

Data Science & Big Data

Introdução ao Python 3 para Ciência de Dados

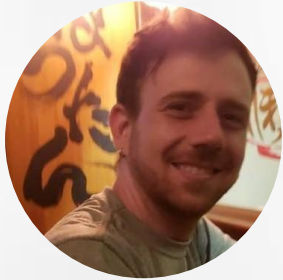
André Grégio



0. Sobre o Curso



Módulo 1 - Professor



- Doutor em Engenharia da Computação.
- Professor-Pesquisador no Departamento de Informática da UFPR.
- Áreas de Interesse e Pesquisa:
 - Segurança Computacional e Ciência de Dados.

Objetivo do Curso

- Aprender a utilizar a linguagem de programação Python para análise de dados.
- Entender os elementos e estruturas básicas da programação
- Criar soluções (programas) para problemas que podem ser resolvidos com o auxílio de um computador

Objetivo do curso

- Abordar ciência de dados de **forma prática**;
- Aplicar os conceitos em problemas reais **corretamente**;
- Incentivar a **exploração** dos dados (compreensão do problema);
- Estudar **técnicas e ferramentas** mais adequadas para problemas específicos;
- Criar consciência sobre organização de dados e **reprodução** de experimentos.

Vamos precisar de...

Um computador convencional

Qualquer sistema operacional!!!

Python 3

Google Colab, no navegador
(<https://colab.research.google.com/>)

ou

VSCode, em seu Desktop/Notebook
(ver tutorial de instalação no Moodle)



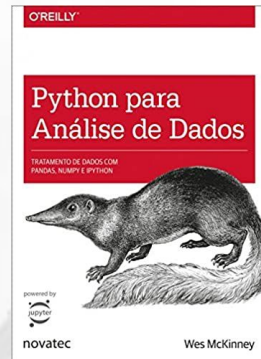
Você precisa...

Ter conhecimentos básicos de usuário:

- Criar diretórios
- Saber o que são arquivos e como organizá-los
- Instalar programas
- Executar comandos

Bibliografia recomendada

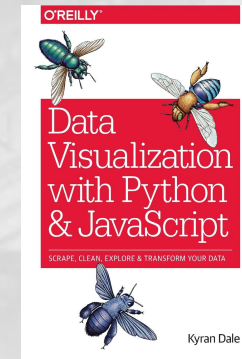
McKinney, W. Python para análise de dados: Tratamento de dados com Pandas, NumPy e IPython. 2019.



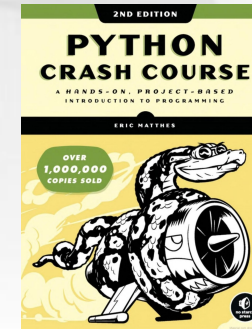
Chen, D. Análise de dados com Python e Pandas. Novatec Editora. 2018.



Dale, K. Data Visualization with Python and JavaScript: Scrape, Clean, Explore & Transform Your Data. O'Reilly Media. 2016.



Matthes, E. Python Crash Course: A Hands-On, Project-Based Introduction to Programming. 2a ed. No Starch Press. 2019.



1.

Introdução a Ciência de Dados



Ciência de Dados

Para quê?
Por quê?
Como?



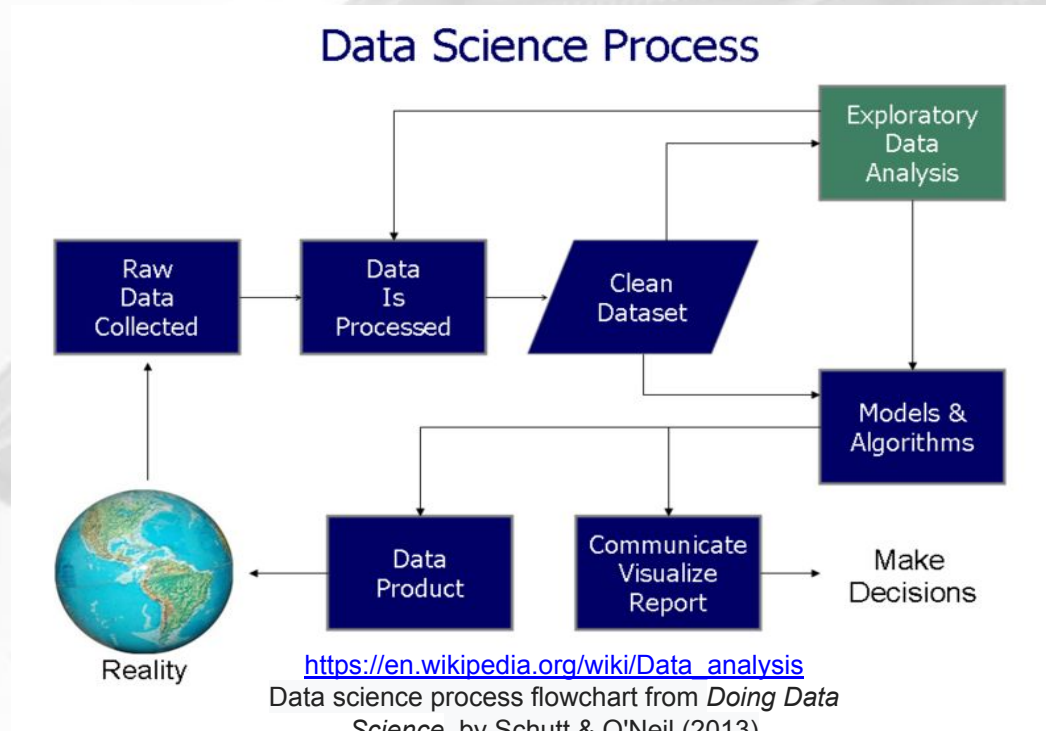
Josh Wills
@josh_wills

Data Scientist (n.): Person who is better at statistics than any software engineer and better at software engineering than any statistician.

1:55 PM · May 3, 2012 · [Twitter Web Client](#)

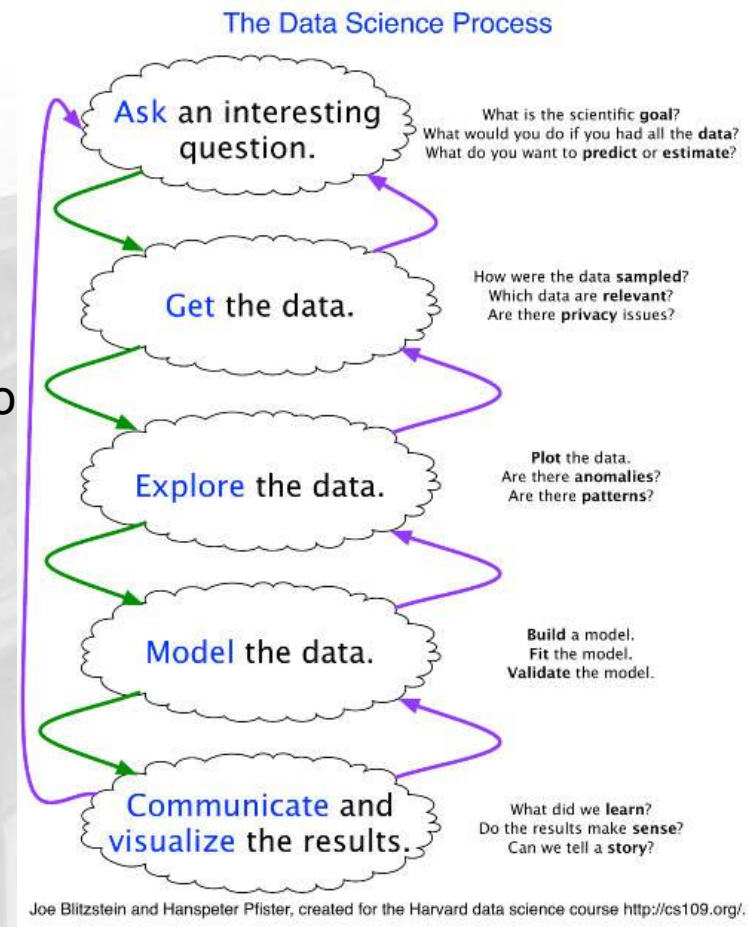


Ciência de Dados (Processo)



