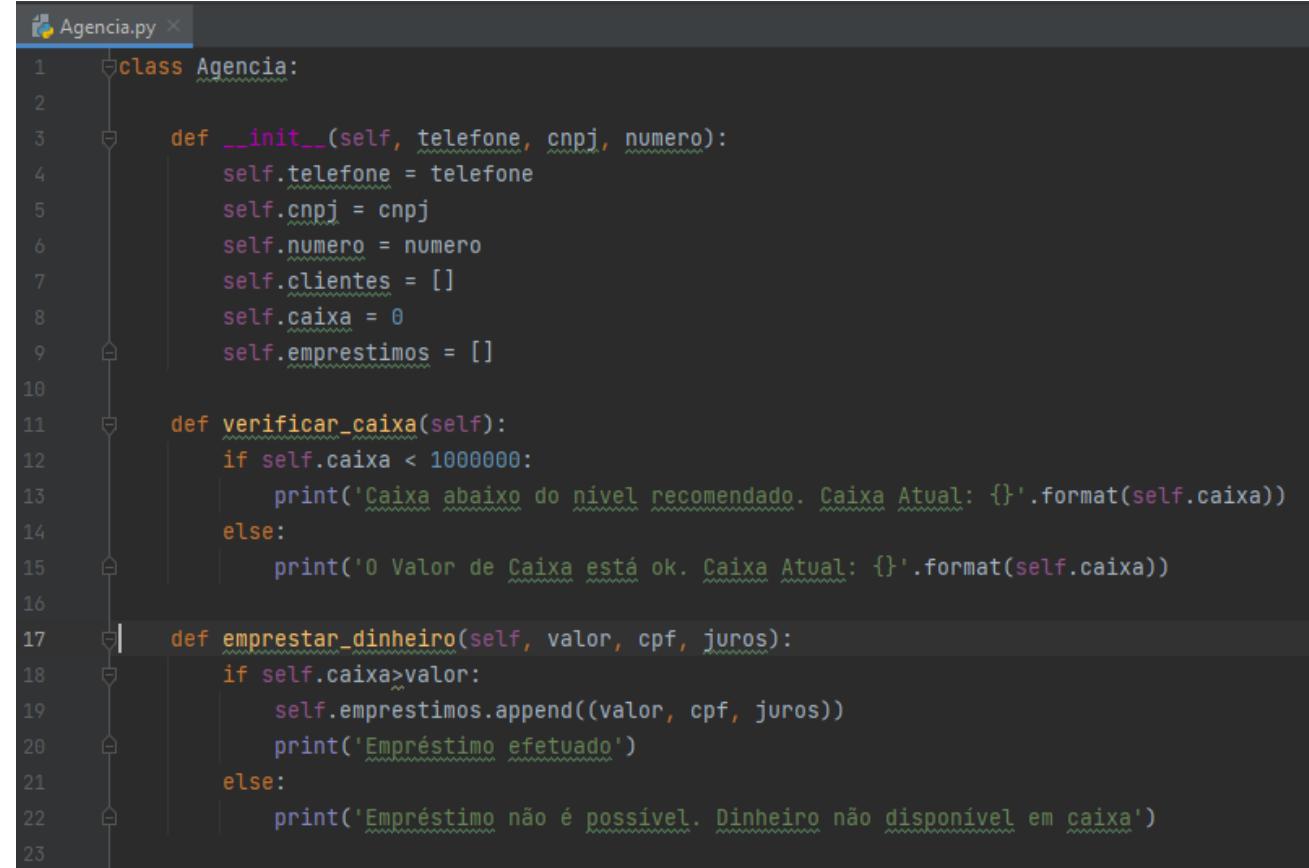


```
Agencia.py
1 class Agencia
2
3     def __init__(self, telefone, cnpj, numero):
4         self.telefone = telefone
5         self.cnpj = cnpj
6         self.numero = numero
7         self.clientes = []
8         self.caixa = 0
9         self.emprestimos = []
```

O primeiro método dessa classe será um verificador de caixa da agência. Ou seja, sempre que o caixa da agência estiver com um montante abaixo de um limite específico, um alerta será emitido.

No nosso exemplo, conforme exemplo ao lado, o limite será R\$1.000.000.

Além disso, teremos o método **emprestar\_dinheiro**, que baseado no montante existente em caixa poderá ou não emprestar o dinheiro à um cliente.



```
1  class Agencia:
2
3      def __init__(self, telefone, cnpj, numero):
4          self.telefone = telefone
5          self.cnpj = cnpj
6          self.numero = numero
7          self.clientes = []
8          self.caixa = 0
9          self.emprestimos = []
10
11     def verificar_caixa(self):
12         if self.caixa < 1000000:
13             print('Caixa abaixo do nível recomendado. Caixa Atual: {}'.format(self.caixa))
14         else:
15             print('O Valor de Caixa está ok. Caixa Atual: {}'.format(self.caixa))
16
17     def emprestar_dinheiro(self, valor, cpf, juros):
18         if self.caixa > valor:
19             self.emprestimos.append((valor, cpf, juros))
20             print('Empréstimo efetuado')
21         else:
22             print('Empréstimo não é possível. Dinheiro não disponível em caixa')
```

Antes de entrarmos em conceitos novos, vamos checar nossas classes criando uma nova agência (`agencia1`) conforme imagem abaixo:

```
agencia1 = Agencia(22223333, 200000000, 4568)

agencia1.caixa=1000000

print(agencia1.__dict__)

agencia1.verificar_caixa()

agencia1.emprestar_dinheiro(10, 11122233344, 0.1)
```

```
{'telefone': 22223333, 'cnpj': 200000000, 'numero': 4568, 'clientes': [], 'caixa': 1000000, 'emprestimos': []}
O Valor de Caixa está ok. Caixa Atual: 1000000
Empréstimo efetuado
```

Até aqui tudo bem, vamos criar mais um método que nos permite adicionar Clientes a nossa agência.

PRONTO! Temos nossa classe agência criada com alguns método básicos.

O que veremos nos próximos slides é como fazer quando temos classes dentro de classes.

Por exemplo:

Existem diversos tipos de agências: físicas, virtuais, premium, etc... Apesar de serem diferentes, são todas agências e existem pelos mesmos motivos.

```
def adicionar_cliente(self, nome, cpf, patrimonio):  
    self.clientes.append((nome, cpf, patrimonio))
```

```
agencia1.adicionar_cliente('Lira', 11122233344, 10000)  
print(agencia1.clientes)
```

```
[('Lira', 11122233344, 10000)]
```

Agora que temos um padrão para o que é uma agência (definido na classe Agencia), podemos criar alguns “tipos de agências”.

Na prática, todas são Agências, mas possuem alguma variação que fazem com que ela seja mesmo que pouco, diferente do padrão.

Ao invés de criarmos do zero, vamos criar nossas agências informando que elas são “filhas” da nossa classe Agencia.

Como fazer isso? Veja o exemplo ao lado. Perceba que dentro do parênteses indicamos Agencia.

Essencialmente o que está sendo criado aqui é uma nova classe que pelo menos POR ENQUANTO, possui as mesmas características da classe Agência.

```
class AgenciaVirtual(Agencia):
    pass

class AgenciaComum(Agencia):
    pass

class AgenciaPremium(Agencia):
    pass
```

Vamos criar uma agencia virtual por exemplo.

A princípio, não definimos nenhum atributo na nossa classe Agencia Virtual nem método `__init__`

Se tentarmos simplesmente criarmos uma agencia virtual usando essa classe, teremos um erro.

Se analisarmos o erro, vamos perceber que faltam 3 argumentos para a criação da instância.

Quais são esses argumentos? Os 3 argumentos que foram definidos na classe Mãe AGENCIA.



```
agencia_virtual = AgenciaVirtual()
```

```
Traceback (most recent call last):
  File "C:/Users/danie/OneDrive/Área de Trabalho/Cartilhas Python/Arquivos apostila/Módulo 30/or
    agencia_virtual = AgenciaVirtual()
TypeError: __init__() missing 3 required positional arguments: 'telefone', 'cnpj', and 'numero'
```

Argumentos herdados da classe “mãe” Agencia

```
agencia_virtual = AgenciaVirtual(22224444, 1520000000, 1000)

print(agencia_virtual.__dict__)
```

```
{'telefone': 22224444, 'cnpj': 1520000000, 'numero': 1000, 'clientes': [], 'caixa': 0, 'emprestimos': []}
```

Mas como fazer se quisermos aproveitar a relação “mãe –filha” mas com flexibilidade de personalizar essa classe?

Conforme apresentado ao lado, criaremos um método `__init__` para nossa agência virtual, mas ao invés de recriarmos todos os atributos iremos indicar quais dos atributos da classe original deverão ser “herdados”.

Usaremos `super()` para indicar essa herança.

No exemplo ao lado estamos herdando da classe agência (Superclasse) o método `init` e os atributos `telefone`, `cnpj` e `numero`.

Perceba que como utilizamos `super()` dentro do método `__init__`, `telefone` e `cnpj`, serão herdados da classe mãe. Só sendo necessária a declaração dentro do método do atributo `site` que é particular a esse tipo de agência

```
class AgenciaVirtual(Agencia):
    def __init__(self, site, telefone, cnpj):
        self.site = site
        super().__init__(telefone, cnpj, numero)
```

Indica que algo será herdado da classe mãe ou superclasse

O que será importado da classe mãe

Agora, podemos instanciar a agência virtual através da nossa nova classe. Perceba que mesmo sem declarar no método `__init__` os atributos `clientes`, e `emprestimos` foram herdados da classe mãe `Agencia`.

```
agencia_virtual = AgenciaVirtual('www.agenciacentral.com.br', 22224444, 15200000000)
agencia_virtual.verificar_caixa()
print(agencia_virtual.__dict__)
```

```
{'site': 'www.agenciacentral.com.br', 'telefone': 22224444, 'cnpj': 15200000000, 'numero': 1000, 'clientes': [], 'caixa': 1000000, 'emprestimos': []}
```

Vamos repetir o mesmo processo utilizado anteriormente mas para as demais agências (Comum e Premium)

Perceba que conceitualmente, nada mudou do exemplo anterior.

Ainda estamos herdando os atributos da classe mãe Agencia, a única diferença é que aqui, a personalização desses atributos é distinta da personalização realizada no exemplo anterior.

Ao instanciarmos agencia\_comum, podemos ver que assim como no caso anterior, todos os atributos definidos na classe mãe foram herdados mas foram personalizados pelo método \_\_init\_\_ da classe AgenciaComum(classe filha).

```
class AgenciaComum(Agencia):  
  
    def __init__(self, telefone, cnpj):  
        super().__init__(telefone, cnpj, randint(1001, 9999))  
        self.caixa=1000000
```

```
agencia_comum = AgenciaComum(33334444, 222000000000)  
agencia_comum.verificar_caixa()  
print(agencia_comum.__dict__)
```

```
{'telefone': 33334444, 'cnpj': 222000000000, 'numero': 7529, 'clientes': [], 'caixa': 1000000, 'emprestimos': []}
```

Na agência premium usaremos o mesmo princípio.

Conforme podemos observar a personalização aqui será no valor do caixa(10.000.000 ao invés de 1.000.000) .

```
class AgenciaPremium(Agencia):  
  
    def __init__(self, telefone, cnpj):  
        super().__init__(telefone, cnpj, randint(1001, 9999))  
        self.caixa = 10000000
```

```
agencia_premium = AgenciaPremium(33333333, 300000000000)  
print('agencia_premium:')  
print(agencia_premium.__dict__)
```

```
agencia_premium:  
{'telefone': 33333333, 'cnpj': 300000000000, 'numero': 8352, 'clientes': [], 'caixa': 10000000, 'emprestimos': []}
```

Se compararmos as 4 agências, poderemos ver que são todas muito próximas pois possuem a mesma base (classe Agência), mas possuem suas particularidades.

```
agencia1:  
{'telefone': 22223333, 'cnpj': 200000000, 'numero': 4568, 'clientes': [], 'caixa': 0, 'emprestimos': []}  
agencia_virtual:  
{'site': 'www.agenciavirtual.com.br', 'telefone': 11111111, 'cnpj': 100000000000, 'numero': 1000, 'clientes': [], 'caixa': 1000000, 'emprestimos': []}  
agencia_comum:  
{'telefone': 22222222, 'cnpj': 200000000000, 'numero': 8943, 'clientes': [], 'caixa': 1000000, 'emprestimos': []}  
agencia_premium:  
{'telefone': 33333333, 'cnpj': 300000000000, 'numero': 8352, 'clientes': [], 'caixa': 10000000, 'emprestimos': []}
```

Assim como feito anteriormente para atributos, também podemos personalizar os métodos de uma classe.

