Curso de Python Básico 2

Os comandos **input()**, **print()**, **int()**, **float()** entre outros são funções nativas (Built-in*) no Python.

input() solicita uma informação do usuário e nos retorna o valor informado.
print() apresenta o valor de uma variável.
int() converte um valor e o retorna como um número inteiro.

No entanto, é possível definir e usar as novas funções.

*https://docs.python.org/3/library/functions.html

Função é uma sequência de comandos que realiza uma tarefa específica.

Em um programa, se há uma sequência de comandos que realiza uma tarefa específica e se repete em vários pontos, este trecho de código "tem cara de ser" uma função.

A vantagem de ter um código (sequência de comandos) dentro de uma função é que, caso precise realizar alguma melhoria ou correção, ela vai ser feita somente dentro da função e não em diversas partes do código.

O código da função tem que ser preferencialmente genérico, funcionar com quaisquer valores, que são passados como parâmetros.
Isso facilita seu reuso.

Para definir uma função

- **Sintaxe:**

```
def <nome da função> ():
```

Quando a função não recebe parâmetros

ou

```
def <nome da função> (<parâmetro(s)>):
    <comandos que quero executar>
    return (caso essa função retorne algum valor)
```

Quando a função recebe parâmetros

Quando a função não tem return, o valor retornado é None

- **Exemplo:**

```
def soma(a, b):
    return a + b
```

definição da função soma

```
print(soma(1, 2))
>>> 3
```

chamada (uso) da função soma

ou

```
print(soma('PyLadies', ' São Paulo'))
>>> PyLadies São Paulo
```

usando strings como parâmetro

- **Exemplo:** função de multiplicação:

```
def multiplica(n1, n2):
    return n1 * n2

n1 = float(input('Informe o primeiro número: '))
n2 = float(input('Informe o segundo número: '))

print(multiplica(n1, n2))
```

definição da função

chamada (uso) da função

```
>>>
Informe o primeiro número: 6
Informe o segundo número: 7
42.0
>>>
```

- **Exemplo:** função de imc

Função para calcular o IMC (Índice de Massa Corporal).

O cálculo é feito dividindo o peso (em kg) pela altura (em metros) ao quadrado ou:

$$\frac{\text{peso}}{\text{altura}^2}$$

definição da função

```
def imc(peso, altura):
    return(peso / (altura ** 2))
```

chamadas (uso) da função

```
print(imc(70, 1.62))
print(imc(45, 1.62))
print(imc(78, 1.62))
```

- Exemplo de função mal planejada:

```
def soma():
    return 5 + 9


print(soma())
>>> 14
```

O código da função tem que ser preferencialmente genérico, funcionar com quaisquer valores, que são passados como parâmetros.
Isso facilita o reuso.
Fixar valores dentro da função pode fazê-la pouco útil.

PROGRAMANDO EM PYTHON

FUNÇÕES

```
retorno = funcao(parametros)
```

```
numero = int('2')
```


```
pipoca = fazer_pipoca()
pipoca_amanteigada = fazer_pipoca(manteiga)

suco1 = bater_no_liquidificador(agua, ['morango',
'kiwi', 'manga'], gelo)
vitamina = bater_no_liquidificador(leite,
['chocolate'], gelo)
```

- **Exemplo:**

```
def fazer_pipoca(manteiga=False):
    return a + b
```

chamadas (uso) da função soma

```
pipoca = fazer_pipoca()
pipoca_amanteigada = fazer_pipoca(True)
```

definição da função com parâmetro opcional.
Ao indicar um valor pré-estabelecido, indica que o parâmetro não precisa ser indicado na chamada da função e que vai assumir o valor pré-estabelecido.
Os parâmetros deste tipo tem que ser os últimos a serem definidos.

pyladies São Paulo

- **Exemplo:**

```
def bater_no_liquidificador(liquido, solidos, gelo=False):
    resultado = solidos
    resultado.append(liquido)
    if gelo is True:
        resultado.append('cubos de gelos')
    return resultado


suco1 = bater_no_liquidificador(agua, ['morango', 'kiwi', 'manga'], gelo)
vitamina = bater_no_liquidificador(leite, ['chocolate'], gelo)
```

AGORA É COM VOCÊ

Em um arquivo, faça um **programa** que:

- pergunte o peso, a altura
- calcule o IMC
- avalie o IMC segundo a tabela da Organização Mundial da Saúde (OMS) abaixo:

Categoria	IMC
Abaixo do peso	Abaixo de 20
Peso normal	20,0 a 24,9
Sobrepeso	25,0 a 29,9
Obesidade	30,0 e acima

Dica:
Use funções.
Os verbos/ações do enunciado do problema dão a indicação de que é uma ou mais funções do programa, seja nativa do Python ou não

pyladies São Paulo

RESPOSTA

```
def avaliar_imc(indice):
    if indice < 20:
        return 'Você está abaixo do peso'
    elif indice < 24.9:
        return 'Você está com o peso normal'
    elif indice < 29.9:
        return 'Você está com sobrepeso'
    else:
        return 'Você está com obesidade'


def imc(peso, altura):
    return peso / (altura ** 2)
```

