**Python Fundamentos para Análise de Dados**

**Capitulo 1 – Introdução**

Por que Todos Deveriam Aprender a Programar? 15/05/2020

“Everybody in this country should learn how to program a computer...because it teaches you how to think.” – Steve Jobs

Visit code.org

Introdução – Parte 1/2 (18/05/2020)

Instrutor: Daniel Mendes

Email: danielm@dsacademy.com.br

Introdução – Parte 2/2 (18/05/2020)

Navegando pela Data Science Academy

A Inteligência está nas Maquinas ou nos Desenvolvedores? (18/05/2020)

‘...computadores são...qualquer coisa que possua uma unidade de processamento programável ...’

A parte inteligente dos smartphones não está no chip de computador e sim nas pessoas que o programaram.

‘...o que está por trás da inteligência artificial são algoritmos criados por desenvolvedores de softwares...os programadores são os responsáveis em dar inteligência aos dispositivos...’

O que é Programação de Computadores?

Programação é o processo de analisar uma tarefa, dividi-la em pedaços ou etapas e escrever uma sequência de instruções que o computador possa entender e executar.

Conhecendo a Linguagem Python

Lançada em 1991, na Holanda, por Guido Van Rossum

Linguagem interpretada

Orientada a Objetos

Portável

Comunidade Ativa

[www.python.org](http://www.python.org)

Por que Cientistas de Dados escolhem Python?

‘...uso geral...isso significa que podemos desenvolver praticamente qualquer tipo de software utilizando a linguagem...coletar, limpar transformar, pré-processar, criar modelo preditivo, avaliar modelo preditivo, criar dashboards...’

‘...Python, R, Scala e Java, são as principais linguagens utilizadas em Data Science...’

Python se tornou uma linguagem de programação que permite criar um canal direto para análise de dados.

Razões da escolha por Python:

- Grande Comunidade;

- Bibliotecas de Análise de Dados (mais de 100 mil);

- Jupyter Notebook (permite programar via browser);

- Facilidade de Aprender;

- Escalabilidade e Portabilidade;

- PyData Satck

(conjunto de pacotes/bibliotecas criada por terceiros e disponibilizadas gratuitamente)

‘...Python é uma linguagem poderosa para análise de dados...uma das bibliotecas mais famosas é o NumPy (Numerical Python), é a base de basicamente tudo que fazemos em Machine Learning e IA...Pandas é uma espécie de Excel para Python’

Keep Calm and Code Python

O Incrível Crescimento da Linguagem Python

Nenhuma linguagem de programação cresceu tanto quanto a linguagem Python nos últimos 6 anos.

E Python é a linguagem de programação que mais deve crescer nos próximos anos.

A linguagem Python também se destaca em relação a várias outras linguagens (e este gráfico não inclui o TensorFlow, a principal biblioteca para construção de modelos de Machine Learning, amplamente usada com Python)

E nos países emergentes (como o Brasil), Python também tem a maior projeção de crescimento entre as principais Linguagens de Programação.

Aprender programação é uma excelente decisão na sua carreira e aprender Python pode ser um excelente caminho!

Python 2 X Python 3

‘...sempre estaremos estudando com a versão 3...’

Anaconda Python

‘...é uma espécie de distribuição...o interpretador padrão não vem com os pacotes instalados...’

IDE´s Python – Anaconda, PyCharm, Rodeo, Spyder e Google Colab

Editor de texto Sublime

Rodeo tem interface muito parecida com RStudio.

‘...Google Colab – como se fosse utilizar a máquina de alguém...tem um GPU simples para testes...’

Instalando o Anaconda Python no Windows 10 (19/05/2020)

.....

Configurando o Anaconda Python Para Executar os Arquivos do Curso (19/05/2020)

‘...Programa 7-Zip recomendo...’

C:\Users\victor>cd Documents

C:\Users\victor\Documents>cd PythonFundamentos

C:\Users\victor\Documents\PythonFundamentos>cd Cap01

C:\Users\victor\Documents\PythonFundamentos\Cap01>jupyter notebook

Trabalhando com o Jupyter Notebook (19/05/2020)

Usando o Interpretador Python de Forma Interativa

C:\Users\victor\Documents\PythonFundamentos\Cap01>dir

dir - mostra conteúdo do diretório

C:\Users\victor\Documents\PythonFundamentos\Cap01>cls

cls - limpa tela

C:\Users\victor\Documents\PythonFundamentos\Cap01>python hello.py

python - é o nome do interpretador, assim chama/abre o interpretador

python hello.py – chama/abre o interpretador E executa o arquivo/script hello.py

Executando script via linha de comando:

C:\Users\victor\Documents\PythonFundamentos\Cap01>python eduTeste.py

Olá, Mundo!

5

Use exit() or Ctrl-Z plus Return to exit

Usando o Google Colab

Usar o Jupyter Notebook online, nenhuma instalação é requerida

Google Colaboratory

\*

Pesquisar quais são os: Principais conceitos matemáticos e conceitos estatísticos

**Capítulo 2 – Variáveis, Tipos e Estruturas de Dados (20/05/2020)**

Introdução

‘...levar em consideração que nunca viu python...’

O que estudaremos nesse capitulo?

- Python Fundamentos

- Números e Operações Matemáticas

- Variáveis e Operadores

- Strings

- Estruturas de dados

. Listas

. Dicionários

. Tuplas

Quiz, Exercícios e Lab01

Por que Isso é Importante?

‘...Leia os anúncios no Fórum de discussão do curso...’

Python Fundamentos

Python é uma linguagem interpretada, clara, de fácil leitura e bastante expressiva

Código Fonte (m.py) -> Byte Code (m.pyc) -> Runtime (PVM)

Basicamente, existem 3 modos de executar programas em Python:

- Modo Shell

- Modo Script (arquivos com extensão .py)

- Modo interativo (Jupyter Notebook)

Indentação é importante (1 tab ou 4 spaces)

‘...empregada com o objetivo de ressaltar a estrutura do algoritmo, e aumentando assim a legibilidade do código...’

Indentação faz parte da sintaxe em Python (obrigatória)

Seja consistente. Use tab ou espaços. Não misture os dois.

Comentários em Python:

Começam com o caracter # ou 3 aspas duplas “””...”””

# Isso é um comentário de uma linha!

“””

Isso é um comentário

Em mais de uma linha

“””

Dicas:

- Clareza é importante. Mantenha seu código limpo e organizado.

- Código esparso é melhor do que código denso.

- Sempre documente seu código.

- Siga os padrões não para criar complexidade, mas para manter a regra.

- Erros nunca serão silenciosos, a menos que propositalmente.

- Simples é melhor que complexo e complexo é melhor que complicado.

- Não se sinta obrigado a criar classes sem uma boa razão.

Números e Operações Matemáticas

Tópicos deste Vídeo:

1) Tipos de números em Python

2) Funções e operações com números

3) Aritmética

4) Operadores

Python possui 2 tipos de números principais:

int -> números inteiros, positivos ou negativos. Ex: -7 e 7

float -> números fracionários, positivos ou negativos. Ex: -7.1 e 7.1

Podemos usar a função type(), para saber qual é o tipo de um número.

Podemos usar as funções int() e float() para converter números.

Built-in Functions (<https://docs.python.org/3/library/index.html>)

**Operações com Números**

**Operadores Aritméticos:**

**Operador Significado Exemplo**

+ Soma 2 + 2 -> 4

- Subtração 3 - 2 -> 1

\* Multiplicação 2 \* 3 -> 6

/ Divisão (OU // para parte inteira da divisão) 10 / 2 -> 5

% Módulo 5 % 2 -> 1

\*\* Potência 4 \*\* 2 -> 16

int() Converte para inteiro int(3.2) -> 3

float() Converte para float float(2) -> 2.0

**Operadores Relacionais:**

**Operador Significado**

== Igualdade/equivalência

!= Desigualdade/Inequivalência

> Maior que

< Menor que

>= Maior que ou igual a

<= Menor que ou igual a

Jupyter Notebook – Números e Operações Matemáticas – Parte 1/2

C:\Users\victor\Documents\PythonFundamentos>cd Cap02

C:\Users\victor\Documents\PythonFundamentos\Cap02>jupyter notebook

cd .. (comando para voltar uma pasta/diretório)

Jupyter Notebook – Números e Operações Matemáticas – Parte 2/2

DSA-Python-Cap02-01-Numeros.ipynb

Para executar código selecionado:

Run

OU

Shift + Enter

Variáveis e Operadores

Variáveis são usadas em nosso código Python para armazenar valores que queremos usar mais tarde.

São espaços em memória que armazenam valores.

Por exemplo, nós podemos armazenar o valor 10 na variável b.

b = 10

O sinal de igual atribui o valor à direita (10) à variável do lado esquerdo (b). Você pode sobrescrever uma variável com um novo valor sempre que quiser. A variável assumirá o novo valor.

A função print() em Python irá imprimir valores na tela (teremos um módulo inteiro sobre funções). Por exemplo:

print(10) -> imprime na tela o valor 10

print(b) -> imprime na tela o valor da variável b

Tudo o que é impresso aparece na tela de saída do Python.