Significa dizer que, poderão ser personalizados dentro da classe filha específica.

Vamos iniciar com o exemplo ao lado da Agencia Virtual.

Nessa agência, temos uma particularidade que é o uso de paypal para transferências. Assim, vamos criar 2 métodos para esse tipo de operação.

Além disso, será necessário a criação de um novo atributo **caixa\_paypal** que receberá o saldo das operações realizadas.

```
class AgenciaVirtual(Agencia):  
  
    def __init__(self, site, telefone, cnpj):  
        self.site = site  
        super().__init__(telefone, cnpj, 1000)  
        self.caixa = 1000000  
        self.caixa_paypal = 0  
  
    def depositar_paypal(self, valor):  
        self.caixa -= valor  
        self.caixa_paypal += valor  
  
    def sacar_paypal(self):  
        self.caixa_paypal -= valor  
        self.caixa += valor
```

Novo atributo criado.  
Só existe na agência virtual

2 novos métodos que existem apenas dentro da classe filha AgenciaVirtual

Ao rodarmos nosso 'novo método na Agencia Virtual, temos o valor esperado conforme apresentado na imagem ao lado.

No entanto, veja o que acontece ao tentarmos rodar o mesmo método em outra agência.

Nada acontece... Isso ocorre pois, conforme falamos anteriormente apesar de todas serem filhas de uma mesma agência, criamos um método que é único a aquela agência.

Um paralelo bobo mas que pode ajudar. Imagine a classe Agencia como uma mãe que teve 3 filhos... Apesar de todos serem parecidos e terem sido criados da mesma forma, um se tornou médico, outro advogado e o terceiro astronauta.

Não podemos pedir para que o filho advogado faça uma cirurgia... Isso é uma habilidade(um método) que só o filho médico sabe fazer.

```
agencia_virtual.depositar_paypal(20000)
print(agencia_virtual.caixa)
print(agencia_virtual.caixa_paypal)
```

```
agencia_comum.depositar_paypal(20000)
print(agencia_comum.caixa)
print(agencia_virtual.caixa_paypal)
```

```
Traceback (most recent call last):
  File "C:/Users/danie/OneDrive/Área de Trabalho/Cartilhas Python/Arquivos a
    agencia_comum.depositar_paypal(20000)
AttributeError: 'AgenciaComum' object has no attribute 'depositar_paypal'
980000
20000
```

O mesmo conceito de herança utilizado para atributos, pode ser aplicado a métodos. Ou seja, usaremos a nossa classe mãe para os métodos padrões mas alterando um pouco a forma como eles irão funcionar.

Veja o exemplo ao lado onde criamos o método `adicionar_cliente` DENTRO da classe `AgenciaPremium`.

Perceba que ao usarmos `def adicionar_clientes` na classe `AgenciaPremium`, estamos criando um método que irá sobrepor o método `adicionar_cliente` da classe `Agencia`.

No entanto, usaremos novamente `super()` para que herdar o que é comum aos dois métodos.

Apenas adicionaremos o que é particular a esse método na classe `AgenciaPremium`.

```
class Agencia:  
    def adicionar_cliente(self, nome, cpf, patrimonio):  
        self.clientes.append((nome, cpf, patrimonio))  
  
class AgenciaPremium(Agencia):  
    def __init__(self, telefone, cnpj):  
        super().__init__(telefone, cnpj, randint(1001, 9999))  
        self.caixa=10000000  
  
    def adicionar_cliente(self, nome, cpf, patrimonio):  
        if patrimonio>1000000:  
            super().adicionar_cliente(nome, cpf, patrimonio)  
        else:  
            print('Cliente não possui o patrimônio mínimo necessário')
```

O método `adicionar_cliente` da classe `AgenciaPremium` herda o método de mesmo nome da classe `Agencia`

Vamos novamente rodar um código que nos permita adicionar clientes nos 4 tipos de agência e compararmos os resultados.

Nosso objetivo é adicionar o cliente Lira a todas as agências.

O patrimônio que usaremos é de 1.000.000 (exatamente o limite da agência premium)

```
agencia1.adicionar_cliente('Lira', 11111111111, 1000000)
print('clientes agencia1:', agencia1.clientes)

agencia_virtual.adicionar_cliente('Lira', 11111111111, 1000000)
print('clientes agencia_virtual:', agencia_virtual.clientes)

agencia_comum.adicionar_cliente('Lira', 11111111111, 1000000)
print('clientes agencia_comum:', agencia_comum.clientes)

agencia_premium.adicionar_cliente('Lira', 11111111111, 1000000)
print('clientes agencia_premium:', agencia_premium.clientes)
```

```
clientes agencia1: [('Lira', 11111111111, 1000000)]
clientes agencia_virtual: [('Lira', 11111111111, 1000000)]
clientes agencia_comum: [('Lira', 11111111111, 1000000)]
Cliente não possui o patrimônio mínimo necessário
clientes agencia_premium: []
```

Perceba que apesar de usarmos o mesmo método para a agencia\_premium tivemos um resultado distinto, visto que o patrimônio não atendia o requisito.

Essa diferença ao usarmos o mesmo método (`adicionar_clientes`) em classes diferentes, se chama **polimorfismo**. Apesar de terem a mesma função de adicionar clientes a uma base, se manifestam de formas diferentes.

Módulo 31

# Interface Gráfico - Tkinter e Criando Sistemas com Python

# Módulo 31 – Interface Gráfico - Tkinter e Criando Sistemas com Python

## Tkinter - Criando uma Janela (1/3)

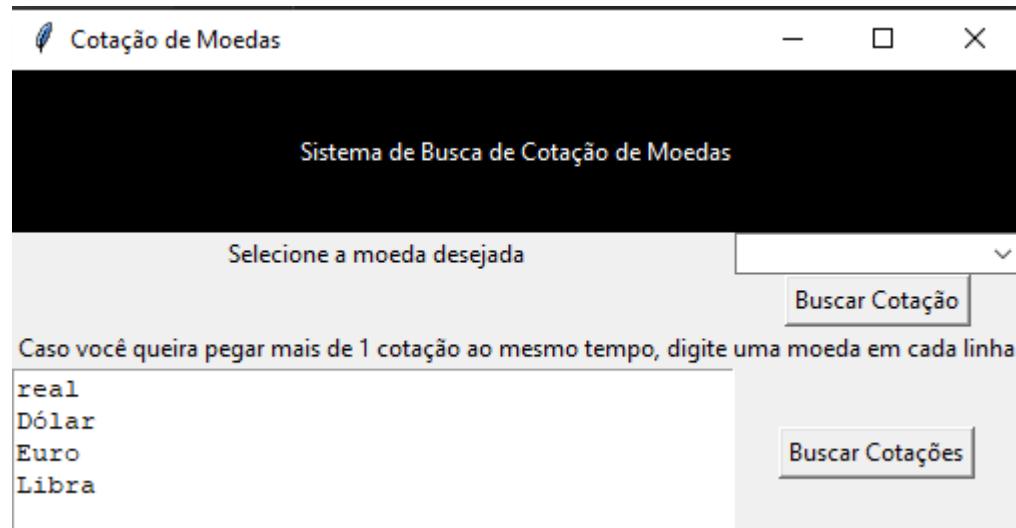
519

Neste módulo, vamos aprender a como criar interface gráficas que permitem melhorar a experiência do usuário ao usar um programa em Python.

Essa biblioteca já “vem dentro” do Python, e portanto, não precisa ser instalada.

Colocamos aqui ao lado, alguns dos exemplos dessa interface gráfica para que seja mais fácil entender o que queremos dizer ☺.

Agora vamos iniciar nosso código.



Primeiro passo é criarmos no nosso Pycharm um novo projeto e um novo arquivo do tipo .py conforme apresentado ao lado.

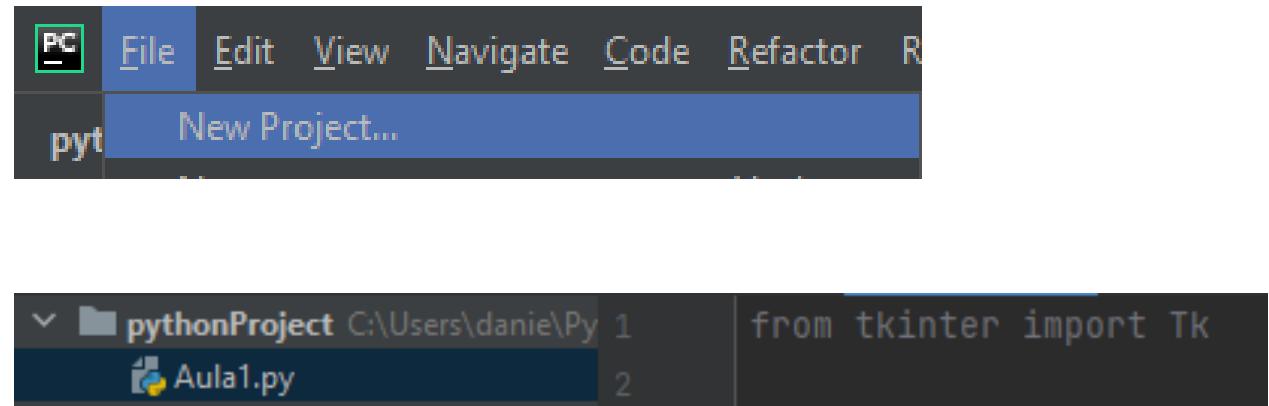
Como já dito anteriormente não precisamos instalar o Tkinter, mas precisamos importa-lo. Para isso, usaremos o comando:

```
from tkinter import Tk
```

Aproveitando o conhecimento de orientação a objetos que vimos anteriormente vamos entender o que significa importar essa biblioteca.

Essa importação significa dizer importar uma Classe para que a partir disso, seja possível criar instâncias/objetos que possam ser utilizados por nós.

É por ai que iremos começar no próximo slide.



Para criarmos uma instância, usaremos a linha de código apresentado ao lado.

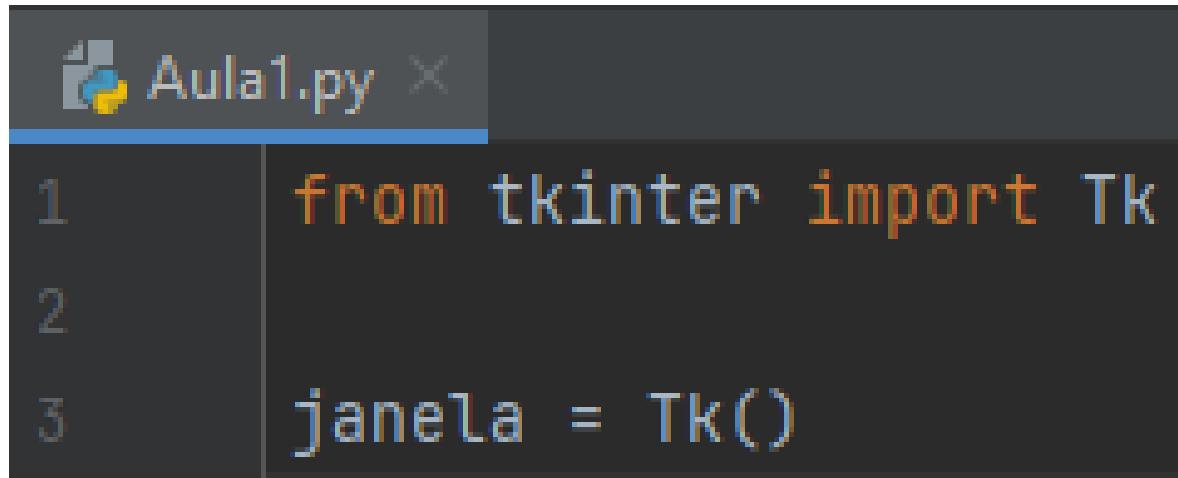
No entanto, se você rodar apenas essa linha de código, nada irá acontecer....