RESPOSTA (continuação)

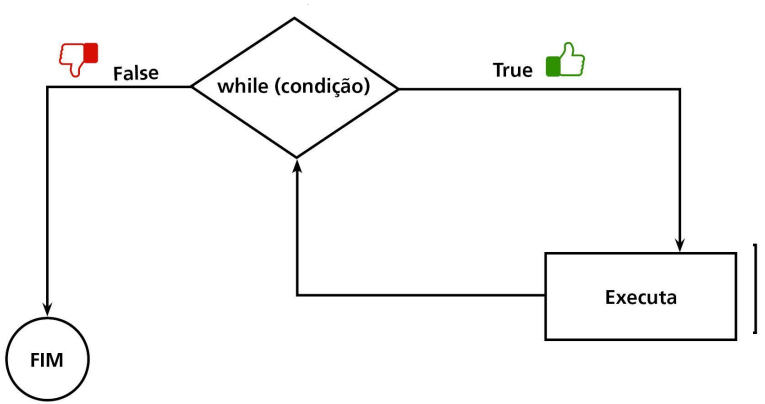
```
peso = float(input('Entre com seu peso em kg:'))
altura = float(input('Entre com sua altura em metros:'))


indice = imc(peso, altura)
print('Seu IMC é {:.2f}'.format(indice))
print(avaliar_imc(indice))
```

PROGRAMANDO EM PYTHON

WHILE (ENQUANTO)

- **while** é usado quando precisamos repetir uma ação algumas vezes ou fazer uma iteração até confirmar uma condição.



- **Sintaxe:**

```
while <condição a ser verificada>:
    <comando que quero executar>
```

WHILE (ENQUANTO)

• **Exemplo 1:** programa que solicita seu peso por vários dias até que o valor informado é nenhum, ou seja, o usuário apenas usou a tecla ENTER.

```
while True:
    resposta = input('Digite seu peso: ')
    if resposta == '':
        break
    peso = float(resposta)
    print('Peso: {}.'.format(peso))
```

WHILE (ENQUANTO)

• **Exemplo 2:** Este programa é similar ao anterior, mas também armazena os valores (em lista) para depois poder imprimir o menor valor, o maior valor e a média, usando funções nativas do Python.

```
pesos = []
while True:
    resposta = input('Digite seu peso: ')
    if resposta == '':
        break
    peso = float(resposta)
    print('Peso: {}.'.format(peso))
    pesos.append(peso)
print('Menor peso: {}.'.format(min(pesos)))
print('Maior peso: {}.'.format(max(pesos)))
print('Média dos pesos: {}.'.format(sum(pesos)/len(pesos)))
```

WHILE (ENQUANTO)

pyladies São Paulo

• **Exemplo 3:** Este programa é similar ao anterior, mas desta vez, solicita além do peso, o nome de uma pessoa. Imprime o menor peso e o maior peso e a pessoa correspondente.

```
lista_peso_pessoa = []
while True:
    resposta = input('Digite o nome e o peso, separados por espaço.
Exemplo: Annie 63.5')
    if resposta == '':
        break
    nome, peso = resposta.split(' ')
    print('Nome {}: {} kg.'.format(nome, peso))
    peso_e_nome = (float(peso), nome)
    lista_peso_pessoa.append(peso_e_nome)
print('Menor peso: {}.'.format(min(lista_peso_pessoa)))
print('Maior peso: {}.'.format(max(lista_peso_pessoa)))
print(sorted(lista_peso_pessoa))
```

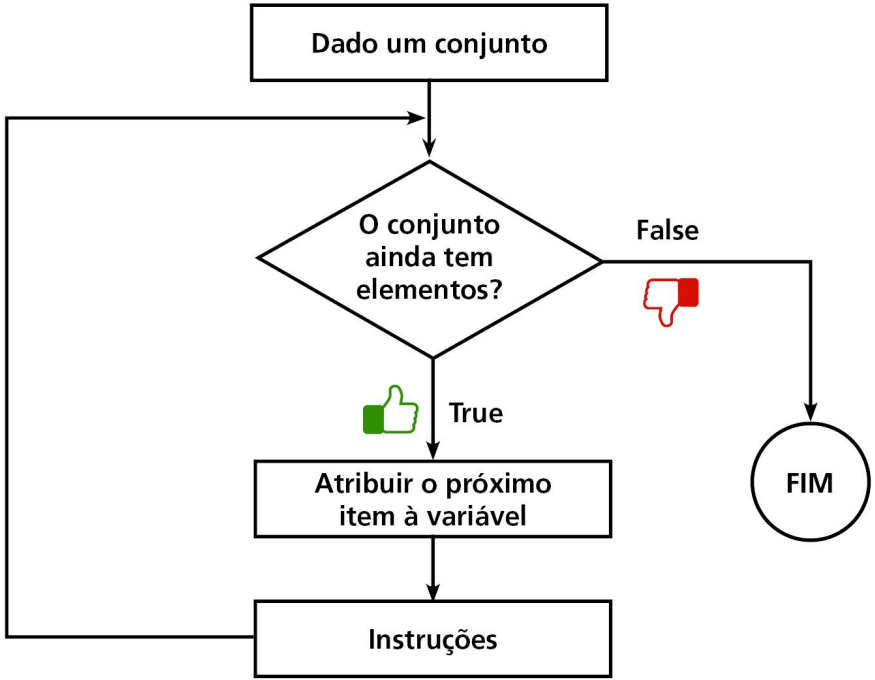
Isso é uma tupla

é uma lista de tuplas e é possível ordená-la

PROGRAMANDO EM PYTHON

FOR (PARA)

