pyladies São Paulo

Bem-vindxs ao curso intermediário I de Python para mulheres

Apoio:

Prêmio Mulheres TechemSampa

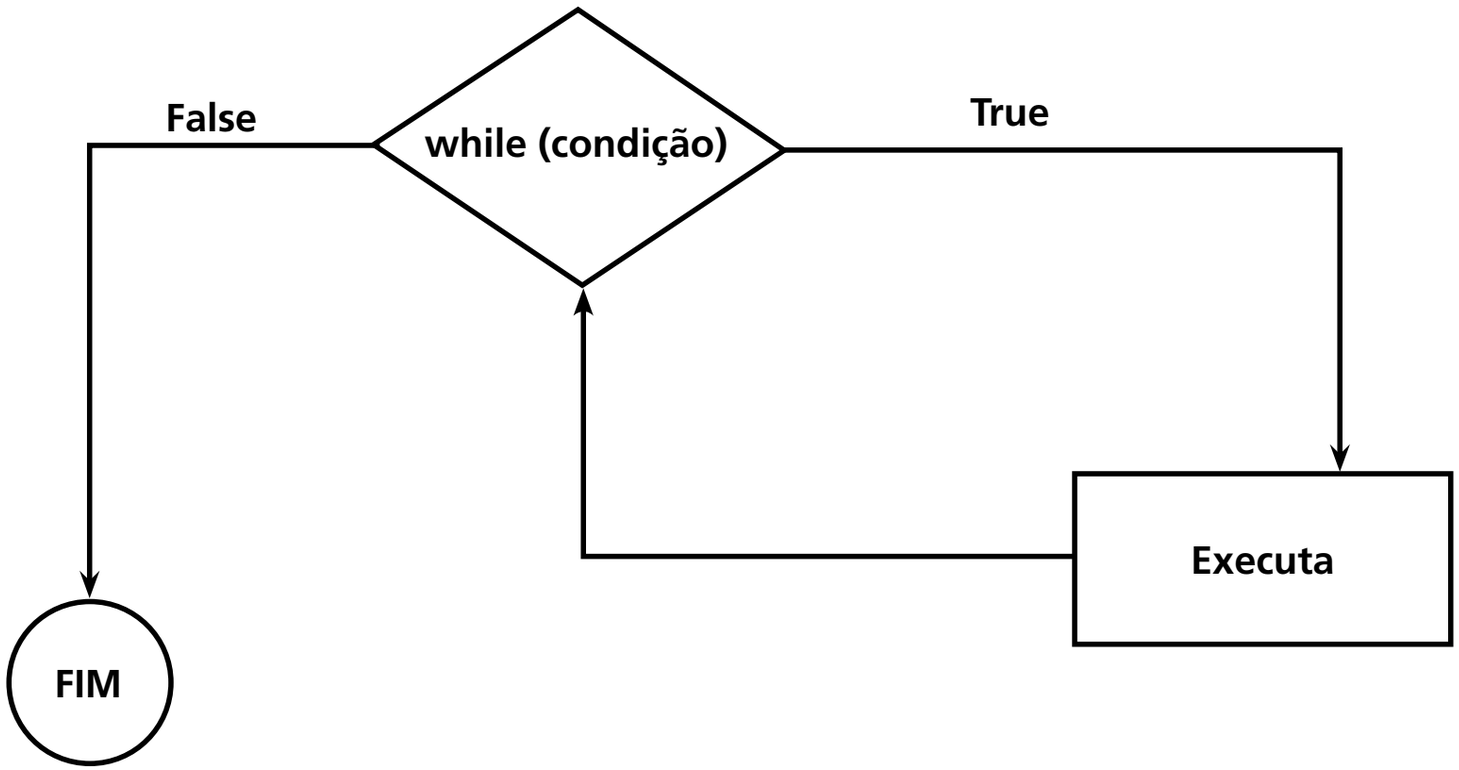
Google for Entrepreneurs

Rede MULHER EMPREENDEDORA

TechSampa

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: soma de 5 números dados pelo usuário.

```
i = 1
soma = 0
while i <= 5:
    nums = int(input('entre com um número: '))
    soma = soma + nums
    i = i + 1
print(soma)
```

acumulador: soma um valor diferente a cada vez que passa por esta linha

contador: soma um valor fixo

WHILE (ENQUANTO)

- **Exemplo 2:**

  Pense na tabuada. Preciso escolher uma delas (x) e multiplicar por todos os números de 1 a 10 (no caso, começa com n = 1).

