PythonBasic_LetsCode

August 17, 2021

0.1 Tipos de Variáveis

Dentre todos tipos de variáveis existem 4 tipos básicos que são fundamentais para o aprendizado inicial de programação.

0.1.1 Variável do Tipo Inteiro " int "

[1]:
```
x = 5
```

[2]:
```
print(x) # usamos a função print() para imprimir na tela o valor atribuído
         # ao que estiver dentro dos parenteses ().
```

```
5
```

[3]:
```
print(x, type(x)) # usamos a função type() para descrever o tipo de dado que␣
↪esta sendo atribuido
                  #à variável.
```

```
5 <class 'int'>
```

0.1.2 Variável do Tipo Float " float = ponto flutuante/numeros decimais"

[4]:
```
preco = 15.01
```

[5]:
```
print(preco, type(preco))
```

```
15.01 <class 'float'>
```

0.1.3 Variável do Tipo String " Str = textos "

[6]:
```
cidade = 'Curitiba' # usamos aspas simples ou duplas para atribuir textos às␣
↪variáveis do tipo str.
```

[7]:
```
print(cidade, type(cidade))
```

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

0.1.4 Variáveis do Tipo Booleana " bool = recebe valores True ou False"

[8]:
```
disponivel = True
```

[9]:
```
print(disponivel, type(disponivel))
```

```
True <class 'bool'>
```

[10]:
```
fechado = False
```

[11]:
```
print( fechado, type(fechado))
```

```
False <class 'bool'>
```

1 Operadores Aritméticos

[12]:
```
x = 50
y = 2
```

[13]:
```
print(x + y) # adição
print(x - y) # subtração
print(x * y) # multiplicação
print(x / y) # divisão
```

```
52
48
100
25.0
```

Obs: No Python o resultado da divisão de dois números inteiros será um valor do tipo float, ou seja, com casas decimais.

[14]:
```
print(x ** y) # exponenciação
print(x // y) # divisão inteira
print(x % y) # resto
```

```
2500
25
0
```

Obs: Na divisão inteira o resultado obrigatoriamente será um número inteiro, desprezando as casas decimais, esse operador não arredonda resultados apenas exclui as casas decimais.

2 Operadores Lógicos

[15]:
```
tem_cafe = True
tem_pao = False
```

[16]:
```
print( not tem_cafe) # operador (not) altera o resultado da expressão
print( tem_cafe or tem_pao) # operador (or) combina expressões booleanas, de␣
 ↪modo que o resultado seja verdadeiro, a não ser que as expressões sejam␣
 ↪falsas.
print( tem_cafe and tem_pao) # operador (and) retorna verdadeiro se ambas␣
 ↪expressões ou resultados forem verdadeiros, caso contrario retorna Falso.
```

```
False
True
False
```

3 Operadores Relacionais

Os operadores relacionais retornam um valor boleano, ou seja, Verdadeiro (True) ou Falso (False).

[17]:
```
dolar = 5.3
real = 1
```

[18]:
```
print( dolar  > real) # maior que
print( dolar  < real) # menor que
print( dolar == real) # igual a
print( dolar >= real) # maior ou igual a
print( dolar <= real) # menor ou igual a
print( dolar != real) # diferente de
```

```
True
False
False
True
False
True
```

4 Estruturas Sequenciais

[19]:
```
idade = input('Informe a sua idade: ')
# a função input(), guarda um valor passado pelo usuario numa variavel, neste␣
 ↪caso pedimos que o usuario informasse a sua idade, que será armazenada na␣
 ↪variavel idade.
print ( idade, type( idade ))
```

```
Informe a sua idade: 32
32 <class 'str'>
```

4.0.1 Obs: a função input(), sempre retornará o valor do tipo de variavel como 'str' = string, devendo ser transformada antes ou depois da operação, em qualquer outro tipo de variavel, denominando-se de casting de variavel.