Por que? Apesar de termos criado a instância, ainda não fornecemos nenhum comando que nos permita abrir essa janela que criamos.

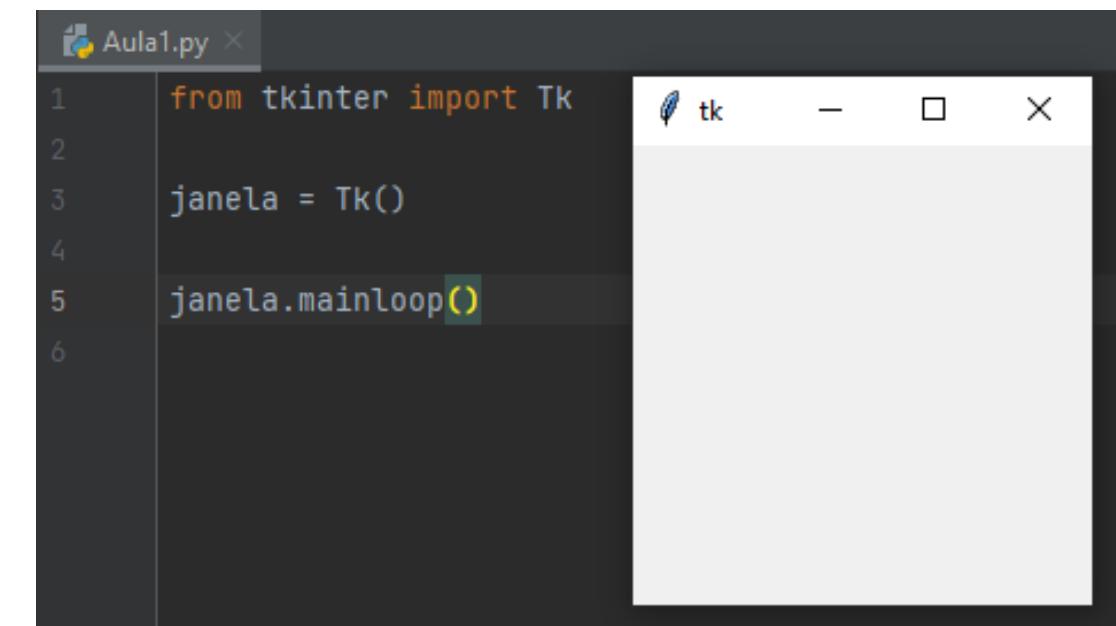
Esse comando é o método `.mainloop()`

Para o Tkinter, uma janela aberta funciona como um looping infinito. Como assim?

O que parece ser uma janela estática e sem graça na verdade é um programa em funcionamento. OK! Não tem muita função, mas o que é importante entender aqui é que ela se mantém aguardando algum comando que chamaremos de evento daqui para frente. Esse looping só se encerrar ao apertarmos X para fechar a página.



```
1 from tkinter import Tk
```



```
1 from tkinter import Tk
2
3 janela = Tk()
4
5 janela.mainloop()
```

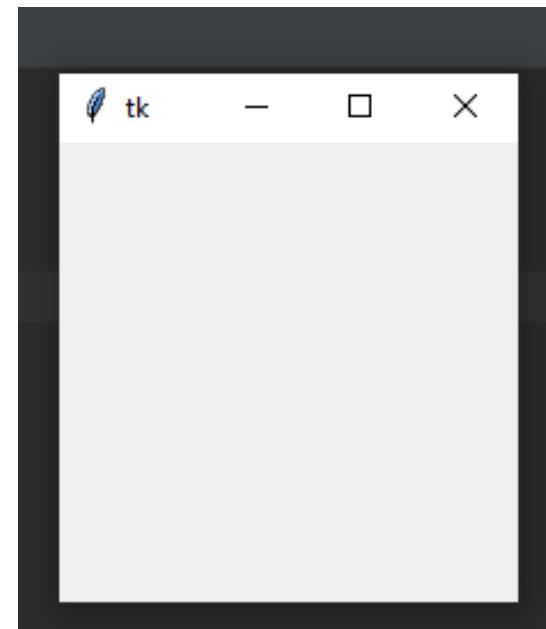
Comparação um pouco bizarra mas que talvez ajude.

Você alguma vez já viu na TV ou em um vídeo do youtube um dos guardas da família real britânica?

Essencialmente durante MUITO tempo ele fica estático sem fazer nada...

Parece inútil(desculpe se você é ou conhece alguém que faz isso), mas ele está o tempo todo fazendo algo. Ele só sairá daquela posição caso haja um evento (algum turista querendo roubar o chapéu dele) ou chegue o fim do seu turno.

Com nossa janela do Tkinter pelo menos a princípio, parece inútil, mas é importante entender que ela está rodando e só irá parar quando clicarmos em fechar.



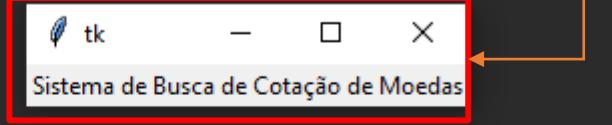
Comparações bizarras a parte, vamos aprender algum um pouco mais útil que será uma caixa de mensagem para o nosso usuário.

Antes, vamos mudar um pouco também a nossa importação para não apenas importarmos a classe Tk mas sim toda a biblioteca TKINTER.

Para termos uma mensagem, precisamos criar um objeto mensagem. Para criação desse objeto usaremos o método **.Label()**.

No entanto, apenas a criação do objeto não é suficiente, precisamos usar outro método que nos permita encapsular esse objeto dentro da Janela do Tkinter. Esse método é o **pack()** conforme apresentado ao lado.

Ao inserirmos algum objeto dentro da janela ela automaticamente altera seu tamanho para o tamanho do objeto.



```
import tkinter as tk
janela = tk.Tk()
mensagem = tk.Label(text="Sistema de Busca de Cotação de Moedas")
mensagem.pack()
janela.mainloop()
```

Cria o objeto  
mensagem

O método **.pack()**  
insere o objeto  
dentro da nossa  
janela

Vamos criar uma nova mensagem.

Perceba que a nossa janela cresceu de tamanho apesar da formatação ainda não ser das mais belas.

Perceba que o passo a passo foi o mesmo.

- 1) Criar objeto;
- 2) Usar .pack() para inserir o objeto na janela.

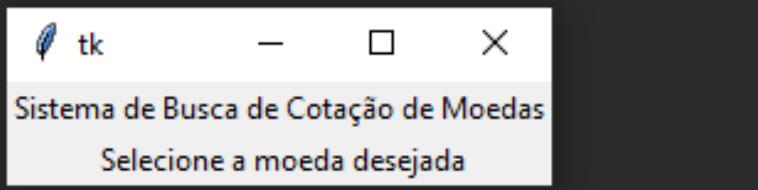
```
import tkinter as tk

janela = tk.Tk()

mensagem = tk.Label(text="Sistema de Busca de Cotação de Moedas")
mensagem.pack()

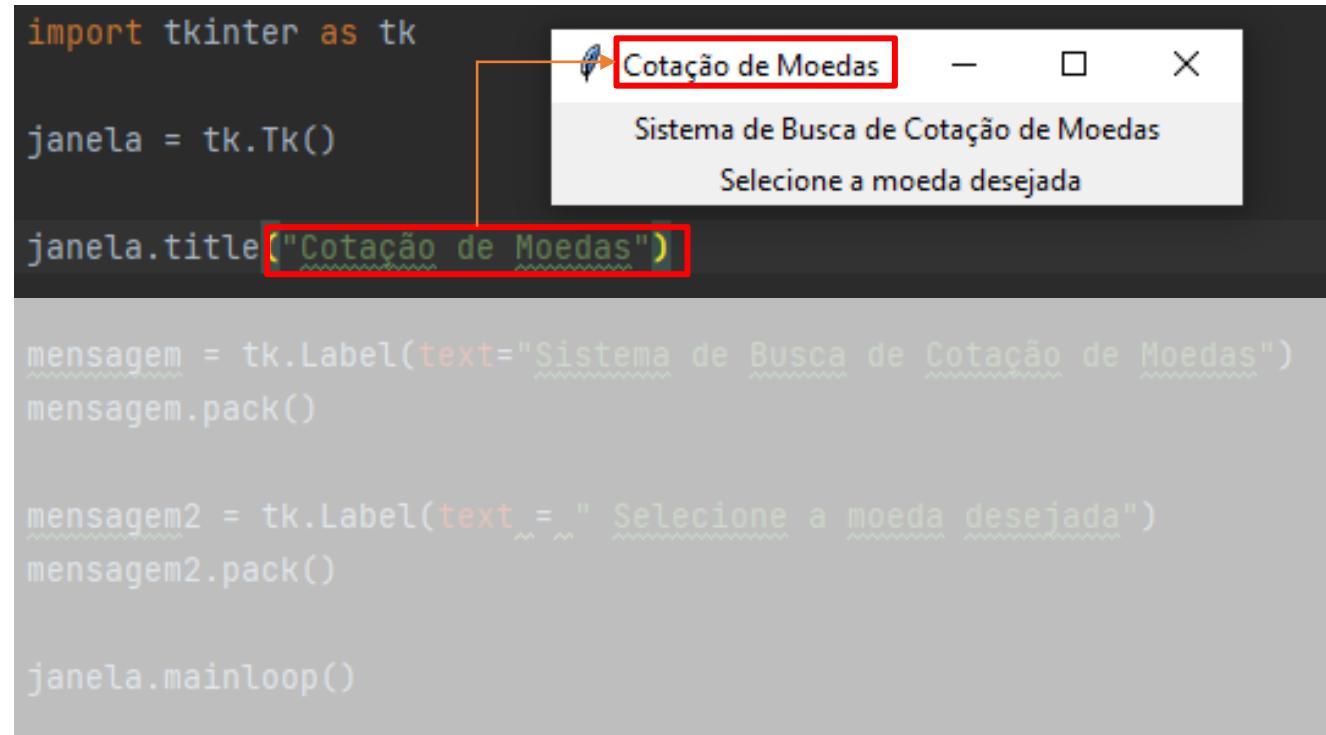
mensagem2 = tk.Label(text="Selecione a moeda desejada")
mensagem2.pack()

janela.mainloop()
```



Você já deve ter percebido que no rótulo da nossa janela temos um tk.

É possível alterar esse nome de forma bem simples conforme apresentado ao lado.



```
import tkinter as tk

janela = tk.Tk()

janela.title("Cotação de Moedas") # Line highlighted with a red box

mensagem = tk.Label(text="Sistema de Busca de Cotação de Moedas")
mensagem.pack()

mensagem2 = tk.Label(text=" Selecione a moeda desejada")
mensagem2.pack()

janela.mainloop()
```

Vamos agora mudar a cor dos nossos objetos.

Perceba que quando escrevemos a mensagem, usamos a keyword **text**.

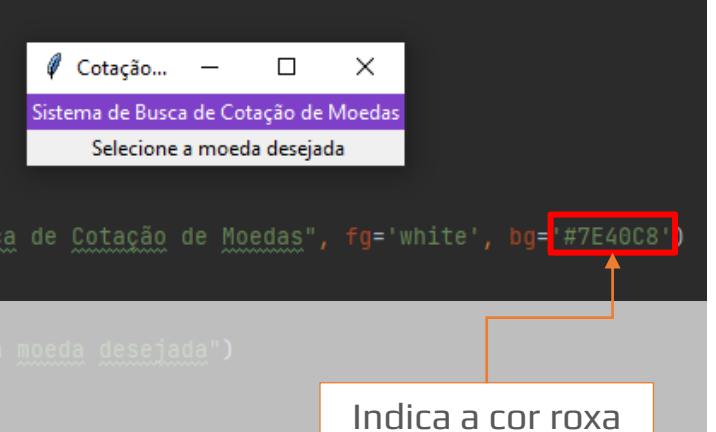
Para mudar a cor de fundo, iremos usar o mesmo conceito.