Ciência de Dados (Estágios)

1. **Questione** o que você quer fazer e qual o objetivo disso (prever, estimar)
2. **Obtenha** o dado e saiba sua amostragem, relevância e questões como privacidade
3. **Explore**, faça gráficos, entenda seu dado (há padrões, anomalias? Precisa limpar?)
4. **Modele** seu dado, valide e teste o modelo
5. **Apresente** os resultados de maneira que sejam compreensíveis e reproduzíveis



Cientista de dados - Habilidades

1. Consultas SQL e Pipelines robustos
2. Transformação de dados e Engenharia de características
3. Controle de versão (Git)
4. Comunicação de histórias
5. Regressão/Classificação
6. Modelos passíveis de explicação
7. Experimentação comparativa
8. Agrupamento
9. Recomendação (*ranking*)
10. Processamento de Linguagem Natural (NLP)

<https://towardsdatascience.com/10-most-practical-data-science-skills-you-should-know-in-2022-9487d7750e8a>



Cientista de dados: Tipos

Todas as áreas de conhecimento têm necessidade de algum tipo de cientista de dados:

Analista de negócios: estudo do domínio (organizações) para interligar TI e Administração via provimento de soluções!

Engenheiro de dados: criam soluções técnicas, compreendem, analisam e abstraem dados.

Contadores de histórias: encontram a narrativa/visualização; comunicadores visuais.

<https://theaims.ac.in/resources/6-reasons-why-being-a-data-scientist-is-hot.html>

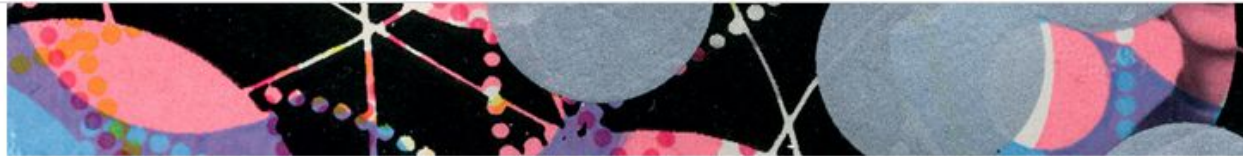


Motivação: Trabalho

https://

Harvard
Business
Review

Data | Data Scientist: The Sexiest Job of the 21st Century



DATA

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

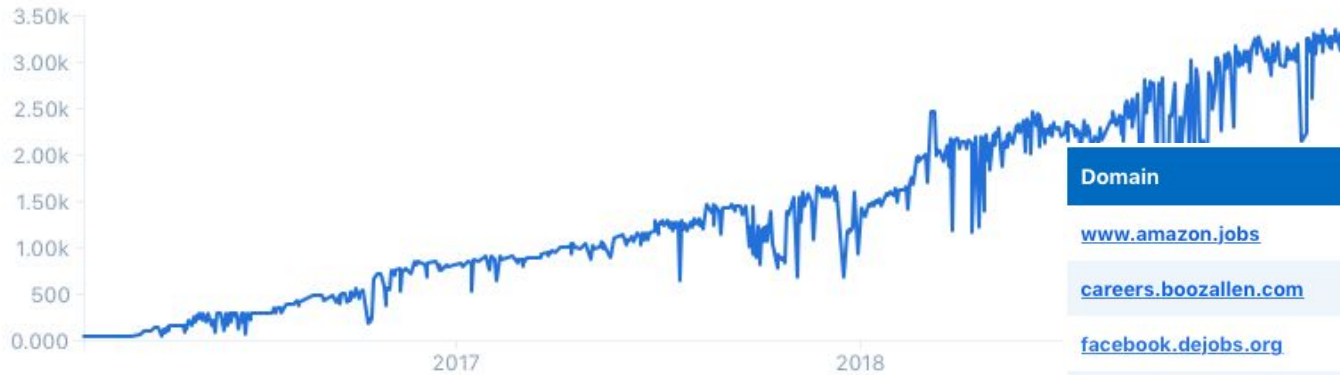
From the October 2012 Issue

 Summary  Save  Share  17 Comment  Print **\$8.95** Buy Copies



Motivação - Demanda

Data Scientist job openings at the world's top companies



Data from Thinknum - [Open dataset](#)

Domain	Title (Count)
www.amazon.jobs	224
careers.boozallen.com	129
facebook.dejobs.org	99
www.capitalonecareers.com	79
tas-accenture.taleo.net	70
careers.walmart.com	65
jobs.jpmorganchase.com	63
jobs.apple.com	56
oracle.taleo.net	49
careers.google.com	48

Motivação - Demanda

Deslocamento entre cientista de dados capaz de:

“Criar um modelo”

vs.

“Extrair conhecimento dos dados”

Capacidade de explicação/compreensão, aplicabilidade em tempo/dados reais!!!



Motivação - Ferramentas

Públicas ou privadas possuem
necessidade de "customização"

Players (alguns) ->

Nicho: Anaconda

Desafiantes: IBM

Líder: SAS

Visionários: Microsoft, Google

Figure 1. Magic Quadrant for Data Science and Machine Learning Platforms



Source: Gartner (February 2020)

<https://www.dominodatalab.com/resources/gartner-magic-quadrant-data-science-platforms/>



Motivação - Segurança

Segurança computacional é totalmente **dependente de dados!**

- Arquivos

- Logs* (registros de auditoria)

- Pacotes de rede

- Artefatos diversos

 - Monitoração de sistemas e aplicações

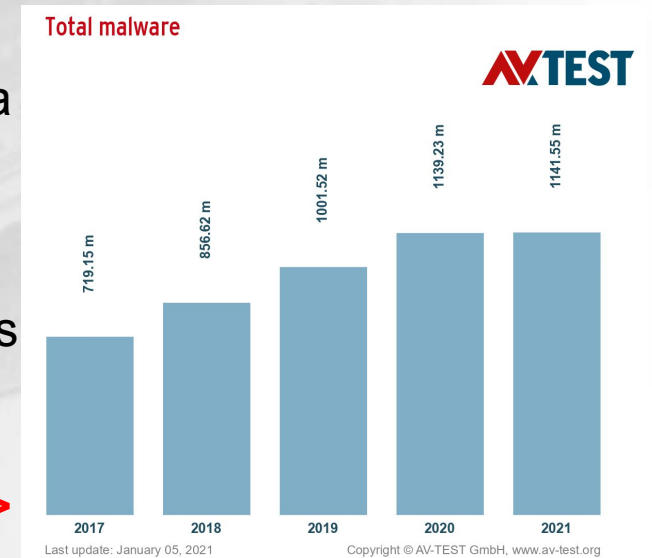
 - Comportamento de usuários

 - Spam/phishing

 - Etc.

