



“Almirante Tamandaré”
São Bernardo do Campo
sbc.sp.senai.br

Fundamentos de *Big Data & Data Analytics* com Python

Prof. Vinícius B. Suterio

Serviço Nacional de Aprendizagem Industrial
Departamento Regional de São Paulo
Área: Tecnologia da Informação

vinicius.suterio@sp.senai.br

2o. Semestre
2023

1.1.1. FIC: *Big Data & Data Analytics* com Python

Objetivo:

- O Curso de Aperfeiçoamento Profissional Fundamentos de *Big Data* e *Data Analytics* com Python tem por objetivo o **desenvolvimento de competências relativas à extração e transformação de dados utilizando softwares específicos**, aplicando técnicas de *machine learning* e gerando informações para **tomada de decisão** seguindo procedimentos e normas com análise estatística.

Requisitos:

- Ter concluído o Ensino Médio;
- Ter, no mínimo, 16 anos;
- Ter conhecimento em linguagem de programação.

Perfil da Qualificação Profissional:

- Solução de problemas com extração e transformação de dados por meio de softwares específicos, usando técnicas de *machine learning* e estatística para auxílio em tomadas de decisões.

1.2.1. Organização curricular: *Big Data & Data Analytics*

LEGISLAÇÃO	UNIDADES CURRICULARES	CARGA HORÁRIA TOTAL (HORAS)
Lei Federal nº 9.394/96 e Decreto Federal nº 5.154/04	Fundamentos de Big Data e Data Analytics com Python	60
	Carga Horária Total	60

1.3.1. Ementário: *Big Data & Data Analytics*

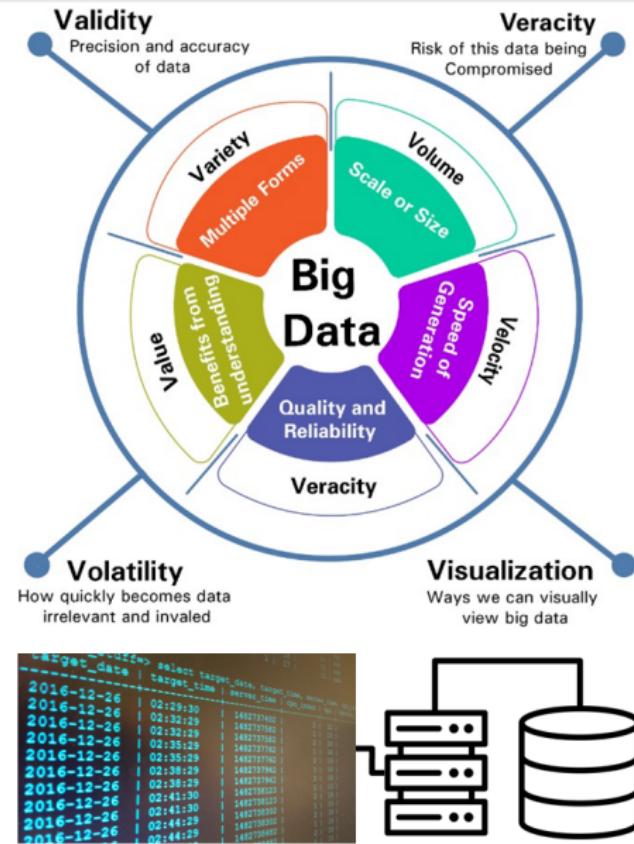
Big Data & Data Analytics - 60h:

- *Big Data:*

- Histórico e definição;
- Aplicações;
- Características dos 5Vs:
 - Volume;
 - Velocidade;
 - Valor;
 - Variedade;
 - Veracidade.
- Etapas dos processos

- *Banco de dados:*

- Definição;
- Características;
- Tipos:
 - SQL;
 - NoSQL.



1.3.1.1. Ementário: *Big Data & Data Analytics*

Big Data & Data Analytics - 60h:

- Comandos em SQL, Spark e PySpark;

- *Machine Learning:*

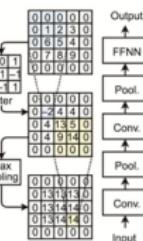
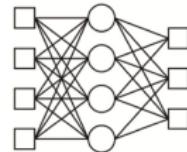
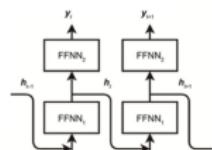
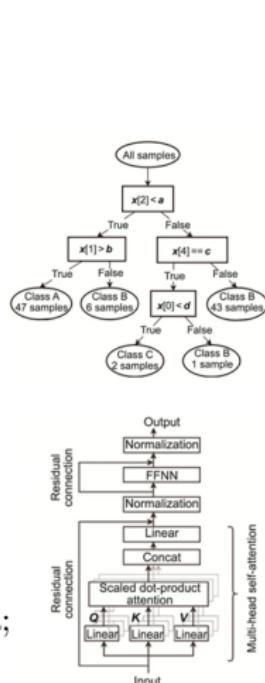
- Regressão linear;
- Árvore de decisão.

- *Dashboards:*

- PowerBI e Tableau.

- *Matemática Estatística:*

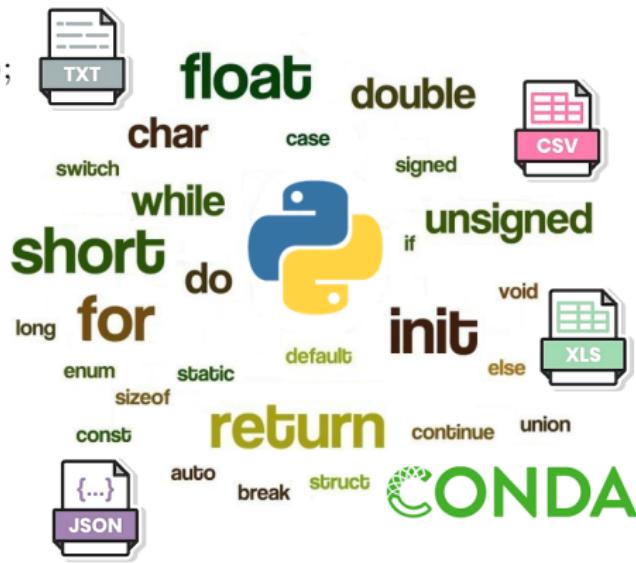
- Definições;
- Funções interativas:
 - Média / moda;
 - Variância;
 - Desvio padrão.
- Estatística descritiva;
- Tabulação de dados;
- Gráficos: linha, coluna, dispersão, etc.



1.3.1.2. Ementário: *Big Data & Data Analytics*

Big Data & Data Analytics - 60h:

- Diretórios e arquivos:
 - *Prompt* de comando e Estrutura;
 - Dados simples (txt, csv, excel e json);
- Linguagem de programação - Python:
 - Histórico;
 - Compiladores/*interpretadores**
 - Bibliotecas: Pip e Conda;
 - Atualizações e repositórios;
- Variáveis e parâmetros:
 - Global/local;
 - *Keywords*, *VarArgs*, *Init* e *Return*;
 - *Self*, herança e dependência;
 - Dados: tipos e classes;
 - Fluxo de controle:
 - *if/else*; *for*; *while*; *break*; *continue*.



1.3.1.3. Ementário: *Big Data & Data Analytics*

Big Data & Data Analytics - 60h:

- Biblioteca Numpy:

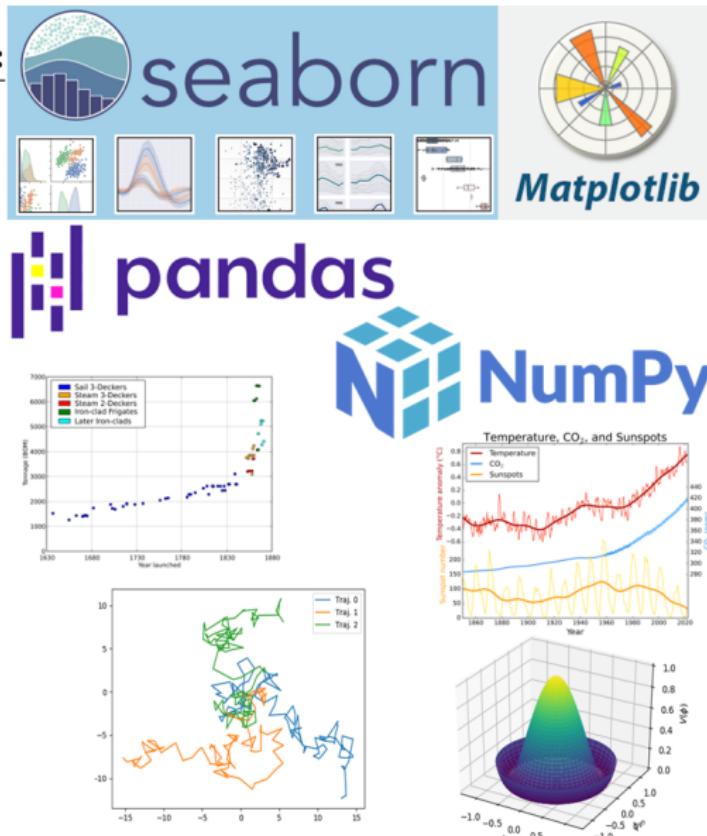
- Comandos;
- Funções;
- Matrizes/equações;
- Gráficos.

- Bibliotecas: Pandas e Seaborn:

- Tabelas/comandos;
- Dados: *import/export*;
- *Dataframes*;
- Dicionários, tuplas e filtros;

- Biblioteca Matplotlib:

- Plotagem;
- Exportação de gráficos;
- KPIs.



1.3.1.7. Ementário: *Big Data & Data Analytics*

Procedimentos de ensino:

- Aulas Teóricas:
 - Lousa/Multimídia;
 - Realização de algoritmos (exemplos e fixação);
 - Elaboração/Análise de projetos utilizando análise de dados com Python.
- Aulas Práticas:
 - Desenvolvimento de lógica utilizando linguagem Python;
 - Familiarizar-se com a linguagem e comandos básicos na modelagem de dados;
 - Manipulação de informações nos bancos de dados (pré-processamento);
 - Obtenção de *features* por meio de análise dos dados utilizando estatística;
 - Técnicas computacionais para avaliação de *features* - criação de indicadores;

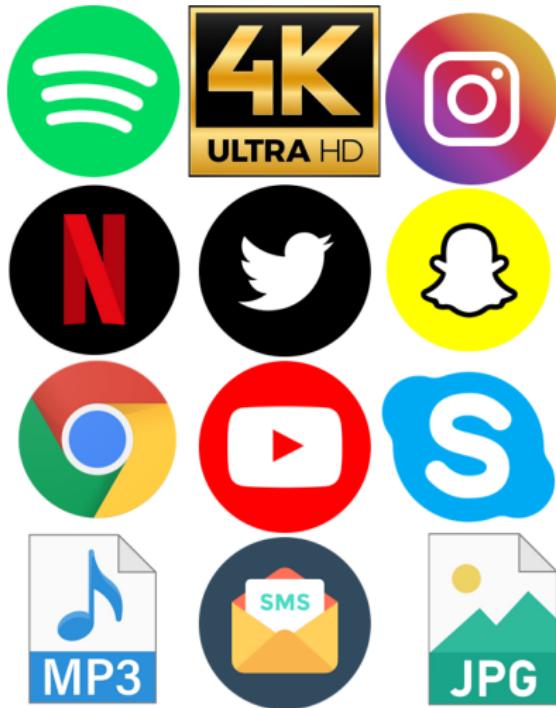
2.1.1. Big Data & Data Analytics

- **Big Data** consiste na área do conhecimento que realiza o **tratamento, análise e obtenção de informações** a partir de um **conjunto de dados** consideravelmente grande;
- Atualmente cientistas dizem que os **dados - data** - são tão importantes quanto a criação/desenvolvimento de algoritmos;
- Aproximadamente **2,5 quintilhões de bytes** ($2,5 \text{ Ebyte} = 2,5 \times 10^{18}$) são criados diariamente e 90% dos dados mundiais foram criados apenas nos **últimos 2 anos**;
- Estima-se que a partir de 2025 o **fornecimento global de dados** atingirá **175 Zbytes** (175 trilhões de Gbytes) **anualmente**;
- Basicamente *data* consiste em **todo tipo de informação** que **se deseja tratar/analisar** que pode ser uni, bi ou multidimensional.



2.1.1.1. Big Data & Data Analytics

- Para que tenhamos uma pequena noção:
 - Música MP3 ≈ 1 a 2,4 MB/min;
 - Foto JPEG ≈ 8 a 10 MB/foto;
 - Vídeo 4K ≈ 350 MB/min;
 - DVD ≈ 8,5 GB (141 horas de música);
 - HD ≈ 1 TB (84 horas de vídeo 4K).
- O quanto de *data* é gerado por minuto?
 - 473.400 *tweets* enviados;
 - 2.083.333 *Snapchats* compartilhados;
 - 97.222 horas de Netflix;
 - 12.986.111 mensagens de texto enviada;
 - 49.380 Posts de *Instagram*;
 - 176.220 ligações no *Skype*;
 - 750.000 músicas transmitidas no *Spotify*;
 - 3.877.140 Pesquisas no Google;
 - 4.333.560 Vídeos assistidos no YouTube.

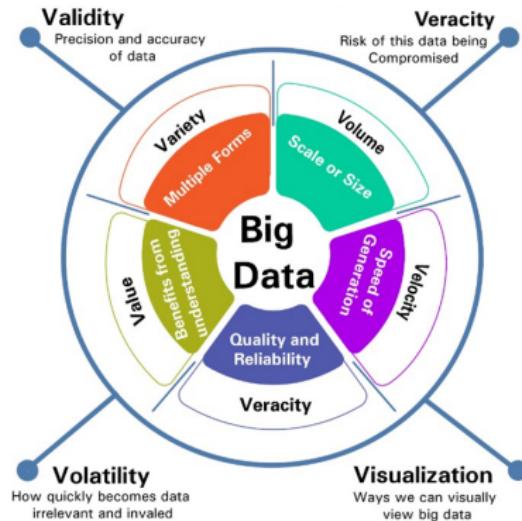


2.1.1.2. Big Data & Data Analytics

- A “explosão” do *Big Data* provavelmente continuará de forma exponencial nos próximos anos e com aproximadamente 50bi de dispositivos conectados por *IoT* gerando “dados”, onde iremos parar?