  Para isso vou iterar o valor de n: atribuo um valor para n e obtenho o valor de x multiplicado por n. Depois incremento o valor de n em mais um inteiro e novamente calculo.

  O código fica:

```
x = int(input('Qual tabuada você quer calcular?'))
print('Tabuada do %d' %x)
n = 1
while n <= 10:
    print('%d x %d = %d' %(x, n, x*n))
    n = n+1
```

WHILE (ENQUANTO)

pyladies São Paulo

AGORA É COM VOCÊ

Tomo lanche todas as noites na faculdade, de segunda a sexta. Faça um código, usando contadores e acumuladores, que calcule quanto gasto por semana.

WHILE (ENQUANTO)

RESPOSTA

```
dia = 1
total = 0
while dia <= 5:
    gasto = float(input('Gastei: R$ '))
    total = total + gasto
    dia = dia + 1
print('Gastei na semana R$ '  ,total)
```

WHILE (ENQUANTO): interrompendo a repetição

Às vezes quero interromper uma repetição no meio de um processo, dependendo do que o usuário digita ou outro motivo. Nestes casos, posso usar o **Break**.

- **Exemplo:** soma de números inteiros até ser digitado zero

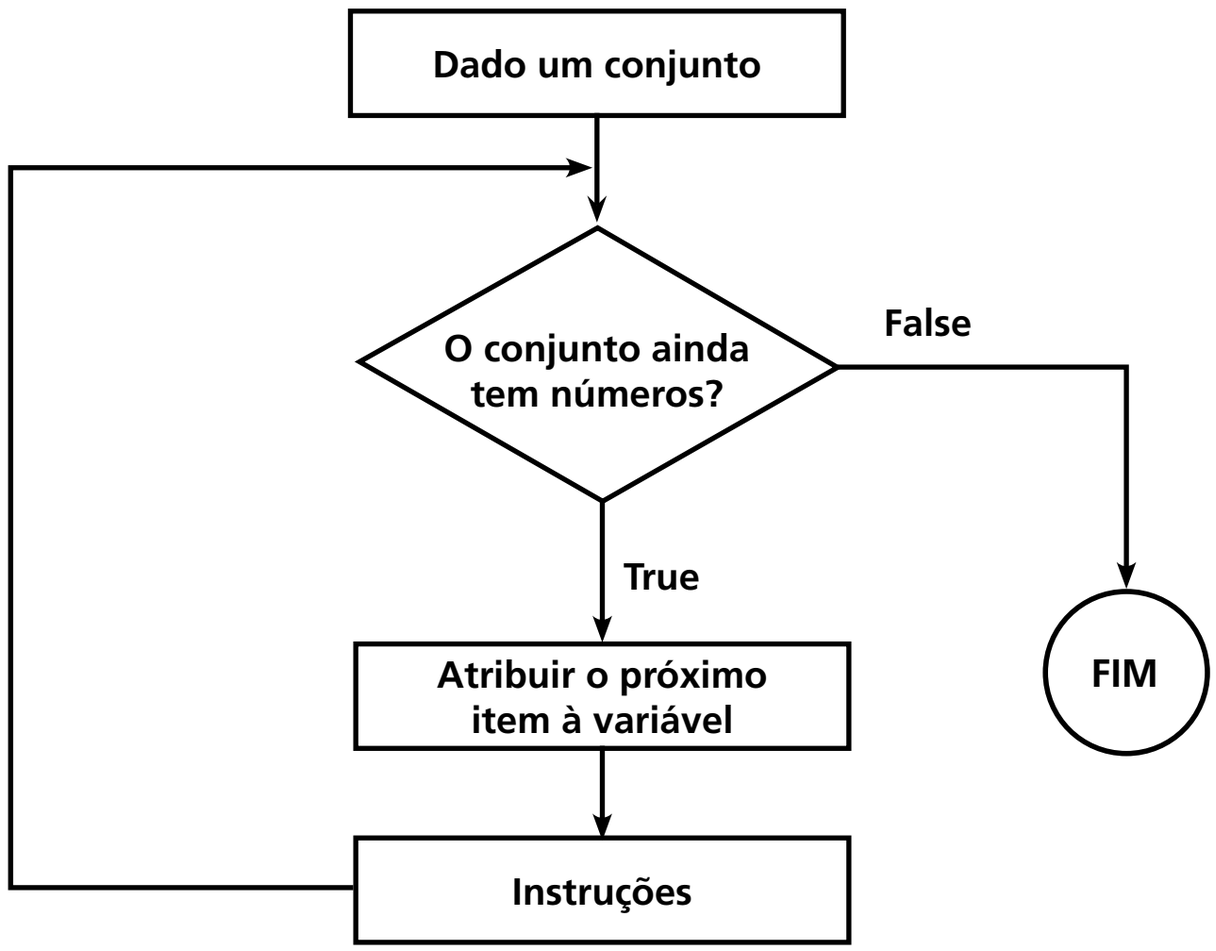
```
soma = 0
while True:
    x = int(input('Digite o número: '))
    if x == 0:
        break
    soma = soma +x
print('Soma: %d' %soma)
```

```
Digite o número: 3
Digite o número: 7
Digite o número: 13
Digite o número: 76
Digite o número: 93
Digite o número: 54
Digite o número: 10
Digite o número: 0
Soma: 256
>>>
```