Existem algumas regras que devem ser seguidas ao definir nomes de variáveis:

1. Os nomes das variáveis não podem começar com um número.

2. Não pode haver espaços no nome; utilize \_ em vez disso.

3. Não é possível usar qualquer um desses símbolos: ‘”,<>/|\()@#$%^&\*~-+!

O item 3 é considerado uma boa pratica de programação (PEP8). Visito os sites abaixo para mais informações:

<https://www.python.org/dev>

<https://docs.python.org/devguide>

Não se pode usar palavras reservadas como nome de variável.

## Não se pode usar palavras reservadas como nome de variável

## 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

Operadores:

Operadores Aritméticos

Operadores Relacionais

**Operadores de Atribuição:**

**Operador Significado Exemplo**

+ Atribuição z = 10

+= Soma z += 10 (equivalente a z = z + 10)

-= Subtração z -= 10 (equivalente a z = z – 10)

\*= Multiplicação z \*= 10 (equivalente a z = z \* 10)

/= Divisão z /= 10 (equivalente a z = z /10)

%= Módulo

\*\*= Potência

//= Divisão Inteira

‘...numa única instrução executo duas operações...’

**Operadores Lógicos**

**Operador Significado Exemplo**

and Se ambos operadores foram True, retorna True (x and y) é True

or Se um dos operadores for True, retorna True (x or y) é True

Not Usado para reverter o estado da lógica Not (x and y) é False

‘...usado para checar duas condições...’

Jupyter Notebook – Variáveis e Operadores

DSA-Python-Cap02-02-Variaveis

Strings e Indexação em Python (21/05/2020)

Strings são usadas em Python para gravar informações em formato texto, como nomes por exemplo. Strings em Python são na verdade uma sequência de caracteres, o que significa, basicamente, que Python mantem o controle de cada elemento da sequência.

Python entende a string “Olá”, como sendo uma sequência de letras em uma ordem especifica. Isso significa que você será capaz de usar a indexação para obter um caractere especifico (como a primeira letra ou a última letra).

Strings – sequencia imutável de caracteres ou apenas 1 caractere

**Indexando Strings**

Já sabemos que Strings são uma sequência. Isso significa que Python pode usar índices para chamar partes da sequência. Vamos aprender como isso funciona.

Em Python, usamos **colchetes []** para representar o índice de uma objeto.

-> em Python, a indexação começa por 0.

Por exemplo, podemos criar a string:

Texto = “Python e Análise de Dados”

Texto[0] = P

Texto[1] = y

Texto[2] = t

É importante ressaltar que a string tem uma importante propriedade conhecida como imutabilidade. Isto significa que uma vez que é criada uma string, os elementos dentro dela não podem ser substituídos ou alterados.

**Funções Built-in de Strings**

Python é uma linguagem orientada a objeto, sendo assim as estruturas de dados possuem atributos (propriedades) e métodos (rotinas associadas às propriedades). Tanto os atributos quanto os métodos são acessados usando ponto (.).

Os métodos estão sob a forma:

objeto.atributo

objeto.método()

objeto.método(parâmetros)

Parâmetros são argumentos extras, que podemos passar para o método.

Jupyter notebook – Strings Parte 1/2

**Criando uma String**

Para criar uma string em Python você pode usar aspas simples ou duplas. Por exemplo:

# Uma única palavra

'Oi'

# Uma frase

'Criando uma string em Python'

# Podemos usar aspas duplas

"Podemos usar aspas duplas ou simples para strings em Python"

# Você pode combinar aspas duplas e simples

"Testando strings em 'Python'"

‘...com o Jupyter Notebook, nós podemos colocar uma string, e imprimir direto na saída, como output...’

**Imprimindo uma String**

print ('Testando Strings em Python')

print ('Testando \nStrings \nem \nPython')

print ('\n')

‘…a representação de caractere **\n** é como se fosse a tecla Enter do teclado…”retorno de carro”...”quebra de linha”...É um caracter...’

**Indexando Strings**

# Atribuindo uma string

s = 'Data Science Academy'

print(s) >>>Data Science Academy

# Primeiro elemento da string.

s[0] >>> 'D'

s[1] >>> 'a'

s[2] >>> 't'

Podemos usar um **:** para executar um slicing que faz a leitura de tudo até um ponto designado. Por exemplo:

# Retorna todos os elementos da string, começando pela posição (lembre-se que Python começa a indexação pela posição 0), até o fim da string.

s[1:] >>>'ata Science Academy'

# A string original permanece inalterada

s >>> 'Data Science Academy'

# Retorna tudo até a posição 3 (é exclusivo, ou seja, excluindo a posição 3)

s[:3] >>> 'Dat'

s[:] >>> 'Data Science Academy'

# Nós também podemos usar a indexação negativa e ler de trás para frente.

s[-1] >>> 'y'

# Retornar tudo, exceto a última letra

s[:-1] >>> 'Data Science Academ'

Jupyter notebook – Strings Parte 2/2

Nós também podemos usar a notação de índice e fatiar a string em pedaços específicos (o padrão é 1). Por exemplo, podemos usar dois pontos duas vezes em uma linha e, em seguida, um número que especifica a frequência para retornar elementos. Por exemplo:

s[::1] >>> 'Data Science Academy'

s[::2] >>> 'Dt cec cdm'

s[::-1] >>> 'ymedacA ecneicS ataD'

**Propriedades de Strings**

s >>> 'Data Science Academy'

# Alterando um caracter

s[0] = 'x' >>> **TypeError**: 'str' object does not support item assignment

# Concatenando strings

s + ' é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de Dados!'

>>> 'Data Science Academy é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de Dados!'

s = s + ' é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de Dados!'

>>>

print(s) >>> Data Science Academy é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de Dados!

# Podemos usar o símbolo de multiplicação para criar repetição!

letra = 'w' >>>

letra \* 3 >>> 'www'

**Funções Built-in de Strings**

s >>> 'Data Science Academy é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de Dados!'

# Upper Case

s.upper() >>> 'DATA SCIENCE ACADEMY É A MELHOR MANEIRA DE ESTAR PREPARADO PARA O MERCADO DE TRABALHO EM CIÊNCIA DE DADOS!'

# Lower case

s.lower() >>> 'data science academy é a melhor maneira de estar preparado para o mercado de trabalho em ciência de dados!'

# Dividir uma string por espaços em branco (padrão)

s.split() >>> ['Data', 'Science', 'Academy', 'é', 'a', 'melhor', 'maneira', 'de', 'estar', 'preparado', 'para', 'o', 'mercado', 'de', 'trabalho', 'em', 'Ciência', 'de', 'Dados!']

# Dividir uma string por um elemento específico

s.split('y') >>> ['Data Science Academ', ' é a melhor maneira de estar preparado para o mercado de trabalho em Ciência de Dados!']

‘...apertar a tecla tab para o intilisense, que são as opções de autocomplete de métodos e atributos

**Funções String**

s = 'seja bem vindo ao universo de python'

s.capitalize() >>> 'Seja bem vindo ao universo de python'

s.count('a') >>> 2

s.find('p') >>> 30

s.center(50, '-') >>> '-------seja bem vindo ao universo de python-------'

s.isalnum() >>> False

s.isalpha() >>> False

s.islower() >>> True

s.isspace() >>> False

s.endswith('o') >>> False

**Comparando Strings**

print("Python" == "R") >>> False

print("Python" == "Python") >>> True

\*

Estruturas de Dados – Listas (21/05/2020)

Neste vídeo, você vai aprender:

1) O que são listas

2) Indexação e slice de listas

3) Métodos básicos de lista

4) Listas aninhadas

Anteriormente, quando discutimos Strings, introduzimos o conceito de uma sequência em Python. As listas podem ser consideradas a versão geral de uma sequência em Python. Ao contrário de Strings, as listas são mutáveis, ou seja, os elementos dentro de uma lista podem ser alterados.

As listas são construídas com o uso de colchetes [] e virgulas separando cada elemento da lista.

Lista = [item1, item2, ..., itemN]

Se você estiver familiarizado com outra linguagem de programação, você pode traçar paralelos entre matrizes em outras linguagens e listas em Python. Listas em Python, no entanto, tendem a ser mais flexíveis do que as matrizes em outras linguagens por dois bons motivos:

1. Listas não têm tamanho fixo (o que significa que não precisamos especificar quão grande uma lista será)

2. Listas não têm restrições de tipo fixo

Uma grande característica de estruturas de dados em Python é que elas suportam aninhamento. Isto significa que podemos usar estruturas de dados dentro de estruturas de dados.