[20]:
```
idade = int(idade)
print( idade, type(idade))
```

```
32 <class 'int'>
```

Outros exemplos de Casting / transformação.

[21]:
```
print(float('123.25'))
print(str('123.25'))
print(bool(''))
print(bool('abc'))
print(bool(0))
print(bool(-2))
```

```
123.25
123.25
False
True
False
True
```

5 Algoritmo

Caso problema: Férias de fim de ano.

[ ]:
```
salario_mensal = input('Digite o seu salário mensal: ')
salario_mensal = float(salario_mensal)

gasto_mensal = input('Digite o seu gasto mensal aproximado: ')
gasto_mensal = float(gasto_mensal)

salario_total = salario_mensal * 12
gasto_total = gasto_mensal * 12

montante_economizado = salario_total - gasto_total
print('O montante que voçê economizou ao final do ano foi de',␣
 ↪montante_economizado)
```

```
Digite o seu salário mensal: 1800
```

6 Estruturas Condicionais

6.0.1 if / else / elif

7 if

Testa uma condição, se ela for verdadadeira, seu conteúdo é executado, caso contrário, seu conteúdo é ignorado.

8 else

Usado quando o if é ignorado, infere-se como alternativa de resposta para o conteúdo testado pelo if, idéia de negação ou opção.

9 elif

Usado para testar diversos casos mutuamente, propõe uma nova condição para ser testada, é uma contração de " else if ".

9.1 Exemplo:

[ ]:
```
valor_passagem = 4.30

valor_corrida = float(input('Qual é o valor da corrida?'))

if (valor_corrida) <= valor_passagem * 5: # O (if) esta como condição Principal.
    print('Pague a corrida.')

elif (valor_corrida) <= valor_passagem * 6: # O (elif) neste caso esta como␣
↪condição secundária.
    print('Aguarde um momento, o valor pode abaixar.')

else:  # O (else) vem como condição final, é quando tudo da ZEBRA dai so resta␣
↪o else pra terminar a conversa.KKK:)
    print('Pegue o ônibus.')
```

10 Estruturas de Repetição

10.1 While

O While é usado para repetir uma tarefa enquanto ela for verdadeira. ### Exemplo:

[ ]:
```
x = 1            # x é a variável contadora que recebe como atribuição o valor␣
↪1;
while x <= 5:    # enquanto x for menor ou igual a 5; (obs: 5 é o limite máximo␣
↪de repetições do while).
    print(x)     # imprima na tela o valor de x;
```

```
    x = x + 1    # para que nao aconteça um loop infinito incrementamos a
↪variável x, onde, x recebe x mais 1.
```

11 Validação de Entrada com While

Podemos utilizar o While para garantir que o usuário informe de maneira correta, o que pedimos que ele faça. ## Exemplo:

[ ]:
```
texto = input('Senha do Administrador')
while texto != 'Admin':
      texto = input(' Senha incorreta !')

print('Acesso Permitido')
```

11.1 Break

Usa-se esta instrução para interromper um loop infinito, neste caso abaixo, a instrução break foi utilizada dentro do bloco de comando if. ### Exemplo:

[ ]:
```
while True:
    resposta = input('Digite OK:')
    if resposta == 'OK':
        break
```

12 Tabuada usando While

[ ]:
```
n = int(input('Tabuada do:'))
x = 0
while x <= 10:
    print( n,'x', x, '=', n*x )
    x = x + 1
```

13 Listas e Tuplas

13.1 Listas [ ]

São coleções ou sequencia ordenada de valores, esta ordenaçao é feita pela indexação de base 0, e a ultima posiçao de cada uma destas estrutura de dados, sera o tamanho dela -1. As Listas são criadas com Colchetes [ ].