O comando **for** opera sobre os itens de qualquer tipo de sequência (iteráveis, por exemplo: lista, string entre outros), na ordem em que eles aparecem na sequência. A variável que aparece na linha do **for** se comporta como cada item da lista.



FOR (PARA)

- **Sintaxe:**

```
for <variável> in <lista>:
    <comando que quero executar>
```

- **Exemplo 1:**

```
alunas = ['Ana', 'Beatriz', 'Caroline', 'Denise', 'Élida',
'Fernanda', 'Glaucia']
for nome in alunas:
    print(nome)
```

FOR (PARA CADA)

• **Exemplo 2:** Este programa pergunta os nomes de pessoas e depois o peso de cada uma delas.

```
nomes = input('Digite vários nomes separados por vírgula: ')
pessoas = nomes.split(',')
print(pessoas)
lista_peso_nome = []
for nome in pessoas:
    resposta = input('Digite o peso de {}: '.format(nome))
    peso = float(resposta)
    print('Nome {}: {} kg.'.format(nome, peso))
    lista_peso_nome.append((peso, nome))
print(sorted(lista_peso_nome))
```

FOR (PARA CADA)

- **Exemplo 3:** Este programa pergunta os nomes de pessoas e depois o peso de cada uma delas. Ao final, apresenta uma lista ordenada por peso e pessoa e outra ordenada por nome e peso

```
nomes = input('Digite vários nomes separados por vírgula: ')
pessoas = nomes.split(',')
print(pessoas)
lista_peso_nome = []
lista_nome_peso = []
for nome in pessoas:
    resposta = input('Digite o peso de {}: '.format(nome))
    peso = float(resposta)
    print('Nome {}: {} kg.'.format(nome, peso))
    lista_peso_nome.append((peso, nome))
    lista_nome_peso.append((nome, peso))
print(sorted(lista_peso_nome))
print(sorted(lista_nome_peso))
```

AGORA É COM VOCÊ

Em um arquivo, faça um **programa** que:

- pergunte o nome de um número de pessoas indefinido, ou seja, o quanto o usuário quiser informar.
- pergunte para cada pessoa, seu peso e altura
- imprima o **nome** da pessoa com **o segundo menor** imc
- imprima **o resultado da avaliação do imc** da pessoa que tem o **penúltimo valor** de imc (dica: use fatiamento)

RESPOSTA POSSÍVEL

```
nomes = input('Digite vários nomes separados por vírgula: ')
pessoas = nomes.split(',')
print(pessoas)
lista = []
for nome in pessoas:
    peso = float(input('Digite o peso de {}: '.format(nome)))
    altura = float(input('Digite o peso de {}: '.format(nome)))
    print('Nome {}: {} kg {} m.'.format(nome, peso, altura))
    lista.append((imc(peso, altura), nome))
lista_ordenada = sorted(lista)
print(lista_ordenada)

nome_segundo_menor_imc = lista_ordenada[1][1]
print(nome_segundo_menor_imc)

penultimo_imc = lista_ordenada[-2][0]
print(avaliar_imc(penultimo_imc))
```

Dicionário é uma coleção não ordenada de pares chave-valor.

Diferentemente de listas e strings em que cada elemento é chamado por um índice numérico, cada elemento em um dicionário é chamado por uma chave (que tem que ser de tipos imutáveis como, por exemplo, string ou numérico ou tupla).

Usa-se dicionário quando se deseja obter a informação por meio de uma chave, quando a ordem dos elementos não é importante.

DICIONÁRIOS

- **Sintaxe:**

```
<variável> = {}
```

dicionário

vazio

- **Exemplo:**

```
pessoa = {}
```

- **Sintaxe:**

```
<variável> = {<chave1>: <valor1>, <chave2>: <valor2>, <chave3>: <valor3>, <chave4>: <valor4>, }
```

dicionário com valores

- **Exemplo:**

```
>>> pessoa = {'nome': 'Alice', 'idade': 12, 'país': 'Das Maravilhas', 'peso': 45.8, 'altura': 1.50, 'imc': 20.35}

# chave é peso, valor é 45.8
>>> pessoa['peso']
45.8
```

DICIONÁRIOS

É possível criar um dicionário a partir do comando **dict()** aplicado a uma **lista** de **tuplas de dois elementos**:

```
>>> tupla1 = ('nome', 'Alice')
>>> tupla2 = ('idade', 12)
>>> tupla3 = ('peso', 45.8)
>>> tupla4 = ('altura', 1.50)

>>> lista = [tupla1, tupla2, tupla3, tupla4]
>>> print(lista)

>>> pessoa = dict(lista)

>>> pessoa
{'nome': 'Alice', 'idade': 12, 'peso': 45.8, 'altura': 1.50,}
```