Veja o exemplo ao lado, lá estamos usando 2 keywords, **bg** e **fg**.

- **bg**: corresponde a cor de fundo;
- **fg**: corresponde a cor da fonte do texto.

### ATENÇÃO!

Perceba que essas configurações apenas são aplicáveis a mensagem 1 e não para toda a janela.



```
import tkinter as tk

janela = tk.Tk()

janela.title("Cotação de Moedas")

mensagem = tk.Label(text="Sistema de Busca de Cotação de Moedas", fg='white', bg="#7E40C8")
mensagem.pack()

mensagem2 = tk.Label(text=" Selecione a moeda desejada")
mensagem2.pack()

janela.mainloop()
```

Indica a cor roxa em hexadecimal

Outra edição possível é alterar o tamanho da nossa janela.

Como podemos esperar, essa edição também será feita através de Keywords. Nesse caso:

- width: largura na unidade *text units*;
- height: altura na unidade *text units*;

Text unit é uma medida que considera o tamanho de um caractere. Por exemplo:



Perceba que apesar da unidade ser a mesma o tamanho de 1 text unit para height é sempre MAIOR que em width.

```
import tkinter as tk

janela = tk.Tk()

janela.title("Cotação de Moedas")

mensagem = tk.Label(text="Sistema de Busca de Cotação de Moedas", fg='white', bg='black', width=50, height=10)
mensagem.pack()

mensagem2 = tk.Label(text=" Seleciona a moeda desejada", height=15, width=70)
mensagem2.pack()

janela.mainloop()
```



width: define a largura da label;  
height: define a altura da label

Até agora, apenas criamos objetos que nos permitem “falar” com o usuário, o contrário, ou seja o usuário não pode interagir com nosso programa.

Para criarmos esses objetos de interação, usaremos o mesmo princípio usado anteriormente.

A diferença está no método que usaremos. Até agora usamos o método `.Label()`. Dessa vez usaremos o método `.Entry()`.

Assim como fizemos anteriormente usaremos o método `.pack()` para inserir na janela.



Alterando os valores de width e height das duas labels existentes melhoramos um pouco a visualização da nossa janela.

Melhorado o layout, precisamos entender algo muito importante. A informação que é escrita pelo usuário não está sendo armazenada em nenhuma variável.

Ao criarmos moeda= tk.Entry() não estamos atribuindo a variável **moeda** o valor digitado pelo usuário. Estamos apenas criando o objeto(caixa em branco).

Em breve vamos entender como coletar essa informação, mas antes, vamos entender um pouco melhor como “arrumar” nossos objetos dentro da nossa janela.

```
import tkinter as tk

janela = tk.Tk()

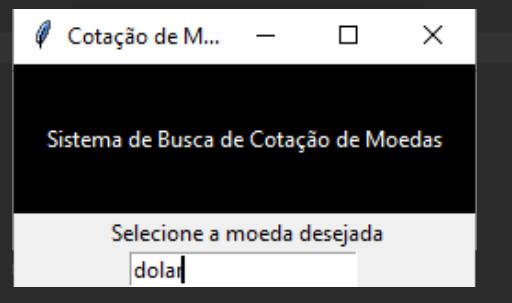
janela.title("Cotação de Moedas")

mensagem = tk.Label(text="Sistema de Busca de Cotação de Moedas", fg='white', bg='black', width=35, height=5)
mensagem.pack()

mensagem2 = tk.Label(text = " Seleccione a moeda desejada")
mensagem2.pack()

moeda = tk.Entry()
moeda.pack()

janela.mainloop()
```



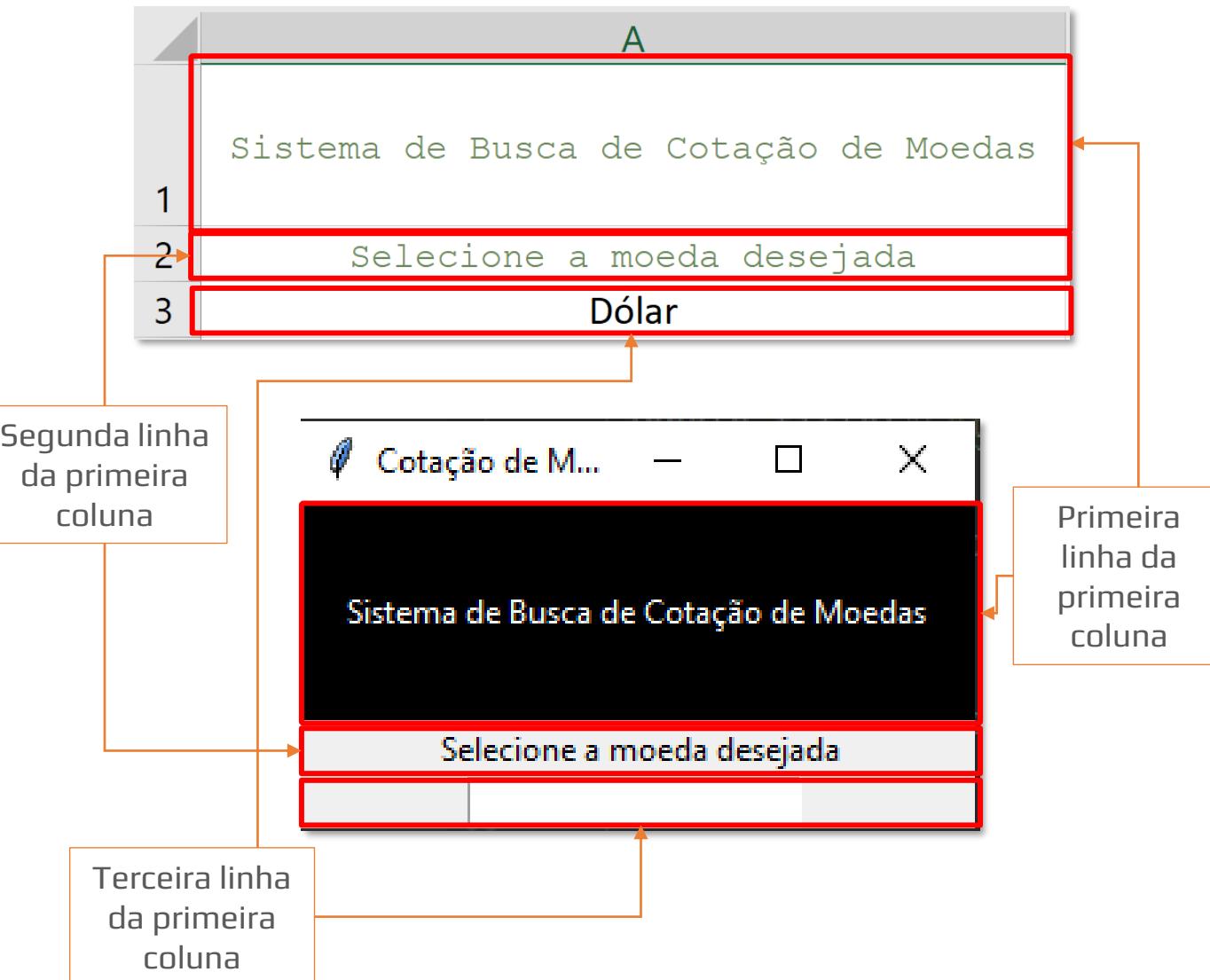
Até agora, sempre que queríamos inserir um objeto dentro da janela usávamos o método `.pack()`.

No entanto, não escolhíamos exatamente onde colocar, ele apenas era colocado na janela (geralmente não de uma forma muito bonita).

Existe outro método chamado `.grid` que nos permite dar “coordenadas” da nossa janela e assim colocar os objetos nos locais mais certos.

Essas coordenadas funcionam com linhas e colunas.

Apenas para exemplo, vamos usar o Excel para entendermos o que está acontecendo.



# Módulo 31 – Interface Gráfico - Tkinter e Criando Sistemas com Python

## Grid - Organizando as Janelas do Tkinter (2/4)

531

Nosso objetivo será alterar a configuração da nossa janela para algo mais parecido com o exemplo abaixo ainda no Excel:

|   | A                                     | B       |
|---|---------------------------------------|---------|
| 1 | Sistema de Busca de Cotação de Moedas |         |
| 2 | Selecione a moeda desejada            | "Caixa" |

Ou seja,

- Sistema de Busca de Cotação de Moedas:
  - Col 1, linha 1;
- Selecione a moeda desejada:
  - Col 1, linha 2;
- “Caixa”:
  - Col 2, linha 2;

Isso que iremos informar ao Python através das Keywords. MAS... LEMBRE-SE, a primeira posição é sempre 0 ou seja, Coluna 1 do Excel = Coluna 0 do Python.

```
import tkinter as tk

janela = tk.Tk()

janela.title("Cotação de Moedas")

mensagem = tk.Label(text="Sistema de Busca de Cotação de Moedas", fg='white', bg='black')
mensagem.grid(row=0, column=0)

mensagem2 = tk.Label(text = " Seleciona a moeda desejada")
mensagem2.grid(row=1, column=0)

moeda = tk.Entry()
moeda.grid(row=1, column=1)

janela.mainloop()
```

Outra funcionalidade oferecida pelo Tkinter é o keyword `columnspan`. Ele nos permite informar se nosso label vai além de uma coluna.

Funciona como uma espécie de “Mesclar células” do Excel.

|   | A                                     | B       |
|---|---------------------------------------|---------|
| 1 | Sistema de Busca de Cotação de Moedas |         |
| 2 | Seleciona a moeda desejada            | "Caixa" |

```
import tkinter as tk

janela = tk.Tk()

janela.title("Cotação de Moedas")

mensagem = tk.Label(text="Sistema de Busca de Cotação de Moedas", fg='white')
mensagem.grid(row=0, column=0, columnspan=2)

mensagem2 = tk.Label(text=" Seleccione a moeda desejada")
mensagem2.grid(row=1, column=0)

moeda = tk.Entry()
moeda.grid(row=1, column=1)

janela.mainloop()
```

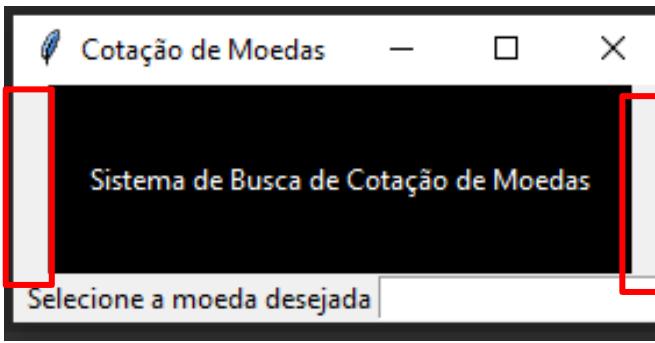
Se olharmos com atenção, veremos que apesar das células terem sido “mescladas”, ainda existe uma “falha” nos cantos da célula.

Por que isso acontece? Como definimos a largura de 35, o Tkinter irá usar 2 colunas mas só até este tamanho.

Se a soma de duas colunas tiver um width de 45 por exemplo, a nossa label de width será posicionada centralizada nos 45 e deixará as bordas “sem preenchimento” ou seja 5 de width a esquerda e 5 de width a direita.

Para garantirmos que esse ajuste sempre será perfeito, usaremos 2 configurações.

- 1) Keyword Sticky;
- 2) Configuração da janela com rowconfigure e columnconfigure.

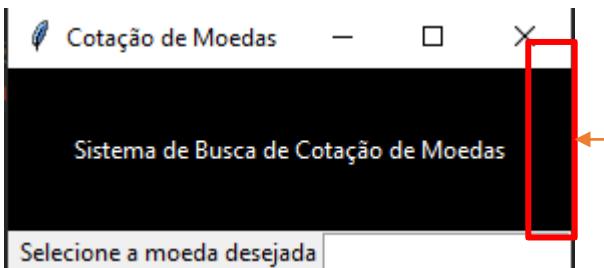


Permitem que a configuração se mantenha mesmo que a tela seja aumentada pelo usuário com o uso do mouse

```
janela.rowconfigure(0, weight=1)
janela.columnconfigure(0, weight=1)

mensagem = tk.Label(text="Sistema de Busca de Cotação de Moedas", fg='white')
mensagem.grid(row=0, column=0, columnspan=2, sticky="NSEW")

mensagem2 = tk.Label(text="Selecionar a moeda desejada")
mensagem2.grid(row=1, column=0)
```



Keyword **sticky**, permite garantir que não haja a sobra.