- Os V's do Big Data:

- **Volume:** Quantidade de dados produzidos/obtidos - podem ou não ter baixa densidade de informação (**relevância**);
- **Velocidade:** O quão rápido os dados são produzidos, coletados e até modificados (tomada de decisão - *real time*);
- **Valor:** O quão relevante são os dados. Mensurar o **grau de significância para o processo**. Quanto irá agregar em valor?
- **Variedade:** Diferentes formatos - alfanuméricos, áudios, vídeos, sensores, etc;
- **Veracidade:** Nível de confiabilidade dos dados. São completos, precisos e reais?



2.1.1.3. Big Data & Data Analytics

- A maior parte dos dados são criados digitalmente em uma variedade de tipos, volumes extraordinários e movem-se em velocidades surpreendentes nas mais diversas aplicações:
 - Desenvolvimento de produtos;
 - Manutenção preditiva;
 - Machine Learning*;
 - Otimização de processos - performance;
 - Eficiência operacional - estratégia, etc;
- Um dos aplicativos de *Big Data* mais utilizado é o de navegação GPS - *Waze* com 90mi de usuários mensais;
- Os primeiros dispositivos GPS dependiam de estatística, mapas e coordenadas. O Waze, devido sua grande quantidade de dados em tempo real, se ajusta dinamicamente com relação ao mapa, alertas, trânsito, etc.



2.2.1. O que utilizaremos em nosso curso?

Software **ANACONDA** consiste em um conjunto de “bibliotecas” com foco nas linguagens de programação **Python** e R que simplificam o gerenciamento e implantação de pacotes.

- Aplicação principalmente em *Data Science: machine learning*, processamento de dados em larga escala, análise preditiva, etc.

O notebook **JUPYTER** consiste em um ambiente computacional web para criação de documentos da plataforma Jupyter. Tal documento é estruturado no formato JSON.

- Contém uma lista ordenada de células, entrada/saída, pode conter código (algoritmo), texto (Markdown), gráficos, etc.

O **VSCODE** é um editor de código-fonte usado para diversas linguagens de programação e ≠'s Sist. Op. Possui recursos de depuração, destaque de sintaxe, refatoração de código, etc.



2.2.2. Download e instalação do ANACONDA

Link do software **ANACONDA**: <https://www.anaconda.com/download>

The screenshot shows a web browser window displaying the Anaconda Distribution download page. The URL in the address bar is [anaconda.com/download](https://www.anaconda.com/download). The page has a dark background with green text and highlights. At the top, there's a navigation bar with links for ANACONDA, Enterprise, Pricing, Resources, and About, along with a Contact Sales button. Below the navigation, the text "Anaconda Distribution" is displayed in green. The main heading is "Free Download" in large white font. A subtext below it says "Everything you need to get started in data science on your workstation." To the left of the subtext is a bulleted list of benefits with green checkmarks: "Free distribution install", "Thousands of the most fundamental DS, AI, and ML packages", "Manage packages and environments from desktop application", and "Deploy across hardware and software platforms". At the bottom of the page, there are two prominent green buttons: "Code in the Cloud" and "Download". Below these buttons, there's a link to "Get Additional Installers" followed by icons for Windows, Mac, and Linux.

2.2.3. Download e instalação do VSCode

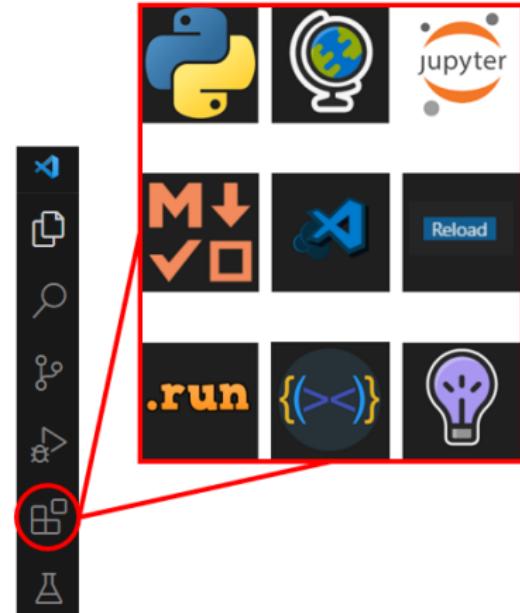
Link do software **VSCode**: <https://code.visualstudio.com/download#>

The screenshot shows the official Visual Studio Code download page at code.visualstudio.com/download#. The page features a dark header with the Visual Studio Code logo and navigation links for Docs, Updates, Blog, API, Extensions, FAQ, Learn, and a prominent blue 'Download' button. A message at the top indicates 'Version 1.83' is available with new features and fixes from September. Below the header, the main title 'Download Visual Studio Code' is displayed, followed by the subtitle 'Free and built on open source. Integrated Git, debugging and extensions.' Three large download buttons are shown: 'Windows' (for Windows 10, 11), 'Linux' (with options for .deb, .rpm, and snap packages), and 'Mac' (for macOS 10.15+). Each button includes a download icon and a list of supported architectures (x64, x86, Arm64, Arm32).

Platform	Architectures
Windows	x64, x86, Arm64, Arm32
Linux	.deb (x64, Arm32, Arm64), .rpm (x64, Arm32, Arm64), Snap (Snap Store)
Mac	.zip (Intel chip, Apple silicon, Universal), CLI (Intel chip, Apple silicon)

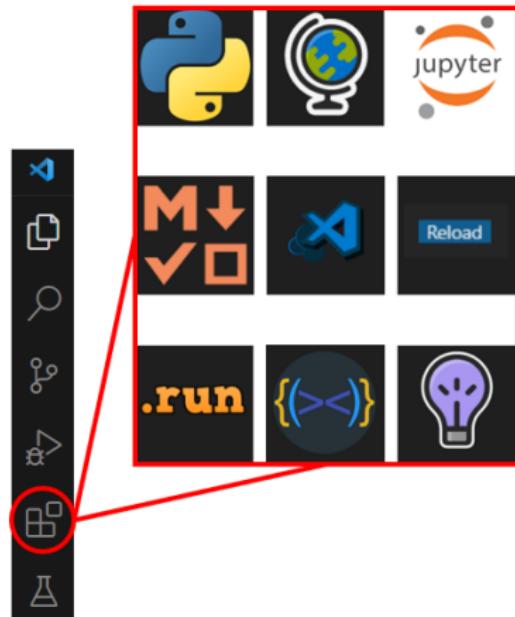
2.2.4. Configurando extensões do VSCode

- *Python, Microsoft*: Permite acesso a diversas funcionalidades para os programadores Phyton usando VSCode - selecionar interpretador, *helper*, etc (essencial);
- *Portuguese (Brazil) Language Pack for Visual Studio, Microsoft*: Pacote português;
- *Jupyter, Microsoft*: Garante a integração com o jupyter notebook e fornece ferramentas que facilitarão o manuseio de dados;
- *Markdown Checkboxes, Matt Bierner*: criação dos checkboxes e interface com formatações que garantem a legibilidade;
- *VSCode-Icons, VSCode Icons Team*: Construção de uma interface mais agradável com um layout mais intuitivo facilitando a busca de arquivos na barra de diretório;



2.2.4.1. Configurando extensões do VSCode

- **Reload:** Facilita o processo de reinicialização do VSCode quando se faz necessário atualizar as configurações;
- **Code Runner, Jun Han:** Execução de trechos de algoritmos em ≠'s linguagens de programação - arquivos com extensão .py;
- **Rainbow Brackets, Mhammed Talhaouy:** formatação intuitiva de parênteses para melhor interpretação do algoritmo;
- **IntelliCode, Microsoft:** IA que sugere o preenchimento das instruções utilizadas (complementa a digitação do programador).

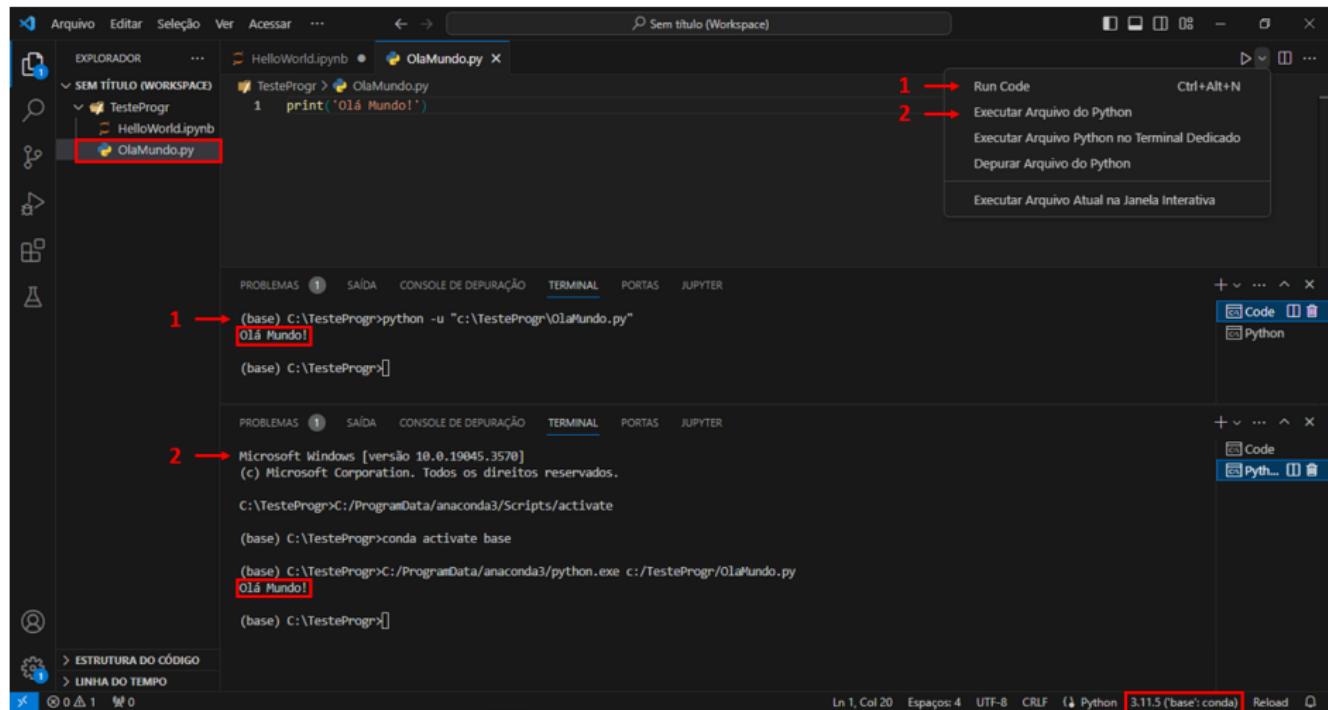


♥ IMPORTANTE ♥

- **Run Code:** Configurar o *Code-runner* → habilitar *Code-runner: Run In Terminal* ✓.
- **Executar Arquivo do Python:** Mudar de *PowerShell* → *Command Prompt*.
- **Configurações:** → habilitar *Jupyter>Interactive Window>Text Editor:Execute Selection* ✓;

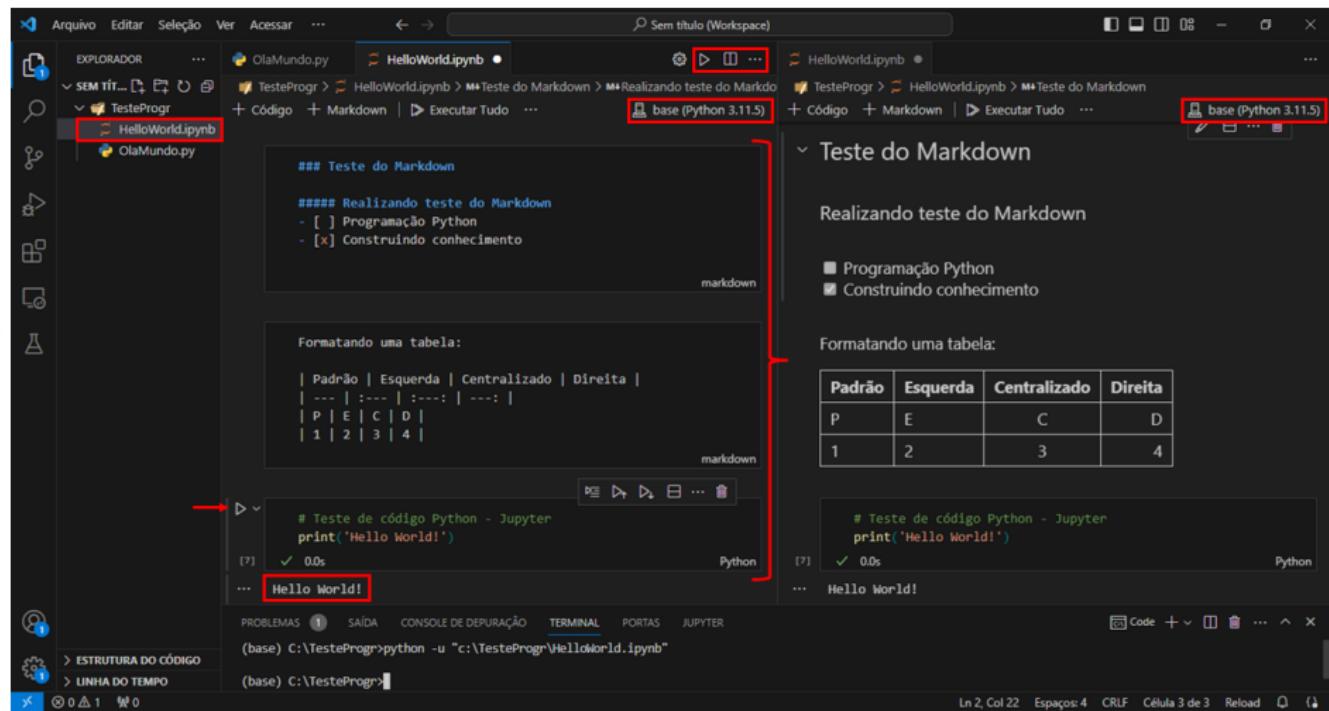
2.2.5. Testando o VSCode + Anaconda

Arquivos Python extensão .py → criação de *scripts*:



2.2.5.1. Testando o VSCode + Anaconda

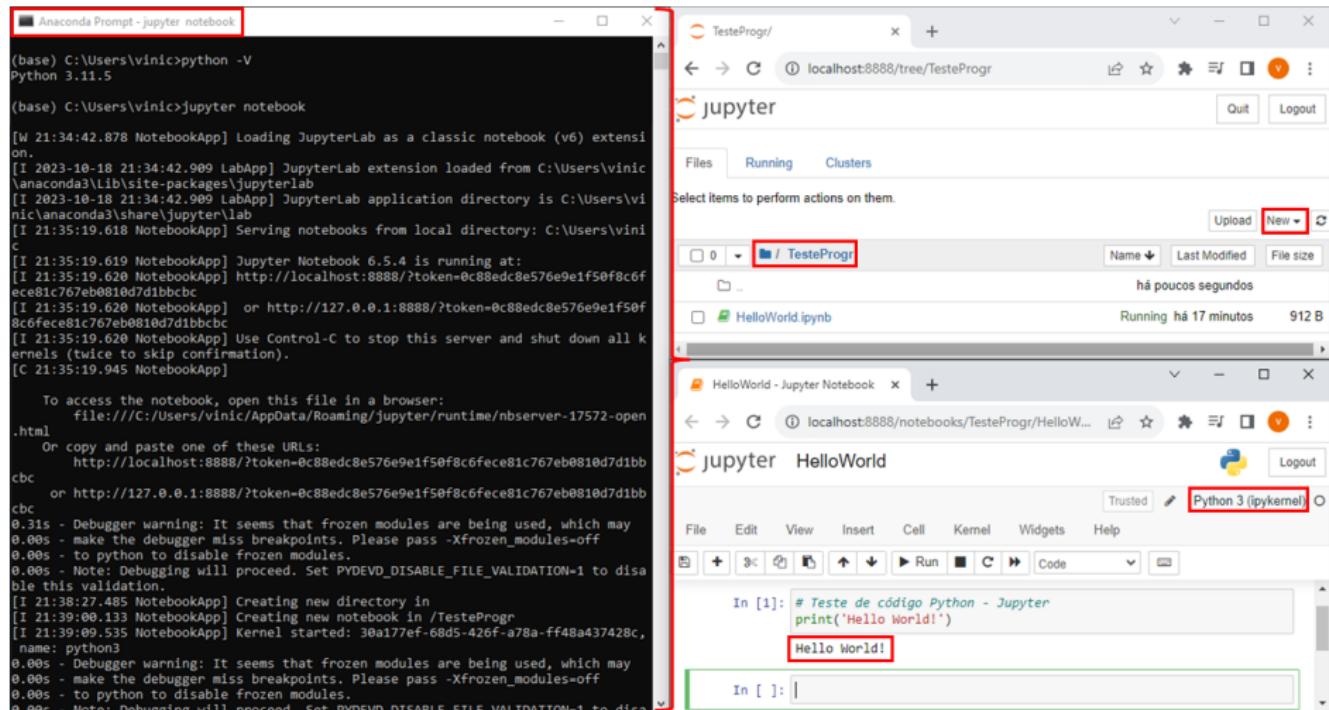
Arquivos Python extensão .ipynb → Jupyter notebook:



[...] → Exportar em HTML e gerar o .pdf utilizando CTRL + P

2.2.5.2. Testando o VSCode + Anaconda

Arquivos Python extensão *.ipynb* → Jupyter notebook → *prompt/navegador*:



2.2.5.3. Testando o VSCode + Anaconda

Crie um *script* (*plot.py*) contendo o código apresentado a seguir.

- O algoritmo fará uso de *duas bibliotecas*: *matplotlib.pyplot* e *numpy*;
- Serão criados *dois vetores temporais*: *t* e *s*;
- O algoritmo irá plotar o *gráfico bidimensional* $t \times s$ - avalie os **dados**.

Crie um segundo *script* (*plot3D.py*) contendo o código do próximo slide.

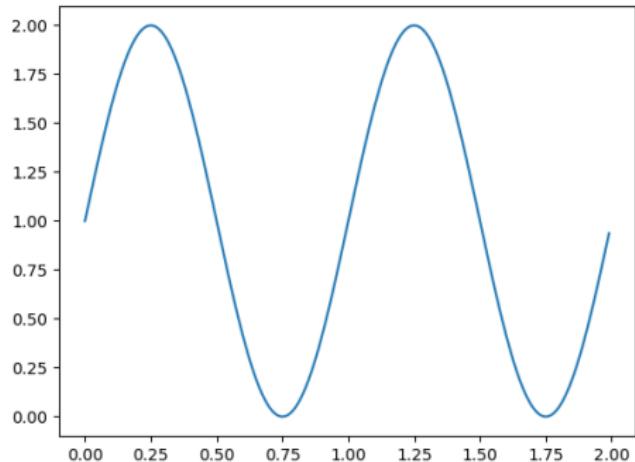
- O algoritmo fará uso das *mesmas bibliotecas*: *matplotlib.pyplot* e *numpy*;
- Serão criados *três matrizes de dados*: *X*, *Y* e *Z*;
- O algoritmo irá plotar o *gráfico tridimensional* $X \times Y \times Z$ - avalie os **dados**.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 t = np.arange(0.0, 2.0, 0.01)
5 s = 1+np.sin(2 *np.pi *t)
6
7 fig, ax =plt.subplots()
8 ax.plot(t,s)
```

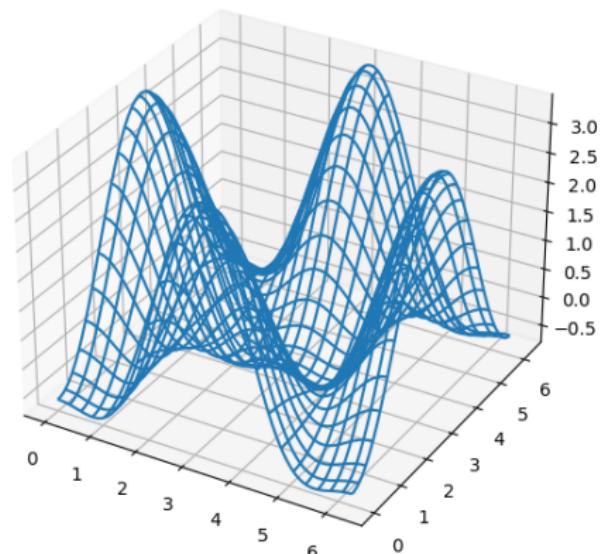
2.2.5.3.1 Testando o VSCode + Anaconda

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 alpha = 0.7
5 phi_ext = 2*np.pi *0.5
6
7 def ColorMap(phi_m,phi_p):
8     return (+alpha -2 * np.cos(phi_p) * np.cos(phi_m)
9             - alpha * np.cos(phi_ext -2*phi_p) )
10
11 phi_m =np.linspace(0, 2*np.pi, 100)
12 phi_p =np.linspace(0, 2*np.pi, 100)
13 X,Y =np.meshgrid(phi_p, phi_m)
14 Z =ColorMap(X, Y).T
15
16 fig = plt.figure(figsize=(8, 6))
17
18 ax =fig.add_subplot(1, 1, 1, projection = '3d')
19
20 p =ax.plot_wireframe(X, Y, Z, rstride=4, cstride=4)
```

2.2.5.3.2. Testando o VSCode + Anaconda



(a) Gráfico 2D - *plot.py*



(b) Gráfico 3D - *plot3D.py*

2.3.1. Contextualização: Linguagem Python

- Apesar do **Python** ter sido criado ao final da década de 80, como um sucessor da linguagem de programação ABC, foi por volta de 2019/20 que ela se “tornou tendência” mantendo-se em alta até os dias de hoje!
- Python** consiste em uma linguagem de **alto nível** (próximo à linguagem humana), quando comparada com **Assembly - baixo nível** → “linguagem de máquina” → zeros e uns;
- Para que um algoritmo seja executado se faz necessário um compilador/interpretador que recebe o código legível pelo usuário e convertem-no para o código “legível” pela máquina;
 - Na linguagem **compilada** (C, C++, Haskell, etc) a máquina traduz o programa diretamente, enquanto a **interpretada** (PHP, Python, JavaScript, etc) se faz necessário um programa que irá ler e executar o código em questão.

```

1  NULL EQU 0
2  STD_OUT_HANDLE EQU -11
3
4  extern ExitProcess
5  extern GetStdHandle
6  extern WriteFile
7
8  global inicio
9
10 section .data
11 mensagem db "HELLO WORLD", 0xA,0xD,0
12 tamanho_mensagem EQU $-mensagem
13
14 section .text
15
16 inicio:
17
18     mov rcx, STD_OUT_HANDLE
19     call GetStdHandle
20
21     mov rcx,rax
22     mov rdx,mensagem
23     mov r8,tamanho_mensagem
24     mov r9,0
25     call WriteFile
26
27     mov rax,NULL
28     call ExitProcess

```

A screenshot of a Jupyter Notebook cell. The code is:

```
print("Hello World")
```

The output shows:

```
[1] ✓ 0.0s
```

At the bottom, it says:

```
... Hello World
```

Python

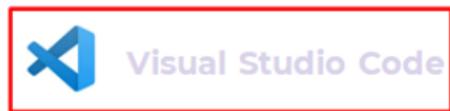
2.3.2. Compiladores - IDE's

Os compiladores/interpretadores são programas que realizam a “tradução” de um algoritmo, que geralmente é criado em alto nível, para um programa equivalente em linguagem de máquina permitindo que um processador possa executá-lo.

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```



```
0101101101010101
1010101010101010
1010101010101010
1010111111110000
10101011111000101
```



Visual Studio Code

Eclipse

Spyder



PyCharm



Wing IDE

2.3.3. Gerenciadores de pacotes e repositórios → ❤

- Os **Gerenciadores de pacotes**, são ferramentas que auxiliam um desenvolvedor a fazer o download de **pacotes/libs/frameworks** permitindo assim com que eles possam ser inseridos e utilizados nos projetos.
- **Repositórios**, segundo o dicionário, consiste em um *“lugar onde se guarda, arquiva e/ou coleciona alguma coisa.”* Nos repositórios é possível armazenar versões do projeto e partilhar conteúdo/ideias/algoritmos❤.



2.3.4. Vantagens da linguagem Python

Vantagens de se escolher Python:

- Aprendizado relativamente fácil:
 - Linguagem gratuita com código aberto, fóruns com suporte aos membros da comunidade e grande quantidade de material didático.
- Sintaxe relativamente simples:
 - Linguagem simples, fácil e intuitiva, quantidade reduzida de códigos na elaboração projetos (\uparrow produtividade \leftrightarrow \downarrow erros).
- Versatilidade:
 - Compatível com diversas plataformas, sistemas operacionais e processadores. Pode ser utilizado em diferentes tipos de ambientes/interfaces, cumpre bem com o propósito.
- Ciência de dados:
 - Aplicação nas mais diversas áreas com análise de dados para definição de novos objetivos: maximização de lucro, minimização de custos.

Funções

- Desenvolver, sistemas, aplicações e soluções para web
- Desenvolver protótipos e coordenar testes
- Manejo de bibliotecas e frameworks

Skills

- Manipulação de sistemas embarcados com Python
- Programação orientada a objetos
- Frameworks e bibliotecas Python

Formação

- Computação, engenharia de sistemas, matemática, etc.
- Certificações e pós-graduação em programação, desenvolvimento para web, entre outros

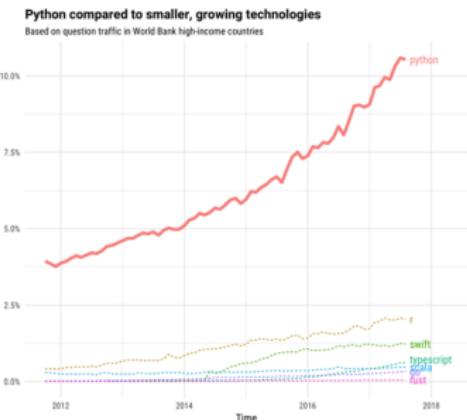
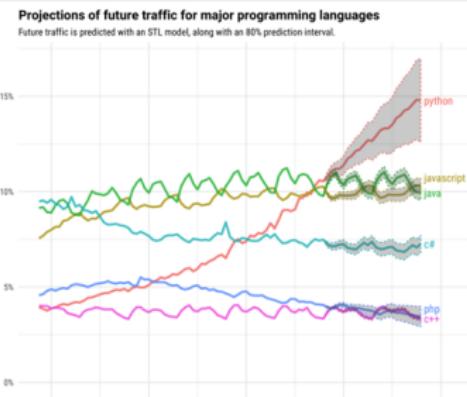
Salário

Estados Unidos: \$ 77.000 / ano
Portugal: € 70.000 / ano
Brasil: R\$ 41.000 / ano

2.3.5. Aplicações da linguagem Python

Aplicações do Python:

- Desenvolvimento Web:
 - Desenvolvimento que inclui funções complexas de *backend* - acesso a banco de dados, outros sites, envio seguro de informações, etc.
- Automação com *scripts* Python:
 - Tarefas cotidianas e exaustivas: renomear/-converter/avaliar arquivos, duplicidade de palavras, enviar e-mails, etc.
- Ciência de dados e *Machine Learning*:
 - Obtenção de informações relevantes em dados analisados por meio da inteligência artificial/computacional: separação de classes, previsão de séries temporais, estatística, etc.
- Desenvolvimento/testes de softwares:
 - Construção, gerenciamento, manutenção, interface gráfica do usuário (GUI), jogos, garantia do funcionamento desejado.