Motivação - Segurança

- Abordagem “tradicional”:
- Hashing, assinaturas e heurísticas manualmente definidas
 - Muito específicas, dependentes da experiência do analista, inflexíveis à variações
- Como lidar com o crescimento dramático de ataques cibernéticos?
 - **2008**: ~1M exemplares de malware conhecidos pela comunidade;
 - **2012**: ~100M;
 - **Últimos 5 anos**: ----- bilhões ----->



Tendências

- Processos "tradicionais" de negócios, tais como:
 - Contratação
 - Propaganda
 - Precificação
 - Estratégias

Demandam transformação digital!

- Tecnologia usada para melhorar os processos
 - N x mais rápido, eficaz, automático

- **Ciência de dados**
 - Ajuda a criar valor através dos dados
 - Modelos robustos: menos palpites, mais análise de grandes volumes de dados



Tendências atuais

Ciência de dados **automatizada**

Armazenamento, limpeza, visualização, exploração ->
ainda muito manuais

Soluções (em andamento):

Auto-Data Cleaning (remoção de ruídos)

Feature engineering (escolha de características mais relevantes)

Auto-Machine Learning (modelagem, treino/teste)

Exemplos:

Padrões químicos (descoberta de padrões frequentes, tomada de decisão)

Proveniência de [meta-] dados (autoria, preservação, curadoria)

Correlação de dados para enriquecimento de informações (exploração, agrupamento)



Tendências atuais

<https://towardsdatascience.com/the-4-hottest-trends-in-data-science-for-2020-3956cd9fc182>

<https://www.wired.com/story/billion-records-exposed-online/>

<https://www.troyhunt.com/the-773-million-record-collection-1-data-reach/>

Segurança & Privacidade de dados

- Confiança do cliente
 - **não** vazamento de dados sensíveis...
- "Garantia" de S&P
 - convence o cliente a prover ainda mais dados!
- Conformidade: LGPD/GDPR
- **ANONIMI[ZAÇÃO][DADE]**
- *Adversarial ML*

The 773 Million Record "Collection #1" Data Breach

LILY HAY NEWMAN

SECURITY 11.22.2019 07:00 AM

1.2 Billion Records Found Exposed Online in a Single Server

Here's the next jumbo data leak, complete with Facebook, Twitter, and LinkedIn profiles.



17 JANUARY 2019

ANDY GREENBERG

SECURITY 01.30.2019 05:31 PM

Hackers Are Passing Around a Megaleak of 2.2 Billion Records

The so-called Collections #1–5 represent a gargantuan, patched-together Frankenstein of rotting personal data.

Tendências atuais

Big Data + Cloud Computing

- Organizações (e pessoas) já possuem mais dados do que um PC processa
 - Documentos, fotos, *backups*, aplicativos
- Necessidades:
 - Bancos de dados robustos
 - Processamento paralelo poderoso
 - Busca/Consulta, correlação, agrupamento, edição
- Poder computacional sob demanda:
 - AWS oferece até 96 vCPUs e 768 GB de RAM
 - Google oferece plataforma para *data analytics* (BigQuery)



Tendências atuais

Processamento de Linguagem Natural (NLP)

- Análise e mineração de dados no passado: NÚMEROS!
 - Mais fáceis de se manipular, coletar, tratar (menos opções de variação arbitrária)
 - Ex.: andre, André, ANDRÉ, ANDRE, andré, andreh, Andre, Andreh, Andre'
 - Atributos não-numéricos podem ser convertidos em números
- E análise de texto?
 - Condensa muita informação (mais rica)
- Análise de textos abrange:
 - Sentimentos (qualidade/intuito do texto)
 - Similaridade (conversas comuns)
 - Categorização (diversas classes textuais)



Data Science Process



OBTAIN



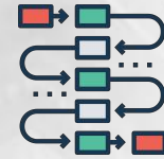
SCRUB



EXPLORE



MODEL



INTERPRET

O

Gather data from
relevant sources

S

Clean data to formats
that machine
understands

E

Find significant patterns
and trends using
statistical methods

M

Construct models to
predict and forecast

N

Put the results into
good use

Originally by Hilary Mason and Chris Wiggins

<https://towardsdatascience.com/5-steps-of-a-data-science-project-lifecycle-26c50372b492>

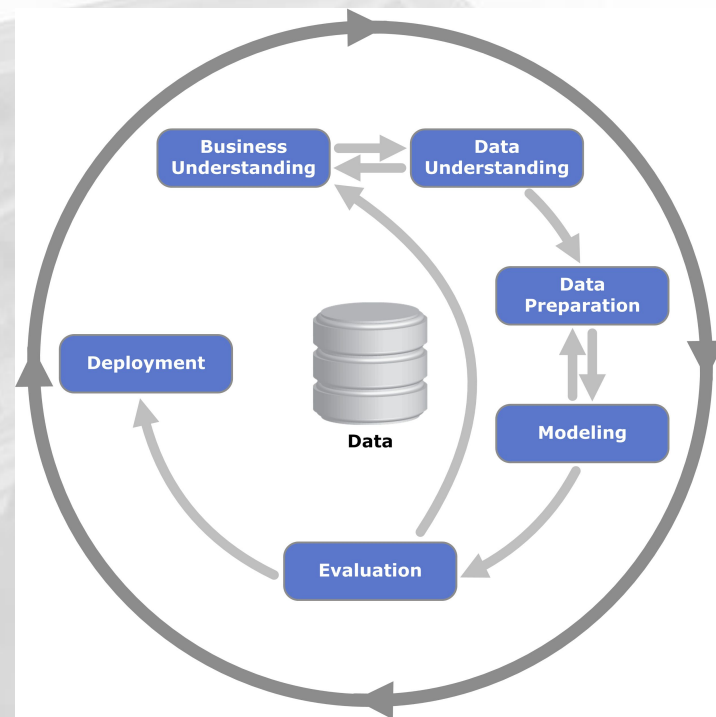
Processo de Ciência de Dados

1. OBTER
 - a. Qual a fonte? Documento digitalizado, SGBD, CSV/JSON/TXT/Binário, objeto real, foto
2. LIMPAR
 - a. Filtrar: eliminar ruído, dado incompleto/corrompido/redundante/irrelevante; Organizar; Substituir; Padronizar => ! "*Garbage in, garbage out*"
3. EXPLORAR
 - a. Inspeccionar o conjunto de dados e suas propriedades; Representação/formato
4. MODELAR
 - a. Construir preditor/classificador/agrupador; reduzir dimensionalidade (só atributos relevantes)
5. INTERPRETAR
 - a. Entender o resultado; visualizar, tirar *insights*; apresentar de maneira compreensível



Recapitulação

- [Ciência|Análise|Mineração] de dados é um **processo** exploratório
- Pensamento crítico-analítico é a melhor ferramenta
- Comunicação deve ser feita via **Moodle**
- Prazos das atividades e questionários devem ser observados rigorosamente!
 - **Em geral, 15 dias para entrega**
 - **Não será feita reabertura por atraso**



https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining



2. Primeiros Passos



Versão do Python

Será usado Python 3 na disciplina

Versão mais recente do Python

Cuidado: Python 3 não é compatível com Python 2, e vice-versa

Versão do Python

Em um terminal digite

```
python --version
```

```
python3 --version
```

O comando que gerar como resposta Python 3.X é o comando que você deve usar em seu computador para acessar o interpretador Python

Interpretador

Para acessar o interpretador Python, digite em um terminal:

`python3`

Exemplo de resultado

Assumindo que no slide anterior o comando que gerou a saída correta foi `python3`, caso contrário, use `python`.

```
python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



Saindo

Para sair do interpretador, insira o comando