Jupyter notebook – Listas Parte 1/2

**Listas**

# Criando uma lista

listadomercado = ["ovos, farinha, leite, maças"]

# Imprimindo a lista

print(listadomercado) >>> ['ovos, farinha, leite, maças']

# Criando outra lista

listadomercado2 = ["ovos", "farinha", "leite", "maças"]

# Imprimindo a lista

print(listadomercado2) >>> ['ovos', 'farinha', 'leite', 'maças']

listadomercado[0] >>> 'ovos, farinha, leite, maças'

listadomercado2[0] >>> 'ovos'

# Criando lista (misturando TIPOS)

lista3 = [12, 100, "Universidade"]

# Imprimindo

print(lista3) >>> [12, 100, 'Universidade']

# Atribuindo cada valor da lista a uma variável.

item1 = lista3[0]

item2 = lista3[1]

item3 = lista3[2]

# Imprimindo as variáveis

print(item1, item2, item3) >>> 12 100 Universidade

**Atualizando um item da lista**

# Imprimindo um item da lista

listadomercado2[2] >>> 'leite'

# Atualizando um item da lista

listadomercado2[2] = "chocolate"

# Imprimindo lista alterada

listadomercado2 >>> ['ovos', 'farinha', 'chocolate', 'maças']

**Deletando um item da lista**

# Out of index. Não é possível deletar um item que não existe na lista

del listadomercado2[4] >>> **IndexError**: list assignment index out of range

# Deletando um item específico da lista

del listadomercado2[3] >>>

# Imprimindo o item com a lista alterada

listadomercado2 >>> ['ovos', 'farinha', 'chocolate']

**Listas de listas (Listas aninhadas)**

Listas de listas são matrizes em Python

# Criando uma lista de listas

listas = [[1,2,3], [10,15,14], [10.1,8.7,2.3]]

# Imprimindo a lista

listas >>> [[1, 2, 3], [10, 15, 14], [10.1, 8.7, 2.3]]

# Atribuindo um item da lista a uma variável

a = listas[0]

a >>> [1, 2, 3]

b = a[0]

b >>> 1

list1 = listas[1]

list1 >>> [10, 15, 14]

# Criando uma lista aninhada (lista de listas)

listas = [[1,2,3], [10,15,14], [10.1,8.7,2.3]]

listas >>> [[1, 2, 3], [10, 15, 14], [10.1, 8.7, 2.3]]

# Atribuindo à variável a, o primeiro valor da primeira lista

a = listas[0][0]

a >>> 1

b = listas[1][2]

b >>> 14

c = listas[0][2] + 10

c >>> 13

d = 10

d >>> 10

e = d \* listas[2][0]

e >>> 101.0

**Concatenando listas**

lista\_s1 = [34, 32, 56]

lista\_s1 >>> [34, 32, 56]

lista\_s2 = [21, 90, 51]

lista\_s2 >>> [21, 90, 51]

# Concatenando listas

lista\_total = lista\_s1 + lista\_s2

lista\_total >>> [34, 32, 56, 21, 90, 51]

**Operador in**

# Criando uma lista

lista\_teste\_op = [100, 2, -5, 3.4]

# Verificando se o valor 10 pertence a lista

print(10 in lista\_teste\_op) >>> False

# Verificando se o valor 100 pertence a lista

print(100 in lista\_teste\_op) >>> True

**Funções Built-in**

# Função len() retorna o comprimento da lista

len(lista\_teste\_op) >>> 4

# Função max() retorna o valor máximo da lista

max(lista\_teste\_op) >>> 100

# Função min() retorna o valor mínimo da lista

min(lista\_teste\_op) >>> -5

# Criando uma lista

listadomercado2 = ["ovos", "farinha", "leite", "maças"]

# Adicionando um item à lista

listadomercado2.append("carne")

listadomercado2 >>> ['ovos', 'farinha', 'leite', 'maças', 'carne']

listadomercado2.append("carne")

listadomercado2 >>> ['ovos', 'farinha', 'leite', 'maças', 'carne', 'carne']

listadomercado2.count("carne") >>> 2

# Criando uma lista vazia

a = []

print(a) >>> []

type(a) >>> list

a.append(10)

a >>> [10]

a.append(50)

a >>> [10, 50]

old\_list = [1,2,5,10]

new\_list = []

# Copiando os itens de uma lista para outra

for item in old\_list:

new\_list.append(item)

new\_list >>> [1, 2, 5, 10]

cidades = ['Recife', 'Manaus', 'Salvador']

cidades.extend(['Fortaleza', 'Palmas'])

print (cidades) >>> ['Recife', 'Manaus', 'Salvador', 'Fortaleza', 'Palmas']

cidades.index('Salvador') >>> 2

cidades.index('Rio de Janeiro') >>> **ValueError**: 'Rio de Janeiro' is not in list

cidades >>> ['Recife', 'Manaus', 'Salvador', 'Fortaleza', 'Palmas']

cidades.insert(2, 110)

cidades >>> ['Recife', 'Manaus', 110, 'Salvador', 'Fortaleza', 'Palmas']

# Remove um item da lista

cidades.remove(110)

cidades >>> ['Recife', 'Manaus', 'Salvador', 'Fortaleza', 'Palmas']

# Reverte a lista

cidades.reverse()

# Imprime a lista

cidades >>> ['Palmas', 'Fortaleza', 'Salvador', 'Manaus', 'Recife']

x = [3, 4, 2, 1]

x >>> [3, 4, 2, 1]

# Ordena a lista

x.sort()

x >>> [1, 2, 3, 4]

\*

Estrutura de Dados – Dicionários (22/05/2020)

Neste vídeo falaremos sobre:

1) O que são dicionários

2) Acessando itens de um dicionário

3) Dicionários aninhados

4) Métodos de Dicionários

Até aqui falamos bastante sobre sequencias em Python, mas agora vamos mudar um pouco o foco e aprender sobre mapeamentos em Python. Se você estiver familiarizado com outras linguagens de programação, pode imaginar os dicionários como tabelas de hash (hash tables).

Os dicionários são construídos com o uso de chaves { } e virgulas separando cada elemento.

Dict = { k1: v1, k2: v2, ..., kn: vn}

‘…Listas [ ] …Dicionários { } … Tuplas ( ) …’

Então, o que são mapeamentos?

Mapeamentos são uma coleção de objetos que são armazenados por uma chave, ao contrário de uma sequência de objetos armazenados por sua posição relativa.

Um dicionário Python consiste de uma chave e, em seguida, um valor associado.

Esse valor pode ser quase qualquer objeto Python.

Dicionários – são mapeamento de chaves e valores (PARES DE CHAVE E VALOR)

{chave1: valor1, chave2: valor2}

‘...esse conceito de dicionário é o mesmo utilizado em alguns bancos de dados não relacionais, NoSQL, o raciocínio é o mesmo...compreendendo o dicionário ficara mais fácil na hora de trabalhar com NoSQL...’

Jupyter Notebook - Estruturas de Dados - Dicionários (22/05/2020)

DSA-Python-Cap02-05-Dicionarios

**Dicionários**

# Isso é uma lista

estudantes\_lst = ["Mateus", 24, "Fernanda", 22, "Tamires", 26, "Cristiano", 25]

estudantes\_lst >>> ['Mateus', 24, 'Fernanda', 22, 'Tamires', 26, 'Cristiano', 25]

# Isso é um dicionário

estudantes\_dict = {"Mateus":24, "Fernanda":22, "Tamires":26, "Cristiano":25}

estudantes\_dict >>> {'Mateus': 24, 'Fernanda': 22, 'Tamires': 26, 'Cristiano': 25}

estudantes\_dict["Mateus"] >>> 24

‘...é assim que funciona o processo de mapeamento de redução(MapReduce), uma das principais atividades em BigData, o mapeamento e redução é na pratica você aplicar essas combinações, e vai reduzindo seus dados até uma quantidade menor de dados, que então podem ser utilizados como resultado do seu processo de análise...”

estudantes\_dict["Pedro"] = 23

estudantes\_dict["Pedro"] >>> 23

estudantes\_dict["Tamires"] >>> 26

estudantes\_dict.clear()

estudantes\_dict >>> {}

del estudantes\_dict

estudantes\_dict >>> **NameError**: name 'estudantes\_dict' is not defined

‘...vamor criar o OBJETO...’

estudantes = {"Mateus":24, "Fernanda":22, "Tamires":26, "Cristiano":25}

estudantes >>> {'Mateus': 24, 'Fernanda': 22, 'Tamires': 26, 'Cristiano': 25}

len(estudantes) >>> 4

estudantes.keys() >>> dict\_keys(['Mateus', 'Fernanda', 'Tamires', 'Cristiano'])

estudantes.values() >>> dict\_values([24, 22, 26, 25])

‘...extrair chaves e valores, isso é muito util em varias etapas de processamento em Machine Learning...’

estudantes.items() >>> dict\_items([('Mateus', 24), ('Fernanda', 22), ('Tamires', 26), ('Cristiano', 25)])

estudantes2 = {"Maria":27, "Erika":28, "Milton":26}

estudantes2 >>>{'Maria': 27, 'Erika': 28, 'Milton': 26}

estudantes.update(estudantes2)

estudantes >>>{'Mateus': 24, 'Fernanda': 22, 'Tamires': 26, 'Cristiano': 25, 'Maria': 27, 'Erika': 28, 'Milton': 26}

dic1 = {}

dic1 >>>{}

dic1["key\_one"] = 2

print(dic1) >>>{'key\_one': 2}

dic1[10] = 5

dic1 >>>{'key\_one': 2, 10: 5}

dic1[8.2] = "Python"

dic1 >>>{'key\_one': 2, 10: 5, 8.2: 'Python'}

dic1["teste"] = 5

dic1 >>>{'key\_one': 2, 10: 5, 8.2: 'Python', 'teste': 5}

dict1 = {}

dict1 >>>{}

dict1["teste"] = 10

# Atenção, pois chave e valor podem ser iguais, mas representam coisas diferentes. !!!

dict1 >>>{'teste': 10, 'key': 'teste'}

dict2 = {}

dict2["key1"] = "Big Data"

dict2["key2"] = 10

dict2["key3"] = 5.6

dict2 >>>{'key1': 'Big Data', 'key2': 10, 'key3': 5.6}

a = dict2["key1"]

b = dict2["key2"]

c = dict2["key3"]

a, b, c >>>('Big Data', 10, 5.6)