[ ]:
```
nomes_paises = ['Brasil','Argentina','China','Canadá', 'Japão']
print(nomes_paises)
```

[ ]:
```
print('Tamanho da lista:', len(nomes_paises)) # a função len() retorna o
↪tamanho da lista.
```

```
[ ]: print('País:', nomes_paises[4]) # podemos localizar um dado passando a sua␣
      ↪posição na lista.
```

```
[ ]: nomes_paises[4] = ' Japão'
     print(nomes_paises)
```

```
[ ]: print('País:', nomes_paises[-1]) # podemos usar numeros negativos para␣
      ↪localizar dados do ultimo para o primeiro.
```

```
[ ]: nomes_paises[4] = 'México' # alterando um dado sobescrevendo ele.
     print(nomes_paises)
```

```
[ ]: print('País:', nomes_paises[4])
     print(nomes_paises)
```

13.1.1 Slicing (Fatiamento)

Esta operação permite o acesso ou a atribuição de vários elementos de uma Lista simultaneamente.

```
[ ]: print(nomes_paises)
```

```
[ ]: print(nomes_paises[1:3]) # Obs: o indice final nao retorna nenhum resultado.
```

```
[ ]: print(nomes_paises[1:-1]) # o numero 1 representa o indice inicial e o numero␣
      ↪-1 o indice final, os : serve como referencia de largura ou especificidade␣
      ↪dos dados a serem pesquisados.
```

```
[ ]: print(nomes_paises[2:]) # omissao do indice final, mesmo assim o Python␣
      ↪interpreta que tera que ir ate o final da lista e mostrar para trazer todos␣
      ↪os resultados.
```

```
[ ]: print(nomes_paises[:3]) # omissão do indice inicial, mesmo assim o Python␣
      ↪interpreta que tera que começar do inicio para trazer todos os resultados␣
      ↪esperados ate o indice final pedido.
```

```
[ ]: print(nomes_paises[:]) # omissão dos indices inicial e final
```

```
[ ]: print(nomes_paises[::2]) # realizando steps "pulos", basta utilizar os indices␣
      ↪inicial e final ou não, seguido de dois pontos : e o tamanho do step␣
      ↪desejado, caso pedimos que pule de dois em dois.
```

```
[ ]: print('Brasil' in nomes_paises) # utilizando o operador logico (in) para␣
      ↪verificar a existencia de um dado na lista, o retorno sera no formato␣
      ↪booleano, ou seja, True(verdadeiro) ou False(falso).
```

```
[ ]: print('Canadá'not in nomes_paises) # podemos utilizar o operador logico (not)␣
      ↪junto com o operador (in), afim de verificar uma informação.
```

13.1.2 Criando Lista e adicionando Dados

[ ]:
```
lista_capitais = [] # criamos uma lista vazia chamada lista_capitais
```

[ ]:
```
lista_capitais.append('Brasília') # o método .append(), adiciona elementos na␣
↪lista.
lista_capitais.append('Buenos Aires')
lista_capitais.append('Pequim')
lista_capitais.append('Bogotá')

print(lista_capitais)
```

[ ]:
```
lista_capitais.insert(2, 'Paris') # o metodo .insert() adiciona de forma␣
↪especifica um elemento na lista.
print(lista_capitais)
```

[ ]:
```
lista_capitais.remove('Paris') # o metodo .remove(), remove um elemento da␣
↪lista.
print(lista_capitais)
```

[ ]:
```
removido = lista_capitais.pop(2)
print(lista_capitais, removido)
```

13.2 Tuplas ( )

As Tuplas sao muito parecidas com as Listas, o grande diferencial é a flexibilidade de atuação dos dados de cada uma, enquanto nas Listas podemos inserir, alterar e remover, nas Tuplas nenhuma dessas operações poderá ser executada, noentanto as Tuplas por serem mais rigidas e especificas em sua construção, nos habilta a trabalhar com operações especiais que nas Listas nao podemos executar.