2.4.1. Mas antes, vamos relembrar... O que é lógica?

- O uso corriqueiro da palavra lógica associa-se à coerência e a racionalidade. Frequentemente associada à matemática, porém pode ser aplicada as demais ciências. Lógica consiste na “Ciência das formas de pensamento”;

Exemplo:

- a-) Todo mamífero é um animal.
Todo cavalo é um mamífero.
Portanto, todo cavalo é um animal.
- a-) Kaiton é país do planeta Stix.
Todos os Xinpins são de Kaiton.
Logo, todos os Xinpins são Stixianos.

Exemplos no dia-a-dia:

- a-) A gaveta está fechada.
A caneta está dentro da gaveta.
Precisamos primeiro abrir a gaveta para depois pegar a caneta.
- a-) Anacleto é mais velho que Felisberto.
Felisberto é mais velho que Marivalda.
Portanto, Anacleto é mais velho que Marivalda.

2.4.2. Lógica de programação e raciocínio?

Mas e a lógica de programação?

- O raciocínio é algo abstrato e intangível que todos possuem. Para que possamos utilizá-lo com facilidade, basta treinarmos e ele poderá ser expresso independente do idioma a ser utilizado;
- De forma similar acontece com a **Lógica de Programação**, que pode ser concebida pela mente treinada e pode ser representada em qualquer das inúmeras linguagens de programação existentes.

O que é um algoritmo?

- O objetivo principal do estudo da lógica de programação é a construção de algoritmos;
- Um **algoritmo** pode ser definido como uma sequência de passos que visam atingir um objetivo bem definido;
- Para especificarmos uma sequência de passos, faz-se necessário utilizar ordem, ou seja, ‘pensar com ordem’, portanto precisamos utilizar lógica;
- Apesar do nome não ser tão usual, o algoritmo está mais presente em nossas vidas do que pensamos, como uma **receita de bolo**.

2.4.3. Exemplo de um algoritmo

Algoritmo 1.1: Troca de lâmpada:

- ① Pegar uma escada;
- ② Posicionar a escada embaixo da lâmpada;
- ③ Buscar uma lâmpada nova;
- ④ Subir na escada;
- ⑤ Retirar a lâmpada velha;
- ⑥ Colocar a lâmpada nova.

Este algoritmo é bem objetivo! - *troca de lâmpada*. Mas e se esta lâmpada não estivesse queimada por exemplo? O algoritmo não previu esta possibilidade.

Algoritmo 1.2: Troca de lâmpada com teste:

- ① Pegar uma escada;
- ② Posicionar a escada embaixo da lâmpada;
- ③ Buscar uma lâmpada nova;
- ④ Acionar o interruptor;
- ⑤ Se a lâmpada não acender, então:
 - subir na escada;
 - retirar a lâmpada queimada;
 - colocar a lâmpada nova.

2.4.3.1. Exemplo de um algoritmo

Ambos os algoritmos anteriores (1.1 e 1.2) são eficazes, pois realizam o **mesmo objetivo**. A grande diferença entre eles é que no (1.1) todas as ações **SÃO executadas** e no (1.2) há uma **condicional** na 5^a ação que, apenas caso seja verdadeira, realizará a substituição da lâmpada - reduzindo de 6 ações para 5.

Mesmo o algoritmo (1.2) sendo mais eficiente ele ainda pode ser melhorado, pois não sabemos se as ações buscar uma escada e uma lâmpada são necessárias.

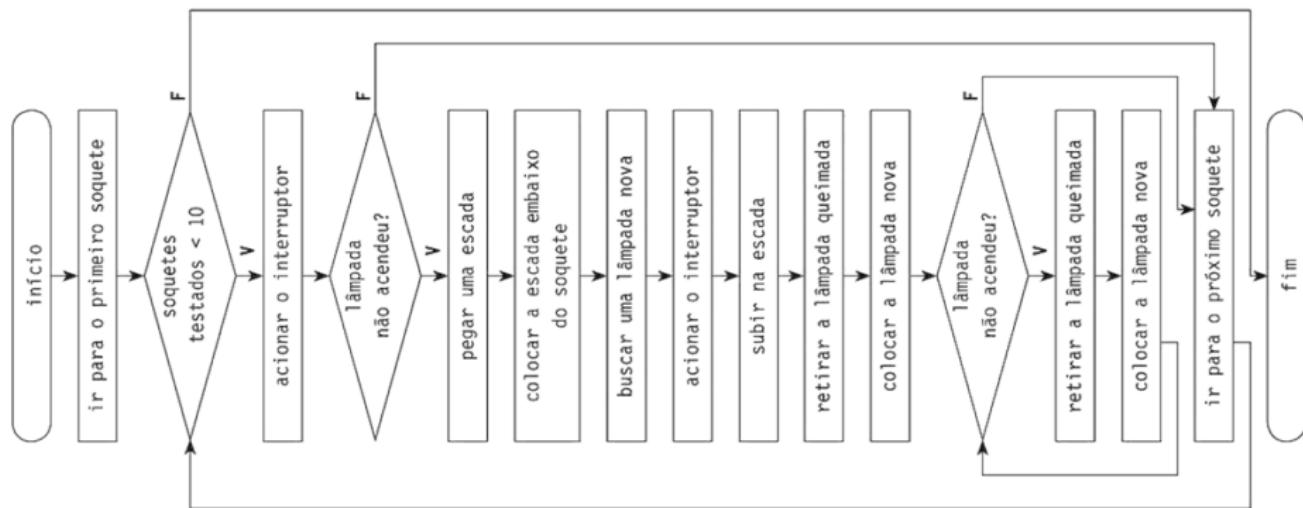
Algoritmo 1.3: Troca de lâmpada com teste no início:

- ① Acionar o interruptor;
- ② Se a lâmpada não acender, então:
 - Pegar uma escada;
 - Posicionar a escada embaixo da lâmpada;
 - Buscar uma lâmpada nova;
 - subir na escada;
 - retirar a lâmpada queimada;
 - colocar a lâmpada nova.

Neste caso as primeiras ações foram reduzidas de 5 para 2. Mas e se mesmo assim, a lâmpada não acender? Podemos representar com uma rotina de repetição! Podemos também otimizar nosso algoritmo buscando a lâmpada e a escada ao mesmo tempo - reduzindo uma ação. E assim construímos nosso algoritmo.

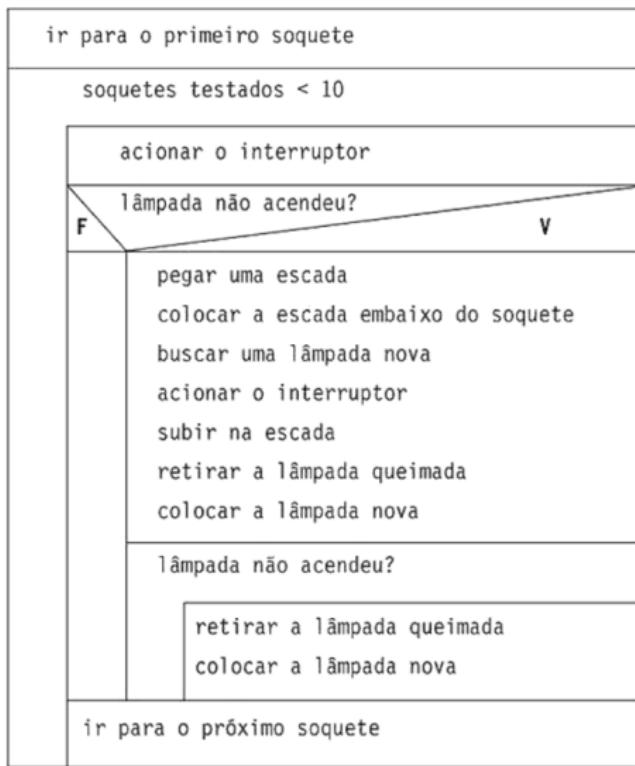
2.4.4. Representação de algoritmo por fluxograma

Um algoritmo pode ser representado por diversas formas, uma delas é bastante usual é o fluxograma. Neste apresenta um algoritmo para realizar a troca de lâmpada com testes para 10 soquetes com repetição:



Mudar a orientação da pagina para vertical no Adobe.

2.4.5. Representação de algoritmo por diagrama de Chapin



2.4.6. Exercícios de fixação: *Google Classroom*

Ex-1) Três senhoras - dona BRANCA, dona ROSA e dona VIOLETA - passeavam pelo parque quando dona ROSA disse:

-Não é curioso que estejamos usando vestidos de cores BRANCA, ROSA e VIOLETA, embora nenhuma de nós esteja usando um vestido de cor igual ao seu próprio nome?

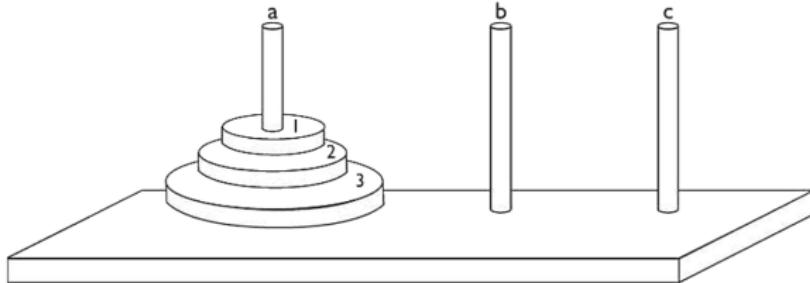
- Uma simples coincidência! - respondeu a senhora com o vestido VIOLETA. Qual a cor do vestido de cada senhora?

Ex-2) Um homem precisa atravessar um rio com um barco que possui capacidade apenas para carregar ele mesmo e mais uma de suas três cargas, que são: um lobo, um bode e um maço de alfafa. O que o homem deve fazer para conseguir atravessar o rio sem perder suas cargas? Escreva um algoritmo mostrando a resposta, ou seja, indicando todas as ações necessárias para efetuar a travessia segura.

2.4.6.1. Exercícios de fixação: *Google Classroom*

Ex-3) Três jesuítas e três canibais precisam atravessar um rio; para tal, dispõe de um barco com capacidade para duas pessoas. Por medida de segurança, não se deve permitir que em alguma margem a quantidade de jesuítas seja inferior à de canibais. Qual solução para efetuar a travessia com segurança? Elabore um algoritmo mostrando a resposta, indicando as ações que concretizam a solução deste problema.

Ex-4) Elabore um algoritmo que move três discos de uma Torre de Hanói, que consiste em três hastes ($a - b - c$), uma das quais serve de suporte para três discos de tamanhos diferentes (1 – 2 – 3), os menores sobre os maiores. Pode-se mover um disco de cada vez para qualquer haste, contanto que nunca seja colocado um disco maior sobre um menor. O objetivo é transferir os três discos para outra haste.



2.5.1. Tipos primitivos

- O computador manipula informações utilizando quatro tipos primitivos, que são utilizados para construção de algoritmos. Tais tipos são:
 - Inteiro: informação numérica que pertença ao conjunto dos números inteiros (negativo, nulo ou positivo). Ex:
 - Gilmar tem 15 irmãos;
 - Aquela escada possui 8 degraus;
 - Meu vizinho comprou 2 carros.
 - Real: informação numérica que pertença ao conjunto dos números reais (negativo, nulo ou positivo). Ex:
 - Ela tem 1,73 metros de altura;
 - Meu saldo bancário é de R\$215,10;
 - No momento estou pesando 82,5Kg
 - Caracter: informação composta de um conjunto de caracteres alfanuméricos: (0...9), alfabeticos (A...Z, a...z) e especiais (#, ?, !, @). Ex:
 - Constava na prova: “Use somente caneta!”;
 - O parque municipal estava repleto de placas: “Não pise na grama”;
 - O nome do vencedor é Felisberto Laranjeira.
 - Lógico: informação composta que pose assumir apenas duas situações (bistáveis). Ex:
 - A porta pode estar aberta ou fechada;
 - A lâmpada pode estar acesa ou apagada.

2.5.2. Informações técnicas

Constantes: dado que não sofre nenhuma variação no decorrer do tempo, ou seja, seu valor é constante desde o início até o fim da execução do algoritmo;

Variável: pode ser alterado em algum instante no decorrer do tempo, ou seja, durante a execução do algoritmo o valor do dado pode sofrer alterações;

Formação de identificadores: os “nomes” das informações são os identificadores e devem seguir as seguintes regras:

- Começar com um caracter alfanumérico;
- Podem ser seguidos por mais caracteres alfanumérico ou numéricos;
- Não devem ser usados caracteres especiais;
- Ex: Alpha, X, BJ156, Notas, Media, 5X, E(13), Nota/2, AWQ*, P&AA.