# Dicionário de listas

dict3 = {'key1':1230,'key2':[22,453,73.4],'key3':['leite','maça','batata']}

dict3 >>>{'key1': 1230, 'key2': [22, 453, 73.4], 'key3': ['leite', 'maça', 'batata']}

dict3['key2'] >>>[22, 453, 73.4]

# Acessando um item da lista, dentro do dicionário

dict3['key3'][0].upper() >>>'LEITE'

# Operações com itens da lista, dentro do dicionário

var1 = dict3['key2'][0] – 2

var1 >>> 20

# Duas operações no mesmo comando, para atualizar um item dentro da lista

dict3['key2'][0] -= 2

dict3 >>> {'key1': 1230, 'key2': [20, 453, 73.4], 'key3': ['leite', 'maça', 'batata']}

**Criando dicionários aninhados**

# Criando dicionários aninhados

dict\_aninhado = {'key1':{'key2\_aninhada':{'key3\_aninhada':'Dict aninhado em Python'}}}

dict\_aninhado >>>{'key1': {'key2\_aninhada': {'key3\_aninhada': 'Dict aninhado em Python'}}}

dict\_aninhado['key1']['key2\_aninhada']['key3\_aninhada'] >>>'Dict aninhado em Python'

‘...Dicionários são excelentes estruturas de dados, principalmente usados em Deep Learning, usamos muito dicionários como uma forma de armazenar o processamento intermediário, para gravar os Pesos de uma rede neural, para gravar o valor do Byas, para gravar as saídas, por exemplo, em acurácia, usamos estruturas de dados como essa.’

Estruturas de Dados – Tuplas (22/05/2020)

Neste vídeo veremos:

1) O que são Tuplas

2) Métodos básicos de Tuplas

3) Imutabilidade

4) Quando usar Tuplas

Em Python, Tuplas são muito semelhantes às Listas, no entanto, ao contrário de Listas, Tuplas são imutáveis, o que significa que não podem ser alteradas. Você usaria Tuplas para apresentar dados que não devem ser alterados, como os dias da semana ou datas em um calendário.

As Tuplas são construídas com o uso de parênteses ( ) e vírgulas separando cada elemento da Tupla.

tupla = (item1, item2, ..., itemN)

Por que se preocupar usando tuplas, uma vez que trata-se de um objeto com limitações e um número menor de métodos disponíveis?

Tuplas não são utilizadas com frequência, como listas por exemplo, mas são usadas quando é necessário imutabilidade. Se em seu programa você precisa ter certeza de que os dados não sofrerão mudança, então tupla pode ser a sua solução. Ela fornece uma fonte conveniente de integridade de dados.

Jupyter Notebook – estruturas de Dados- Tuplas (22/05/2020)

DSA-Python-Cap02-06-Tuplas

# Criando uma tupla

tupla1 = ("Geografia", 23, "Elefantes")

# Imprimindo a tupla

tupla1 >>>('Geografia', 23, 'Elefantes')

# Tuplas não suportam append()

tupla1.append("Chocolate") >>>**AttributeError**: 'tuple' object has no attribute 'append'

# Tuplas não suportam delete de um item específico

del tupla1["Gatos"] >>>**TypeError**: 'tuple' object does not support item deletion

# Tuplas podem ter um único item

tupla1 = ("Chocolate")

tupla1 >>>'Chocolate'

tupla1 = ("Geografia", 23, "Elefantes")

tupla1[0] >>>'Geografia'

# Verificando o comprimento da tupla

len(tupla1) >>>3

# Slicing, da mesma forma que se faz com listas

tupla1[1:] >>>(23, 'Elefantes')

tupla1.index('Elefantes') >>>2

# Tuplas não suportam atribuição de item

tupla1[1] = 21 >>>**TypeError**: 'tuple' object does not support item assignment

# Deletando a tupla

del tupla1

tupla1 >>>**NameError**: name 'tupla1' is not defined

# Criando uma tupla

t2 = ('A', 'B', 'C')

t2 >>>('A', 'B', 'C')

# Tuplas não suportam atribuição de item

t2[0] = 'D' >>>**TypeError**: 'tuple' object does not support item assignment

# Usando a função list() para converter uma tupla para lista

lista\_t2 = list(t2)

lista\_t2 >>>['A', 'B', 'C']

lista\_t2.append('D')

# Usando a função tuple() para converter uma lista para tupla

t2 = tuple(lista\_t2)

t2 >>>('A', 'B', 'C', 'D')

Resumo – O que Vimos Até Aqui (22/05/2020)

**Tipo de Objeto Categoria Mutável? Formato**

Números Numérico Não

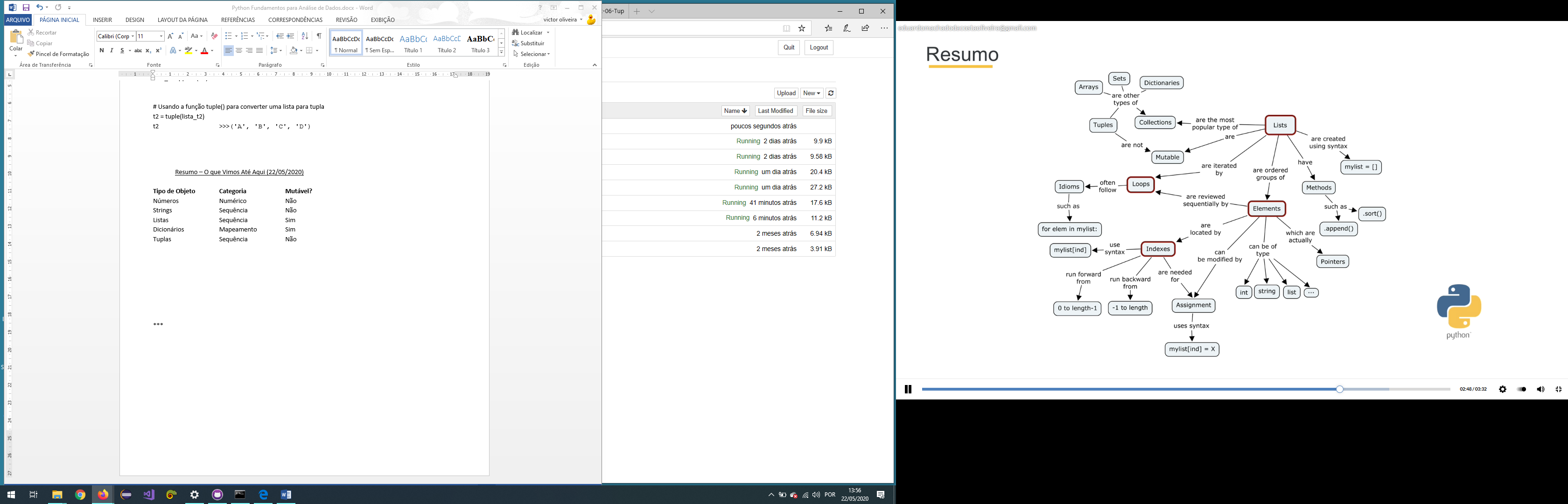
Strings Sequência Não

Listas Sequência Sim **[ ]**

Dicionários Mapeamento Sim **{ }**

Tuplas Sequência Não **( )**

‘…Listas [ ] …Dicionários { } … Tuplas ( ) …’



Você vai perceber, que o conhecimento adquirido neste capitulo, será utilizado em todos os demais capítulos e quando trabalharmos com Análise de Dados.

Agora, faça os exercícios, compare suas respostas com o gabarito fornecido, faça o quiz, acesse a seção de links úteis com material complementar e encontre comigo no Laboratório01, quando vamos trabalhar com um game em Pyhton.

Exercicios

# Exercício 1 - Imprima na tela os números de 1 a 10. Use uma lista para armazenar os números.

lista = [1,2,3,4,5,6,7,8,9,10]

print(lista)

>>>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Exercício 2 - Crie uma lista de 5 objetos e imprima na tela

lista = ['chocolate', 'banana', 'abacaxi', 'uva', 'morango']

print(lista)

>>> ['chocolate', 'banana', 'abacaxi', 'uva', 'morango']

# Exercício 3 - Crie duas strings e concatene as duas em uma terceira string

frase1 = 'Se beber '

frase2 = 'não dirija!'

frase\_final = frase1 + frase2

print(frase\_final)

>>>Se beber não dirija!