[ ]:
```
nome_paises = ('Brasil','Argentina', 'Uruguai', 'Paraguai', 'Chile',)
print(nome_paises, type(nome_paises))
```

[ ]:
```
nome_estado = 'Curitiba', # declarando uma tupla de um elemento apenas é␣
↪essencial que coloquemos a (,) para que o Python entenda que se trata de uma␣
↪tupla, caso contrario o elemento sera interpretado com uma string.
print(nome_estado, type(nome_estado))
```

[ ]:
```
len(nome_paises) # retorna a quantidade de itens na tupla
```

[ ]:
```
a, b, c, d, e = nome_paises # operação especial da tupla, chamamos de␣
↪unpacking, que consiste em atribuirmos uma variavel para cada valor ou dado␣
↪da tupla de forma simultanea.
print(a, b, c)
```

[ ]:
```
nome_paises[0] # indexação dos elementos e o fatiamento/slicing tambem se␣
↪mantem igual as Listas.
```

```
[ ]: print(*nome_paises) # print com * sera exibido apenas os valores sem estar na␣
      ↪estrutura de dados
     print(nome_paises)  # print sem * sera exibido a estrutura de dados da tupla
```

14 Strings I

Características e metodos utilizados

```
[ ]: empresa = ' Google '
     empresa = " Google " # para declarar uma string podemos utilizar aspas simples␣
      ↪e aspas duplas
     print(empresa)
```

```
[ ]: empresa = " Let's Code " # declarando uma string com apostrofo entre aspas␣
      ↪duplas
     print(empresa)
```

```
[ ]: frase = " O professor Pietro da Let's Code disse: \"Hoje tem festa\" " # usando␣
      ↪barra invertida para cancelar a operação de um limitar de string, neste caso␣
      ↪da citação foi aspas duplas.
     print (frase)
```

```
[ ]: empresa = 'Google '
     print(empresa [0]) # usando o slicing para fatiar strings
     print(empresa[:3]) # retorna os caracteres nas posições 0,1 e 2
```

14.1 Método split ( )

Separa a string em diferentes palavras, retornando o resultado como uma estrutura de Lista

```
[ ]: nomes_cidades = 'São Paulo, Belo Horizonte, Rio de Janeiro, Brasília'
     nomes_cidades = nomes_cidades.split(', ') # usando o método split() para␣
      ↪separar a nossa string em diferentes palavras separadas por (, e espaço),␣
      ↪lembrando que o metodo split() por padrao separa por espaço as palavras de␣
      ↪uma string e retorna como elemento de uma estrutura de Lista.
     print(nomes_cidades)
```

14.2 Método strip ( ) = retira espaços excessivos

```
[ ]: cabecalho = '                    MENU PRINCIPAL                '
     print(cabecalho.strip()) # o metodo strip() serve para retirar espaços execivos␣
      ↪de uma frase ou algum texto, ou de dados nao estruturados.
```

```
[ ]: nome_cidade = 'rIo DE jaNEirO'

     print(nome_cidade.title()) # metodo title() deixa as letras iniciais de cada␣
      ↪palavra em caixa alta (maiúscula)
```

```
print(nome_cidade.capitalize()) # metodo capitalize() deixa a letra inicial da␣
 ↪palavra em caixa alta (maiúscula)
print(nome_cidade.lower()) # metodo lower() deixa todas letras em caixa baixa␣
 ↪(minúscula)
print(nome_cidade.upper()) # metodo upper() deixa todas letras em caixa alta (␣
 ↪maiúscula)
```

OBS: Uma das maneiras mais usadas para utilizar o metodo lower() e upper() é a comparação de valores ja predefinidos.

14.3 Método in ( ) = esta contido

[ ]:
```
mensagem = ' Você viu o que o Pietro disse na salara ontem? '
fui_citado = ' Pietro ' in mensagem # o metodo in() nas strings é usado quando␣
 ↪queremos verificar se a informação é verdadeira ou falsa.
fui_citado2 = ' Luiza ' in mensagem # neste exemplo da palavra Luiza nao foi␣
 ↪citada na mensagem inicial, retornando um valor boleano False
print(fui_citado, fui_citado2)
```

15 String ll

Formatando utilizando operadores aritmeticos e outros.