Declaração de variáveis: as informações são guardadas em dispositivos eletrônicos, **memória**. Tais memórias possuem ‘Subespaços’ responsáveis por armazenar as variáveis de tipos iguais juntas. Porém, cada uma no seu lugar - guardadas individualmente. Ex:

- **Inteiro**: X;
- **Caracter**: Nome, Endereco, Data;
- **Real**: ABC, XPTO, Peso, Dolar;
- **Logico**: resposta, H186;

2.5.3. Operadores matemáticos

Tabela 2.1 Operadores aritméticos

Operador	Função	Exemplos
+	Adição	2 + 3, X + Y
-	Subtração	4 - 2, N - M
*	Multiplicação	3 * 4, A * B
/	Divisão	10/2, X1/X2

Tabela 2.2 Potenciação e radiciação

Operador	Função	Significado	Exemplos
pow(x,y)	Potenciação	x elevado a y	pot(2,3)
sqrt(x)	Radiciação	Raiz quadrada de x	rad(9)

Tabela 2.3 Operador de resto e quociente de divisão inteira

Operador	Função	Exemplos
A % B	Resto da divisão	9 mod 4 resulta em 1 27 mod 5 resulta em 2
A / B	Quociente da divisão	9 div 4 resulta em 2 27 div 5 resulta em 5

Tabela 2.4 Precedência entre os operadores aritméticos

Prioridade	Operadores
1 ^a	parênteses mais internos
2 ^a	pow sqrt
3 ^a	* / % Alguns operadores mudam de linguagem para linguagem
4 ^a	+ -

2.5.4. Operadores lógicos

Tabela 2.5 Operadores relacionais

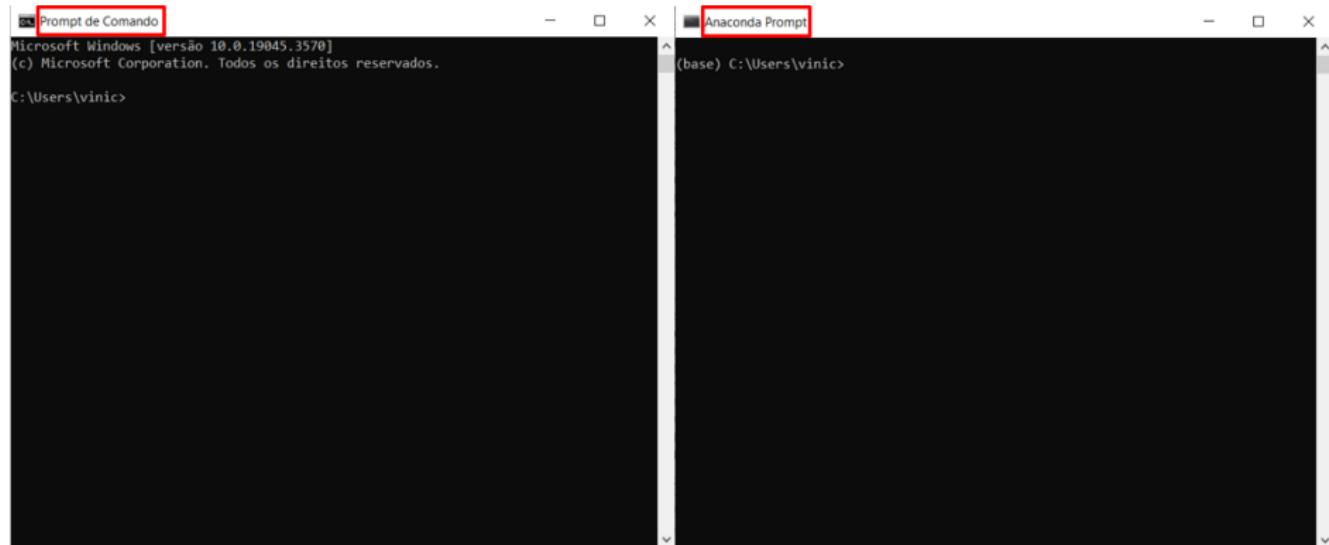
Operador	Função	Exemplos
<code>==</code>	Igual a	<code>3 = 3, X = Y</code>
<code>></code>	Maior que	<code>5 > 4, X > Y</code>
<code><</code>	Menor que	<code>3 < 6, X < Y</code>
<code>>=</code>	Maior ou igual a	<code>5 >= 3, X >= Y</code>
<code><=</code>	Menor ou igual a	<code>3 <= 5, X <= Y</code>
<code>!=</code>	Diferente de	<code>8 <> 9, X <> Y</code>

Tabela 2.6 Operadores lógicos

Operador	Função
<code>!</code>	Negação
<code>&&</code>	Conjunção (e)
<code> </code>	Disjunção (ou)

3.2.1. O que é e o que faz um *prompt* de comandos ?

- O *prompt* de comandos consiste em um programa que emula o campo de entrada em uma tela de interface do usuário **baseada em texto**. Tal ferramenta possui aparência da interface gráfica do usuário (GUI) do *Windows*.



3.2.1.1. O que é e o que faz um *prompt* de comandos ?

Anaconda Prompt



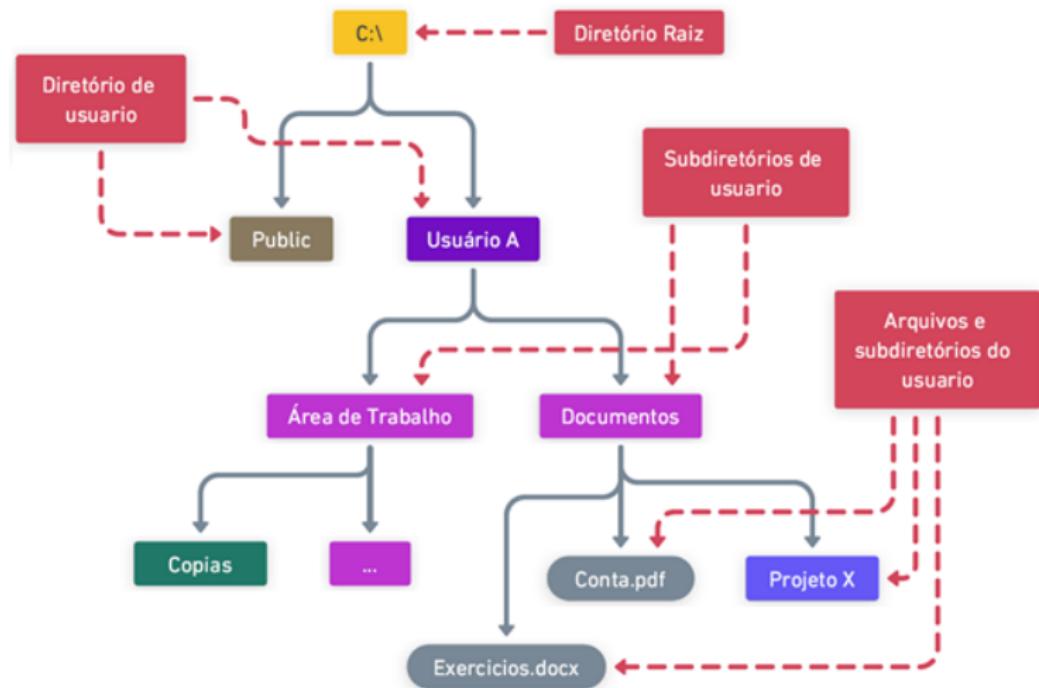
```
(base) C:\Users\vinic>python
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 10
>>> print("Hello World!")
Hello World!
>>> a
10
>>> import numpy as np
>>> t = np.arange(0, 10, 1)
>>> t
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> a
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> print(a[1,2])
6
>>> a[1][2]=100
>>> a
array([[ 1,   2,   3],
       [ 4,   5, 100],
       [ 7,   8,   9]])
>>> t[2]=200
>>> t
array([ 0,   1, 200,   3,   4,   5,   6,   7,   8,   9])
>>> exit()

(base) C:\Users\vinic>
```

Comandos básicos: DIR, COPY, MOVE, MKDIR, CD, RD, CLS, etc... → [HELP](#)

3.2.2. Estrutura de diretórios

Exemplo de hierarquia de diretórios do *Windows*



3.2.3. Dados simples



Txt - Arquivo de texto “cru” sem formatação (.txt)

A screenshot of a Windows Notepad window titled "dados - Bloco de Notas". The window contains the following text:

```
Arquivo Editar Formatar Exibir Ajuda
Aluno
aluno@email.com
senha: 123456
```

The window has standard operating system controls (minimize, maximize, close) at the top right. At the bottom, there are status indicators: "Ln 3, Col 14", "100%", "Windows (CRLF)", and "UTF-8".

3.2.3.1. Dados simples



Csv - Comma Separated Values - valores separados por vírgula: arquivo de texto estruturado, onde seus campos são separados por vírgulas (.csv)



```
1 Nome;Idade;Genero;Email;  
2 Joao;15;M;joaopedefeijao@hotmail.com;  
3 Maria;16;F;maryacomym@gmail.com;  
4 Cleberson;22;M;cleocleo@uol.com;  
5
```

3.2.3.2. Dados simples



Excel - Pasta de trabalho com várias planilhas (.xls .xlsx .xlsm etc...)

A screenshot of Microsoft Excel showing a data table. The table has columns labeled 'Nome', 'Idade', 'Genero', and 'Email'. The data rows are:

	Nome	Idade	Genero	Email
2	Joao	15	M	joaopedefeijsao@hotmail.com
3	Maria	16	F	maryacomym@gmail.com
4	Cleberson	22	M	cleocleo@uol.com
5				
6				
7				
8				
9				
10				
11				

The Excel ribbon is visible at the top, showing tabs like Arquivo, Página Inicial, Inserir, Layout da Página, Fórmulas, Dados, Revisão, Exibir, Desenvolvedor, and Ajuda. The formula bar shows 'D16'. The status bar at the bottom indicates 'Planilha1' and 'Acessibilidade: não disponível'.

3.2.3.3. Dados simples



Json - Java Script Object Notation: é um arquivo leve de troca de informações/dados entre sistemas → possui estrutura definida (.json)

```
● ● ●
1  {
2      "produtos": [
3          {
4              "id": 1,
5              "name": "caneca",
6              "price": 5.45,
7              "color": "white"
8          },
9          {
10             "id": 2,
11             "name": "garrafa",
12             "price": 75,
13             "color": "white"
14         }
15     ]
16 }
```

3.2.4. Python - Keywords

Python

>>>

```
>>> help("keywords")
```

Here is a list of the Python keywords. Enter any keyword to get more help.

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

3.3.1. Qual a definição de um banco de dados?



Banco de dados (BD) consiste em uma **coleção organizada de informações/dados estruturados**.

Normalmente **armazenados eletronicamente em um sistema de computador e controlado por um sistema de gerenciamento de banco de dados**.

As características de um BD são:

- Tem um propósito específico;
- O conjunto deve ser modelável;
- Representa um aspecto do mundo real;
- Pode variar em tamanho e complexidade;
- Deve ter controle de redundância;
- Deve permitir o uso de dados simultâneos;
- Deve ser padronizado.

3.4.1. Primeiro contato com banco de dados em Python

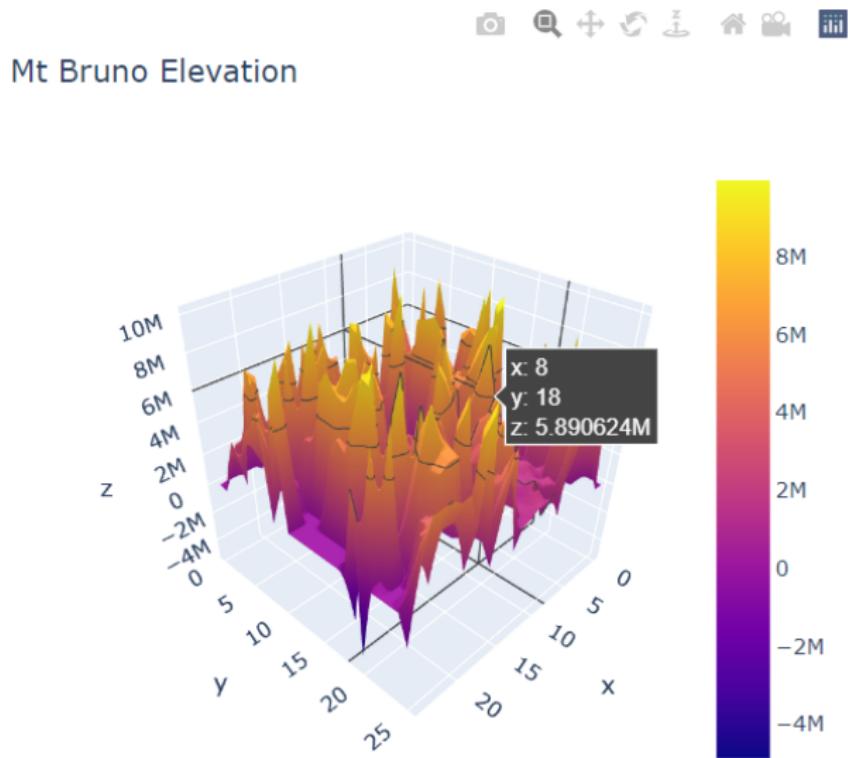
O banco de dados contido no link a seguir consiste em dados de latitude, longitude e elevações - *Mt Bruno Elevation*. Realize seu download e importe os dados e avaliando a plotagem do gráfico obtido a partir dos mesmos.

```
1 import plotly.graph_objects as go
2 import pandas as pd
3
4 # Leitura de dados .csv
5 z_dataCSV =pd.read_csv('DataCSV.csv')
6
7 # Apresentação dados em CSV
8 fig =go.Figure(data=[go.Surface(z=z_dataCSV.values)])
9
10 fig.update_layout(title='Mt Bruno Elevation – .csv', autosize=False,
11                   width=500, height=500,
12                   margin=dict(l=90, r=50, b=65, t=90))
13
14 fig.show()
```

Link para o banco de dados - *Mt Bruno Elevation*:

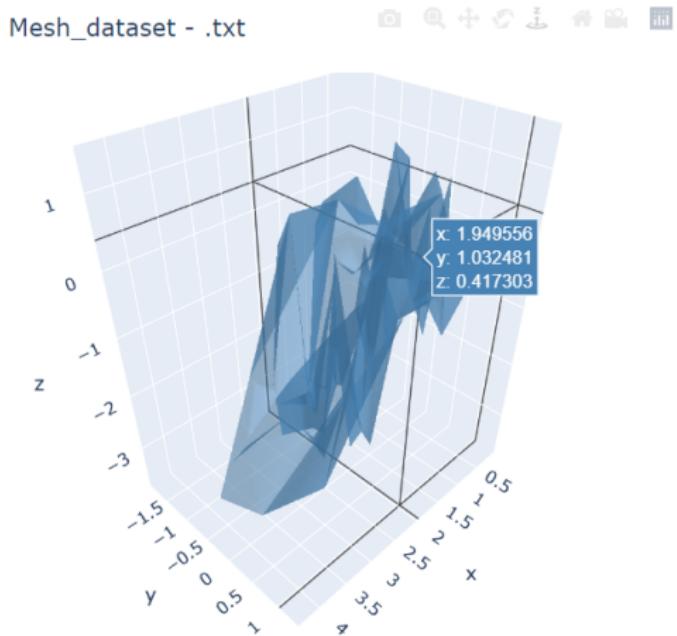
https://raw.githubusercontent.com/plotly/datasets/master/api_docs/mt_bruno_elevation.csv

3.4.1.1. Primeiro contato com banco de dados em Python



3.4.1.2. Primeiro contato com banco de dados em Python

Faça as pesquisas necessárias na internet para realizar a importação de dados na extensão .txt em Python e fazendo uso da mesma biblioteca: *plotly.graph_objects*, plote o gráfico apresentado a seguir.



https://raw.githubusercontent.com/plotly/datasets/master/mesh_dataset.txt

3.5.1. Definições banco de dados SQL e NoSQL



Structured Query Language:

“Linguagem de consulta estruturada”

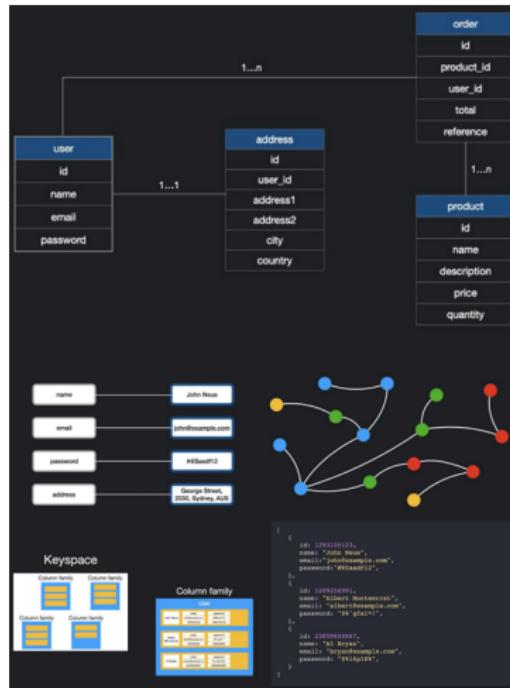
SQL consiste em uma linguagem de programação usada por grande parte dos bancos de dados **relacionais** para consultar, manipular, definir e fornecer controle de acesso. Ou seja, gerenciar dados relacionais em um sistema de BD ou para processamento de fluxo de dados.

No Only Structured Query Language:

“Não somente Linguagem de consulta estruturada”

NoSQL é uma linguagem de programação usada para BDs **não relacionais**. Ele pode conter SQL, porém há outras formas de manipular os dados. Fornece um mecanismo para armazenamento e recuperação de dados modelados de formas diferentes das relações tabulares (BD relacionais).

3.5.1.1. Definições banco de dados SQL e NoSQL



Afinal, o que é um banco relacional?

É um formato de banco rigidamente estruturado, baseado em tabelas. Os campos tem relacionamento entre si.

E o que é um banco não relacional?

É qualquer banco de dados que não segue o modelo relacional fornecido pelos sistemas tradicionais de gerenciamento de banco de dados relacionais (SGBDR). Apresentam os seguintes tipos:

- *Key-value stores;*
- *Graph stores;*
- *Column stores;*
- *Document stores.*

3.5.2. Alguns comandos muito utilizados em SQL

Alguns comandos em SQL:

- **CREATE**: cria novas tabelas em um banco de dados;
- **ALTER**: alterar uma tabela já criada;
- **INSERT**: adiciona registros a uma tabela;
- **UPDATE**: atualiza os registros já inseridos;
- **DELETE**: exclui registros de uma tabela;
- **SELECT**: busca registros na tabela;
- **GRANT**: permite acesso a objetos do banco de dados;
- **REVOKE**: remove o acesso a objetos do banco de dados;
- **DENY**: bloqueia o acesso para objetos e usuários;específicos
- **DROP**: exclui uma tabela do banco de dados.

3.5.3. Exemplos de comandos em SQL

CREATE: criar novas tabelas em um BD.

```
CREATE TABLE usuarios (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nome VARCHAR(50),
    email VARCHAR(100),
    senha VARCHAR(50));
```

ALTER: alterar uma tabela já criada em um BD.

```
ALTER TABLE usuarios ADD data_nascimento VARCHAR(10);
```

INSERT: adicionar registros a uma tabela de um BD.

```
INSERT INTO usuarios(nome, email, senha, data_nascimento)
VALUES ('Joao','joao_e_maria@gmail.com','Maria123','21/05/1989');
```

UPDATE: atualizar os registros de uma tabela já criada em um BD.

```
UPDATE usuarios SET senha="Ana456" WHERE nome = "Joao"
```

DELETE: excluir os registros de uma tabela já criada em um BD.

```
DELETE FROM usuarios WHERE email="joao_e_maria@gmail.com"
```

3.5.4. Aplicação comando *CREATE*

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect("MyBD_py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Define a Query e executa
10 c.execute("""CREATE TABLE IF NOT EXISTS usuarios (
11     id INTEGER PRIMARY KEY AUTOINCREMENT,
12     nome VARCHAR(50),
13     email VARCHAR(100),
14     senha VARCHAR(50)
15 )""")
16
17 print ("Tabela Usuarios Criada com sucesso")
18
19 # Grava no banco
20 BD.commit()
21
22 # Fechar a conexão com banco de dados
23 BD.close()
```

3.5.5. Aplicação comando *ALTER*

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect("MyBD_py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Define a Query e executa
10 c.execute("""ALTER TABLE usuarios ADD data_nascimento VARCHAR(10)""")
11
12 print ("Tabela Usuarios Alterada com sucesso")
13
14 # Grava no banco
15 BD.commit()
16
17 # Fechar a conexão com banco de dados
18 BD.close()
```

3.5.6. Aplicação comando *INSERT*: Versão 1

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect("MyBD_py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Define a Query e executa
10 c.execute("""INSERT INTO usuarios(nome, email, senha, data_nascimento)
11 VALUES ('Joao','joao_e_maria@gmail.com','Maria123','21/05/1989')""")
12 c.execute("""INSERT INTO usuarios(nome, email, senha, data_nascimento)
13 VALUES ('Roberto','beto@gmail.com','Beto123','15/11/2000')""")
14 c.execute("""INSERT INTO usuarios(nome, email, senha, data_nascimento)
15 VALUES ('Marcia','Marinha@gmail.com','Marcinha123','07/01/1993')""")
16
17 # Mostra mensagem para o usuario
18 print("Usuarios Adicionados com sucesso V1")
19
20 # Grava no banco
21 BD.commit()
22
23 # Fechar a conexão com banco de dados
24 BD.close()
25
26 # Para visualizar os dados no BD (comentar da linha 9 à 26)
27 #c.execute("SELECT * FROM usuarios")
28 #print(c.fetchall())
```

3.5.6.1. Aplicação comando *INSERT*: Versão 2

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect ("MyBD.py.db", timeout=60)
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Cria a lista de usuarios
10 lista = [
11     ('Fernando', 'Nando89@gmail.com', 'Nando123', '21/05/1989'),
12     ('Gilmar', 'Gil@gmail.com', 'Gilmar123', '07/12/1999'),
13     ('Carlos', 'carlinhos1988@gmail.com', 'Carlinhos123', '18/06/1988')
14 ]
15
16 # Define a Query e executa
17 c.executemany ("""INSERT INTO usuarios(nome, email, senha, data_nascimento) VALUES (?,?,?,?,?)""", lista )
18
19 # Mostra mensagem para o usuario
20 print ("Usuarios Adicionados com sucesso V2")
21
22 # Grava no banco
23 BD.commit()
24
25 # Fechar a conexão com banco de dados
26 BD.close()
```

3.5.6.2. Aplicação comando *INSERT*: Versão 3

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect ("MyBD.py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Carrega as informações do usuario
10 p_nome =input ('Nome: ')
11 p_email = input ('Email: ')
12 p_senha = input ('Senha: ')
13 p_data_de_nascimento = input ('Data de Nascimento (dd/mm/yyyy): ')
14
15
16 # Define a Query e executa
17 c.execute("""INSERT INTO usuarios(nome, email, senha, data_nascimento) VALUES (?,?,?,?,?)""",
18     (p_nome, p_email, p_senha, p_data_de_nascimento))
19
20 # Mostra mensagem para o usuario
21 print ("Usuarios Adicionados com sucesso V3")
22
23 # Grava no banco
24 BD.commit()
25
26 # Fechar a conexão com banco de dados
27 BD.close()
```

3.5.7. Aplicação comando *UPDATE*: Versão 1

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect("MyBD.py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Carrega as informações do usuario
10 id_usuario =1
11 novo_nome ="Luiza"
12 novo_email ="luiza@uol.com.br"
13 nova_senha ="souLinda123"
14 nova_data_de_nascimento ="02/02/2002"
15
16 # Define a Query e executa
17 c.execute("""UPDATE usuarios SET nome=? , email=? , senha=? , data_nascimento=? WHERE id=? """,
18     (novo_nome, novo_email, nova_senha, nova_data_de_nascimento, id_usuario))
19
20 # Grava no banco
21 BD.commit()
22
23 # Mostra mensagem para o usuario
24 print(f" Usuarios { id_usuario } modificado com sucesso V1")
25
26 # Fechar a conexão com banco de dados
27 BD.close()
```

3.5.7.1. Aplicação comando *UPDATE*: Versão 2

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD = sqlite3.connect("MyBD.py.db")
5
6 # Inicializa o cursor onde sera feita as Queries
7 c = BD.cursor()
8
9 # Define a Query e executa
10 c.execute("""SELECT * FROM usuarios""")
11
12 # Mostra os dados da tabela para o usuario
13 for linha in c.fetchall():
14     print(linha)
15
16 # Carrega as informações do usuario
17 id_usuario = input('Id do usuario: ')
18 novo_nome = input('Nome: ')
19 novo_email = input('Email: ')
20 nova_senha = input('Senha: ')
21 nova_data_de_nascimento = input('Data de Nascimento (dd/mm/yyyy): ')
22
23 # Define a Query e executa
24 c.execute("""UPDATE usuarios SET nome=?, email=?, senha=?, data_nascimento=? WHERE id=?""",
25             (novo_nome, novo_email, nova_senha, nova_data_de_nascimento, id_usuario))
26
27 # Grava no banco
28 BD.commit()
29
30 # Mostra mensagem para o usuario
31 print(f"Usuarios {id_usuario} modificado com sucesso V2")
32
33 # Fechar a conexão com banco de dados
34 BD.close()
```

3.5.8. Aplicação comando *DELETE*

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect("MyBD_py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Carrega as informações do usuario
10 id_usuario = "5"
11
12 # Define a Query e executa
13 c.execute("""DELETE FROM usuarios WHERE id= ? """, (id_usuario))
14
15 # Grava no banco
16 BD.commit()
17
18 # Mostra mensagem para o usuario
19 print(f'Usuario { id_usuario } deletado com sucesso')
20
21 # Fechar a conexão com banco de dados
22 BD.close()
```

3.5.9.4. Exemplo de comandos em SQL

SELECT: busca registros de uma tabela já criada em um BD.

SELECT * FROM usuarios

SELECT * FROM usuarios WHERE nome = "Ana"

SELECT nome FROM usuarios WHERE senha = "Joao123"

GRANT: permite acesso a objetos de uma tabela já criada em um BD.

GRANT SELECT, INSERT UPDATE ON usuarios TO usuario_db_01

REVOKE: remove o acesso a objetos de uma tabela já criada em um BD.

REVOKE SELECT ON usuarios FROM usuario_db_01

DENY: bloqueia o acesso a objetos de uma tabela já criada em um BD.

DENY SELECT ON usuarios TO usuario_db_01

DROP: exclui uma tabela de um BD ou o próprio BD.

DROP TABLE usuarios → c.execute(" " " DROP TABLE usuarios" " ")

DROP DATABASE db

3.5.10.6.1. Aplicação comando *SELECT*

```
1 import sqlite3
2
3 # Inicializa a conexão com o banco de dados
4 BD =sqlite3.connect("MyBD_py.db")
5
6 # Inicializa o cursor onde sera feita as Querys
7 c =BD.cursor()
8
9 # Define a Query e executa
10 c.execute("""SELECT * FROM usuarios""")
11
12 # Mostra mensagem para o usuario
13 for linha in c.fetchall():
14     print(linha)
15
16 # Fechar a conexão com banco de dados
17 BD.close()
```

3.5.11. Exercício de fixação

Desenvolva um banco de dados (*BD.ipynb*) no qual deverá ser criado utilizando os comandos aprendidos anteriormente. O conteúdo do banco de dados será escolhido por você. Por exemplo:

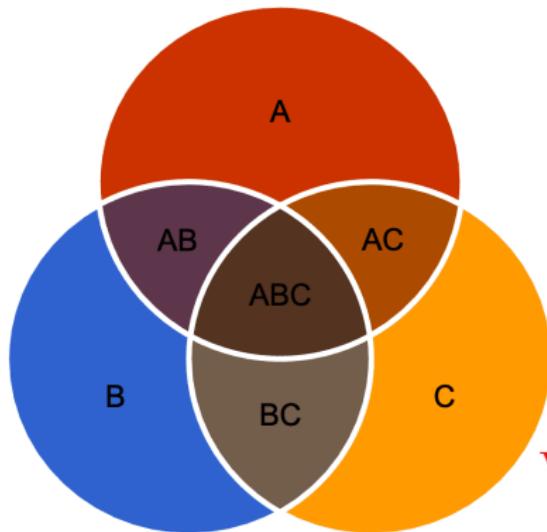
- Estoque de uma loja;
- Controle de clientes;
- Dados obtidos a partir de uma linha de produção;
- Dados médicos de pacientes;
- Fabricação de veículos, etc.