# Exercício 4 - Crie uma tupla com os seguintes elementos: 1, 2, 2, 3, 4, 4, 4, 5 e depois utilize a função count do

# objeto tupla para verificar quantas vezes o número 4 aparece na tupla

tup1 = (1, 2, 2, 3, 4, 4, 4, 5)

tup1.count(4)

>>>3

# Exercício 5 - Crie um dicionário vazio e imprima na tela

dict3 = {}

print(dict3)

>>>{}

# Exercício 6 - Crie um dicionário com 3 chaves e 3 valores e imprima na tela

dict = {'k1':'martelo', 'k2':'serrote', 'k3':'machado'}

print(dict)

>>>{'k1': 'martelo', 'k2': 'serrote', 'k3': 'machado'}

# Exercício 7 - Adicione mais um elemento ao dicionário criado no exercício anterior e imprima na tela

dict['k4'] = 'parafuso'

print(dict)

>>>{'k1': 'martelo', 'k2': 'serrote', 'k3': 'machado', 'k4': 'parafuso'}

# Exercício 8 - Crie um dicionário com 3 chaves e 3 valores. Um dos valores deve ser uma lista de 2 elementos numéricos.

# Imprima o dicionário na tela.

dict2 = {'chave1':'Geografia', 'chave2':'Biologia', 'chave3':[70, 90]}

print(dict2)

>>>{'chave1': 'Geografia', 'chave2': 'Biologia', 'chave3': [70, 90]}

# Exercício 9 - Crie uma lista de 4 elementos. O primeiro elemento deve ser uma string,

# o segundo uma tupla de 2 elementos, o terceiro um dcionário com 2 chaves e 2 valores e

# o quarto elemento um valor do tipo float.

# Imprima a lista na tela.

lst = ['String1', (90, 92), {'k1':'v1', 'k2':'v2'}, 99.98]

print(lst)

>>>['String1', (90, 92), {'k1': 'v1', 'k2': 'v2'}, 99.98]

# Exercício 10 - Considere a string abaixo. Imprima na tela apenas os caracteres da posição 1 a 18.

frase = 'Cientista de Dados é o profissional mais sexy do século XXI'

frase[0:18]

>>> 'Cientista de Dados'

Obs: Antes de achar que o resultado acima está errado, reflita junto comigo:

Em uma corrida ou maratona quem sobre ao podium? Aqueles que ficam nas posições 1, 2 e 3, certo? Existe posição zero? Não! Pois bem!

O exercício acima pede os caracteres nas posições de 1 a 18 (não existe posição zero). Em Python, a indexação (que representa o índice) começa por zero. Logo, posições de 1 a 18 estão nos índices de 0 a 18 (pois 18 é exclusivo e portanto temos de 0 a 17, logo 18 posições).

Lab01

C:\>cd Users

C:\Users>cd victor

C:\Users\victor>cd Documents

C:\Users\victor\Documents>cd PythonFundamentos

C:\Users\victor\Documents\PythonFundamentos>cd Cap02

C:\Users\victor\Documents\PythonFundamentos\Cap02>cd Lab01

C:\Users\victor\Documents\PythonFundamentos\Cap02\Lab01>python game\_v1.py

Qual nível você gostaria de jogar? 1/2/3/4/5

‘...estude o código do arquivo game\_v1.py e comente o script...tente enteder tudo que esta aqui...pesquise, busque na internet, consulte documentação, é uma excelente forma de praticar...’

Solução Lab01

‘game\_v2.py’

tkinter é uma API Python para criar ambiente gráfico

\*\*\*

**Cápitulo 3 - Loops, Condicionais, Métodos e Funções (25/05/2020)**

Introdução

O que estudaremos neste cápitulo?

- Condicionais If/Else/Elif

- Estruturas de Repetição

- Métodos

- Funções

- Outras operações

Bônus – IA x Machine Learning x Deep Learning

Deep Learning Book (www.deeplearningbook.com.br)

1950´s

Artificial Inteligence

Early antificial intelligence stirs excitement

1980´s

MACHINE LEARNING

Machine leraning begins to flourish

2010´s

DEEP LEARNING

Deep learning breakthroughs drive AI boom

Since an early flush of optimism in the 1950´s, smaller subsets os artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Condicionais – If/Else/Elif

O condicional IF nos permite dizer ao computador para executar ações com base em um determinado conjunto de resultados.

Verbalmente, podemos imaginar que estamos dizendo ao computador:

“Ei, caso isso aconteça, execute esta ação.”

Ex.:

If (expressão 1):

print(“comando executado caso a expressão 1 seja Verdadeira”)

else:

print(“comando executado caso a expressão 1 seja Falsa”)

Ex.:2

If (expressão 1):

print(“comando executado caso a expressão 1 seja Verdadeira”)

elif (expressão 2):

print(“comando executado caso a expressão 1 seja Falsa e expressão 2 seja Verdadeira”)

else:

print(“comando executado caso a expressão 1 e 2 sejam Falsas”)

Indentação

É importante ter uma boa compreensão de como funciona o recuo em Python (indentação) para manter a estrutura em ordem no seu código.

Jupyter Notebook -Condicionais

**Condicional If**

# Condicional If

if 5 > 2:

print("Python funciona!")

>>> Python funciona!

# Statement If...Else

if 5 < 2:

print("Python funciona!")

else:

print("Algo está errado!")

>>> Algo está errado!

6 > 3

>>> True

3 > 7

>>> False

4 < 8

>>> True

4 >= 4

>>> True

if 5 == 5:

print("Testando Python!")

>>> Testando Python!

if True:

print('Parece que Python funciona!')

>>> Parece que Python funciona!

**Obs.: só executa quando verdadeiro!**

# Atenção com a sintaxe

if 4 > 3

print("Tudo funciona!")

>>> **SyntaxError:** invalid syntax

# Atenção com a sintaxe

if 4 > 3:

print("Tudo funciona!")

>>> **IndentationError:** expected an indented block

Jupyter Notebook – Condicionais Aninhados

**Condicionais Aninhados**

idade = 18

if idade > 17:

print("Você pode dirigir!")

>>> Você pode dirigir!

Nome = "Bob"

if idade > 13:

if Nome == "Bob":

print("Ok Bob, você está autorizado a entrar!")

else:

print("Desculpe, mas você não pode entrar!")

>>> Ok Bob, você está autorizado a entrar!

idade = 13

Nome = "Bob"

if idade >= 13 **and** Nome == "Bob":

print("Ok Bob, você está autorizado a entrar!")

>>> Ok Bob, você está autorizado a entrar!

idade = 12

Nome = "Bob"

if (idade >= 13) **or** (Nome == "Bob"):

print("Ok Bob, você está autorizado a entrar!")

>>> Ok Bob, você está autorizado a entrar!

**Obs.:o parênteses ( ) não é obrigatório mas é uma boa prática!**

Jupyter Notebook – Condicionais Elif

**Elif**

dia = "Terça"

if dia == "Segunda":

print("Hoje fará sol!")

else:

print("Hoje vai chover!")

>>> Hoje vai chover!

if dia == "Segunda":

print("Hoje fará sol!")

elif dia == "Terça":

print("Hoje vai chover!")

else:

print("Sem previsão do tempo para o dia selecionado")

>>> Hoje vai chover!

Obs: ‘...Elif mostra que é um código Pythonista...’rs

Jupyter Notebook – Condicionais, Operadores Lógicos e Placeholders

**Operadores Lógicos**

idade = 18

nome = "Bob"

if idade > 17:

print("Você pode dirigir!")

>>> Você pode dirigir!

idade = 18

if idade > 17 and nome == "Bob":

print("Autorizado!")

>>> Autorizado!

# Usando mais de uma condição na cláusula if

disciplina = input('Digite o nome da disciplina: ')

nota\_final = input('Digite a nota final (entre 0 e 100): ')

if disciplina == 'Geografia' and nota\_final >= '70':

print('Você foi aprovado!')

else:

print('Lamento, acho que você precisa estudar mais!')

>>> Digite o nome da disciplina: Geografia

Digite a nota final (entre 0 e 100): 70

Você foi aprovado!

**--> Fique atento aos espaços entre a margem e cada um dos seus comandos. Falaremos mais sobre indentação ao longo do curso. A indentação faz parte da sintaxe da linguagem Python.**

Obs.:

Placeholders (existem algumas maneiras)

print('Você foi aprovado em %s com média final %r!' %(disciplina, nota\_final))

print('Você foi aprovado em {} com média final {}!'.format(disciplina, nota\_final))

print(f'Você foi aprovado em {disciplina} com média final {nota\_final}!')

Estruturas de Repetição – Loop For (26/05/2020) 19%

For

Valida cada item em uma série de valores

**for** item **in** série-de-items:

Executar comandos

**for** item **in** série de items:

**if** item > 0:

Executar comandos

‘...permite executar um comando(s) um numero **determinado** de vezes...’

Podemos utilizar o loop for em objetos sequenciais como:

- strings

- listas

- tuplas

- elementos de dicionários

- arquivos

Jupyter Notebook – Loop for

DSA-Python-Cap03-02-For

‘...iteração é executar uma ação diversas vezes...’

**Loop For**

# Criando uma tupla e imprimindo cada um dos valores

tp = (2,3,4)

for i in tp:

print(i)

>>>2

3

4

# Criando uma lista e imprimindo cada um dos valores

ListaDoMercado = ["Leite", "Frutas", "Carne"]

for i in ListaDoMercado:

print(i)

>>>Leite

Frutas

Carne

# Imprimindo os valores no intervalo entre 0 e 5 (exclusive)

for contador in range(0,5):

print(contador)

>>>0

1

2

3

4

# Imprimindo na tela os números pares da lista de números

lista = [1,2,3,4,5,6,7,8,9,10]

for num in lista:

if num % 2 == 0:

print (num)

>>>2

4

6

8

10

# Listando os números no intervalo entre 0 e 20, com incremento em 2

for i in range(0,20,2):

print(i)

>>>0

2

4

6

8

10

12

14

16

18

# Strings também são sequências

for caracter in 'Python':

print (caracter)

>>>P

y

t

h

o

n

Jupyter Notebook – Loop for Aninhado e Outras Operações

**Loops Aninhados**

# Loops aninhados

for i in range(0,2):

for a in range(0,5):

print(a)

>>> 0

1

2

3

4

0

1

2

3

4

# Operando os valores de uma lista com loop for

listaB = [32,53,85,10,15,17,19]

soma = 0

for i in listaB:

double\_i = i \* 2

soma += double\_i

print(soma)

>>>462

# Operando os valores de uma lista com loop for

listaB = [10,15,5,30]

soma = 0

for i in listaB:

double\_i = i \* 2

soma += double\_i

print(soma)

>>>20

50

60

120

Obs: ‘...soma é uma variável **global**, e double\_i é uma variável **local**...’