DICIONÁRIOS

- Para criar um novo valor ou atualizar o valor de um elemento do dicionário:

```
>>> pessoa = {'nome': 'Alice', 'idade': 12, 'peso': 45.8}

>>> pessoa['peso'] = 8

>>> pessoa
{'nome': 'Alice', 'idade': 12, 'peso': 8}


>>> pessoa['nacionalidade'] = 'inglesa'

>>> pessoa
{'nome': 'Alice', 'idade': 12, 'peso': 8, 'nacionalidade': 'inglesa'}
```

DICIONÁRIOS

pyladies São Paulo

- Para apagar um elemento:

```
>>> pessoa = {'nome': 'Annie', 'idade': 12, 'peso': 45.8,
'nacionalidade': 'canadense'}

>>> del pessoa['nome']

>>> pessoa
{'idade': 12, 'peso': 45.8, 'nacionalidade': 'canadense'}

>>> pessoa.pop('peso')

>>> pessoa
{'idade': 12, 'nacionalidade': 'canadense'}
```

pyladies São Paulo

- Para acrescentar e/ou atualizar vários elementos ao dicionário:

```
>>> pessoa = {'nome': 'Laura', 'idade': 12, 'peso': 45.8,}

>>> pessoa.update({'nome': 'Annie', 'nacionalidade': 'canadense'})

>>> pessoa
{'nome': 'Annie', 'idade': 12, 'peso': 45.8, 'nacionalidade':
'canadense'}
```

DICIONÁRIOS

- Para obter o valor de um elemento e retornar um valor default caso o elemento não exista:

```
>>> pessoa = {'nome': 'Laura', 'idade': 12, 'peso': 45.8,}

>>> pessoa.get('cidade', 'Londres')
Londres

>>> pessoa.get('nome', 'Branca de Neve')
Laura
```

pyladies São Paulo

AGORA É COM VOCÊ

Crie um dicionário que contenha a tradução da palavra GATO para os idiomas inglês (cat), espanhol (gato) e francês (chat).

Escreva os comandos para imprimir (print) a palavra gato em cada um dos idiomas nesta sequência: inglês, espanhol, francês.

RESPOSTAS POSSÍVEIS

```
>>> gatos = {}
>>> gatos['inglês'] = 'cat'
>>> gatos['espanhol'] = 'gato'
>>> gatos['francês'] = 'chat'
```

ou

```
>>> gatos = {'inglês': 'cat', 'espanhol': 'gato', 'francês':
'chat', }
>>> gatos['inglês']
>>> gatos['espanhol']
>>> gatos['francês']
```

- Para obter, respectivamente, todos os elementos, os valores, as chaves, usamos as seguintes funções de dicionários: `.items()`, `.values()`, `.keys()`

```
>>> gatos = {'inglês': 'cat', 'espanhol': 'gato', 'francês': 'chat', }

>>> gatos.items()
dict_items([('inglês','cat'),('francês','chat'),('espanhol','gato')])

>>> gatos.values()
dict_values(['cat', 'chat', 'gato'])

>>> gatos.keys()
dict_keys(['inglês', 'francês', 'espanhol'])
```

dict_items,
dict_values,
dict_keys
são tipos iteráveis

dict_items, dict_values, dict_keys são tipos iteráveis como as listas, por exemplo.
Então é possível aplicar o comando **for**

```
gatos = {'inglês': 'cat', 'espanhol': 'gato', 'francês': 'chat', }

for item in gatos.items():
    print(item)

for item in gatos.keys():
    print(item)

for item in gatos.values():
    print(item)
```

pyladies São Paulo

AGORA É COM VOCÊ

Crie um dicionário que contenha a tradução da palavra GATO para os idiomas inglês (cat), espanhol (gato) , francês (chat), alemão (Katze), italiano (gatto).

Escreva os comandos que imprimirão a palavra gato em cada um dos idiomas, lembrando de indicar em que idioma a palavra está, por exemplo:
português gato ou gato português
O importante é mostrar a palavra e o idioma correspondente
Dica: use o comando **for**

RESPOSTAS POSSÍVEIS

```
for item in gatos.items():
    print(item)
```

ou

```
for item in gatos.items():
    print(item[0] + ' ' + item[1])
```

ou

```
for item in gatos.items():
    print(item[1] + ' ' + item[0])
```

pyladies São Paulo

AGORA É COM VOCÊ

Use o seguinte dicionário do exercício anterior:

```
gatos = {'inglês': 'cat', 'espanhol': 'gato',
'francês': 'chat', 'alemão': 'Katze', 'italiano':
'gatto',}
```

Faça um programa que imprima:

1) Os **ítens** do dicionário em ordem crescente
2) Os **valores** do dicionário em ordem decrescente
3) As **chaves** que contenham a 'ês' no final e em ordem crescente
4) Os **valores** dos ítens cujas chaves que não contenham 'ês' no final