[ ]:
```
cumprimento = ' Ola, '
nome = 'Efraim'
print(cumprimento + nome) # concatenando strings com operadores aritmeticos.
```

[ ]:
```
print(nome * 5) # multiplicando strings
```

[ ]:
```
nome = 'Efraim' # concatenando diferentes variaveis como strings, retornando um␣
 ↪frase
idade = 32
n_filhos = 5
print(nome + ' tem ' + str(idade) + ' anos e ' + str(n_filhos) + ' filhos.')
```

15.1 Método .format ( )

Usado para formatar o texto ou a mensagem, atribuindo valores nas chaves { }

[ ]:
```
print(' {} tem {} anos e {} filhos' .format(nome, idade, n_filhos)) #␣
 ↪utilizando chaves para atribuir os valores passados pelo método .format()
```

[ ]:
```
preco_gasolina = 3.476
print(' O preço da gasolina é de R$ {:.2f}'.format(preco_gasolina)) #␣
 ↪formatando usando chaves e atribuindo a quantidade de casas decimais que␣
 ↪serao apresentadas.
```

15.1.1 Atalho do Método .format()

[ ]:
```
print(f' {nome} tem {idade} anos e {n_filhos} filhos.') # usa-se p (f) na␣
↪frente de uma string para chamar o metodo .format()
```

16 Dicionários

Nos Dicionários a declaração é feita dentro de chaves a partir deste momento os dados serao compostos por chave e valor, onde a chave geralmente será uma string e seu valor pode ser qualquer tipo de dado

[ ]:
```
dados_cidades = {

    'nome':' São Paulo',
    'estado': ' São Paulo',
    'area_km2': 1521,
    'populacao_milhoes': 12.18,
                }
print(dados_cidades, type(dados_cidades))
```

[ ]:
```
print(dados_cidades['nome'])
```

[ ]:
```
dados_cidades['Pais'] = 'Brasil' # adicionando um novo par chave e valor usa-se␣
↪o colchete
print(dados_cidades)
```

[ ]:
```
dados_cidades['area_km2'] = 1600 # alterando valor de uma chave no dicionario␣
↪usa-se o colchete
print(dados_cidades)
```

[ ]:
```
dados_cidades_2 = dados_cidades # qualquer alteraçao feita no dicionario 2 sera␣
↪feita tambem no dicionario principal
dados_cidades_2['nome'] = 'Santos'# foi alterado o nome em ambos os dicionarios
print(dados_cidades_2)
print(dados_cidades)
```

[ ]:
```
dados_cidades_3 = dados_cidades.copy() # o metodo .copy() garante que o␣
↪dicionario sera copiado e que a partir deste momento, qualquer alteração␣
↪sera executada apenas no dicionario que recebeu a copia do dicionario␣
↪principal.
dados_cidades_3['estado'] = 'Rio de Janeiro'
print(dados_cidades_3)
```

[ ]:
```
print(dados_cidades)
```

[ ]:
```
print(dados_cidades)
```

[ ]:
```
novos_dados = {
    'populacao_milhoes': 15,
    'fundacao': '25/01/1554'
}

dados_cidades.update(novos_dados) # o metodo .update() é usado para atualizar e concatear outros dicionarios ou outras informaçoes no dicionario especifico
print(dados_cidades)
```

[ ]:
```
print(dados_cidades.get('prefeito')) # o metodo .get() é usado para verificar se existe a informaçao buscada no dicionario, e seu retorno sera a expressao NONE, ao inves de mostrar um erro.
```