PROGRAMANDO EM PYTHON

FOR (PARA)

O comando **for** itera sobre os itens de qualquer tipo de sequência (lista ou string), 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**:

```
a = ['Ariadyne', 'Beatriz', 'Caroline', 'Duda',
'Élida', 'Jussara', 'Veronica']

for i in a:
    if i.startswith('C') and i.endswith('e'):
        print(i)
```

FOR (PARA)

- **Exemplo 2**:

  Somar todos os números ímpares
  da Lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```
print('Soma de números ímpares')
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
somaImpar = 0
for x in lista:
    if (x % 2) != 0:
        somaImpar = somaImpar + x
        print('Somando o número ímpar: %d' %x)
        print('A soma dos ímpares é: %d' %somaImpar)
print('A soma de numeros ímpares da lista é %d'
%somaImpar)
```

FOR (PARA)

pyladies São Paulo

AGORA É COM VOCÊ

Dada a lista
linguagens = ['Java', 'JavaScript', 'PHP', 'C', 'Python'], verifique apenas linguagens que comecem com a letra P e imprima na tela com letras maiúsculas.

PROGRAMANDO EM PYTHON

FOR (PARA)

pyladies São Paulo

RESPOSTA

```
linguagens = ['Java', 'JavaScript', 'PHP', 'C', 'Python']

for i in linguagens:
    if i.startswith('P'):
        print(i.upper())
```

```
PHP
PYTHON
>>>
```

DIFERENÇAS ENTRE O WHILE E FOR

O **while** executa uma repetição até que uma determinada condição seja verdadeira.

O **for** executa uma repetição baseada em um número de vezes pré-determinado.

FUNÇÕES

Funções são sub-rotinas no código que servem para executar um procedimento muitas vezes, evitando que você tenha que reescrevê-lo mais de uma vez.

Uma funcionalidade importante é o fato que, caso precise realizar alguma alteração ou correção, ela vai ser feita nesta sub-rotina e não em diversas partes do código.

FUNÇÕES

- **Sintaxe**:

Quando a função não recebe parâmetros:

```
def <nome da função> ():
```

ou

Quando a função recebe parâmetros:

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

Voltemos ao exemplo da troca da lâmpada...

Quando chamamos o zelador para trazer a escada ou chamamos a Pat para fazer o pezinho para alcançar a lâmpada, estamos usando a ideia de função.

Toda vez que for necessário alcançar a lâmpada, precisaremos chamar o zelador ou a Pat.

Tanto o zelador, quanto a Pat têm a mesma função: nos elevar até a lâmpada. Só são maneiras diferentes de fazer a mesma coisa. Da mesma forma, podemos escrever uma função de maneiras diferentes, depende do estilo do programador, o modo como ele pensa.

O comando **input()**, usado nos exemplos, é na verdade uma função nativa no Python.

Ela solicita uma informação do usuário e nos retornar o valor informado.

Agora vamos criar a nossa própria função :)

FUNÇÕES

- **Exemplo**:

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

chamada da Função soma

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

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

usando como parâmetro strings

FUNÇÕES

- **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))
```

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

pyladies São Paulo

AGORA É COM VOCÊ

Faça uma função para pedir ao usuário dois números e calcular a divisão entre eles.