“NSEW” indica que a correção acontecerá nas 4 direções:

- (N) – Norte
- (S) – Sul
- (E) – Leste
- (W) – Oeste

Com o que temos até agora, temos uma janela que o usuário pode inserir a moeda, mas não consegue por exemplo confirmar o que digitou.

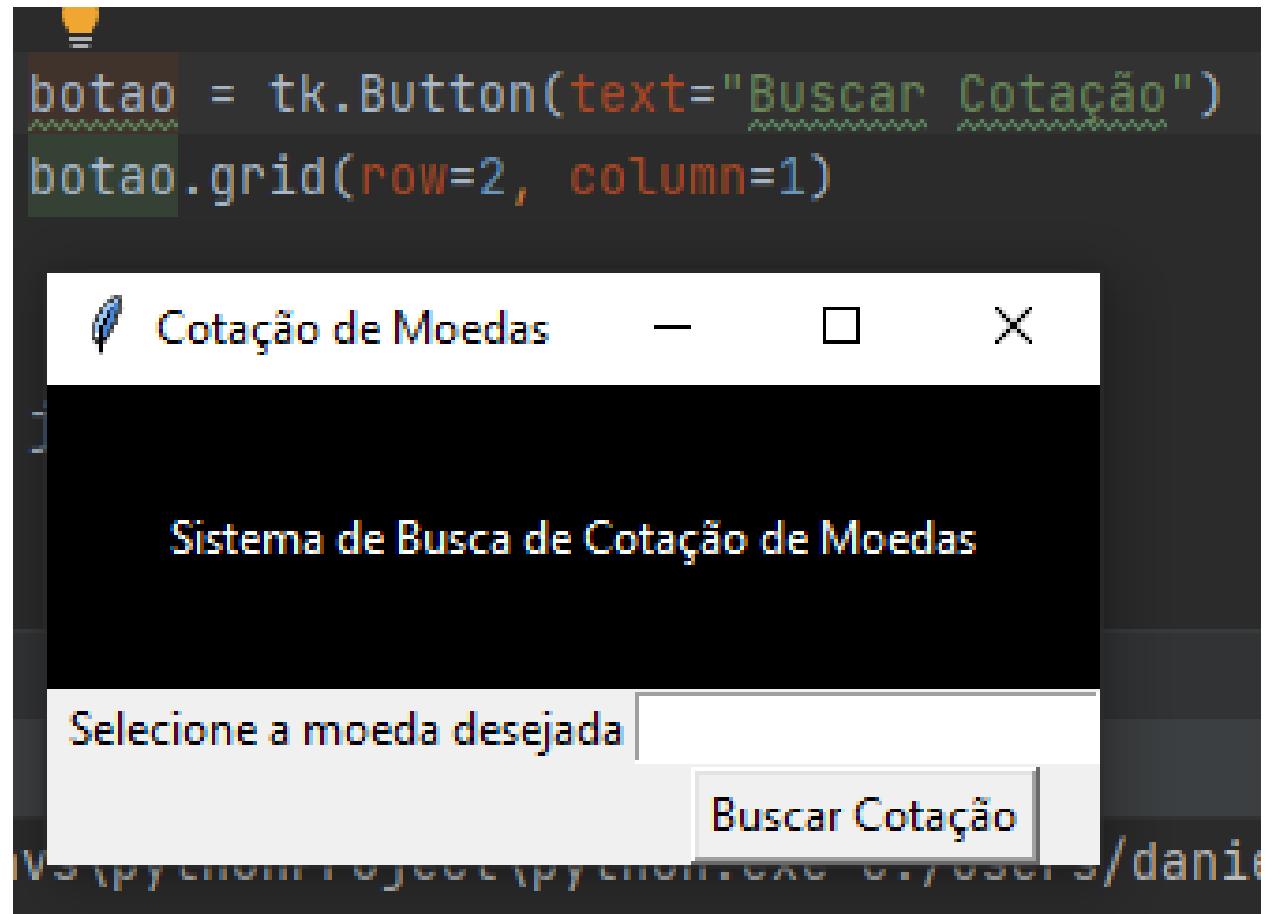
Geralmente, essas interações são geradas via botões. E isso é o que iremos aprender agora ☺.

Para criação de um Botão, usaremos o método `.Button()` conforme apresentado ao lado.

Perceba que criamos um novo objeto chamado botão.

Assim como vimos anteriormente, vamos posicionar esse botão logo abaixo da caixa de entrada:

`(row=2, column=1)`



Por enquanto, esse botão não possui nenhuma função. Se clicarmos nada irá acontecer.

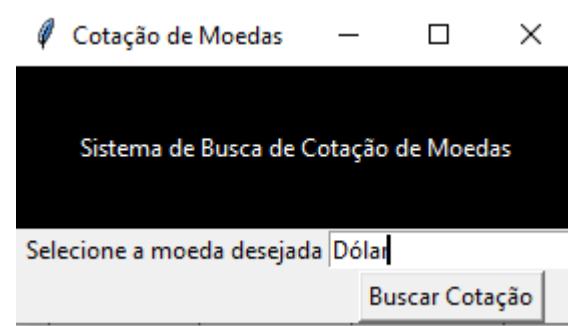
Para que esse botão execute alguma operação, iremos precisar informar ao Python através do keyword **command**.

A operação a ser executada também precisa ser definida.

Por enquanto, vamos apenas criar uma função que printa dentro do nosso terminal do Python o valor digitado pelo usuário.

Perceba que dentro do método **buscar\_cotação()** utilizamos o método **.get()**.

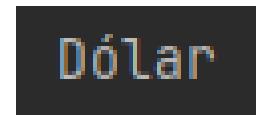
Esse método permite acessar o objeto **moeda** e “pegar” a informação existente nesse objeto.



```
def buscar_cotação():
    print(moeda.get())

botao = tk.Button(text="Buscar Cotação", command=buscar_cotação)
botao.grid(row=2, column=1)
```

The code shows a function definition `def buscar_cotação():` and its use in creating a Tkinter button. An orange arrow points from the `buscar_cotação()` call in the button's `command` parameter to the function definition above it.



Vamos melhorar um pouco mais a experiência do nosso usuário.

Até agora, o método criado apenas coleta a informação digitada pelo usuário e escreve no Console do Python (que não é visto pelo usuário).

A fim de exemplo, vamos criar cotações fixas para 3 moedas distintas:

- Dólar = 5.47;
- Euro = 6.68;
- Bitcoin=20.000;

Nosso objetivo é retornar essas cotações sempre que o usuário digitar o nome da moeda desejada.

```
dicionario_cotacoes = {  
    'Dólar': 5.47,  
    'Euro': 6.68,  
    'Bitcoin': 20000  
}
```

Cotações exemplo geradas dentro do dicionário. Nosso objetivo é acessar esse dicionário sempre que o usuário digitar alguma das 3 moedas listadas.

Já havia sido criado um método `buscar_cotacao`. Vamos editá-lo para que seja possível alcançar o nosso objetivo.

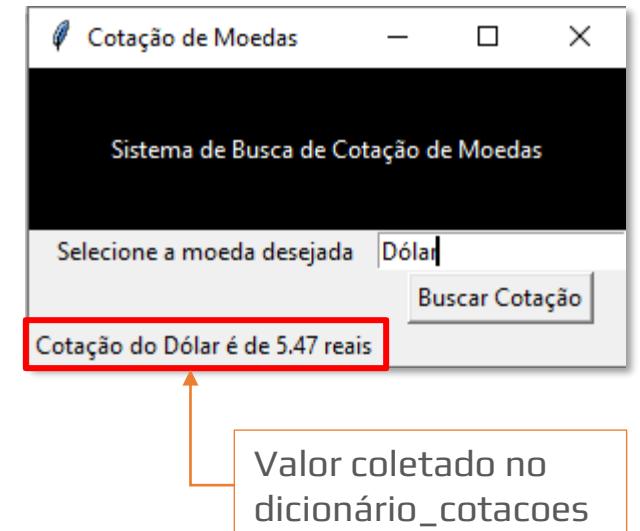
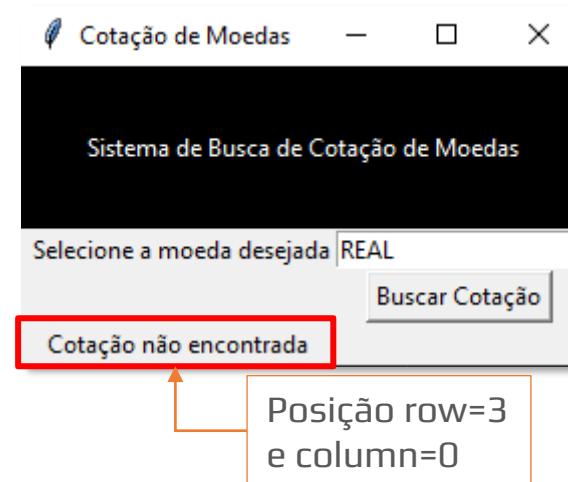
Antes usávamos a estrutura `moeda.get` para printar o que o usuário digitava no campo aberto. Agora, iremos usá-lo como mecanismo de busca no dicionário que acabamos de criar.

Dessa busca existem 2 resultados possíveis.

- 1) A moeda digitada existe e a consulta é bem sucedida;
- 2) A moeda digitada não existe e a consulta não é bem sucedida;

Para qualquer dos resultados, o usuário obterá uma resposta na posição `row=3` e `column=0` ou seja, 4 linha e primeira coluna.

```
def buscar_cotacao():
    moeda_preenchida = moeda.get()
    cotacao_moeda = dicionario_cotacoes.get(moeda_preenchida)
    mensagem_cotacao = tk.Label(text = "Cotação não encontrada")
    mensagem_cotacao.grid(row=3, column=0)
    if cotacao_moeda:
        mensagem_cotacao["text"] = f'Cotação do {moeda_preenchida} é de {cotacao_moeda} reais'
```



Uma forma reduzirmos a possibilidade do usuário cometer erros ao digitar a informação é fornecer aos usuários quais são as opções possíveis.

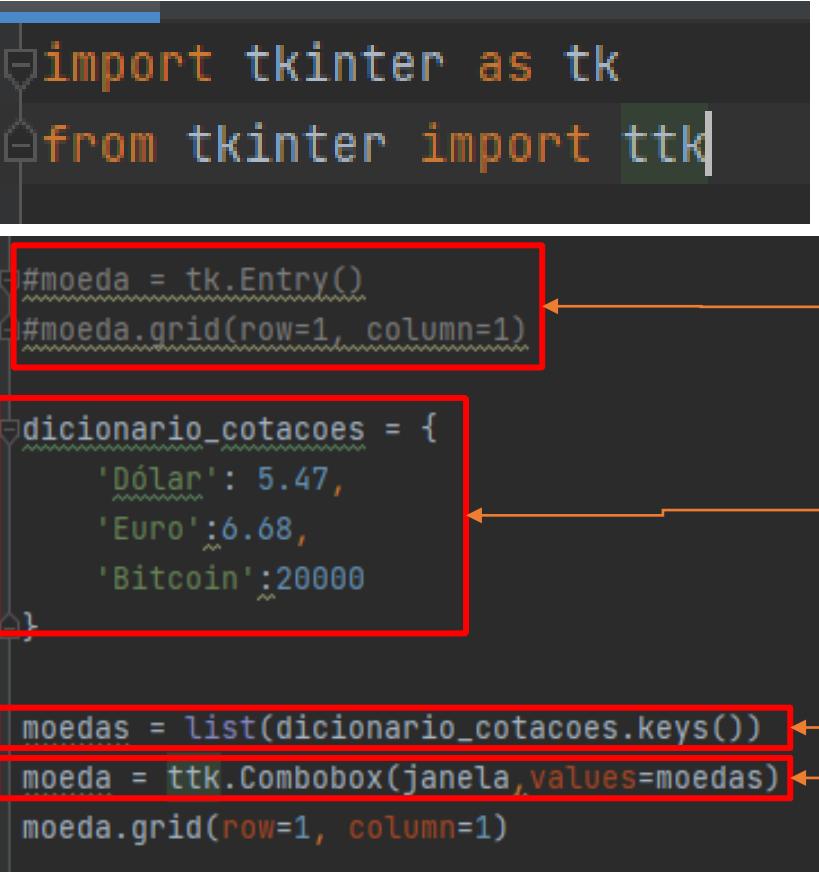
Para isso, vamos precisar criar uma lista suspensa para que o usuário possa selecionar apenas entre as moedas que temos a cotação.