# Loops em lista de listas

listas = [[1,2,3], [10,15,14], [10.1,8.7,2.3]]

for valor in listas:

print(valor)

>>>[1, 2, 3]

[10, 15, 14]

[10.1, 8.7, 2.3]

# Contando os itens de uma lista

lista = [5,6,10,13,17]

count = 0

for item in lista:

count += 1

print(count)

>>>5

# Contando o número de colunas

lst = [[1,2,3],[3,4,5],[5,6,7]]

primeira\_linha = lst[0]

count = 0

for column in primeira\_linha:

count = count + 1

print(count)

>>>3

# Pesquisando em listas

listaC = [5, 6, 7, 10, 50]

# Loop através da lista

for item in listaC:

if item == 5:

print("Número encontrado na lista!")

>>>Número encontrado na lista!

# Listando as chaves de um dicionário

dict = {'k1':'Python','k2':'R','k3':'Scala'}

for item in dict:

print(item)

>>>k1

k2

k3

# Imprimindo chave e valor do dicionário. Usando o método items() para retornar os itens de um dicionário

for k,v in dict.items():

print (k,v)

>>>k1 Python

k2 R

k3 Scala

Estruturas de Repetição – Loop While

O loop While em Python é uma das formas mais comuns para executar iteração.

A instrução while sera executada repetidamente, seja uma única instrução ou grupo de instruções, desde que uma condição seja verdadeira.

Valida cada item em uma série de valores

while (expressão1):

print(“comando executado caso a expressão1 seja Verdadeira”)

Jupyter Notebook – Loop While

**While**

# Usando o loop while para imprimir os valores de 0 a 9

counter = 0

while counter < 10:

print(counter)

counter = counter + 1

>>>0

1

2

3

4

5

6

7

8

9

# Também é possível usar a claúsula else para encerrar o loop while

x = 0

while x < 10:

print ('O valor de x nesta iteração é: ', x)

print (' x ainda é menor que 10, somando 1 a x')

x += 1

else:

print ('Loop concluído!')

>>> valor de x nesta iteração é: 0

x ainda é menor que 10, somando 1 a x

O valor de x nesta iteração é: 1

x ainda é menor que 10, somando 1 a x

O valor de x nesta iteração é: 2

x ainda é menor que 10, somando 1 a x

O valor de x nesta iteração é: 3

x ainda é menor que 10, somando 1 a x

O valor de x nesta iteração é: 4

x ainda é menor que 10, somando 1 a x

O valor de x nesta iteração é: 5

x ainda é menor que 10, somando 1 a x

O valor de x nesta iteração é: 6

x ainda é menor que 10, somando 1 a x

O valor de x nesta iteração é: 7

x ainda é menor que 10, somando 1 a x

O valor de x nesta iteração é: 8

x ainda é menor que 10, somando 1 a x

O valor de x nesta iteração é: 9

x ainda é menor que 10, somando 1 a x

Loop concluído

Jupyter Notebook – Pass, Break, Continue

**Pass, Break, Continue**

counter = 0

while counter < 100:

if counter == 4:

break

else:

pass

print(counter)

counter = counter + 1

>>> 0

1

2

3

for verificador in "Python":

if verificador == "h":

continue

print(verificador)

>>> P

y

t

o

n

obs.: ‘...usando continue para pular um item...’

**While e For juntos**

for i in range(2,30):

j = 2

counter = 0

while j < i:

if i % j == 0:

counter = 1

j = j + 1

else:

j = j + 1

if counter == 0:

print(str(i) + " é um número primo")

counter = 0

else:

counter = 0

>>> 2 é um número primo

3 é um número primo

5 é um número primo

7 é um número primo

11 é um número primo

13 é um número primo

17 é um número primo

19 é um número primo

23 é um número primo

29 é um número primo

Estruturas de Repetição - Range

A função range() nos permite criar uma lista de números em um intervalo especifico.

A função range() tem o seguinte formato:

**range([start], [stop], [step])**

[start] – número que inicia sequência

[stop] – número que encerra a sequência (não é incluído na sequência)

[step] – diferença entre cada número da sequência

Ex,: range(50, 101, 2)

Começar em 50 ate 101 (excludente) pulando de 2 em 2

52, 54, 56, 58...98, 100

Jupyter Notebook - Range

DSA-Python-Cap03-04-Range

# Imprimindo números pares entre 50 e 101

for i in range(50, 101, 2):

print(i)

>>> 50

52

54

56

58

60

62

64

66

68

70

72

74

76

78

80

82

84

86

88

90

92

94

96

98

100

for i in range(3, 6):

print (i)

>>> 3

4

5

for i in range(0, -20, -2):

print(i)

>>> 0

-2

-4

-6

-8

-10

-12

-14

-16

-18

lista = ['Morango', 'Banana', 'Abacaxi', 'Uva']

lista\_tamanho = len(lista)

for i in range(0, lista\_tamanho):

print(lista[i])

>>> Morango

Banana

Abacaxi

Uva

# Tudo em Python é um objeto

type(range(0,3))

>>> range

Exercicios – Loops Condicionais

DSA-Python-Cap03-Exercicios-Loops-Condiconais

# Exercício 1 - Crie uma estrutura que pergunte ao usuário qual o dia da semana. Se o dia for igual a Domingo ou

# igual a sábado, imprima na tela "Hoje é dia de descanso", caso contrário imprima na tela "Você precisa trabalhar!"

dia\_semana = str(input('Qual o dia da semana hoje? ')).lower()

if (dia\_semana == 'domingo' or dia\_semana == 'sabado'):

print("Hoje é dia de descanso")

else:

print("Você precisa trabalhar!")

# Exercício 2 - Crie uma lista de 5 frutas e verifique se a fruta 'Morango' faz parte da lista

frutas = ['banana', 'maçã', 'laranja', 'limão', 'morango']

#1

#morango = frutas.index('morango')

#if (morango > -1):

# print(f'Morango faz parte da lista. Esta na posição {morango}')

#2

#print('morango' in frutas)

#3

for item in frutas:

if (item == 'morango'):

print('Morango faz parte da lista de frutas')

>>> Morango faz parte da lista de frutas

# Exercício 3 - Crie uma tupla de 4 elementos, multiplique cada elemento da tupla por 2 e guarde os resultados em uma

# lista

tupla4 =(1, 2, 3, 4)

lista\_dobro = []

for i in tupla4:

lista\_dobro.append(i\*2)

print(lista\_dobro)

>>> [2, 4, 6, 8]

???colocar solução minha ou dele???

Colocar solução da DSA

21%

O que são Métodos? (27/05/2020) 21%

Nos já vimos alguns exemplos de métodos quando aprendemos sobre Estruturas de dados em Python.

Os métodos são essencialmente funções incorporadas em objetos.

Mais tarde, vamos aprender sobre como criar nossos próprios métodos que utilizam Programação Orientada a Objetos (OOP) e classes.

Métodos permitem executar ações específicas no objeto e podem também ter argumentos, exatamente como uma função.

Os métodos são executados sob a forma:

objeto.metodo(arg1, arg2, etc...)

Com Jupyter Notebook podemos ver rapidamente todos os métodos possíveis para um objeto, usando a tecla TAB. Por exemplo, os métodos (alguns) para o objeto lista são:

- append

-count

- extend

- insert

- pop

- remove

- reverse

-sort

Jupyter notebook - Métodos

DSA-Python-Cap03-05-Metodos

**Métodos**

# Criando uma lista

lst = [100, -2, 12, 65, 0]

# Usando um método do objeto lista

lst.append(10)

# Imprimindo a lista

lst

>>>[100, -2, 12, 65, 0, 10]

# Usando um método do objeto lista

lst.count(10)

>>>1

# A função help() explica como utilizar cada método de um objeto

help(lst.count)

>>>Help on built-in function count:

count(value, /) method of builtins.list instance

Return number of occurrences of value.

# A função dir() mostra todos os métodos e atributos de um objeto

dir(lst)

>>>['\_\_add\_\_',

'\_\_class\_\_',

'\_\_contains\_\_',

'\_\_delattr\_\_',

'\_\_delitem\_\_',

'\_\_dir\_\_',

'\_\_doc\_\_',

'\_\_eq\_\_',

'\_\_format\_\_',

'\_\_ge\_\_',

'\_\_getattribute\_\_',

'\_\_getitem\_\_',

'\_\_gt\_\_',

'\_\_hash\_\_',

'\_\_iadd\_\_',

'\_\_imul\_\_',

'\_\_init\_\_',

'\_\_init\_subclass\_\_',

'\_\_iter\_\_',

'\_\_le\_\_',

'\_\_len\_\_',

'\_\_lt\_\_',

'\_\_mul\_\_',

'\_\_ne\_\_',

'\_\_new\_\_',

'\_\_reduce\_\_',

'\_\_reduce\_ex\_\_',

'\_\_repr\_\_',

'\_\_reversed\_\_',

'\_\_rmul\_\_',

'\_\_setattr\_\_',

'\_\_setitem\_\_',

'\_\_sizeof\_\_',

'\_\_str\_\_',

'\_\_subclasshook\_\_',

'append',

'clear',

'copy',

'count',

'extend',

'index',

'insert',

'pop',

'remove',

'reverse',

'sort']

a = 'Isso é uma string'

# O método de um objeto pode ser chamado dentro de uma função, como print()

print (a.split())

>>>['Isso', 'é', 'uma', 'string']

O que são Funções?