Informe:
- o valor exato da divisão
- o valor inteiro
- o resto

FUNÇÕES

RESPOSTA

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

    div = n1 / n2
    print('Total: %f' %div)

    div = n1 / n2
    print('Inteiro: %d' %div)

    div = n1 % n2
    print('Resto: %f' %div)

Divide()
```

```
>>>
Informe o primeiro número: 53
Informe o segundo número: 16
Total: 3.312500
Inteiro: 3
Resto: 5.000000
>>>
```

DICIONÁRIOS

Um **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 pode ser qualquer tipo de dado, não necessariamente numérico).

- **Sintaxe**:

```
variavel = {}  para dicionário vazio

variavel = {'chave 1': <valor 1>, 'chave 2': <valor 2>,
…, 'chave n': <valor n>}
```

DICIONÁRIOS

- **Exemplo:**

```
>>> cat = {'cor': 'branca', 'idade': 9, 'raça': 'SRD'}
>>> cat['cor']
>>> 'branca'
```

- **Para mudar um valor**: basta redefinir o valor associado à chave

  - **Sintaxe:**

```
<variavel>['chave'] = 'novo valor'
```

  - **Exemplo:**

```
>>> cat['raça'] = 'russo branco'
>>> cat
{'idade': 9, 'cor': 'branca', 'raça': 'russo branco'}
```

DICIONÁRIOS

- **Removendo itens em um dicionário usando del**

  - **Sintaxe:**

```
del <variavel>['chave']
```

  - **Exemplo:**

```
>>> del cat['raça']
>>> cat
{'idade': 9, 'cor': 'branca', 'sexo': 'fêmea'}
```

DICIONÁRIOS

pyladies São Paulo

AGORA É COM VOCÊ

Crie um dicionário para traduzir a palavra gato para as seguintes línguas: inglês, espanhol, francês, alemão e italiano (use o tradutor do Google :p).

Acrescente em um segundo momento amor em finlandês. Em um terceiro momento delete o par chave:valor em italiano.

(hahahha... este não tem resposta :p)

http://www3.ifrn.edu.br/~jurandy/fdp/doc/aprenda-python/capitulo_10.html
http://www.tutorialspoint.com/python/python_dictionary.htm

DICIONÁRIOS

```
>>> cat['comprimento']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'comprimento'
```

Para evitar erro ao acessar o valor de uma chave inexistente:
usar a operação `get()` do dicionário:

- **Sintaxe:**

  `<dicionario>.get(<chave>, [<valor pré-definido>])`

  sendo que `<valor pré-definido>` é opcional

- **Exemplo:**

```
>>> cat = {'idade': 9, 'cor': 'branca', 'sexo': 'fêmea'}
>>> cat.get('nome')
>>> cat.get('nome', 'Gatinho sem nome')
'Gatinho sem nome'
```

pyladies São Paulo

AGORA É COM VOCÊ

Obtenha o valor da chave 'ano de nascimento', use o valor 'desconhecido' como valor pré-definido, para os seguintes dicionários:

```
cat = {'idade': 9, 'cor': 'branca',
'sexo': 'fêmea'}

gatinho = {'idade': 1, 'cor': 'branca',
'sexo': 'fêmea', 'ano de nascimento':
'2015'}
```

DICIONÁRIOS

pyladies São Paulo

RESPOSTA

```
>>> cat.get('ano de nascimento', 'desconhecido')
'desconhecido'
>>> gatinho.get('ano de nascimento', 'desconhecido')
'2015'
```

DICIONÁRIOS

Para retornar os pares chave-valor: use a operação `items()` do dicionário.

- **Sintaxe:**

```
<dicionario>.items()
```

- **Exemplo:**

```
>>> cat = {'nome': 'Filoca', 'mãe': True, 'filhotes':
9, 'idade': 4, 'raça': 'indefinida', 'cor': ['preta',
'branca'], 'localização': (15,20)}

>>> cat.items()

dict_items([('mãe', True), ('idade', 4), ('nome',
'Filoca'), ('cor', ['preta', 'branca']), ('localização',
(15, 20)), ('filhotes', 9), ('raça', 'indefinida')])
```

DICIONÁRIOS

Para retornar as chaves: use a operação `keys()` do dicionário.

- **Sintaxe:**