```
quit()
```

Interpretador

O interpretador aceita comandos Python, e os executa assim que você tecla enter

```
python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

O comando `print("SEU TEXTO")` imprime na tela o texto solicitado

Interpretador

```
python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("ola mundo")
ola mundo
>>>
```

Realizando Operações

O interpretador é capaz de realizar operações e exibir o resultado
Alguns operadores que podem ser usados:

- + Adição
- Subtração
- / Divisão
- * Multiplicação
- ** Exponenciação
- % Módulo (resto da divisão)



Realizando Operações

```
python3
```

```
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
```

```
[GCC 9.3.0] on linux
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> print("ola mundo")
```

```
ola mundo
```

```
>>> 2**3
```

```
8
```

```
>>> 5%3
```

```
2
```

```
>>> 5%2
```

```
1
```

```
>>> 3+4
```

```
7
```

```
>>>
```

Variáveis

Uma variável **armazena** um valor na memória de trabalho do computador

Para criar uma variável, basta dar um **nome** para ela, seguido de um **=** para atribuir um valor

Variáveis

```
>>> qtde_alunos = 30  
>>> nome = "Maria"  
>>> pi = 3.14
```

Variáveis

```
>>> qtde_alunos = 30  
>>> nome = "Maria"  
>>> pi = 3.14
```

Nome da variável

Valor armazenado

Variáveis

```
>>> qtde_alunos = 30  
>>> nome = "Maria"  
>>> pi = 3.14
```

Na computação, chamamos frases de **strings**. Se a variável deve armazenar uma string, coloque a string entre aspas duplas.

Variáveis

- Cada variável no seu programa deve ter um nome **único**
- Nomes de variáveis **devem**:
 - Conter somente letras, números e underscores (“_”)
 - Começar com letra ou underscore, nunca números
 - Não possuir espaços em branco

Variáveis

- Cada variável no seu programa deve ter um nome **único**
- Nomes de variáveis **devem**:
 - Conter somente letras, números e underscores (“_”)
 - Começar com letra ou underscore, nunca números
 - Não possuir espaços em branco

Exemplo de nomes **inválidos**



```
>>> 1frase = "uma string qualquer"  
>>> frase 1 = "outra string"  
>>> num# = 50  
>>> 2 = 'c'
```



Variáveis

Use nomes descritivos e curtos. Por exemplo, uma variável que vai armazenar a quantidade de alunos na sala pode ser

`qtde_alunos`

Alguns nomes **ruins para essa** variável

`quantidade_de_alunos_na_sala`

`x`

`nave`



Variáveis

Não usar palavras reservadas (keywords) e nomes de funções de Python

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



3. Mais sobre variáveis

O que é uma variável

- Seu computador possui uma memória de trabalho
- No seu computador pessoal, geralmente (e imprecisamente) essa memória é chamada de “Memória RAM”

O que é uma variável

A memória pode ser vista como um vetor
isto é, várias “caixinhas”, uma seguida da outra

Podemos assumir que no começo do programa, nenhuma “caixinha”
está sendo usada

Memória Principal



O que é uma variável

Quanto é criada uma variável, o interpretador pega uma “caixinha” vazia, e dá esse nome para a caixinha. Agora os valores atribuídos à variável são armazenados nela

```
>>> pi = 3.14
```

Memória Principal

pi

3.14						
------	--	--	--	--	--	--

...



Imprimindo

- Você pode imprimir o valor de uma variável usando

```
print(nome_variavel)
```

Não use aspas

- Você pode alterar o valor da variável, ou fazer contas com o valor armazenado na variável

Exemplo

```
>>> qtde_alunos = 200  
>>> print(qtde_alunos)  
200
```

Memória Principal

qtde_alunos

200						
-----	--	--	--	--	--	--

...



Exemplo

```
>>> qtde_alunos = 200  
>>> print(qtde_alunos)  
200  
>>> qtde_alunos = 220  
>>> print(qtde_alunos)  
220
```

Memória Principal

qtde_alunos

220						
-----	--	--	--	--	--	--

...



Exemplo

```
>>> qtde_alunos = 200
>>> print(qtde_alunos)
200
>>> qtde_alunos = 220
>>> print(qtde_alunos)
220
>>> alunos_extr = 30
>>> qtde_alunos = qtde_alunos + alunos_extr
>>> print(qtde_alunos)
250
```

Memória Principal

qtde_alunos alunos_extr

250	30					
-----	----	--	--	--	--	--

...



Tipos de variáveis

- O tipo de uma variável diz o tipo de dado que ela armazena
- Em Python, a **tipagem é dinâmica**

 **Infer automaticamente** qual o tipo da variável na memória

Tipos de variáveis

Para descobrir o tipo de uma variável no momento, use a função

```
>>> type(nome_variavel)
```

Exemplo

```
>>> peso = 10  
>>> type(peso)  
<class 'int'>
```

A variável peso armazena números **inteiros** no momento.

Exemplo

```
>>> peso = 10
>>> type(peso)
<class 'int'>
>>> peso = 10.3
>>> type(peso)
<class 'float'>
```

A variável peso armazena um número em **ponto flutuante** no momento. Um ponto-flutuante é uma **aproximação** para se representar um número racional.

Exemplo

```
>>> peso = 10
>>> type(peso)
<class 'int'>
>>> peso = 10.3
>>> type(peso)
<class 'float'>
>>> nome = "Maria"
>>> type(nome)
<class 'str'>
```

A variável nome armazena uma **string**.

Operações sobre variáveis

Embora subtrair um texto do outro não funcione como esperado, é possível **somar** texto.

```
>>> frase1 = "Curso de"  
>>> frase2 = "Python básico"  
>>> frase1 + ' ' + frase2  
'Curso de Python básico'
```

Esta operação é chamada de **concatenar**!

Operações sobre variáveis

Portanto, algumas das operações mencionadas anteriormente funcionam em variáveis do tipo *string*, realizando a concatenação de palavras, frases ou caracteres:

```
>>> frase1 = "Curso de"  
>>> frase2 = "Python básico"  
>>> frase1 + ' ' + frase2  
'Curso de Python básico'
```

```
>>> letra = 'a'  
>>> print(10*letra)  
aaaaaaaaaa
```

4. Criando Scripts

Scripts

Um script é uma série de comandos que são executados **em ordem, de cima para baixo**

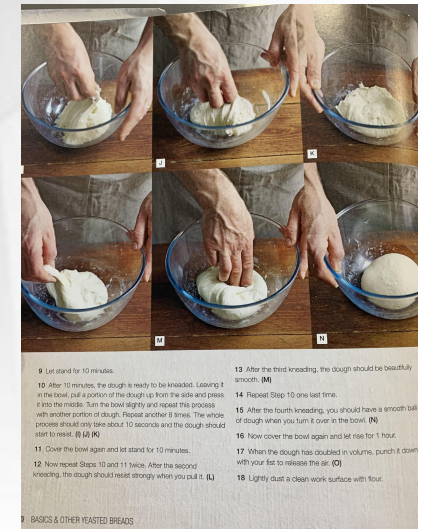
Em outras palavras, estamos criando um **programa em Python**

Ou como preferimos na computação, estamos implementando um **algoritmo**

Breve revisão de algoritmos

Procedimento computacional bem definido para:

1. Sistematizar instruções/**atividades repetitivas**
2. Obter uma **solução** para um determinado tipo de **problema**



Breve revisão de algoritmos

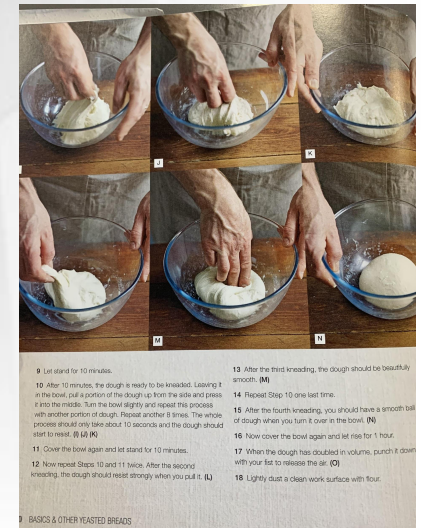
Procedimento computacional bem definido para:

1. Sistematizar instruções/**atividades repetitivas**
2. Obter uma **solução** para um determinado tipo de **problema**

a. Recebe algum valor (ou conjunto de valores) de **ENTRADA**

b. Produz algum valor (ou conjunto de valores) de **SAÍDA**

Algoritmo é uma **sequência de passos** que transforma a **ENTRADA** na **SAÍDA** e deve poder ser reproduzido!