Elabore um *Markdown* descrevendo as etapas do código, e por fim, elabore um arquivo em .pdf.

4.2.1. Variáveis globais × Variáveis locais

Variáveis globais:

- Consiste em uma variável **acessível em todos** os escopos de um algoritmo;
- Tal variável é **definida fora** do corpo de uma função específica;
- Geralmente considerada “inadequada” devido estar **sujeita a alterações** em qualquer local e parte de um *script*;
- Muito utilizadas em **programação concorrente** para que as informações possam permeiar em diferentes seções do código que não possuem relação entre si → *threads* e sinais.



Variáveis locais:

- Consiste em uma variável **acessível apenas** dentro da função na qual foi declarada;
- Só pode ser **alterada dentro do referido escopo** em que foi definida.

4.2.1.1. Variáveis globais × Variáveis locais

```
1 nome = "Jose" # José como variável global
2
3 def EscreveGlob():
4     print(f"{nome} esta em aula!")
5
6 def EscreveInfoGlob():
7     print(f"{nome} esta estudando")
8
9 EscreveGlob()
10 EscreveInfoGlob()
11 nome = "Fernanda" # Mudando uma variável global
12
13 def EscreveLoc():
14     nome = "Maria" # Maria como variável local
15     print(f"{nome} esta assistindo aula no SENAI!")
16
17 def EscreveInfoLoc():
18     nome = "Maria" # Maria como variável local
19     print(f"{nome} esta na aula de Data Analytics com Python!")
20
21 EscreveLoc()
22 EscreveInfoLoc()
23
24 print(nome + ' É uma variável global e Maria é local!')
```

4.2.2. Varargs - argumentos variáveis e comando *return*

Varargs:

```
def calcular_imposto(valor, perc_ir):
    ir = valor * perc_ir
    iss = valor * 0.05
    csll = valor * 0.0375
    pis = valor * 0.03
    return ir + iss + csll + pis

print(calcular_imposto(1000 , perc_ir=0.275))

def calcular_imposto(valor, *args):
    total_imposto = 0
    for item in args:
        total_imposto += valor * item
    return total_imposto

print(calcular_imposto(1000, 0.275, 0.05, 0.0375, 0.03))
```

392.5

- Quando em um método **não se sabe ao certo quantos parâmetros serão passados**: vetor, lista, objetos, etc, pode-se definir um determinado parâmetro do tipo *varargs*;
- Pode ser considerado um “facilitador” devido a sua versatilidade em um *script*;
- Ao invés de criar um *array* ou *lista* e colocar os valores dentro dele é possível chamá-lo diretamente passando ‘n’ valores e os **parâmetros são automaticamente adicionados** em um *array* do mesmo tipo do *varargs*.

Return:

- É um comando que retorna algum valor que deseja ser obtido após a execução de uma **função**, por exemplo. Uma função que calcula Pitágoras deve retornar a hipotenusa.

4.2.2.1. Varargs - argumentos variáveis e comando *return*

```
1 def soma(*args):
2     resultado = 0
3
4     for x in args:
5         resultado+= x
6
7     print(resultado)
8
9 soma(1,2,3,4,5,6)
10 soma(1,3)
11
12 import numpy as np
13 cont = 0
14 def Matrix3_3(cont, *args):
15     Mat = np.zeros([int(len(args)/3),int(len(args)/3)])
16     for x in range(int(len(args)/3)):
17         for y in range(int(len(args)/3)):
18             Mat[x][y] = args[cont]
19             cont += 1
20     print(Mat)
21
22 Matrix3_3(cont, 22,32,123,553,22,12,56,23,98)
```

4.2.2.1. Varargs - argumentos variáveis e comando *return*

```
1 import math
2
3 def MenorNumero(*args):
4
5     oMenor = math.inf
6     for x in args:
7         if (oMenor >= x):
8             oMenor = x
9
10    return (oMenor)
11
12 print('O menor número é: ' + str(MenorNumero(14,5,99,4,9,63)))
13
14 def MaiorNumero(*args):
15
16     oMaior = -math.inf
17     for x in args:
18         if (oMaior <= x):
19             oMaior = x
20
21    return (oMaior)
22
23 print('O maior número é: ' + str(MaiorNumero(14,5,99,4,9,63)))
```

4.2.3. Init e Self no Python

```
# Vamos criar uma classe carro
class carro():
    # método init ou construtor
    def __init__(self, modelo, cor):
        self.modelo = modelo
        self.cor = cor

    def mostrar(self):
        print("O Modelo é", self.modelo )
        print("A Cor é", self.cor )

# cada objeto possui diferentes self com atributos
audi = carro("audi a4", "azul")
ferrari = carro("ferrari 488", "verde")
audi.mostrar()
ferrari.mostrar()

O Modelo é audi a4
A Cor é azul
O Modelo é ferrari 488
A Cor é verde

[10] carro.mostrar(audi)

O Modelo é audi a4
A Cor é azul

[11] carro.mostrar(ferrari)

O Modelo é ferrari 488
A Cor é verde
```

__init__ :

- Também chamada de “método construtor” é responsável por criar o objeto de uma determinada classe;
- Ou seja, é uma função especial utilizada para inicializar o objeto quando vai criar uma instância daquela classe.

self:

- É um parâmetro que faz referência a instância de uma determinada classe;
- É responsável por vincular os atributos com os argumentos enviados para uma função ou método → **1º argumento a ser passado.**

4.2.3.1. Init e Self no Python

```
1  class Pessoa:  
2      def __init__(self, nome, idade):  
3          self.nome = nome  
4          self.idade = idade  
5  
6      def apresenta(self):  
7          print(f'{self.nome} tem {self.idade} anos!')  
8  
9  Pessoa1 = Pessoa("Joao","15")  
10 Pessoa2 = Pessoa("Maria","18")  
11  
12 print(Pessoa1.nome)  
13 print(Pessoa1.idade)  
14  
15 Pessoa.apresenta(Pessoa2)
```

4.2.3.2. Init e Self no Python

```
1 class Fruta:  
2     def __init__(self, tipo, cor) -> None:  
3         self.tipo = tipo  
4         self.cor = cor  
5  
6     def MostrarPropriedades(self):  
7         print(f"Sou um(a) {self.tipo} e minha cor é {self.cor}!")  
8  
9 NovaFruta = Fruta("Banana", "Amarela")  
10 NewFruit = Fruta("Melancia", "Verde")  
11  
12 NovaFruta.MostrarPropriedades()  
13 NewFruit.MostrarPropriedades()
```

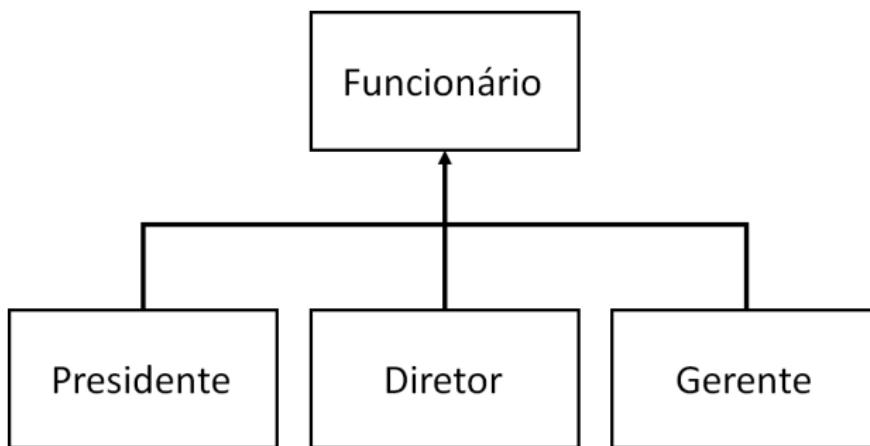
4.2.4. Herança e dependência no Python

Herança:

- A **herança** é um conceito do paradigma da **orientação à objetos** que determina que uma **classe (filha)** pode herdar atributos e métodos de uma outra **classe (pai)** e, assim, evitar que haja muita repetição de código.

Dependência:

- É a **referência de uma biblioteca, objeto, arquivo, etc**, que se faz necessário para que haja o correto funcionamento de uma função/classe/objeto, etc.



4.2.4.1. Herança e dependência no Python

```
1 class Animal():
2     def __init__(self, nome, cor):
3         self.__nome = nome
4         self.__cor = cor
5
6     def comer(self):
7         print(f"O {self.__nome} {self.__cor} está comendo")
8
9 class Gato(Animal):
10    def __init__(self, nome, cor):
11        super().__init__(nome, cor)
12
13 class Cachorro(Animal):
14    def __init__(self, nome, cor):
15        super().__init__(nome, cor)
16
17 class Coelho(Animal):
18    def __init__(self, nome, cor):
19        super().__init__(nome, cor)
20
21
22 gato = Gato("Bichano", "Branco")
23 cachorro = Cachorro("Totó", "Preto")
24 coelho = Coelho("Pernalonga", "Cinza")
25
26 gato.comer()
27 cachorro.comer()
28 coelho.comer()
```

```
1 # Dependência
2
3 import math
4
5 print(math.pi)
6 print(math.cos(0))
```

4.3.1. Dashboard / Tableau: uma interface gráfica

Dashboards / Tableau:

- Em **tecnologia da informação**, pode-se dizer que consiste em um painel composto por uma interface gráfica (*knobs*, mostradores, *bar chart*, etc) que fornece visualizações rápidas dos principais indicadores de desempenho relevantes para um objetivo ou processo de negócios específico;
- Em outras palavras o “Painel de bordo” ou *dashboard* é utilizado como um painel de indicadores que fornece uma representação ilustrada do desempenho dos negócios em toda a organização;
- Um exemplo de aplicação são os **indicadores chave de desempenho (KPIs)** dos quais ajudam as empresas a **medir, monitorar e até gerenciar seu desempenho, alcançando assim seus objetivos.**



4.3.2.1. Dashboard / Tableau: uma interface gráfica

Dashboards / Tableau:

- Para que seja possível criar *Dashboards* em **Python** se faz necessário a instalação da biblioteca **streamlit** via **CMD** por meio do comando:

pip intall streamlit

- Após a instalação da biblioteca e criação do *script* referente ao projeto em questão, novamente, via **CMD**, é possível obter o *dashboard* pelo comando:

streamlit run nomeScript.py

```
(base) C:\Users\vinic\OneDrive\Professor2023_1\SENAI\BigDataPython\SlideAulasBigPython\Algoritmos>pip install streamlit
Requirement already satisfied: streamlit in c:\programdata\anaconda3\lib\site-packages (1.28.2)
Requirement already satisfied: altair<6,>4.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (5.1.2)
Requirement already satisfied: blinker<2,>1.0.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (1.7.0)
Requirement already satisfied: cachetools<6,>4.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (5.3.2)
Requirement already satisfied: click<9,>7.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (8.0.4)
Requirement already satisfied: importlib-metadata<7,>1.4 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (6.0.0)
Requirement already satisfied: numpy<2,>1.19.3 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (1.24.3)
Requirement already satisfied: packaging<24,>=16.8 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (23.1)
Requirement already satisfied: pandas<3,>=1.3.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (2.0.3)
Requirement already satisfied: pillow<11,>7.1.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (9.4.0)
Requirement already satisfied: protobuf<5,>=3.20 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (4.25.1)
Requirement already satisfied: pyarrow<6.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (11.0.0)
Requirement already satisfied: python-dateutil<3,>=2.7.3 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (2.8.2)
Requirement already satisfied: requests<3,>=2.27 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (2.31.0)
Requirement already satisfied: rich<14,>=10.14.0 in c:\programdata\anaconda3\lib\site-packages (from streamlit) (13.7.0)
:
Requirement already satisfied: pyrsistent!=0.17.0,!0.17.1,!0.17.2,>0.14.0 in c:\programdata\anaconda3\lib\site-packages (from jsonschema)=3.0->altair<6,>4.0->streamlit) (0.18.0)
Requirement already satisfied: mdurl~0.1 in c:\programdata\anaconda3\lib\site-packages (from markdown-it-py)>=2.2.0->rich<14,>=10.14.0->streamlit) (0.1.0)

(base) C:\Users\vinic\OneDrive\Professor2023_1\SENAI\BigDataPython\SlideAulasBigPython\Algoritmos>streamlit run Dashboarduber.py

You can now view your Streamlit app in your browser.

Local URL: http://localhost:8502
Network URL: http://192.168.0.220:8502
```