Para isso, iremos importar uma extensão do Tkinter com o comando abaixo:

```
from tkinter import ttk
```

Ao invés de deixarmos um espaço para que o usuário possa digitar, vamos substituir por uma lista suspensa.

Perceba que os valores disponíveis são as chaves do dicionário\_cotações criado anteriormente.



```
import tkinter as tk
from tkinter import ttk

#moeda = tk.Entry()
#moeda.grid(row=1, column=1)

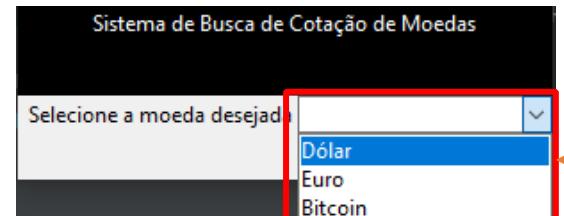
dicionario_cotacoes = {
    'Dólar': 5.47,
    'Euro': 6.68,
    'Bitcoin': 20000
}

moedas = list(dicionario_cotacoes.keys())
moeda = ttk.Combobox(janela, values=moedas)
moeda.grid(row=1, column=1)
```

Retiramos o campo Entry para podermos usar a lista suspensa

Cria uma lista com as chaves do dicionario\_cotacoes

Combobox cria uma lista suspensa no objeto janela e as opções oferecidas são os itens da lista criada com as chaves do dicionario



Essa interface gráfica, não é a mais comum, mas mesmo assim, vamos falar um pouco sobre ela.

Essencialmente o que vamos acrescentar a nossa janela, é uma grande caixa de texto que nos permitiria por exemplo buscar várias cotações ao mesmo tempo e não apenas uma.

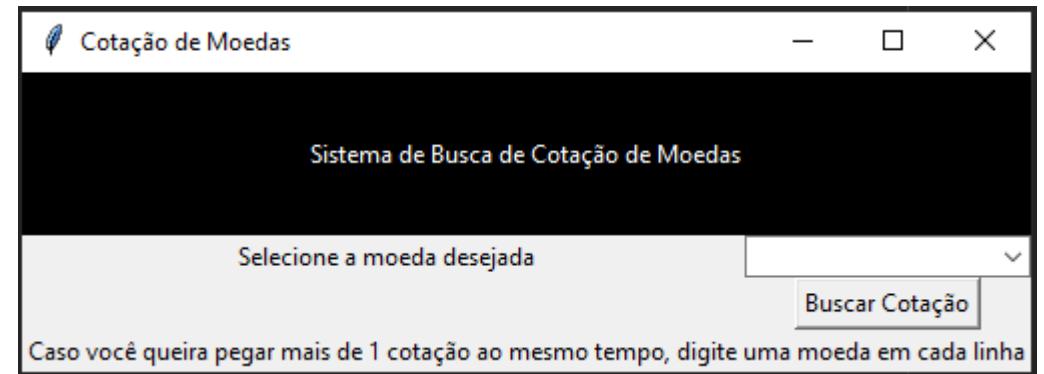
Para isso, vamos precisar criar 3 objetos distintos:

- mensagem3 (Label);
- caixa\_texto (Text);
- Botao\_múltiplas cotações (Button).

O exemplo ao lado apresenta o código do primeiro objeto.

Perceba que utilizamos row=4 e não 3. Por que?  
Se checarmos o método de buscar cotações, vamos perceber que o resultado é plotado na linha de posição 3 .

```
mensagem3=tk.Label(text="Caso você queira pegar mais de 1 cotação ao mesmo tempo, digite uma moeda em cada linha")
mensagem3.grid(row=4, column=0, columnspan=2)
```



O segundo objeto é nossa caixa de texto. Por padrão, ao usarmos `Text`, estamos criando uma caixa de texto muito maior do que gostaríamos, conforme apresentado abaixo:

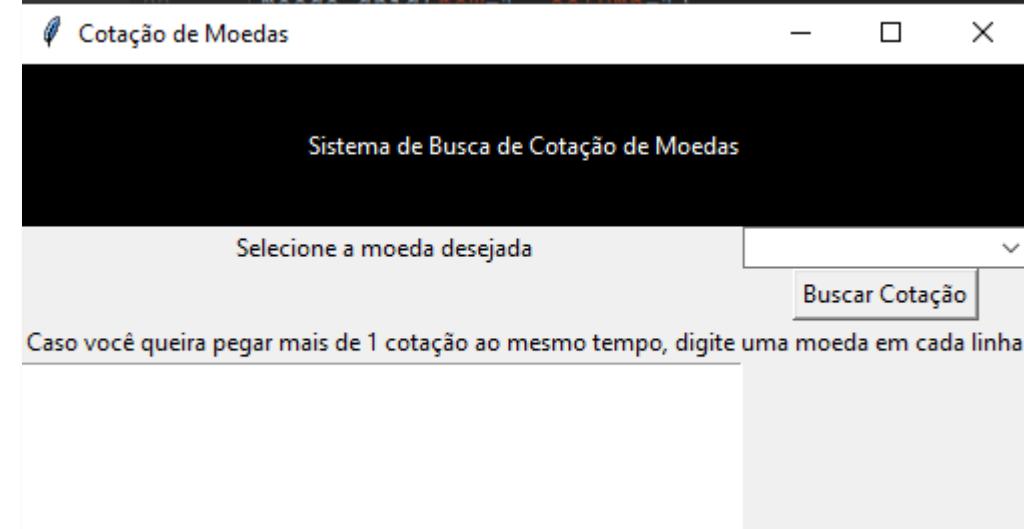


Por isso, é interessante ( pelo menos para nosso caso), restringirmos o tamanho utilizando as Keywords `width` e `height` conforme a imagem ao lado.

A keyword `Sticky`, como já visto anteriormente nos permitirá “garantir que todo o campo seja preenchido.

The code shown is Python code for creating a text input field. It uses the `tk.Text` class from the Tkinter module. An orange arrow points from a callout box labeled "Cria caixa de texto" to the first line of the code, which is `caixa_texto = tk.Text(...)`. The code also includes the `grid` method to position the text input field in the window.

```
caixa_texto = tk.Text(..., width=10, height=5)
caixa_texto.grid(row=5, column=0, sticky='nswe')
```

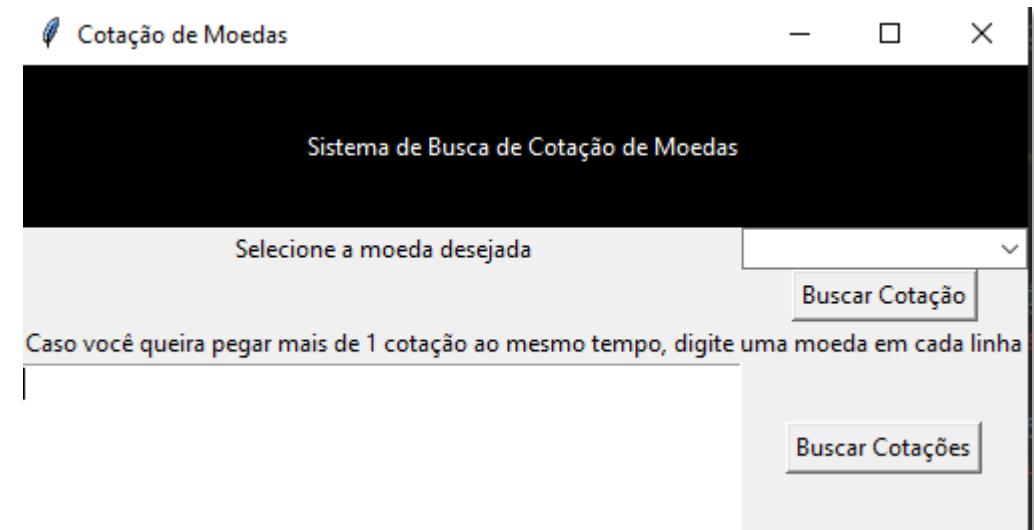


O último objeto que iremos criar, será o botão de buscar cotações assim como fizemos anteriormente, mas dessa vez para a busca via caixa de texto.

Antes de entendermos o método `buscar_cotacoes`, vamos analisar as linhas de código permitem a criação do botão.

```
def buscar_cotacoes():
    pass

botao_multiplascotacoes = tk.Button(text="Buscar Cotações", command=buscar_cotacoes)
botao_multiplascotacoes.grid(row=5, column=1)
```



Vamos entender agora o nosso método `buscar_cotacoes`.

Quando tínhamos apenas 1 valor, usávamos o método `get()` sem nenhum argumento para “pegar” a informação.

Aqui, como temos diversas linhas na nossa caixa de texto precisamos dar 2 inputs:

- Índice da linha inicial;
- Índice da linha final;

Essa linha de código irá nos fornecer todos os textos dentro da caixa de texto.

Vamos precisar separar essa informação em uma lista onde cada item representará uma linha da lista conforme podemos ver na imagem ao lado.

Índice inicial da coleta dos dados

```
def buscar_cotacoes():
    texto = caixa_texto.get("1.0", tk.END)
    lista_moedas = texto.split('\n')
    print(lista_moedas)
```

Coletará até a última linha existente na caixa de texto

Separar cada linha em um item da lista específico

Cotação de Moedas

Sistema de Busca de Cotação de Moedas

Selezione a moeda desejada

Buscar Cotação

Caso você queira pegar mais de 1 cotação ao mesmo tempo, digite uma moeda em cada linha

real  
Dólar  
Euro  
Libra

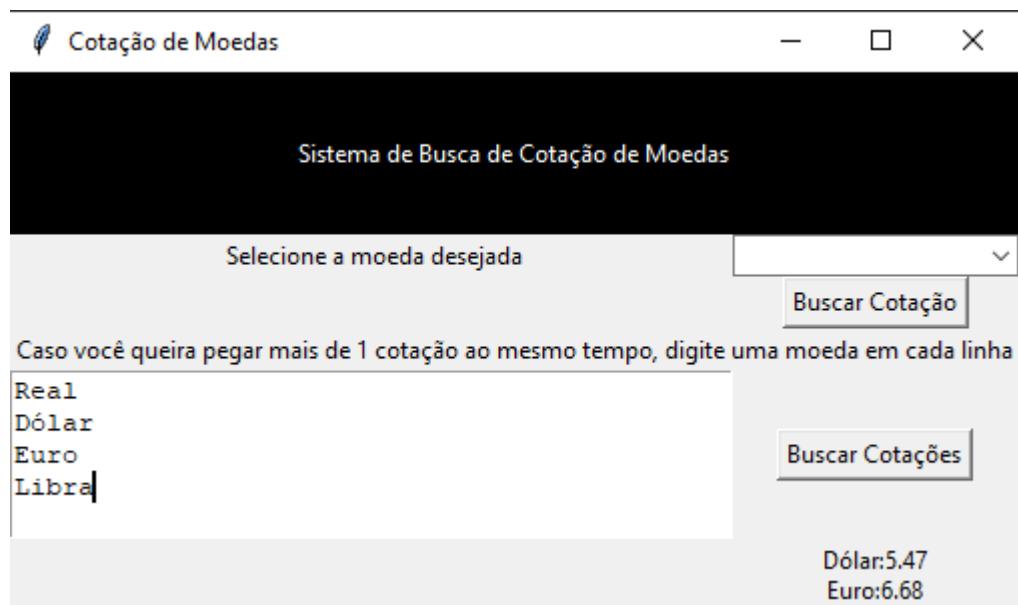
Buscar Cotações

```
['real', 'Dólar', 'Euro', 'Libra', '']
```

O que precisamos agora é validar se as moedas solicitadas na caixa de texto estão na nossa base.