16.0.1 3 Métodos que transformam um Dicionário em Lista

[ ]:
```
print(dados_cidades.keys()) # retorna uma lista de chaves de um dicionario
print('-------------------------')
print(dados_cidades.values()) # retorna uma lista de valores de um dicionario
print('-------------------------')
print(dados_cidades.items()) # retorna uma lista de tuplas (chave,valor) de um dicionario
```

17 Estrutura de Repetição: For

[ ]:
```
nomes_cidades = ['São Paulo','Londres','Tóquio','Paris'] # loop for com Listas
for nome in nomes_cidades: # usamos a variavel "nome" para representar todos os dados contidos na lista de " nomes_cidades"
    print(nome)
```

[ ]:
```
contador = 0
nomes_cidades = ['São Paulo','Londres','Tóquio','Paris'] # mesmo exemplo de loop so que usando o loop WHILE
while contador < len(nomes_cidades):
    print(nomes_cidades[contador])
    contador = contador + 1
```

[ ]:
```
nomes_cidades = 'São Paulo','Londres','Tóquio','Paris' # loop for usado em Tuplas
for nome in nomes_cidades:
    print(nome)
```

[ ]:
```
cidade = {                # o loop for em Dicionario é um pouco diferente porque nao se usa a variavel representativa de todos os dados do Dicionario, neste cado usa-se a propria chave como referencia
    'nome': 'São Paulo',
    'estado': 'São Paulo',
```

```
    'popoulacao': 12.2
}
for chave in cidade:
    print(f'{chave} : {cidade[chave]}') # para mostrar na tela os dados do dicionario
```

[ ]:
```
nomes_cidades = ['São Paulo','Londres','Tóquio','Paris']

for posicao in range(len(nomes_cidades)) :

    print(posicao) # mostra a posicao dos elementos da lista

    nomes_cidades[posicao] = 'Rio de Janeiro' # altera os elementos de acordo com sua posicao
print(nomes_cidades)
```

17.0.1 Outras maneiras de utilizar a função range()

[ ]:
```
print(list(range(10))) # cria uma lista de 0 ate 10 menos 1
print(list(range(2, 10))) # cria uma lista que inicia no numero 2 ate 10 menos 1
print(list(range(2, 10, 2))) # cria uma lista que inicia no numero 2 ate 10 menos 1, mais um incremento de 2, ou seja a lista sera criada com numeros pulando de 2 em 2.
```

18 Funções

[ ]:
```
def hello():        # criando uma função com a palavara reservada "def", logo depois nomeamos a função com a palavra "hello"
    print('Olá, Mundo!') # pedimos que ao chamar a função hello(), mostre na tela do usuario a mensagem citada.
hello()
```

[ ]:
```
def calcula_media(valor1, valor2, valor3): # estrutura de uma função definida para retornar uma media de 3 valores.
    soma = valor1 + valor2 + valor3
    media = soma / 3
    return media
```

[ ]:
```
resultado = calcula_media(9, 4, 5) # executando a função calcula_media definida acima.
print(resultado)
```

[ ]:
```

```

```
def calcula_media(valor1=0, valor2=0, valor3=0):  # os parametros default " 0␣
↪", precisa estar declarado apos os parametros que nao possuem os paramentros␣
↪default, caso contrario o Python apresentara um erro
    soma = valor1 + valor2 + valor3
    media = soma / 3
    return media
```

[ ]:
```
resultado = calcula_media()
print(resultado)
```

[ ]:
```
def calcula_media(*args): # o parametro *args é usado com convençao mais␣
↪poderia ser usado qualquer outro nome.
    print(args, type(args)) # ainda falando sobre o padrao *args, ele retorna␣
↪uma tupla de dados
calcula_media(10, 8, 9)
```

[ ]:
```
def calcula_media(*args, margem):
    soma = sum(args) # sum() é a função soma
    media = soma / len(args)
    return media + margem
calcula_media(10, 8, 9, margem = 0.3)
```