4.3.3.2. Dashboard / Tableau: uma interface gráfica

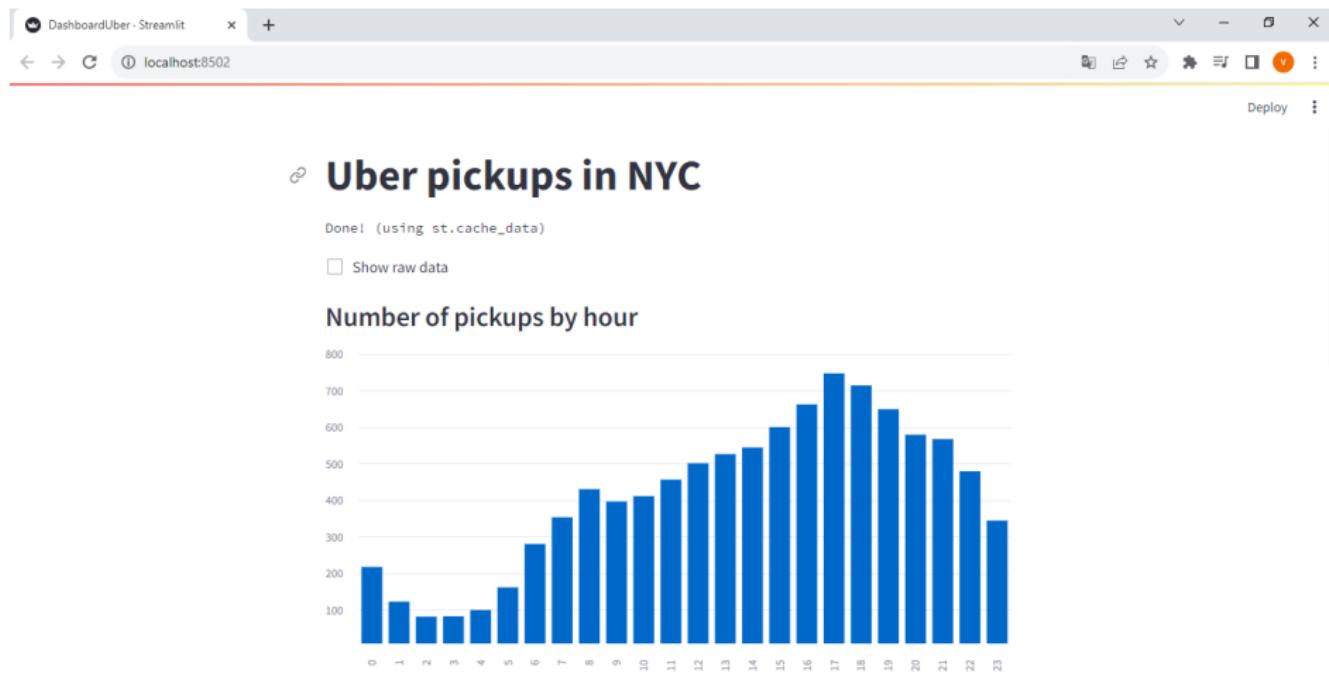
Dashboards / Tableau:

- O link a seguir: <https://docs.streamlit.io/> fornece a documentação do *streamlit* desde a instalação até exemplos de aplicação prática;
- O *script* a seguir apresenta um *dashboard* da utilização de Ubers na cidade de Nova York por meio de um banco de dados .csv durante as 24h do dia;

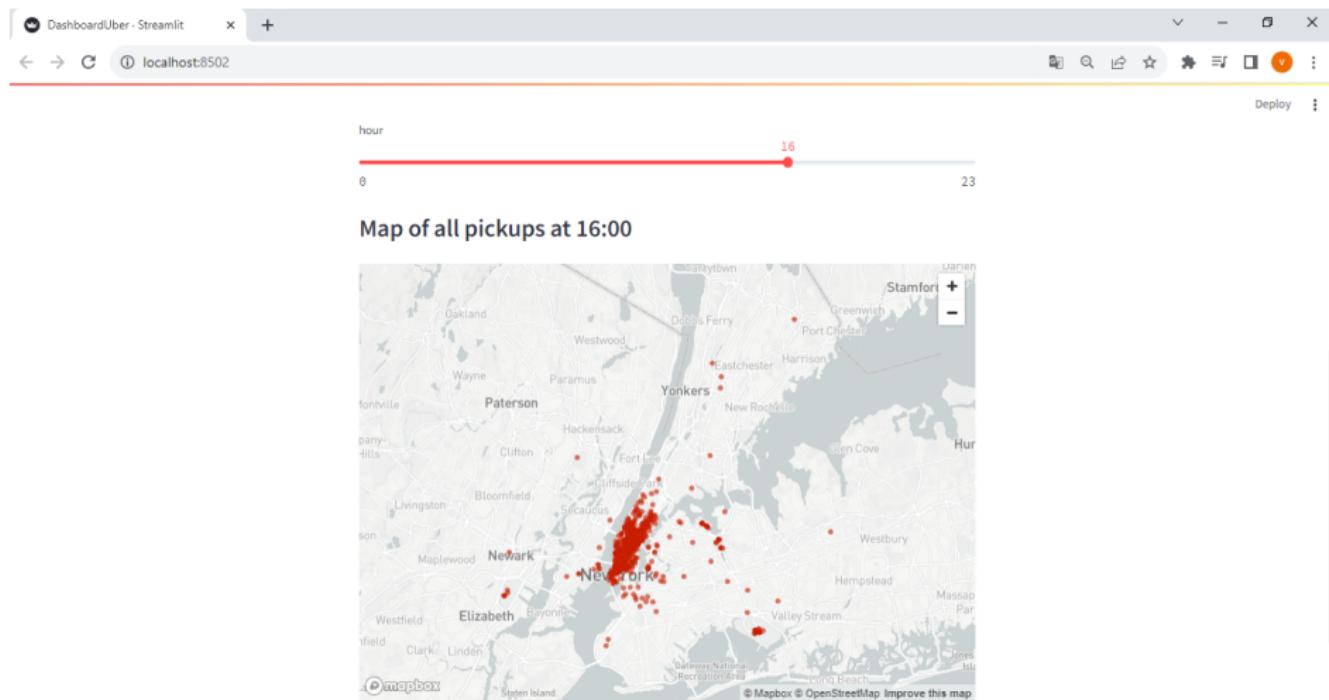
The screenshot shows a Streamlit application running in a browser window. The title of the dashboard is "Uber pickups in NYC". Below the title, there is a message "Done! (using st.cache_data)". There is a checked checkbox labeled "Show raw data". A section titled "Raw data" displays a table with 10 rows of data. The columns are "date/time", "lat", "lon", and "base". The data represents Uber pickup locations in New York City at various times on September 1, 2014.

	date/time	lat	lon	base
0	2014-09-01 00:01:00	40.2201	-74.0021	B02512
1	2014-09-01 00:01:00	40.75	-74.0027	B02512
2	2014-09-01 00:03:00	40.7559	-73.9864	B02512
3	2014-09-01 00:06:00	40.745	-73.9889	B02512
4	2014-09-01 00:11:00	40.8145	-73.9444	B02512
5	2014-09-01 00:12:00	40.6735	-73.9918	B02512
6	2014-09-01 00:15:00	40.7471	-73.6472	B02512
7	2014-09-01 00:16:00	40.6613	-74.2691	B02512
8	2014-09-01 00:32:00	40.3745	-73.9999	B02512
9	2014-09-01 00:33:00	40.7633	-73.9773	B02512

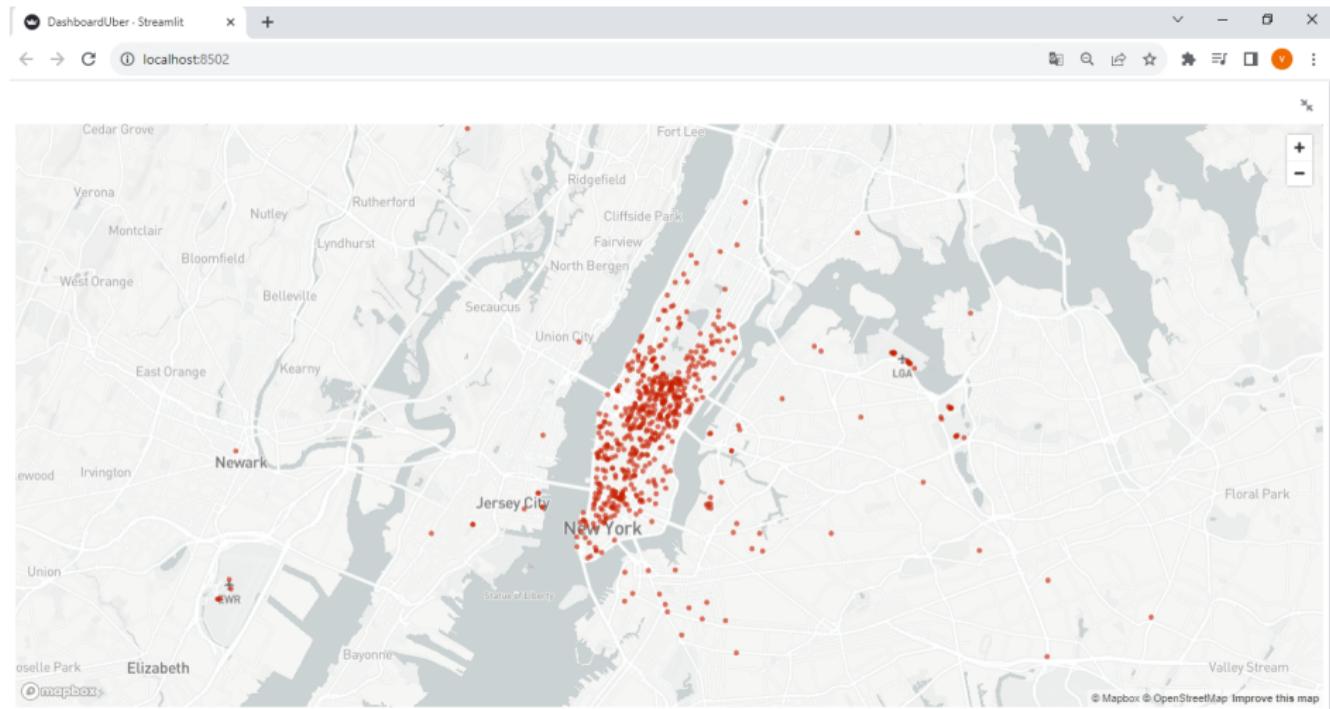
4.3.4.2.1. Dashboard / Tableau: uma interface gráfica



4.3.5.2.2. Dashboard / Tableau: uma interface gráfica



4.3.6.2.2.1. Dashboard / Tableau: uma interface gráfica

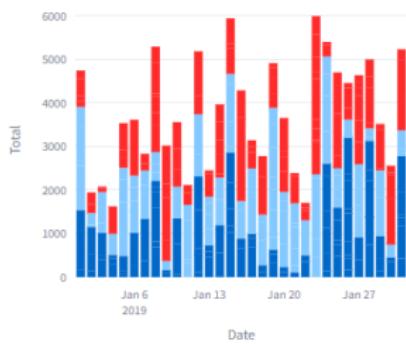


4.3.6.6.1. Dashboard / Tableau: uma interface gráfica

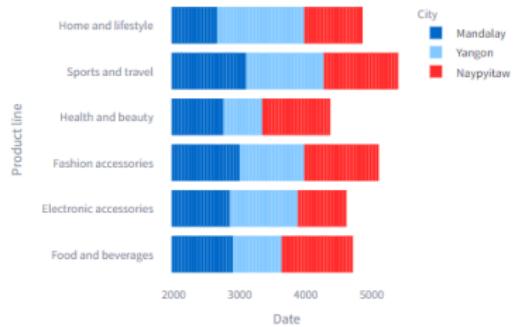
```
1 import streamlit as st
2 import pandas as pd
3 import numpy as np
4
5 st.title('Uber pickups in NYC')
6
7 DATE_COLUMN = 'date/time'
8 DATA_URL = ('https://s3-us-west-2.amazonaws.com/'
9             'streamlit-demo-data/uber-raw-data-sep14.csv.gz')
10
11 @st.cache_data
12 def load_data(nrows):
13     data = pd.read_csv(DATA_URL, nrows=nrows)
14     lowercase = lambda x: str(x).lower()
15     data.rename(lowercase, axis='columns', inplace=True)
16     data[DATE_COLUMN] = pd.to_datetime(data[DATE_COLUMN])
17     return data
18
19 data_load_state = st.text('Loading data ... ')
20 data = load_data(10000)
21 data_load_state.text("Done! (using st.cache_data)")
22
23 if st.checkbox('Show raw data'):
24     st.subheader('Raw data')
25     st.write(data)
26
27 st.subheader('Number of pickups by hour')
28 hist_values = np.histogram(data[DATE_COLUMN].dt.hour, bins=24, range=(0, 24))[0]
29 st.bar_chart(hist_values)
30
31 # Some number in the range 0–23
32 hour_to_filter = st.slider('hour', 0, 23, 17)
33 filtered_data = data[data[DATE_COLUMN].dt.hour == hour_to_filter]
34
35 st.subheader('Map of all pickups at %s:00' % hour_to_filter)
36 st.map(filtered_data)
```

4.3.7. Dashboard no banco: supermarket_sales.csv

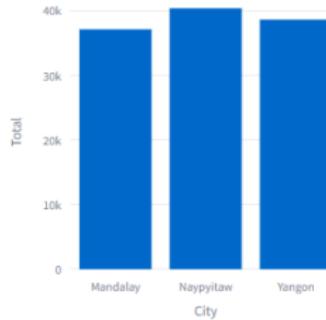
Faturamento por dia



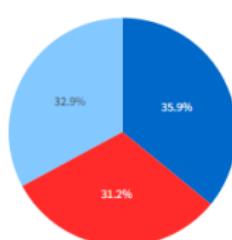
Faturamento por tipo de produto



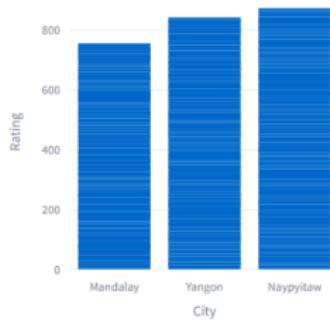
Faturamento por filial



Faturamento por tipo de pagamento



Avaliação



REFERÊNCIAS



[1] José UNPINGCO.

Python Programming for Data Analysis (2020)
Springer.



[2] Paul DEITEL e Harvey DEITEL.

Into to Python for computer science and data science - Learning to program with AI, Big Data and the Cloud (2021)
Pearson Education Limited.



[3] Eric MATTHES.

Curso intensivo de PYTHON - uma introdução prática e baseada em projetos à programação (2016)
Novatec Ltda.



[4] Wes MCKINNEY.

Python for Data Analysis - 1 Ed. (2012)
O'Reilly Media, Inc.



[5] Siegmund BRANDT.

Data Analysis - statistical and computational Methods for Scientists and Engineers
- 4 Ed. (2014).
Springer.



SENAI

Departamento Regional
DE SÃO PAULO
www.sp.senai.br