Sempre que a cotação existir, iremos adicionar o valor da cotação em uma lista que aqui, chamamos `mensagem_cotacoes`.

```
def buscar_cotacoes():
    texto = caixa_texto.get("1.0", tk.END)
    lista_moedas = texto.split('\n')
    mensagem_cotacoes = []
    for item in lista_moedas:
        cotacao = dicionario_cotacoes.get(item)
        if cotacao:
            mensagem_cotacoes.append(f'{item}:{cotacao}')
    mensagem4 = tk.Label(text='\n'.join(mensagem_cotacoes))
    mensagem4.grid(row=6, column=1)
```



Criando um novo arquivo .py, vamos aprender uma nova funcionalidade do tkinter. O Checkbox.

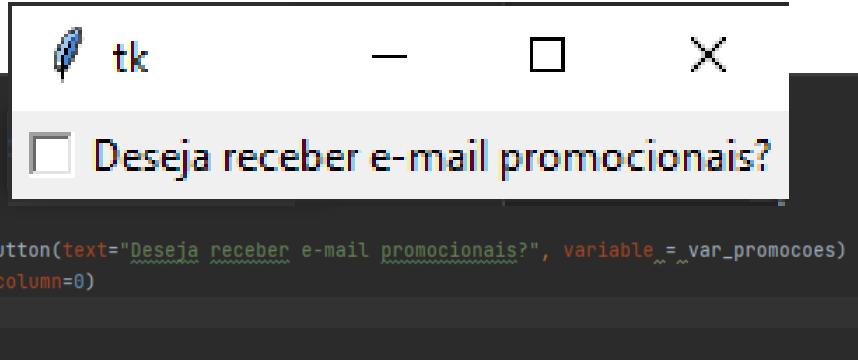
Já sabemos como posicionar e criar objetos dentro da janela.

O que muda com o checkbox, é que ele não é como um botão que ao clicar roda um método. Ele possui estados.

Ele pode estar MARCADO ou NÃO MARCADO. Para cada um desses estados temos um valor associado:

- Marcado : 1;
- Não marcado: 0.

Para armazenarmos essa informação, criaremos a variável `var_promoções` que será uma variável do tipo `IntVar()`.



The screenshot shows a Python code editor on the left and a running Tkinter application window on the right. The code in the editor is:

```
import tkinter as tk
janela = tk.Tk()
var_promocoes = tk.IntVar()
checkbox_promocoes = tk.Checkbutton(text="Deseja receber e-mail promocionais?", variable=var_promocoes)
checkbox_promocoes.grid(row=0, column=0)
janela.mainloop()
```

The window title bar says "tk". The window contains a question "Deseja receber e-mail promocionais?" followed by a checkbox. The checkbox is currently unchecked.

Por enquanto, nada acontece no nosso checkbox.  
Marcado ou desmarcado, ele gera o mesmo resultado.

Vamos criar um botão que nos permita coletar a informação se o checkbox está marcado ou não.

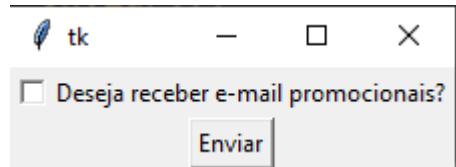
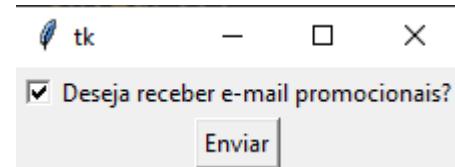
Para isso, além da criação do objeto, iremos precisar criar um método que capture esse valor.

Em um primeiro momento iremos apenas “printar essa informação no console do Python e no próximo slide entramos mais a fundo.

```
def enviar():
    print(var_promocoes.get())

botao_enviar = tk.Button(text="Enviar", command = enviar)
botao_enviar.grid(row=1, column=0)

janela.mainloop()
```



1 Valor 1 indica que o checkbox está MARCADO  
0 Valor 0 indica que o checkbox NÃO ESTÁ MARCADO

# Módulo 31 – Interface Gráfico - Tkinter e Criando Sistemas com Python

## Checkbox (Checkbutton) – (3/3)

546

Vamos agora acrescentar outro checkbox que permita o usuário de autorizar os termos de uso e políticas de privacidade.

Além disso, iremos adicionar um IF no nosso método `enviar()` que nos permitirá “traduzir” os valores 0 e 1 resultantes do checkbox.

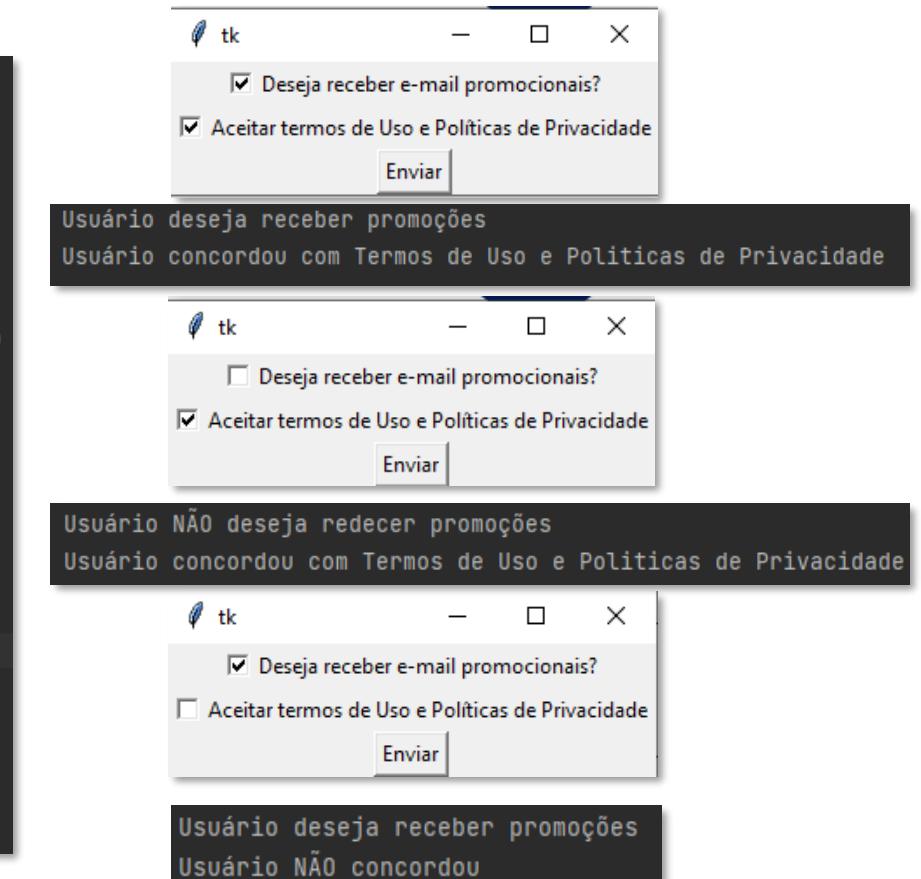
```
janela = tk.Tk()

var_promocoes = tk.IntVar()
checkbox_promocoes = tk.Checkbutton(text="Deseja receber e-mail promocionais?", variable=var_promocoes)
checkbox_promocoes.grid(row=0, column=0)

var_politicas = tk.IntVar()
checkbox_politicas = tk.Checkbutton(text="Aceitar termos de Uso e Políticas de Privacidade", variable=var_politicas)
checkbox_politicas.grid(row=1, column=0)

def enviar():
    if var_promocoes.get():
        print('Usuário deseja receber promoções')
    else:
        print('Usuário NÃO deseja receber promoções')
    if var_politicas.get():
        print('Usuário concordou com Termos de Uso e Políticas de Privacidade')
    else:
        print('Usuário NÃO concordou')

botao_enviar = tk.Button(text="Enviar", command=enviar)
botao_enviar.grid(row=2, column=0)
```



Temos um caso muito parecido com o caso anterior. Essencialmente a diferença está que o checkbox é individualizado. Ou seja, todos os checkbox podem ser marcados ou não.

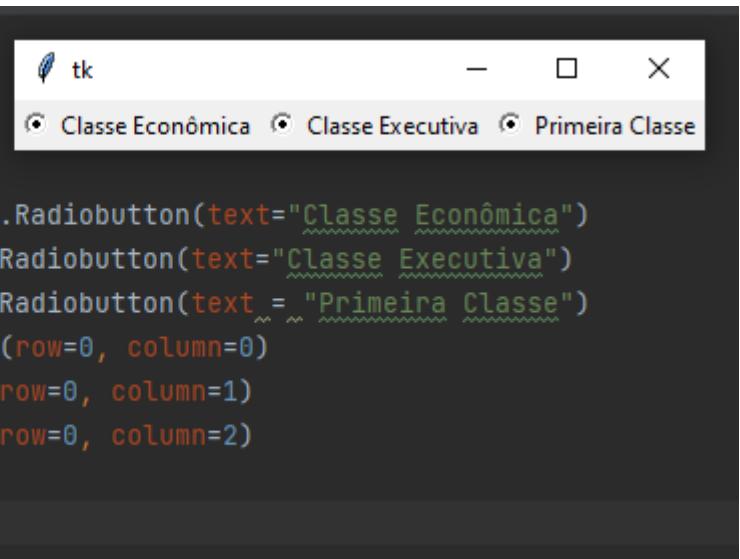
O radiobutton só permite entre as opções 1 escolha.

Para criar nossos radiobuttons, usaremos essencialmente os conceitos apresentados anteriormente.

No entanto, perceba que ao criarmos utilizado o código ao lado, todas as opções estão marcadas e ao tentarmos clicar para desmarcá-las, nada acontece.

Isso acontece, pois ainda não há relação entre eles.

Para isso, vamos precisar criar algumas variáveis auxiliares que nos permitam criar esse relacionamento.



```
import tkinter as tk

janela = tk.Tk()

botao_classeeconomica = tk.Radiobutton(text="Classe Econômica")
botao_classeexecutiva = tk.Radiobutton(text="Classe Executiva")
botao_primeiraclasse = tk.Radiobutton(text="Primeira Classe")

botao_classeeconomica.grid(row=0, column=0)
botao_classeexecutiva.grid(row=0, column=1)
botao_primeiraclasse.grid(row=0, column=2)

tk.mainloop()
```

No caso do radiobutton, iremos usar uma mesma variável para TODAS as opções. Isso permitirá o relacionamento entre as opções.

No caso do Checkbox, utilizamos uma variável do tipo IntVar, que nos fornecia os valores 0 ou 1.

Aqui, usaremos o mesmo conceito, mas não para valores 0 ou 1 e sim um StringVar, que pode ter 3 valores de String distintos:

- Classe Econômica;
- Classe Executiva;
- Primeira Classe.

Essas informações, serão atribuídas a partir do parâmetro value indicado em cada um dos objetos.

No entanto, ainda nos falta algo que colete a informação informada. Para isso, iremos criar na próxima página um botão de enviar.

```
import tkinter as tk

janela = tk.Tk()

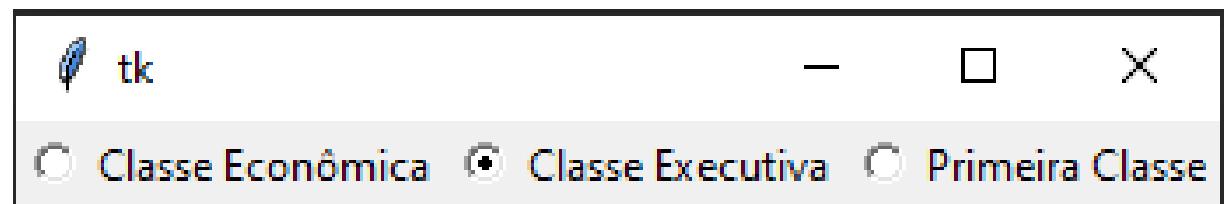
var_aviao = tk.StringVar()

botao_classeeconomica = tk.Radiobutton(text="Classe Econômica", variable=var_aviao, value="Classe Econômica")
botao_classeexecutiva = tk.Radiobutton(text="Classe Executiva", variable=var_aviao, value="Classe Executiva")
botao_primeiraclasse = tk.Radiobutton(text="Primeira Classe", variable=var_aviao, value="Primeira Classe")

botao_classeeconomica.grid(row=0, column=0)
botao_classeexecutiva.grid(row=0, column=1)
botao_primeiraclasse.grid(row=0, column=2)

tk.mainloop()
```

Parâmetro value irá fornecer informações para a variável var\_aviao



No caso do radiobutton, iremos usar uma mesma variável para TODAS as opções. Isso permitirá o relacionamento entre as opções.