[ ]:
```
def print_info(**kwargs): # o retorno do parametro **kwargs, é um dicionario␣
↪composto de chave / valor.
    print(kwargs, type(kwargs))
print_info(nome = 'Efraim', sobrenome = 'Kristhianno')
```

19 Python: Manipulação de Arquivos

[ ]:
```
arquivo = open('DomCasmurro.txt', 'r', encoding = 'utf-8') # a função open()␣
↪abre o arquivo e a função " r" abre o arquivo no modo leitura
texto = arquivo.read() # a função .read(), executa a leitura do arquivo
print(texto)
arquivo.close() # a função .close(), fecha o arquivo
```

[ ]:
```
arquivo = open('DomCasmurro.txt', 'r', encoding = 'utf-8') #
linha = arquivo.readline() # a função .readline(), executa a leitura de uma␣
↪linha por vez.

while linha != '':
    print(linha, end='') # (end=''), significa que o Python não executara uma␣
↪quebra automática de linha.
    linha = arquivo.readline()
arquivo.close()
```

```
[ ]: arquivo = open('DomCasmurro.txt', 'r', encoding = 'utf-8') # utilizando o (loop␣
     ↪for) para ler o arquivo

     for linha in arquivo:
         print(linha, end= '')
     arquivo.close()
```

```
[ ]: with open('DomCasmurro.txt', 'r', encoding = 'utf-8') as arquivo: # usando o␣
     ↪comando 'with' o arquivo é aberto no modo 'r' leitura, em seguida é fechado␣
     ↪automaticamente.
         texto = arquivo.read()
         print(texto)
```

```
[ ]: with open('arquivo_teste.txt', 'w', encoding = 'utf-8') as arquivo:  # para␣
     ↪criar um arquivo do zero ou sobescrever um arquivo existente utilizamos 'w'␣
     ↪de 'write = escrita'.
         arquivo.write('Essa é a primeira linha que eu escrevi usando Python.\n')
         arquivo.write('Essa é a primeira linha que eu escrevi usando Python.\n')
```

```
[ ]: with open('arquivo_teste.txt', 'r', encoding = 'utf-8') as arquivo: # o 'r',␣
     ↪executa a leitura do arquivo
         print(arquivo.read())
```

```
[ ]: with open('arquivo_teste.txt', 'a', encoding = 'utf-8') as arquivo: #␣
     ↪utilizamos o 'a', significa .append() que adiciona uma linha ou mensagem ao␣
     ↪arquivo.
         arquivo.write('Essa é a terceira linha de Python.\n')
```

```
[ ]: with open('arquivo_teste.txt', 'r', encoding = 'utf-8') as arquivo:
         print(arquivo.read())
```

20 Manipulção de Arquivos CSV

```
[ ]: import csv
```

```
[ ]: with open('brasil_covid.csv', 'w', encoding = 'utf-8') as arquivo_csv:
         escritor = csv.writer(arquivo_csv)
```

```
[ ]: with open('brasil_covid.csv', 'r', encoding = 'utf-8') as arquivo_csv: #␣
     ↪abrindo o arquivo csv no modo leitura 'r', e dando apelido 'as' de␣
     ↪arquivo_csv.
         leitor = csv.reader(arquivo_csv) # .reader(), criando um leitor para o␣
     ↪arquivo_csv
         for linha in leitor:
             print(linha)
```

```
[ ]: with open('brasil_covid.csv', 'r', encoding = 'utf-8') as arquivo_csv:
        leitor = csv.reader(arquivo_csv)
        header = next(leitor) # usamos a funçao next(), para pular uma interação,␣
     ↪neste caso o header / cabecalho do arquivo_csv
        for linha in leitor: # lê-se, para cada linha no arquivo denominado de␣
     ↪'leitor'
            if float(linha[2]) > 1: # lê-se, convertendo para tipo float o dado da␣
     ↪linha na posição 2, pedimos que se linha 2 for maior do que 1, print na tela␣
     ↪a linha.
                print(linha)
```

```
[ ]: with open('users.csv', 'w', encoding = 'utf-8', newline ='') as arquivo_users:␣
     ↪# criando um arquivo 'csv', utilizando o 'w' de escritor
        escritor = csv.writer(arquivo_users) # setando um escritor para o␣
     ↪arquivo_users
        escritor.writerow(['nome', 'sobrenome','email','genero'])
        escritor.writerow(['Efraim', 'Kristhianno', 'ekriator@gmail.com',␣
     ↪'masculino']) # a utilização do comando (newline =''), reduzo espaçamento␣
     ↪entre as linhas e as colunas
```

```
[ ]: with open('users.csv', 'r', encoding = 'utf-8' ) as arquivo_users:
        print(arquivo_users.read())
```