Scripts

Um script é um arquivo de texto com comandos

Utilize **qualquer editor de texto simples** (e.g., notepad) para criar o arquivo

IDE

Alguns editores possuem mais “recursos” apropriados para programação. Esse tipo de editor também é chamado de IDE (Integrated Development Environment):

Exemplos com suporte para python

- Visual Studio Code (code.visualstudio.com)
- MU (codewith.mu)
- Sublime (www.sublimetext.com)
- Geany (www.geany.org)
- IDLE (vem com o Python no Mac OS e Windows, apt no Linux)
- PyCharm (www.jetbrains.com/pycharm)

IDE

Objetivos do Editor de Texto/IDE:

- Criar programas completos
- Salvar os programas para execução posterior
- *Highlighting* para facilitar leitura (cores para comandos/palavras reservadas)
- Execução e depuração em tempo real
- Automação, atalhos
- Adição de funcionalidades via *plugins*

Primeiro Script

- Crie um diretório em seu computador, em um lugar de sua preferência
 - Esse diretório vai conter o **seu projeto**

Dicas:

- Crie em um local fácil para você lembrar e acessar
- Use o tutorial disponibilizado neste curso

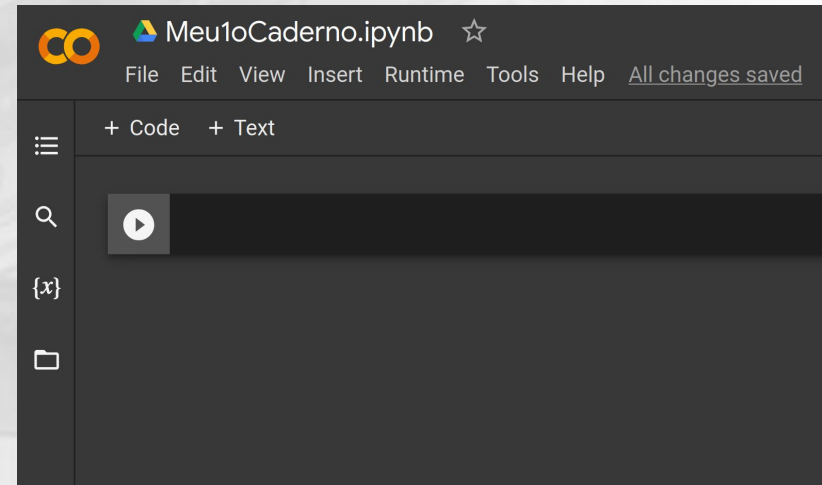
Primeiro Script

Podemos também fazer com Google Colab, mas em vez de um *script*, você estará criando um *Python Notebook*

- Interpreta célula por célula
- Permite integração com *Drive*
- Possibilita exportar tanto em
 - *.ipynb* (*notebook*)

quanto em

- *.py* (*script*)