No caso do Checkbox, utilizamos uma variável do tipo IntVar, que nos fornecia os valores 0 ou 1.

Aqui, usaremos o mesmo conceito, mas não para valores 0 ou 1 e sim um StringVar, que pode ter 3 valores de String distintos:

- Classe Econômica;
- Classe Executiva;
- Primeira Classe.

Essas informações, serão atribuídas a partir do parâmetro **value** indicado em cada um dos objetos.

No entanto, ainda nos falta algo que colete a informação informada. Para isso, iremos criar na próxima página um botão de enviar.

```
import tkinter as tk

janela = tk.Tk()

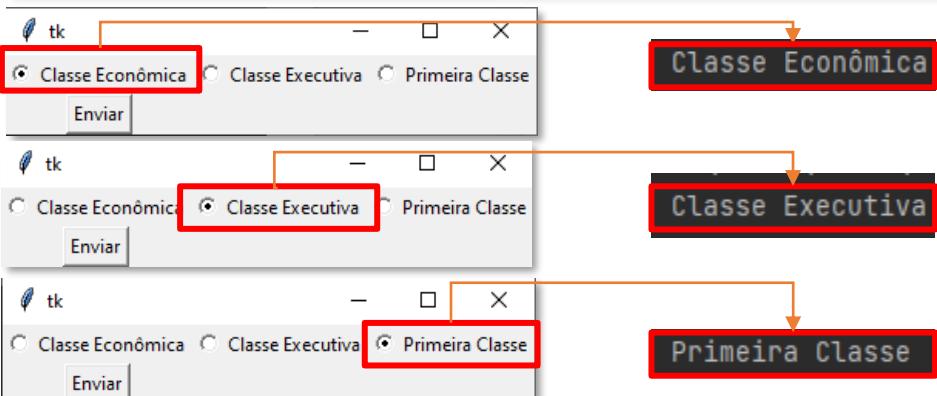
var_aviao = tk.StringVar(value="Nenhuma Opcão Marcada")

botao_classeeconomica = tk.Radiobutton(text="Classe Econômica", variable=var_aviao, value="Classe Econômica")
botao_classeexecutiva = tk.Radiobutton(text="Classe Executiva", variable=var_aviao, value="Classe Executiva")
botao_primeiraclasse = tk.Radiobutton(text="Primeira Classe", variable=var_aviao, value="Primeira Classe")
botao_classeeconomica.grid(row=0, column=0)
botao_classeexecutiva.grid(row=0, column=1)
botao_primeiraclasse.grid(row=0, column=2)

def enviar():
    print(var_aviao.get())

botao_enviar = tk.Button(text = "Enviar", command=enviar)
botao_enviar.grid(row=1, column=0)

tk.mainloop()
```



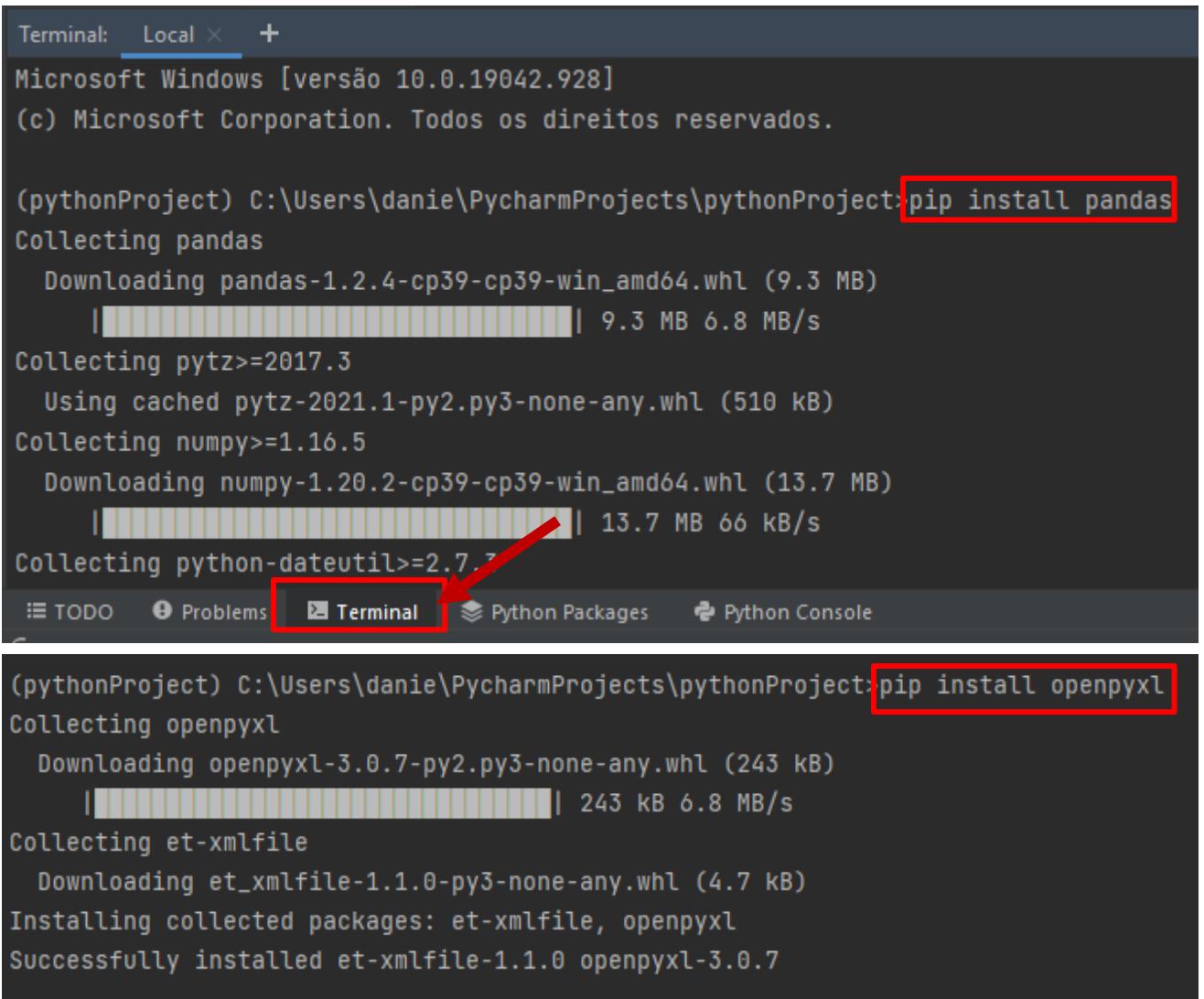
Vamos rever agora um tipo de interação que já vimos anteriormente que é como solicitar ao usuário que ele indique os arquivos que gostaria de utilizar.

Esse arquivo importado pode ser um arquivo Excel por exemplo. Por isso, vamos instalar o Pandas que nos permitirá importar essa base após a seleção do usuário.

Lembrando que agora estamos no Pycharm e portanto, precisamos instalá-lo.

Para isso, vá no TERMINAL e digite `pip install pandas`, conforme apresentado ao lado.

No entanto, no Pycharm o Pandas precisa de outra biblioteca para que possa funcionar. Esse biblioteca é chamada `openpyxl`



The screenshot shows a PyCharm interface with a terminal window open. The terminal title bar says "Terminal: Local". The terminal content shows two pip installation commands:

```
Terminal: Local +  
Microsoft Windows [versão 10.0.19042.928]  
(c) Microsoft Corporation. Todos os direitos reservados.  
  
(pythonProject) C:\Users\danie\PycharmProjects\pythonProject>pip install pandas  
Collecting pandas  
  Downloading pandas-1.2.4-cp39-cp39-win_amd64.whl (9.3 MB)  
   |████████████████████████████████| 9.3 MB 6.8 MB/s  
Collecting pytz>=2017.3  
  Using cached pytz-2021.1-py2.py3-none-any.whl (510 kB)  
Collecting numpy>=1.16.5  
  Downloading numpy-1.20.2-cp39-cp39-win_amd64.whl (13.7 MB)  
   |████████████████████████████████| 13.7 MB 66 kB/s  
Collecting python-dateutil>=2.7.7  
  
...  
  
(pythonProject) C:\Users\danie\PycharmProjects\pythonProject>pip install openpyxl  
Collecting openpyxl  
  Downloading openpyxl-3.0.7-py2.py3-none-any.whl (243 kB)  
   |████████████████████████████████| 243 kB 6.8 MB/s  
Collecting et-xmlfile  
  Downloading et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)  
Installing collected packages: et-xmlfile, openpyxl  
Successfully installed et-xmlfile-1.1.0 openpyxl-3.0.7
```

A red box highlights the command `pip install pandas`. A red arrow points from the bottom of the first terminal window to the top of the second terminal window, indicating the continuation of the session.

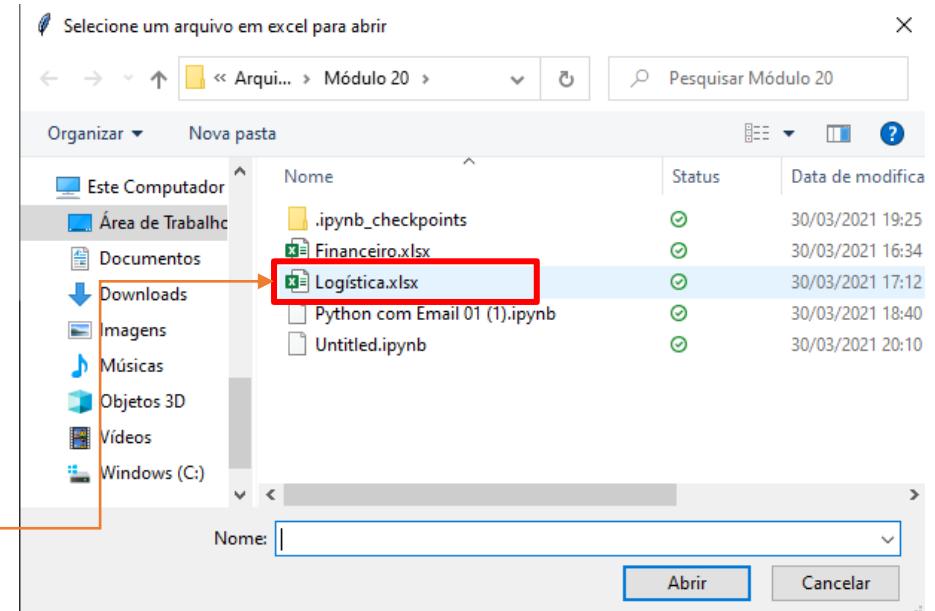
# Módulo 31 – Interface Gráfico - Tkinter e Criando Sistemas com Python

## Pedir para o Usuário Selecionar Arquivo (2/2)

550

Feita as instalações, vamos para nosso código.

```
from tkinter.filedialog import askopenfilename  
import pandas as pd  
  
caminho_arquivo = askopenfilename(title = "Selecione um arquivo em excel para abrir")  
  
df = pd.read_excel(caminho_arquivo)  
  
print(df)
```



|    | Data       | Juros de Dívida |
|----|------------|-----------------|
| 0  | 2020-01-01 | 37255           |
| 1  | 2020-02-01 | 37548           |
| 2  | 2020-03-01 | 25400           |
| 3  | 2020-04-01 | 11455           |
| 4  | 2020-05-01 | 41126           |
| 5  | 2020-06-01 | 30901           |
| 6  | 2020-07-01 | 16669           |
| 7  | 2020-08-01 | 19907           |
| 8  | 2020-09-01 | 20736           |
| 9  | 2020-10-01 | 12701           |
| 10 | 2020-11-01 | 44532           |

**OBRI  
GADO!**