RESPOSTAS POSSÍVEIS

1) Os **ítens** do dicionário em ordem crescente

```
for item in sorted(gatos.items()):
    print(item)
```

ou

```
print(sorted(gatos.items()))
```

RESPOSTAS POSSÍVEIS

2) Os **valores** do dicionário em ordem decrescente

```
for item in sorted(gatos.values(), reverse=True):
    print(item)
```

ou

```
print(sorted(gatos.values(), reverse=True))
```

reverse é um parâmetro opcional da função sorted

RESPOSTA POSSÍVEL

3) As **chaves** que contenham a 'ês' no final e em ordem crescente

```
for item in sorted(gatos.keys()):
    if item.endswith('ês'):
        print(item)
```

RESPOSTA POSSÍVEL

4) Os **valores** dos ítens cujas chaves que não contenham 'ês' no final

```
for item in sorted(gatos.keys()):
    if not item.endswith('ês'):
        print(gatos[item])
```

MÓDULOS: são os arquivos .py

PACOTES: são pastas que contém os módulos e obrigatoriamente um dos módulos tem que ser **__init__.py** mesmo que seu conteúdo é vazio.

Conforme vamos escrevendo as instruções, o programa vai ficando muito longo e difícil de dar manutenção.
Sendo assim, um programa pode ser composto por vários módulos.

As definições podem ser escritas em um módulo e podem ser importadas para ser usada em vários outros módulos ao invés de repetir o mesmo trecho de código (redefinições) em várias partes do programa.

MÓDULOS E PACOTES

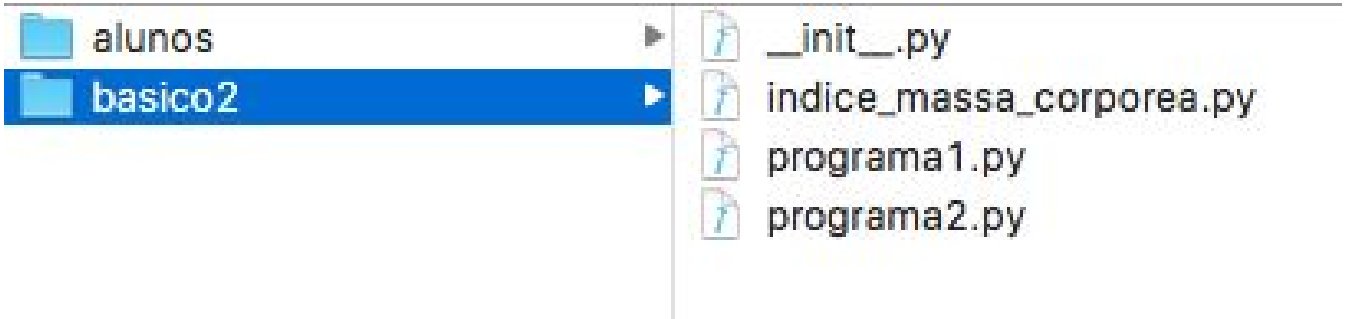
PROGRAMA

- **PACOTES (ou bibliotecas)**
- **MÓDULOS (são os arquivos .py)**

- **PACOTES (ou bibliotecas)**
- **MÓDULOS (são os arquivos .py)**

- definições de funções
  variáveis etc

Por exemplo, podemos ter o módulo **indice_massa_corporea.py** contendo as função **imc()** e **avaliar_imc()** como já vimos anteriormente.



pyladies São Paulo

conteudo de indice_massa_corporea.py

```
def avaliar_imc(indice):
    if indice < 20:
        return 'Você está abaixo do peso'
    elif indice < 24.9:
        return 'Você está com o peso normal'
    elif indice < 29.9:
        return 'Você está com sobrepeso'
    else:
        return 'Você está com obesidade'

def imc(peso, altura):
    return peso / (altura ** 2)
```


```
# conteudo do programa1 (por exemplo)

from . import indice_massa_corporea

valor_imc = indice_massa_corporea.imc(55, 1.50))
print(indice_massa_corporea.avaliar_imc(valor_imc))

# conteudo do programa2 (por exemplo)

from indice_massa_corporea import imc
from indice_massa_corporea import avaliar_imc

valor_imc = imc(55, 1.50)
print(avaliar_imc(valor_imc))
```

A origem dos pacotes e módulos de um programa pode ser:

1. da biblioteca padrão do Python, ou seja, que é instalada ao instalar Python (https://docs.python.org/3/library/);
2. de terceiros, ou seja, que tem que ser instalado a parte (https://pypi.python.org/pypi, github, etc.);
3. os que você criou para seu programa

A PEP8 (pep eight) (www.python.org/dev/peps/pep-0008/) – um guia de estilo de programação para Python – recomenda a ordem de importação acima. E cada grupo separado por uma linha.
Além disso, todas as importações devem estar no início do módulo, como boa prática.