Vamos agora estudar o que é uma função em Python. As funções serão um dos nossos principais recursos, quando construirmos quantidades cada vez maiores de código para resolver problemas.

Função é um dispositivo que agrupa um conjunto de instruções para que ela possam ser executadas mais de uma vez. Funções também permitem especificar os parâmetros que podem servir como entrada para as funções.

Em um nível mais fundamental, a construção de funções nos permite reutilizar código, sem ter que escrevê-lo novamente. Nas aulas de strings, utilizamos a função len() para obter o comprimento de uma string. Com funções, escrevemos o código uma única vez e repetimos a mesma instrução, fazendo a chamada à função, quantas vezes forem necessárias.

O formato geral de uma função é:

def nome da função(arg1, arg2):

‘’’Aqui vão os comentários, documentando sua função’’’

<aqui vai seu código>

<Retorno desejado pela função>

E por que Função é importante?

Funções em Python são uma forma de escrever a sua lógica em único pacot e utilizá-la em diferentes lugares no seu código e quantas vezes quiser.

**Entrada Saída**

Argumentos -> Função -> Resultado

Variáveis Globais -> Variaveis Locais -> Varáveis Globais

Arquivos/ Streams de Dados -> Outras Funções -> Arquivos/Streams de Dados

Jupyter Notebook – Funções – Parte 1/3

DSA-Python-Cap03-06-Funcoes

**Funções**

# Definindo uma função

def primeiraFunc():

print('Hello World')

primeiraFunc()

>>> Hello World

# Definindo uma função com parâmetro

def primeiraFunc(nome):

print('Hello %s' %(nome))

primeiraFunc('Aluno')

>>> Hello Aluno

def funcLeitura():

for i in range(0, 5):

print("Número " + str(i))

funcLeitura()

>>> Número 0

Número 1

Número 2

Número 3

Número 4

# Função para somar números

def addNum(firstnum, secondnum):

print("Primeiro número: " + str(firstnum))

print("Segundo número: " + str(secondnum))

print("Soma: ", firstnum + secondnum)

# Chamando a função e passando parâmetros

addNum(45, 3)

>>> Primeiro número: 45

Segundo número: 3

Soma: 48

**Variáveis locais e globais**

# Variável Global

var\_global = 10 # Esta é uma variável global

def multiply(num1, num2):

var\_global = num1 \* num2 # Esta é uma variável local

print(var\_global)

multiply(5, 25)

>>> 125

print(var\_global)

>>> 10

# Variável Local

var\_global = 10 # Esta é uma variável global

def multiply(num1, num2):

var\_local = num1 \* num2 # Esta é uma variável local

print(var\_local)

multiply(5, 25)

>>> 125

print(var\_local)

>>> **NameError**: name 'var\_local' is not defined

**Funções Built-in**

abs(-56)

>>> 56

abs(23)

>>> 23

bool(0)

>>> False

bool(1)

>>> True

**Funções str, int, float**

# Erro ao executar por causa da conversão

idade = input("Digite sua idade: ")

if idade > 13:

print("Você pode acessar o Facebook")

>>> **TypeError**: '>' not supported between instances of 'str' and 'int'

‘…erro de Tipo de Dado…’

# Usando a função int para converter o valor digitado

idade = int(input("Digite sua idade: "))

if idade > 13:

print("Você pode acessar o Facebook")

>>> Digite sua idade: 15

Você pode acessar o Facebook

int("26")

>>> 26

float("123.345")

>>> 123.345

str(14)

>>> '14'

len([23,34,45,46])

>>> 4

‘...cada Tipo de Dado tem Metodos diferentes...’

array = ['a', 'b', 'c']

max(array)

>>> 'c'

min(array)

>>> 'a'

array = ['a', 'b', 'c', 'd', 'A', 'B', 'C', 'D']

array

>>>['a', 'b', 'c', 'd', 'A', 'B', 'C', 'D']

max(array)

>>>'d'

min(array)

>>>'A'

list1 = [23, 23, 34, 45]

sum(list1)

>>> 125

Jupyter Notebook – Funções – Parte 3/3

‘...SKLearn...’

“A scikit-learn é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python”

‘...Pypi...’

“O Python Package Index, abreviado como PyPI e também conhecido como Cheese Shop, é o repositório de software oficial de terceiros para Python. É análogo ao CPAN, o repositório para Perl. Alguns gerenciadores de pacotes, incluindo o pip, usam o PyPI como a fonte padrão para os pacotes e suas dependências.”

**Criando funções usando outras funções**

import math

def numPrimo(num):

'''

Verificando se um número

é primo.

'''

if (num % 2) == 0 and num > 2:

return "Este número não é primo"

for i in range(3, int(math.sqrt(num)) + 1, 2):

if (num % i) == 0:

return "Este número não é primo"

return "Este número é primo"

numPrimo(541)

>>>'Este número é primo'

**Fazendo split dos dados**

# Fazendo split dos dados

def split\_string(text):

return text.split(" ")

texto = "Esta função será bastante útil para separar grandes volumes de dados."

# Isso divide a string em uma lista.

print(split\_string(texto))

>>>['Esta', 'função', 'será', 'bastante', 'útil', 'para', 'separar', 'grandes', 'volumes', 'de', 'dados.']

# Podemos atribuir o output de uma função, para uma variável

token = split\_string(texto)

token

>>>['Esta',

'função',

'será',

'bastante',

'útil',

'para',

'separar',

'grandes',

'volumes',

'de',

'dados.']

caixa\_baixa = "Este Texto Deveria Estar Todo Em LowerCase"

def lowercase(text):

return text.lower()

lowercased\_string = lowercase(caixa\_baixa)

lowercased\_string

>>>'este texto deveria estar todo em lowercase'

# Funções com número variável de argumentos

def printVarInfo( arg1, \*vartuple ):

# Imprimindo o valor do primeiro argumento

print ("O parâmetro passado foi: ", arg1)

# Imprimindo o valor do segundo argumento

for item in vartuple:

print ("O parâmetro passado foi: ", item)

return;

# Fazendo chamada à função usando apenas 1 argumento

printVarInfo(10)

>>>O parâmetro passado foi: 10

printVarInfo('Chocolate', 'Morango', 'Banana')

>>>O parâmetro passado foi: Chocolate

O parâmetro passado foi: Morango

O parâmetro passado foi: Banana

Expressões Lambda

Uma das características mais útei em Python (e para iniciantes, um pouco confuso) é a expressão lambda. Expressões lambda nos permitem criar funções “anônimas”. Isto significa que podemos fazer rapidamente funções ad-hoc sem a necessidade de definir uma função usando a palavra reservada **def**.

Objetos de função desenvolvidos executando expressões lambda funcionam exatamente da mesma forma como aqueles criados e atribuídos pela palavra reservada def. Mas há algumas diferenças fundamentais que fazem lambda útil em funções especializadas:

- O corpo do lambda é uma única expressão, não um bloco de intruções.

- O corpo do lambda é semelhante a uma instrução de retorno do corpo def.

Expressões lambda realmente são úteis, quando usadas em conjunto com as funções map(), filter() e reduce().

Expressões lambda são usadas para criar funções simples.

São também chamadas *funções* *in-line* ou apenas *funções* *anônimas*

Exemplo

**lambda x: x\*\*2.**

‘...usamos a palavra reservada lambda; o primeiro x representa o parâmetro de entrada; e o x\*\*2 é o retorno da função...’

‘...ou seja, recebemos o parâmetro de entrada x e retornamos o parâmetro de entrada elevado a segunda potencia (x\*\*2)...basicamente o que fazemos é executar uma função, só que em apenas uma única instrução...não temos um bloco de intruções como temos com a palavra reservada def...’

Obs. Edu: seria um tipo de operador unário ou ternário?!

Diferença entre def e lambda para criar funções:

def -> cria um objeto, e atribui um nome a ele (nome da função)

lambda -> cria um objeto, mas o retorna como um resultado em tempo de execução

Jupyter Notebook – Expressões Lambda

DSA-Python-Cap03-07-Lambda

**Expressões Lambda**

# Definindo uma função - 3 linhas de código

def potencia(num):

result = num\*\*2

return result

potencia(5)

>>>25

# Definindo uma função - 2 linhas de código

def potencia(num):

return num\*\*2

potencia(5)

>>>25

# Definindo uma função - 1 linha de código

def potencia(num): return num\*\*2

potencia(5)

>>>25

# Definindo uma expressão lambda

potencia = lambda num: num\*\*2

potencia(5)

>>>25

# Lembre: operadores de comparação retornam boolean, true or false

Par = lambda x: x%2==0

Par(3)

>>>False

Par(4)

>>>True

first = lambda s: s[0]

first('Python')

>>>'P'

reverso = lambda s: s[::-1]

reverso('Python')

>>>'nohtyP'

‘…eu trago a sequência de caracteres da string a partir do final…’

addNum = lambda x,y : x+y

addNum(2,3)

>>>5

‘...a expressão lambda é uma forma de você criar uma função sem precisar defini-la a através de um nome; como nos fazemos com a palavra reservada def; isso nos da flexibilidade, facilidade, legibilidade, menos linha de código, entre alguma outras vantagens...’