21 Criando um cadastro no formato .CSV

```
[ ]: header = ['nome', 'sobrenome'] # criando um cabeçalho no formato de lista com␣
     ↪os valores de 'nome' e 'sobrenome'
     dados = [] # criando uma nova lista vazia por nome de dados, onde serao␣
     ↪atribuidos os valores das perguntas ao usuario.
     opt = input('O que deseja fazer?\n1 - Cadastrar\n0 - Sair\n') # interação com␣
     ↪usuario
     while opt != '0': # condiçao para o loop while, tradução: enquanto a pergunta␣
     ↪for diferente da opção '0', siga para proximas instruçoes.
        nome = input('Qual seu nome?')
        sobrenome = input('Qual seu sobrenome?')
        dados.append([nome, sobrenome]) # neste momento utilizamos o metodo .
     ↪append() na variavel dados que vai armazenar os valores de nome e sobrenome␣
     ↪em formato de lista.
        opt = input('O que deseja fazer?\n1 - Cadastrar\n0 - Sair\n') # e segue o␣
     ↪loop infinito....ate que se digite a opçao '0' para sair.

     print(dados)# mostra na tela todos os dados armazenados das variaveis nome e␣
     ↪sobrenome.

     with open('users.csv', 'w', newline='') as arquivo_csv: # criando um arquivo .
     ↪csv no modo escrita, ou seja pronto para editar
```

```
    writer = csv.writer(arquivo_csv) # criando um escritor, que recebe a funçao␣
↪escritor(.writer) da biblioteca (csv) passando como parametro nosso arquivo␣
↪apelidado de 'arquivo_csv'
    writer.writerow(header)# neste momento pedimos que o escritor, transcreva␣
↪as linhas da variavel (header) ou seja a lista ['nome','sobrenome']
    writer.writerows(dados)# neste momento pedimos que o escritor, transcreva␣
↪as linhas da variavel (dados) em listas

with open('users.csv', 'r') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        print(row)
```

22 APIs

Application Programming Interface

[ ]:
```
!pip install requests
```

[ ]:
```
import requests
```

[ ]:
```
url = 'https://api.exchangerate-api.com/v6/latest' # definindo a url para qual␣
↪iremos fazer a chamada da api

req = requests.get(url) # fazendo a requisao da api utilizando o metodo .
↪get(url), podemos utilizar tambem o metodo .post()

print(req.status_code) # verificando o status do codigo da url com proposito de␣
↪identificar se existe o famoso 'erro 440', se estiver tudo certo com a url o␣
↪retorno sera 200
```

[ ]:
```
dados = req.json() # o metodo .json(), pega os dados da api e transforma no␣
↪formato de Dicionario

print(dados)
```

[ ]:
```
valor_reais = float(input('Informe o valor em R$ a ser convertido\n'))
cotacao = dados['rates']['BRL'] # com base nos dados da api utilizamos␣
↪especificamente a moeda base para conversao o dolar que por sua vez na␣
↪descriçao rates guarda a taxa de conversao para outras moedas inclusivo o␣
↪Real.
print(f'R${valor_reais} em dolar valem US$ {(valor_reais / cotacao):.2f}')
```