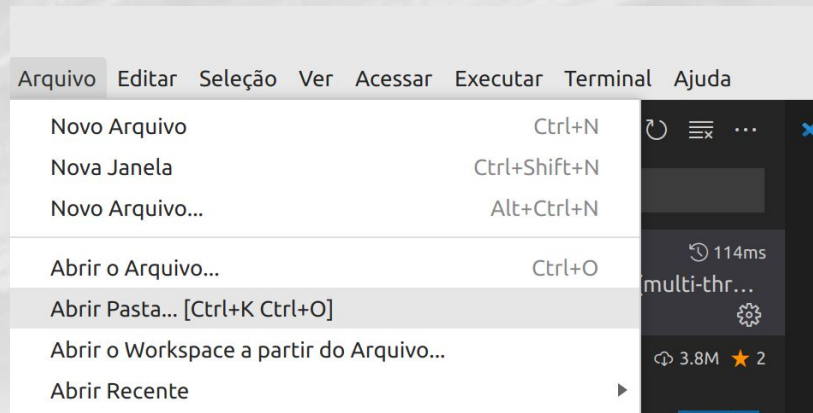


Primeiro Script

Abra o VSCode

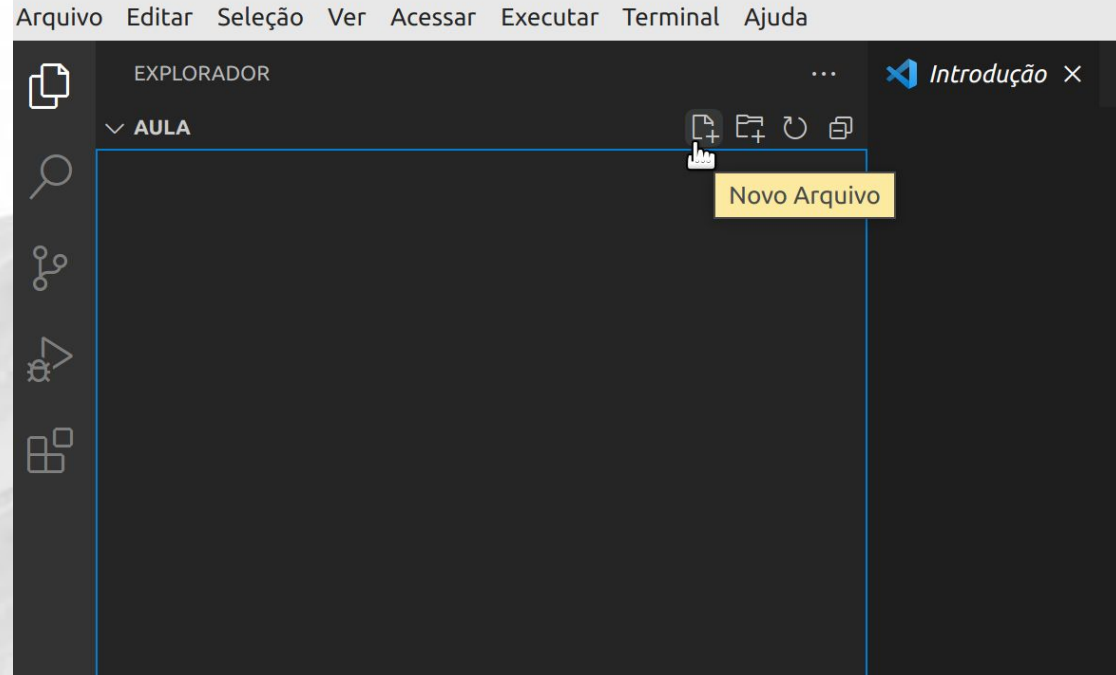
Menu *Arquivo* -> *Abrir Pasta ...*

Abra a pasta recém criada



Primeiro Script

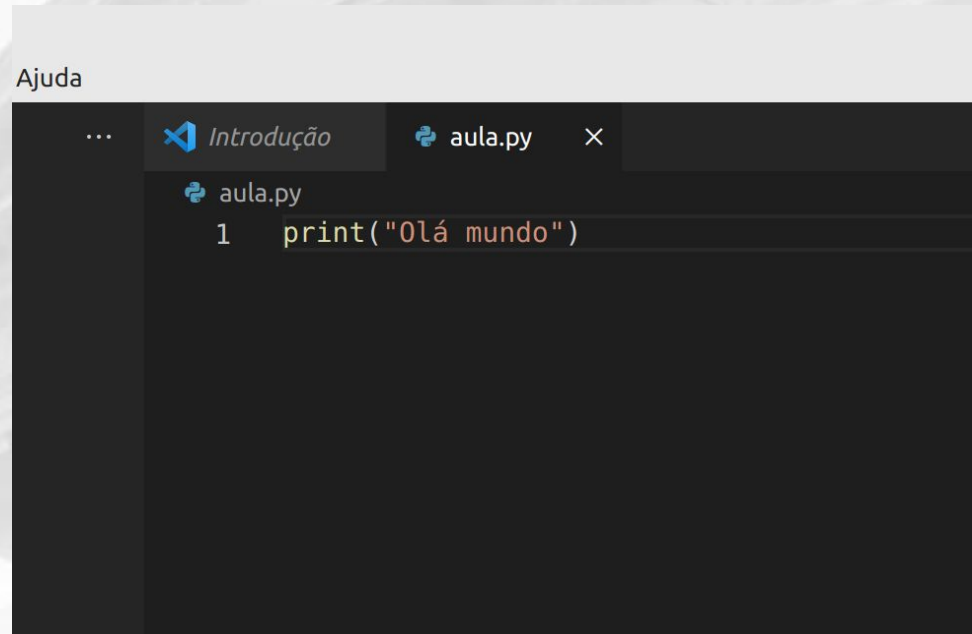
Crie um arquivo chamado aula.py



Em Python, é obrigatório que arquivos de script terminem com a extensão .py

Primeiro Script

Insira o seguinte comando no arquivo aula.py

A screenshot of a code editor window. The title bar at the top says "Ajuda". Below it, there are two tabs: "Introdução" with a blue icon and "aula.py" with a Python icon. The "aula.py" tab is active. The code editor shows a single line of Python code:

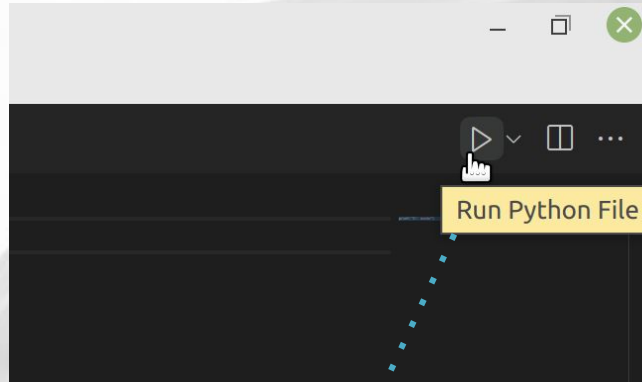
```
1 print("Olá mundo")
```

 The text is white on a dark background, with line numbers on the left.

```
Ajuda
... Introdução aula.py x
aula.py
1 print("Olá mundo")
```


Primeiro Script

Execute



```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  Python + - [ ] [X] ^ X
PS C:\Users\Andre Gregio\Documents\UFPR\CursoPython> & "C:/Users/Andre Gregio/AppData/Local/Programs/Python/Python310/python.exe" "c:/Users/Andre Gregio/Documents/Olá mundo!"
PS C:\Users\Andre Gregio\Documents\UFPR\CursoPython> 
```



Script x Executar no Interpretador

O interpretador Python opera em modo **REPL**:

Read

Evaluate

Print

Loop

O *script* é executado como uma sequência dos comandos, sem pausas (a menos que haja necessidade de interação, **como o teclado**, ou erros)

5.

Leitura de Dados Externos

Lendo do Teclado

Para solicitar um dado do usuário e armazenar o que ele digitou em uma variável, usa-se a função *input*

```
>>> variavel_destino = input("Texto opcional para exibir")
```

Lendo do Teclado

```
print("Olá mundo!")  
nome = input("Digite seu nome: ")  
print("Seu nome é", nome)
```

Lendo do Teclado

```
print("Olá mundo!")
nome = input("Digite seu nome: ")
print("Seu nome é", nome)
idade = input("Digite sua idade: ")

idade_apos = input("Com quantos anos você quer aposentar: ")
anos_apos = idade_apos - idade
print("Faltam", anos_apos, " para você aposentar")
```

Lendo do Teclado

```
print("Olá mundo!")
nome = input("Digite seu nome: ")
print("Seu nome é", nome)
idade = input("Digite sua idade: ")

idade_apos = input("Com quantos anos você quer aposentar: ")
anos_apos = idade_apos - idade
print("Faltam", anos_apos, " para você aposentar")
```

Oops!

Traceback (most recent call last):

File "/home/usuario/aula/aula.py", line 6, in <module>

anos_apos = idade_apos - idade

TypeError: unsupported operand type(s) for -: 'str' and 'str'



Conversão de tipos (*casting*)

Python assume que tudo que é lido do teclado é uma **string**, isto é, um texto ou uma **sequência de caracteres**

```
idade = input("Digite sua idade: ")  
idade_apos = input("Com quantos anos você quer aposentar: ")  
anos_apos = idade_apos - idade
```

Subtrair um texto de outro não faz sentido!



Conversão de tipos (*casting*)

Se você quer que a variável armazene o dado como outro tipo que não string, leia da seguinte forma:

```
>>> variavel_destino = tipo(input("Texto opcional para exibir"))
```

- Substitua *tipo* pelo tipo correto, como int ou float
 - Isso fará uma **conversão** de string para o tipo solicitado

Exemplo

```
print("Olá mundo!")
nome = input("Digite seu nome: ")
print("Seu nome é", nome)
idade = int(input("Digite sua idade: "))

idade_apos = int(input("Com quantos anos você quer aposentar: "))
anos_apos = idade_apos - idade
print("Faltam", anos_apos, " para você aposentar")
```

6.