ARQUIVOS

Para manipular arquivos é necessário usar módulo os da biblioteca padrão do Python.

```
>>> import os
>>> os.listdir('/users/pyladies/desktop')
>>> os.makedirs('/users/pyladies/desktop/')
```

Para manipular arquivos é necessário usar módulo os da biblioteca padrão do Python.

```
import os
```

Para listar os arquivos de uma pasta

- **Sintaxe:**

```
os.listdir(<caminho da pasta>)
```

- **Exemplo:**

```
os.listdir('/users/pyladies/desktop/')
```

Para criar todas as pastas de um caminho

- **Sintaxe:**

```
os.makedirs(<caminho da pasta>)
```

- **Exemplo:**

```
os.makedirs('/users/pyladies/desktop/curso básico 2')
```

Para saber se um caminho é uma pasta

- **Sintaxe:**

```
os.path.isdir(<caminho da pasta>)
```

- **Exemplo:**

```
os.path.isdir('/users/pyladies/desktop/curso básico 2')
```

Para saber se um caminho é um arquivo

- **Sintaxe:**

```
os.path.isfile(<caminho da pasta>)
```

- **Exemplo:**

```
os.path.isfile('/users/pyladies/desktop/curso básico 2/texto.txt')
```

ARQUIVOS: LEITURA E ESCRITA DE ARQUIVO TEXTO

Os programas trabalham com entrada e saída de dados.
Até então usamos input() e print()para fazer respectivamente entrada e saída.

Arquivos são uma das formas de fazer entrada e saída, ou seja, leitura e escrita de arquivos. Por exemplo:

Entrada: quando programa lê um texto de um arquivo para contar a quantidade de cada palavra no texto.

Saída: guardar um relatório em arquivo.

PROGRAMANDO EM PYTHON

ARQUIVOS: ESCRITA

Para fazer a escrita de um arquivo:

- **Sintaxe:**

```
with open(<caminho do arquivo>, 'w')  as <variavel>:
    <variavel>.write(<conteudo do arquivo>)
```

w é de write (escrita)

apaga o conteúdo que já está no arquivo antes de escrever

- **Exemplo:**

```
with open('/users/pyladiessp/desktop/meuarquivo.txt', 'w') as
arquivo:
    arquivo.write('Olá!')
    arquivo.write('Estou aprendendo escrita em arquivo com Python')
```

PROGRAMANDO EM PYTHON

ARQUIVOS: ESCRITA

Para fazer a escrita de um arquivo, mantendo o seu conteúdo:

a é de append (escrita acrescentando conteúdo sem apagar o que já está no arquivo)

- **Sintaxe:**

```
with open(<caminho do arquivo>, 'a')  as <variavel>:
    <variavel>.write(<conteudo do arquivo>)
```

- **Exemplo:**

```
with open('/users/pyladiessp/desktop/meuarquivo.txt', 'a') as
arquivo:
    arquivo.write('Olá!')
    arquivo.write('Estou aprendendo escrita em arquivo com Python,
acrescentando o conteúdo ao arquivo')
```

ARQUIVOS: LEITURA

Para fazer a leitura de um arquivo:

r é de read (leitura)

- **Sintaxe:**

```
with open(<caminho do arquivo>, 'r') as <variavel>:
    <conteudo do arquivo> = <variavel>.read()
```

- **Exemplo:**

```
with open('/users/pyladiessp/desktop/meuarquivo.txt', 'r')
as arquivo:
    conteudo = arquivo.read()
    print(conteudo)
```

pyladies São Paulo

csv são arquivos que contém dados tabulados.

ARQUIVOS: ESCRITA DE ARQUIVO CSV

Use `newline=''` para que funcione em qualquer sistema operacional

Para fazer a escrita de um arquivo csv:

- **Sintaxe:**

```
import csv
with open(<caminho do arquivo>, 'w', newline='') as <variavel>:
    writer = csv.DictWriter(<variavel>, fieldnames=<lista nomes das
colunas>)
    writer.writeheader()
    writer.writerow(<dicionario com nome coluna e valor>)
    writer.writerow(<dicionario com nome coluna e valor>)
    writer.writerow(<dicionario com nome coluna e valor>)
```

If newline='' is not specified, newlines embedded inside quoted fields will not be interpreted correctly, and on platforms that use \r\n linendings on write an extra \r will be added. It should always be safe to specify newline='', since the csv module does its own (universal) newline handling.

ARQUIVOS: ESCRITA DE ARQUIVO CSV

- **Exemplo:**

```
import csv
with open('pessoas.csv', 'w', newline='') as csvfile:
    colunas = ['nome', 'peso', 'altura']
    writer = csv.DictWriter(csvfile, fieldnames=colunas)
    writer.writeheader()
    writer.writerow({'nome': 'Annie','peso': 35.5, 'altura':
1.30,})
    writer.writerow({'nome': 'Claire','peso': 39.5, 'altura':
1.35,})
```

ARQUIVOS: LEITURA DE ARQUIVO CSV

Para fazer a leitura de um arquivo csv:

- **Sintaxe:**

```
import csv
with open(<caminho do arquivo>, newline='') as <variavel>:
    reader = csv.DictReader(<variavel>)
    for linha in reader:
        print(linha)
        <comandos que usem linha>
```

ARQUIVOS: LEITURA ARQUIVO CSV

Para fazer a leitura de um arquivo csv:

- **Exemplo:**

```
import csv
with open('pessoas.csv', newline='') as arquivo:
    reader = csv.DictReader(arquivo)
    for linha in reader:
        print(linha)
        print(linha['nome'], linha['peso'])
```

AGORA É COM VOCÊ

Faça um programa que leia dados fornecidos pelo usuário e os escreva um arquivo csv cujas colunas são: Nome, Peso e Altura.