```
<dicionario>.keys()
```

- **Exemplo:**

```
>>> cat = {'nome': 'Filoca', 'mãe': True, 'filhotes':
9, 'idade': 4, 'raça': 'indefinida', 'cor': ['preta',
'branca'], 'localização': (15,20)}
>>> cat.keys()
dict_keys(['localização', 'idade', 'nome', 'filhotes',
'raça', 'cor', 'mãe'])
```

DICIONÁRIOS

Para retornar os valores: use a operação `values()` do dicionário.

- **Sintaxe:**

```
<dicionario>.values()
```

- **Exemplo:**

```
>>> cat = {'nome': 'Filoca', 'mãe': True, 'filhotes':
9, 'idade': 4, 'raça': 'indefinida', 'cor': ['preta',
'branca'], 'localização': (15,20)}

>>> cat.values()

dict_values([(15, 20), 4, 'Filoca', 9, 'indefinida',
['preta', 'branca'], True])
```

pyladies São Paulo

AGORA É COM VOCÊ

Pratique `items()`, `keys()` e `values()` com:

```
cat = {'idade': 9, 'cor':
'branca', 'sexo': 'fêmea'}
```

DICIONÁRIOS

RESPOSTA

```
>>> cat
{'idade': 9, 'cor': 'branca', 'sexo': 'fêmea'}
>>> cat.items()
dict_items([('idade', 9), ('cor', 'branca'), ('sexo',
'fêmea')])
>>> cat.keys()
dict_keys(['idade', 'cor', 'sexo'])
>>> cat.values()
dict_values([9, 'branca', 'fêmea'])
```

DICIONÁRIOS

Como o resultado de `keys()` e `values()` são iteráveis, ou seja, se comportam como listas, é possível usar com `for`:

- **Sintaxe:**

```
for <item> in <dicionario>.keys():
    print(<item>)
```

e

```
for <item> in <dicionario>.values():
    print(<item>)
```

DICIONÁRIOS

- **Exemplos:**

```
>>> cat = {'idade': 9, 'cor': 'branca', 'sexo': 'fêmea'}
>>> for chave in cat.keys():
...     print(chave)
cor
idade
sexo
>>> for valor in cat.values():
...     print(valor)
branca
9
fêmea
```

pyladies São Paulo

AGORA É COM VOCÊ

Pratique o `for` com `keys()` e `values()` do seguinte dicionário:

```
cat = {'nome': 'Filoca', 'mãe':
True, 'filhotes': 9, 'idade': 4,
'raça': 'indefinida', 'cor': ['preta',
'branca'], 'localização': (15,20)}
```

DICIONÁRIOS

RESPOSTA

```
>>> cat = {'nome': 'Filoca', 'mãe': True, 'filhotes':
9, 'idade': 4, 'raça': 'indefinida', 'cor': ['preta',
'branca'], 'localização': (15,20)}
>>> for chave in cat.keys():
...     print(chave)
mãe
cor
nome
raça
localização
idade
filhotes
```

continua --->

DICIONÁRIOS

RESPOSTA (cont.)

```
>>> for valor in cat.values():
...     print(valor)
True
['preta', 'branca']
Filoca
indefinida
(15, 20)
4
9
```

DICIONÁRIOS

Como o resultado de `items()` também é um iterável, ou seja, se comporta como lista, é possível usar com `for`:

- **Sintaxe:**

```
for <chave>, <valor> in <dicionario>.items():
    print(<chave>)
    print(<valor>)
```

DICIONÁRIOS

- **Exemplo:**

```
>>> cat = {'idade': 9, 'cor': 'branca', 'sexo': 'fêmea'}
>>> for chave, valor in cat.items():
...     print(chave)
...     print(valor)
idade
9
sexo
fêmea
cor
branca
```

pyladies São Paulo

AGORA É COM VOCÊ

Pratique o `for` com `items()` do seguinte dicionário:

```
cat = {'nome': 'Filoca', 'mãe':
True, 'filhotes': 9, 'idade': 4,
'raça': 'indefinida', 'cor': ['preta',
'branca'], 'localização': (15,20)}
```

DICIONÁRIOS

RESPOSTA