Você não precisa de uma IDE



Executando o script

- Uma IDE como o VSCode facilita (muito) a criação e execução do script, mas você não precisa dele
- Um *script* nada mais é do que um arquivo de texto

Executando o script no Linux

- Abra um terminal onde o arquivo está salvo
- Para executar o script sem precisar de nenhuma IDE, basta executar o comando

```
python3 nome_seu_script.py
```

A terminal window with a dark background. The title bar shows 'gregio@R2D2:~/Documents/UFPB/DSBD2023'. The prompt is 'gregio@R2D2:~/Documents/UFPB/DSBD2023 \$'. The command 'python3 script1.py' has been entered. The output shows 'Olá mundo!' followed by a prompt 'Digite seu nome:'.

```
gregio@R2D2:~/Documents/UFPB/DSBD2023 $ python3 script1.py
Olá mundo!
Digite seu nome:
```

Executando o script no Windows

PowerShell

```
Windows PowerShell

PS C:\Users\abedg\UFPR> more .\teste.py.txt
a = 1
b = 2
print(a+b)

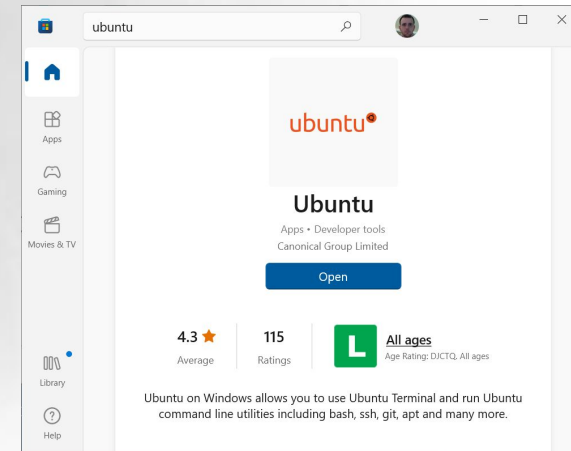
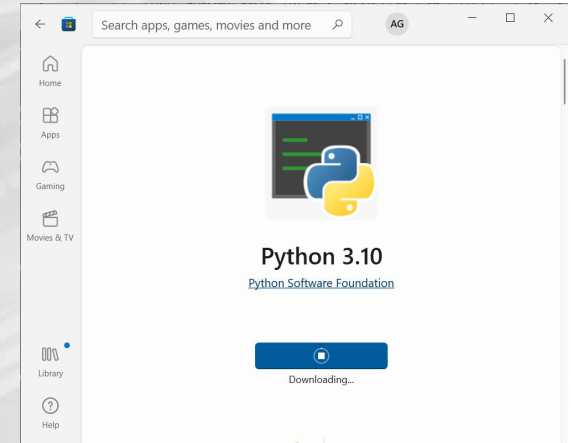
PS C:\Users\abedg\UFPR> python3.exe .\teste.py.txt
3

PS C:\Users\abedg\UFPR>
```

Ubuntu no Windows

```
gregio@DESKTOP-RMOQKI2: ~
gregio@DESKTOP-RMOQKI2:~$ cat teste.py
#!/usr/bin/python3

print("oi")
gregio@DESKTOP-RMOQKI2:~$ python3 teste.py
oi
gregio@DESKTOP-RMOQKI2:~$
```



Leitura da Entrada Padrão

O módulo “sys” provê acesso a funções para interagir com o interpretador, como ler argumentos da entrada padrão.

Para um programa receber o valor “5” via linha de comando, como abaixo

```
$ python meu_script.py 5  
Você digitou o número 5.
```

O interpretador tem que obter o argumento dado e processá-lo dentro do script.



Leitura da entrada padrão

```
$ python meu_script.py 5  
Você digitou o número 5.
```

← Terminal

← Código-fonte

```
import sys  
entrada = sys.argv[1]  
print("Você digitou o número {}".format(entrada))
```


Leitura da entrada padrão

```
$ python meu_script.py 5  
Você digitou o número 5.
```

```
import sys  
entrada = sys.argv[1]  
print("Você digitou o número  
{0}.".format(entrada))
```

lista de argumentos da
linha de comando
passada para o script
Python!

Leitura da entrada padrão

índice 0	índice 1	índice 2	...	índice N-1	...
arg1	arg2	arg3	...	argN	...

lista de argumentos
da linha de comando
passada para o script
Python!

Leitura da entrada padrão

primeiro elemento
(índice 0) é o próprio
programa!

```
$ python meu_script.py 5 -15 nome 0 sim  
print(sys.argv)
```

```
['meu_script.py', '5', '-15', 'nome', '0',  
'sim']
```

Leitura da entrada padrão/função

```
import sys
def fat(n):
    res = 1
    for i in range(2,n+1):
        res *= i
    print(res)
```

```
n = int(sys.argv[1])
fat(n)
```

Leitura de arquivos via *stdin*

Passo a passo:

1. Considere um arquivo de texto
 - 1.1. Receba o caminho do arquivo
 - 1.2. Abra o arquivo
 - 1.3. Leia o arquivo linha por linha

Leitura de arquivos via *stdin*

Passo a passo:

1. Considere um arquivo de texto

- 1.1. Receba o caminho do arquivo (`$ python3 processa_texto.py /Docs/poesia.txt`)

```
arq_nome = sys.argv[1]
```

- 1.2. Abra o arquivo

- 1.3. Leia o arquivo linha por linha

Leitura de arquivos via *stdin*

Passo a passo:

1. Considere um arquivo de texto

- 1.1. Receba o caminho do arquivo (\$ python3 processa_texto.py [/Docs/poesia.txt](#))

```
arq_nome = sys.argv[1]
```

- 1.2. Abra o arquivo

- 1.3. Leia o arquivo linha por linha

Leitura de arquivos via *stdin*

Passo a passo:

1. Considere um arquivo de texto

- 1.1. Receba o caminho do arquivo (`$ python3 processa_texto.py /Docs/poesia.txt`)

```
arq_nome = sys.argv[1]
```

- 1.2. Abra o arquivo

```
arq_handle = open(arq_nome)
```

- 1.3. Leia o arquivo linha por linha

Leitura de arquivos via *stdin*

Passo a passo:

1. Considere um arquivo de texto

- 1.1. Receba o caminho do arquivo (`$ python3 processa_texto.py /Docs/poesia.txt`)

```
arq_nome = sys.argv[1]
```

- 1.2. Abra o arquivo

```
arq_handle = open(arq_nome) # aberto para leitura!
```

- 1.3. Leia o arquivo linha por linha

```
texto = arq_handle.read()
```



Leitura de arquivos via *stdin*

Passo a passo:

1. Considere um arquivo de texto

1.1. Receba o caminho do arquivo (`$ python3 processa_texto.py /Docs/poesia.txt`)

```
arq_nome = sys.argv[1]
```

1.2. Abra o arquivo

```
arq_handle = open(arq_nome) # aberto para leitura!
```

1.3. Leia o arquivo linha por linha

```
texto = arq_handle.read() # lê para uma string, não linha a linha...
```



Leitura de arquivos via *stdin*

```
import sys

arq_nome = sys.argv[1] # caminho para arquivo
arq_handle = open(arq_nome) # abre para leitura
texto = arq_handle.read() # lê para uma string

print(texto)
print(type(texto))
```

Leitura de arquivos via *stdin*

```
$ python3 processa_texto.py texto.txt
```

```
linha 1
```

```
linha 2
```

```
linha 3
```

```
<class 'str'>
```

Leitura de arquivos via *stdin*

```
$ python3 processa_texto.py texto.txt
```

```
linha 1  
linha 2  
linha 3  
<class 'str'>
```

texto.txt
linha 1
linha 2
linha 3

```
import sys
```

```
arq_nome = sys.argv[1] # caminho para arquivo  
arq_handle = open(arq_nome)# abre para leitura  
texto = arq_handle.read() # lê para uma string  
print(texto)  
print(type(texto))
```

Leitura de arquivos via *stdin* - read()

```
import sys

texto = open(sys.argv[1]).read()
linhas = texto.split("\n")
print(linhas)
print(type(linhas))
```

Leitura de arquivos via *stdin* - read()

```
import sys

texto = open(sys.argv[1]).read()
linhas = texto.split("\n")
print(linhas)
print(type(linhas))
```

```
# texto.txt
linha 1
linha 2
linha 3
```

```
$ python3 processa_texto.py texto.txt
['linha 1', 'linha 2', 'linha 3', '']
<class 'list'>
```

Leitura de arquivos via *stdin* - `readline()`

```
import sys

texto = open(sys.argv[1]).readline()
linhas = texto.split("\n")
print(linhas)
print(type(linhas))
```