Dica: Procure criar funções para futuro reuso.
O arquivo csv poderia ficar assim, por exemplo:

Nome	Peso	Altura
Rosa	45	1.40
Laura	53.5	1.65
Adriana	60.3	1.87
Clarice	67.8	1.58

RESPOSTA POSSÍVEL

```
def ler_dados():
    nomes = input('Digite vários nomes separados por vírgula: ')
    pessoas = nomes.split(',')
    print(pessoas)
    lista = []
    for nome in pessoas:
        peso = float(input('Digite o peso de {}: '.format(nome)))
        altura = float(input('Digite o peso de {}:
'.format(nome)))
        print('Nome {}: {} kg {} m.'.format(nome, peso, altura))
        lista.append((nome, peso, altura))
    return lista
```

No lugar de tupla, você também pode usar dicionários

continua...

RESPOSTA POSSÍVEL (continuação)

```
import csv

def escrever_dados(nome_arquivo, pessoas):
    colunas = ['Nome', 'Peso', 'Altura']
    with open(nome_arquivo, 'w', newline='') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=colunas)
        writer.writeheader()
        for nome, peso, altura in pessoas:
            pessoa = {}
            pessoa['Nome'] = nome
            pessoa['Peso'] = peso
            pessoa['Altura'] = altura
            writer.writerow(pessoa)
```

continua...

RESPOSTA POSSÍVEL (continuação)

```
pessoas = ler_dados()

# pessoas é uma lista de tuplas (Nome, Peso, Altura)
escrever_dados('pessoas.csv', pessoas)
```

Observe como o programa é pequeno. Contém apenas 2 linhas.
As funções fazem todo o trabalho e elas podem ser reusadas (importadas) em outros programas.

OUTRA RESPOSTA POSSÍVEL

```
def ler_dados():
    nomes = input('Digite vários nomes separados por vírgula: ')
    pessoas = nomes.split(',')
    print(pessoas)
    lista = []
    for nome in pessoas:
        peso = float(input('Digite o peso de {}: '.format(nome)))
        altura = float(input('Digite o peso de {}: '.format(nome)))
        print('Nome {}: {} kg {} m.'.format(nome, peso, altura))
        pessoa = {'Nome': nome, 'Peso': peso, 'Altura': altura)
        lista.append(pessoa)
    return lista
```

No lugar de tupla, você também pode usar dicionário

continua...

OUTRA RESPOSTA POSSÍVEL (continuação)

```
import csv

def escrever_dados(nome_arquivo, pessoas):
    colunas = ['Nome', 'Peso', 'Altura']
    with open(nome_arquivo, 'w', newline='') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=colunas)
        writer.writeheader()
        for pessoa in pessoas:
            # p = {}
            # p['Nome'] = pessoa['Nome']
            # p['Peso'] = pessoa['Peso']
            # p['Altura'] = pessoa['Altura']
            # writer.writerow(p)
            writer.writerow(pessoa)
```

pessoa e p são dicionários cujas chaves são iguais: Nome, Peso, Altura, ou seja, p é igual a pessoa. Então no lugar de 5 linhas podemos usar uma única. Se as chaves fossem diferentes, teria que ser feito a equivalência das chaves.

continua...

OUTRA RESPOSTA POSSÍVEL (continuação)

```
pessoas = ler_dados()

# pessoas é um dicionário {Nome, Peso, Altura}
escrever_dados('pessoas.csv', pessoas)
```

Note que o programa não mudou, apesar de o conteúdo das duas funções tenha mudado.

pyladies São Paulo

AGORA É COM VOCÊ

Faça um programa que leia em um arquivo csv que contenha as colunas: Nome, Peso, Altura.

Dica: Procure criar funções para futuro reuso.

RESPOSTA POSSÍVEL

```
import csv

def ler_dados_do_csv(nome_arquivo):
     resultado = []
    with open(nome_arquivo, newline='') as arquivo:
        reader = csv.DictReader(arquivo)
        for linha in reader:
            resultado.append(linha)
    return resultado

# pessoas é uma lista de dicionários
pessoas = ler_dados_do_csv('pessoas.csv')
print(pessoas)
```

pyladies São Paulo

AGORA É COM VOCÊ

Complete o programa, sendo que o que ele faz é:

1. Ler em um arquivo csv os seguintes dados de Nome, Peso e Altura
2. Calcular o imc() e avaliar o valor do imc().
3. Escrever um novo arquivo csv com os dados originais mais os dados calculados: imc e a avaliação

Dica: Importação de módulos e reuso de funções.
Que tal ter um módulo com apenas imc e avaliar_imc?
Que tal ter um módulo só para ler e escrever csv?
Lembre-se de que um programa pode ser formado por mais de um módulo (arquivo .py)