24%

Exercicios – Métodos e Funções

# Exercício 1 - Crie uma função que imprima a sequência de números pares entre 1 e 20 (a função não recebe parâmetro) e

# depois faça uma chamada à função para listar os números

def listaPar():

for i in range(2, 21, 2):

print(i)

listaPar()

-- - - - - -

# Exercício 2 - Crie uam função que receba uma string como argumento e retorne a mesma string em letras maiúsculas.

# Faça uma chamada à função, passando como parâmetro uma string

def listaString(texto):

print(texto.upper())

return

listaString('Rumo à Análise de Dados')

>>>RUMO À ANÁLISE DE DADOS OS

--

# Exercício 3 - Crie uma função que receba como parâmetro uma lista de 4 elementos, adicione 2 elementos a lista e

# imprima a lista

def novaLista(lista):

print(lista.append(5))

print(lista.append(6))

lista1 = [1, 2, 3, 4]

novaLista(lista1)

print(lista1)

>>> None

None

[1, 2, 3, 4, 5, 6]

--

# Exercício 4 - Crie uma função que receba um argumento formal e uma possível lista de elementos. Faça duas chamadas

# à função, com apenas 1 elemento e na segunda chamada com 4 elementos

def printNum( arg1, \*lista ):

print (arg1)

for i in lista:

print (i)

return;

# Chamada à função

printNum( 100 )

printNum( 'A', 'B', 'C' )

>>> 100

A

B

C

--

# Exercício 5 - Crie uma função anônima e atribua seu retorno a uma variável chamada soma. A expressão vai receber 2

# números como parâmetro e retornar a soma deles

soma = lambda arg1, arg2: arg1 + arg2

print ("A soma é : ", soma( 452, 298 ))

>>> A soma é : 750

--

# Exercício 6 - Execute o código abaixo e certifique-se que compreende a diferença entre variável global e local

total = 0

def soma( arg1, arg2 ):

total = arg1 + arg2;

print ("Dentro da função o total é: ", total)

return total;

soma( 10, 20 );

print ("Fora da função o total é: ", total)

>>> Dentro da função o total é: 30

Fora da função o total é: 0

--

# Exercício 7 - Abaixo você encontra uma lista com temperaturas em graus Celsius

# Crie uma função anônima que converta cada temperatura para Fahrenheit

# Dica: para conseguir realizar este exercício, você deve criar sua função lambda, dentro de uma função

# (que será estudada no próximo capítulo). Isso permite aplicar sua função a cada elemento da lista

# Como descobrir a fórmula matemática que converte de Celsius para Fahrenheit? Pesquise!!!

Celsius = [39.2, 36.5, 37.3, 37.8]

Fahrenheit = map(lambda x: (float(9)/5)\*x + 32, Celsius)

print (list(Fahrenheit))

>>> [102.56, 97.7, 99.14, 100.03999999999999]

--

# Exercício 8

# Crie um dicionário e liste todos os métodos e atributos do dicionário

dic = {'k1': 'Natal', 'k2': 'Recife'}

dir(dic)

>>> ['\_\_class\_\_',

'\_\_contains\_\_',

'\_\_delattr\_\_',

'\_\_delitem\_\_',

'\_\_dir\_\_',

'\_\_doc\_\_',

'\_\_eq\_\_',

'\_\_format\_\_',

'\_\_ge\_\_',

'\_\_getattribute\_\_',

'\_\_getitem\_\_',

'\_\_gt\_\_',

'\_\_hash\_\_',

'\_\_init\_\_',

'\_\_init\_subclass\_\_',

'\_\_iter\_\_',

'\_\_le\_\_',

'\_\_len\_\_',

'\_\_lt\_\_',

'\_\_ne\_\_',

'\_\_new\_\_',

'\_\_reduce\_\_',

'\_\_reduce\_ex\_\_',

'\_\_repr\_\_',

'\_\_setattr\_\_',

'\_\_setitem\_\_',

'\_\_sizeof\_\_',

'\_\_str\_\_',

'\_\_subclasshook\_\_',

'clear',

'copy',

'fromkeys',

'get',

'items',

'keys',

'pop',

'popitem',

'setdefault',

'update',

'values']

--

import pandas as pd

pd.\_\_version\_\_

>>> '1.0.3'

--

# Exercício 9

# Abaixo você encontra a importação do Pandas, um dos principais pacotes Python para análise de dados.

# Analise atentamente todos os métodos disponíveis. Um deles você vai usar no próximo exercício.

import pandas as pd

dir(pd)

>>> ['BooleanDtype',

'Categorical',

'CategoricalDtype',

'CategoricalIndex',

'DataFrame',

'DateOffset',

'DatetimeIndex',

'DatetimeTZDtype',

'ExcelFile',

'ExcelWriter',

'Float64Index',

'Grouper',

'HDFStore',

'Index',

'IndexSlice',

'Int16Dtype',

'Int32Dtype',

'Int64Dtype',

'Int64Index',

'Int8Dtype',

'Interval',

'IntervalDtype',

'IntervalIndex',

'MultiIndex',

'NA',

'NaT',

'NamedAgg',

'Period',

'PeriodDtype',

'PeriodIndex',

'RangeIndex',

'Series',

'SparseDtype',

'StringDtype',

'Timedelta',

'TimedeltaIndex',

'Timestamp',

'UInt16Dtype',

'UInt32Dtype',

'UInt64Dtype',

'UInt64Index',

'UInt8Dtype',

'\_\_builtins\_\_',

'\_\_cached\_\_',

'\_\_doc\_\_',

'\_\_docformat\_\_',

'\_\_file\_\_',

'\_\_getattr\_\_',

'\_\_git\_version\_\_',

'\_\_loader\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

'\_\_path\_\_',

'\_\_spec\_\_',

'\_\_version\_\_',

'\_config',

'\_hashtable',

'\_is\_numpy\_dev',

'\_lib',

'\_libs',

'\_np\_version\_under1p14',

'\_np\_version\_under1p15',

'\_np\_version\_under1p16',

'\_np\_version\_under1p17',

'\_np\_version\_under1p18',

'\_testing',

'\_tslib',

'\_typing',

'\_version',

'api',

'array',

'arrays',

'bdate\_range',

'compat',

'concat',

'core',

'crosstab',

'cut',

'date\_range',

'describe\_option',

'errors',

'eval',

'factorize',

'get\_dummies',

'get\_option',

'infer\_freq',

'interval\_range',

'io',

'isna',

'isnull',

'json\_normalize',

'lreshape',

'melt',

'merge',

'merge\_asof',

'merge\_ordered',

'notna',

'notnull',

'offsets',

'option\_context',

'options',

'pandas',

'period\_range',

'pivot',

'pivot\_table',

'plotting',

'qcut',

'read\_clipboard',

'read\_csv',

'read\_excel',

'read\_feather',

'read\_fwf',

'read\_gbq',

'read\_hdf',

'read\_html',

'read\_json',

'read\_orc',

'read\_parquet',

'read\_pickle',

'read\_sas',

'read\_spss',

'read\_sql',

'read\_sql\_query',

'read\_sql\_table',

'read\_stata',

'read\_table',

'reset\_option',

'set\_eng\_float\_format',

'set\_option',

'show\_versions',

'test',

'testing',

'timedelta\_range',

'to\_datetime',

'to\_numeric',

'to\_pickle',

'to\_timedelta',

'tseries',

'unique',

'util',

'value\_counts',

'wide\_to\_long']

--

# \*\*\*\*\*\*\*\*\*\*\*\*\* Desafio \*\*\*\*\*\*\*\*\*\*\*\*\* (pesquise na documentação Python)

# Exercício 10 - Crie uma função que receba o arquivo abaixo como argumento e retorne um resumo estatístico descritivo

# do arquivo. Dica, use Pandas e um de seus métodos, describe()

# Arquivo: "binary.csv"

import pandas as pd

file\_name = "binary.csv"

def retornaArq(file\_name):

df = pd.read\_csv(file\_name)

return df.describe()

retornaArq(file\_name)

>>> admit gre gpa rank

count 5.000000 5.000000 5.000000 5.000000

mean 0.600000 616.000000 3.480000 3.000000

std 0.547723 185.148589 0.421307 1.224745

min 0.000000 380.000000 2.930000 1.000000

25% 0.000000 520.000000 3.190000 3.000000

50% 1.000000 640.000000 3.610000 3.000000

75% 1.000000 660.000000 3.670000 4.000000

max 1.000000 880.000000 4.000000 4.000000

obs. Edu, ajuda para memorizar:

‘…Pai panda leia o seguinte arquivo de formato csv: ‘binary.csv’ ( pai = pd.read\_csv(‘binary.csv’))

‘...agora que já leu, Pai descreva resumidamente o arquivo ( pai.describe() )

Quiz

listaB = [32,53,85,10,15,17,19]

soma = 0

for i in listaB:

double\_i = i \* 2

soma += double\_i

print(soma)

>>> 462