Leitura de arquivos via *stdin* - `readline()`

```
import sys

texto = open(sys.argv[1]).readline()
linhas = texto.split("\n")
print(linhas)
print(type(linhas))
```

```
# texto.txt
linha 1
linha 2
linha 3
```

```
$ python3 processa_texto.py texto.txt
linha 1
<class 'str'>
```

Leitura de arquivos via *stdin* - `readlines()`

```
import sys

texto = open(sys.argv[1]).readlines()
linhas = texto.split("\n")
print(linhas)
print(type(linhas))
```

Leitura de arquivos via *stdin* - `readlines()`

```
import sys

texto = open(sys.argv[1]).readlines()
linhas = texto.split("\n")
print(linhas)
print(type(linhas))
```

```
# texto.txt
linha 1
linha 2
linha 3
```

```
$ python3 processa_texto.py texto.txt
['linha 1\n', 'linha 2\n', 'linha 3\n']
<class 'list'>
```

Leitura de arquivos via *stdin* - composição

```
import sys

texto = open(sys.argv[1]).read().splitlines()
print(texto)
print(type(texto))
```

Leitura de arquivos via *stdin* - composição

```
import sys

texto = open(sys.argv[1]).read().splitlines()
print(texto)
print(type(texto))
```

```
$ python3 processa_texto.py texto.txt
['linha 1', 'linha 2', 'linha 3']
<class 'list'>
```

```
# texto.txt
linha 1
linha 2
linha 3
```



Leitura de arquivos via *stdin*

```
import sys

arq = open(sys.argv[1])
texto = arq.read().splitlines()
print(texto)
print(type(texto))
arq.close()
```

Lembrar de
fechar o
arquivo!

Escrita de arquivos (modos)

```
arq_saida = open(sys.argv[2], 'w')
```

- 'w' abre o arquivo em modo *escrita*
- Se o arquivo **não** existir, cria um novo
- Senão, sobrescreve!

Escrita de arquivos (modos)

```
arq_saida = open(sys.argv[2], 'a')
```

- 'a' abre o arquivo em modo *append*
- Se o arquivo **não** existir, cria um novo
- Senão, insere no final!

7.

Importação de novas funcionalidades



Importando

- É possível **importar** bibliotecas com funções já prontas
 - Existem diversas formas
- Dica: a **primeira** coisa presente no seu script devem ser as importações, no formato:

```
import nome_arquivo
```

Math

- A biblioteca math adiciona diversas funções relacionadas a operações matemáticas
- Veja a documentação
 - docs.python.org/3/library/math.html

Raiz Quadrada

```
import math

valor = 4
raiz = math.sqrt(valor)
print("A raiz de ", valor, "é", raiz)
```

Raiz Quadrada

```
import math  
  
valor = 4  
raiz = math.sqrt(valor)  
print("A raiz de ", valor, "é", raiz)
```

A função `sqrt` da biblioteca `math` vai calcular a raiz quadrada da variável `valor`, e armazenar o resultado na variável `raiz`.



Documentação

Se você precisar saber mais sobre uma função, você pode:

- Procurar nos livros recomendados
 - [Think Python 2e – Green Tea Press](#)
 - (Em português: <https://penseallen.github.io/PensePython2e/>)
- Procurar em na documentação oficial
 - **<https://docs.python.org/pt-br>**
- Procurar na Internet, tomando cuidado com o *site*
 - Um *site* bastante utilizado pela comunidade em geral é o **<https://pt.stackoverflow.com/>**



Organização de código

Um documento interessante é o guia de estilo do Python:

<https://www.python.org/dev/peps/pep-0008/>

Organização de código

Comentários são importantes para “organizar o código”.
Em Python, um comentário de código começa com #

```
#A linha abaixo lê o nome do usuário da entrada padrão  
nome = input("Digite seu nome: ")
```


Estilo de codificação

Python é extremamente dependente de **IDENTAÇÃO**

- ▶ Funções e blocos **NÃO** usam chaves...
- ▶ Comandos e funções requerem ":" ao final (veremos depois)
- ▶ Pode-se usar <TAB> ou espaços para criar níveis:
 - ▶ Separação de funções, blocos ou aninhamentos
- ▶ Comentários:
 - ▶ Linha → # um comentário
 - ▶ Múltiplas linhas → """ texto em várias linhas """

Meu primeiro .py

- ▶ Em um editor de texto qualquer, abra um arquivo novo e o nomeie com a extensão do Python, por exemplo “teste.py”:

1) `# meu primeiro programa em Python`

2) `print("oi")`

- ▶ Comentários precedidos por “#”
- ▶ Execução na linha de comando: `python teste.py`



Função básica - print()

O comando *print()*:

- ▶ Sintaxe:

- ▶ `print(valor1, valor2, ..., valorN)`
- ▶ Se nenhum argumento é passado, imprime uma linha em branco
- ▶ Os valores a serem impressos são separados por um espaço:
 - `print("Resposta:", 6 * 7, "!!!")`

Condicionais básicos

- ▶ if (*condição*):
 - ▶ Instrução 1
 - ▶ Instrução 2
- ▶ elif (*outra condição*):
 - ▶ Instrução 3
- ▶ else:
 - ▶ Instrução 4
- ▶ Instrução 5 (fora do IF-ELIF-ELSE)

Laços básicos

- ▶ `for i in range(5):`
 - ▶ `print("%d " %i)`
 - ▶ 0
 - ▶ 1
 - ▶ 2
 - ▶ 3
 - ▶ 4
- ▶ `while True:`
 - ▶ `print("laço infinito!")`

Recomendações Finais

Exercícios

1. Replique tudo que foi ensinado durante a aula para fixar os conhecimentos.
2. Resolva os exercícios postados no site da disciplina para testar os seus conhecimentos.
3. Resolva os exercícios dos próximos *slides*

Exercício 1

Faça um programa que pergunta o peso (em kg) e a altura (em metros) do usuário. O programa deve calcular o Índice de Massa Corpórea (IMC) do usuário, que é dado por $\text{peso}/(\text{altura} \times \text{altura})$ e exibir o valor de IMC na tela.

Dica: use float para ler o input. Para separar as casas decimais quando estiver digitando, utilize ponto (e.g., digite 1.8 para uma altura de um e oitenta). Isso pode ser diferente em alguns sistemas. Por exemplo, alguns terminais de sistemas Mac utilizam vírgula para separar as casas decimais.

Exercício 2

Escreva um programa que solicita o raio de um círculo, e exibe o perímetro e a área desse círculo.

Para calcular a área, utilize o operador de exponenciação `**`.

Exercício 3

Faça um programa que leia os coeficientes a , b e c de uma equação de segundo grau. O programa deve calcular por Bhaskara as raízes dessa equação. Para esse problema, assuma que a equação sempre vai ter raízes reais (o usuário não vai digitar valores a , b e c que levam a uma equação que não possui raízes). Caso a equação possua apenas uma raiz, como a equação x^2 , repita a raiz na resposta duas vezes.

Exemplo de execução do programa:

```
Digite a: 2
Digite b: 0
Digite c: 0
Raíz 1: 0
Raíz 2: 0
```

Exercício 4

Dado um retângulo de lado $a=7$ e área 63, escreva:

- Uma instrução em Python que calcule o valor do lado b .
- Uma instrução que calcule a diagonal deste retângulo.
- Uma função que calcule a área e a diagonal de qualquer retângulo, dados os valores dos lados a e b .

Dúvidas?

Faça seu Quiz no Moodle e...

Até a próxima aula!