Programa

```
# Este é o módulo principal (principal.py) do meu programa

# COMPLETAR.
# Dica 1: comece com as importações
# Dica 2: defina a função calcular_e_avaliar_imc_das pessoas

# PROGRAMA PRINCIPAL
pessoas = meu_csv.ler_dados('pessoas.csv')
# pessoas é uma lista de dicionários (Nome, Peso, Altura)
pessoas = calcular_e_avaliar_imc_das_pessoas(pessoas)
colunas = ['Nome','IMC','Avaliação do IMC','Peso','Altura',]
meu_csv.escrever_dados('avaliacao_imc.csv', colunas, pessoas)
```

Pasta do programa

arquivos dentro da pasta

pasta

```
meu_programa { __init__.py
               meu_csv.py
               indice_massa_corporea.py
               principal.py
```

RESPOSTA POSSÍVEL

```
# Este é o início do módulo (arquivo) principal do meu programa

import meu_csv
import indice_massa_corporea

def calcular_e_avaliar_imc(pessoas):
    dados_atualizados = []
    for pessoa in pessoas:
        imc = indice_massa_corporea.imc(pessoa['Peso'],
pessoa['Altura'])
        resultado = indice_massa_corporea.avaliar_imc(imc)
        pessoa.update({'IMC': imc, 'Avaliação do IMC': resultado}
        dados_atualizados.append(pessoa)
    return dados_atualizados
```

RESPOSTA POSSÍVEL (continuação)

```
# Este é o módulo meu_csv.py. Note que ele pode ser reusado facilmente
import csv

def ler_dados_do_csv(nome_arquivo):
    resultado = []
    with open(nome_arquivo, newline='') as arquivo:
        reader = csv.DictReader(arquivo)
        for linha in reader:
            resultado.append(linha)
    return resultado

def escrever_dados_em_csv(nome_arquivo, colunas, dados):
    with open(nome_arquivo, 'w', newline='') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=colunas)
        writer.writeheader()
        for item in dados:
            writer.writerow(item)
```

RESPOSTA POSSÍVEL (continuação)

```
# Este é o módulo indice_massa_corporea.py. Note que ele pode ser reusado facilmente
def avaliar_imc(indice):
    if indice < 20:
        return 'Você está abaixo do peso'
    elif indice < 24.9:
        return 'Você está com o peso normal'
    elif indice < 29.9:
        return 'Você está com sobrepeso'
    else:
        return 'Você está com obesidade'


def imc(peso, altura):
    return peso / (altura ** 2)
```

ONDE ESTUDAR ONLINE

- www.codecademy.com/pt
- www.sololearn.com/Course/Python
- pythontutor.com
- www.pycursos.com/python-para-zumbis
- coursera.org
- www.urionlinejudge.com.br/judge/pt/login

REFERÊNCIAS

- http://wiki.python.org.br/PrincipiosFuncionais
- Curso Python para Zumbis
- Curso “An Introduction to Interactive Programming in Python” - Coursera
- http://www.peachpit.com/articles/article.aspx?p=1312792&seqNum=6
- https://docs.python.org/release/2.3.5/whatsnew/section-slices.html
- http://www.bbc.co.uk/education/guides/zqh49j6/revision/3
- https://www.youtube.com/watch?v=SYioCdLPmfw
- https://pt.wikibooks.org/wiki/Python/Conceitos_b%C3%A1sicos/Tipos_e_operadores
- http://www.dcc.ufrj.br/~fabiom/mab225/02tipos.pdf
- http://pt.stackoverflow.com/questions/62844/como-se-insere-n%C3%BAmeros-complexosem-python
- www.arquivodecodigos.net/principal/dicas_truques_categoria2.php?linguagem=12&categoria1=1&categoria2=59

REFERÊNCIAS

- www.arquivodecodigos.net/principal/dicas_truques_categoria2.php?linguagem=12&categoria1=1&categoria2=51
- www.dotnetperls.com/lower-python
- https://pt.wikipedia.org/wiki/Algoritmo
- http://wiki.python.org.br/SoftwarePython
- http://wiki.python.org.br/EmpresasPython
- https://powerpython.wordpress.com/2012/03/16/programas-e-jogos-feitos-em-python/
- http://tutorial.djangogirls.org/pt/python_installation/index.html
- https://powerpython.wordpress.com/2012/03/19/aula-python-17-estrutura-de-decisao/
- https://under-linux.org/entry.php?b=1371
- http://aprenda-python.blogspot.com.br/2009/10/nova-formatacao-de-strings.html
- https://docs.python.org/3/library/string.html#formatspec
- https://www.python.org/dev/peps/pep-3101/
- https://novatec.com.br/livros/automatize-tarefas-macantes-com-python/

MÃO NA MASSA

Procurem trabalhar em grupo e
trocar informações.

Tendo dúvidas, estamos à disposição

PyLadiesSP

PyLadiesSãoPaulo

PyLadiesSP

@PyLadiesSP

Meetup PyLadiesSP

saopaulo@pyladies.com

Mulheres que amam programar e ensinar Python