```
>>> cat = {'nome': 'Filoca', 'mãe': True, 'filhotes':
9, 'idade': 4, 'raça': 'indefinida', 'cor': ['preta',
'branca'], 'localização': (15,20)}
>>> for chave, valor in cat.items():
...     print(chave)
...     print(valor)
mãe
True
cor
['preta', 'branca']
nome
Filoca
raça
indefinida
localização
(15, 20)
idade
4
filhotes
9
```

Ao percorrer as chaves, os valores e os items de um dicionários podemos observar que não ficam na mesma ordem em que foram inseridos no dicionário.

O motivo disso é que os dicionários são tabelas de dispersão *(hash table)*. Seus dados são armazenados na memória de forma a obter os valores desejados rapidamente (otimizar a busca). Leia mais sobre tabela de dispersão para entender como funciona seu armazenamento (https://pt.wikipedia.org/wiki/Tabela_de_dispers%C3%A3o).

Isso quer dizer também que conforme novos itens são inseridos no dicionário, a ordem de apresentação dos itens também muda.

DICIONÁRIOS

Para ter o dicionário ordenado pelas chaves, use a função `sorted()`.

- **Sintaxe:**

```
sorted(<dicionario>.keys())
```

Da mesma forma, para valores e items:

```
sorted(<dicionario>.values())
sorted(<dicionario>.items())
```

- **Exemplo:**

```
for k in sorted(cat.keys()):
    print(k)
    print(cat.get(k))
```

DICIONÁRIOS

pyladies São Paulo

AGORA É COM VOCÊ

Use o dicionário:

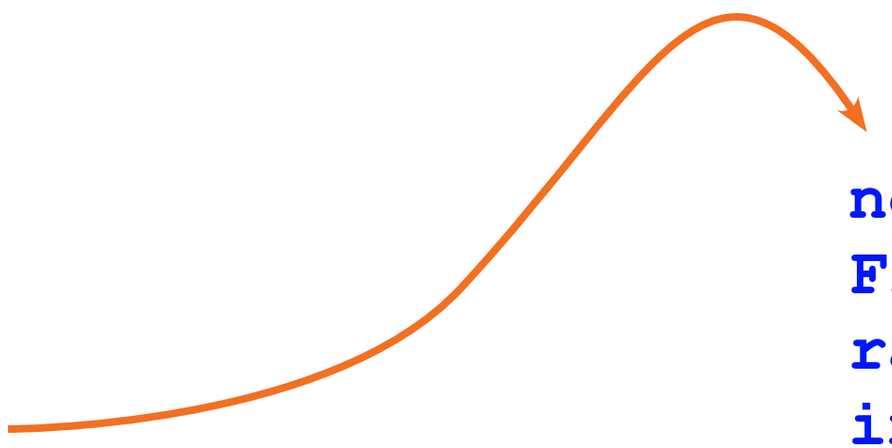
```
cat = {'nome': 'Filoca', 'mãe':
True, 'filhotes': 9, 'idade':
4, 'raça': 'indefinida', 'cor':
['preta', 'branca'], 'localização':
(15,20)}
```

e imprima as chaves e os valores do dicionário ordenado pelas chaves.

DICIONÁRIOS

RESPOSTA

```
>>> cat = {'nome': 'Filoca', 'mãe': True, 'filhotes':
9, 'idade': 4, 'raça': 'indefinida', 'cor': ['preta',
'branca'], 'localização': (15,20)}
>>> for k in sorted(cat.keys()):
...     print(k)
...     print(cat.get(k))
cor
['preta', 'branca']
filhotes
9
idade
4
localização
(15, 20)
mãe
True
```



```
nome
Filoca
raça
indefinida
```

DICIONÁRIOS

Para controlar a ordem dos itens em um dicionário, você pode utilizar `OrderedDict` do módulo `collections`. Ele preserva exatamente a ordem original de inserção de dados em uma iteração. Por exemplo:

```
from collections import OrderedDict
d = OrderedDict()
d['foo'] = 1
d['bar'] = 2
d['spam'] = 3
d['grok'] = 4

# Apresenta "foo 1", "bar 2", "spam 3", "grok 4"

>>> for key in d:
...     print(key, d[key])
```

DICIONÁRIOS

Um OrderedDict mantém internamente uma lista duplamente ligada que ordena as chaves de acordo com a ordem de inserção. Quando um novo item for inicialmente inserido, ele será colocado no final dessa lista. Novas atribuições subsequentes em uma chave existente não alterarão a ordem. Esteja ciente de que um OrderedDict tem um tamanho que corresponde a mais do dobro de um dicionário normal, por causa da lista ligada extra criada.

Python Cookbook - David Beazley (https://novatec.com.br/livros/python-cookbook/capitulo9788575223321.pdf)

DICIONÁRIOS

Vamos criar um arquivo chamado dicionarios.py e escrever nele três funções:

- mostra_chaves
- mostra_valores
- mostra_pares_chave_valor

Use o `for` para `items()`, `keys()` e `values()` de um dicionário.
O dicionário é o parâmetro das funções.

Para testar, vamos usar como valor do dicionário:

```
cat = {'nome': 'Filoca', 'mãe': True, 'filhotes': 9, 'idade':
4, 'raça': 'indefinida', 'cor': ['preta', 'branca'],
'localização': (15,20)}
```

DICIONÁRIOS


```
# no arquivo dicionarios.py
def mostra_chaves(d):
    for item in d.keys():
        print(item)

def mostra_valores(d):
    for item in d.values():
        print(item)

def mostra_chaves_valores(d):
    for k, v in d.items():
        print(k)
        print(v)
        print('---')
```

continua -->

DICIONÁRIOS

```
cat = {'nome': 'Filoca', 'mãe': True, 'filhotes': 9,
'idade': 4, 'raça': 'indefinida', 'cor': ['preta',
'branco'], 'localização': (15,20)}
```

```
mostra_chaves(cat)
mostra_valores(cat)
mostra_chaves_valores(cat)
```

DICIONÁRIOS

Tanto chave como valor não precisam ser sempre do mesmo tipo, nem ser somente um sequência de caracteres (string).

- **Exemplo:**

```
coordenadas = {(1,1):'cafeteria ', (300,990):  'pet shop ',
(2,4): { '1 andar ': None,  '2 andar ': [ 'restaurante ',
'bar ',  'banheiros '],  '3 andar ':  'estacionamento '},
(5, 100, 400000):  'topo da colina '}
```

Neste exemplo, as chaves são tuplas e os valores são str, dict, list.

AGORA É COM VOCÊ

Copie o arquivo dicionarios.py e renomeie para coordenadas.py.

Altere para colocar o uso das três funções anteriormente criadas com o dicionário coordenadas.

DICIONÁRIOS

pyladies São Paulo

RESPOSTA

```
# no arquivo coordenadas.py
def mostra_chaves(d):
    for item in d.keys():
        print(item)

def mostra_valores(d):
    for item in d.values():
        print(item)

def mostra_chaves_valores(d):
    for k, v in d.items():
        print(k)
        print(v)
        print('---')
```

continua --->

DICIONÁRIOS

RESPOSTA (cont.)

```
coordenadas = {(1,1):'cafeteria ', (300,990):  'pet
shop ', (2,4): { '1 andar ': None,  '2 andar ': [
'restaurante ',  'bar ',  'banheiros '],  '3 andar
':  'estacionamento '}, (5, 100, 400000):  'topo da
colina '}

mostra_chaves(coordenadas)
mostra_valores(coordenadas)
mostra_chaves_valores(coordenadas)
```

EXERCÍCIO EXTRA

Considere o dicionário:

```
coord_gatos = {'pompom': (1,1), 'Mingau': (4,0), 'Fifi':
(2,4), 'Fofona': (4,0)}
```

Em um arquivo chamado localiza_gatos.py, escreva uma função chamada achei_uns_gatos que percorra o dicionário e imprima o nome do gato que está em uma dada coordenada.

Teste a função com as coordenadas (4, 0) e (10,10)

Crie um arquivo com nome de gatos.py

DICIONÁRIOS

RESPOSTA

```
# no arquivo localiza_gatos.py

def achei_uns_gatos(coord_gatos, coord_do_local):
    for nome_do_gato, coord_do_gato in coord_gatos.items():
        if coord_do_local == coord_do_gato:
            print(nome_do_gato)

coord_gatos = {'pompom': (1,1), 'Mingau': (4,0), 'Fifi':
(2,4), 'Fofona': (4,0)}

achei_uns_gatos(coord_gatos, (4, 0))
achei_uns_gatos(coord_gatos, (10, 10))
```

MÓDULOS E PACOTES

Conforme vamos escrevendo as instruções, o programa vai ficando muito longo e difícil de dar manutenção. Sendo assim, o melhor é dividí-los em vários arquivos, que são os **módulos**.

Os módulos são arquivos com extensão **.py** que contém definições e declarações (funções, variáveis, etc).

As definições em um módulo podem ser importadas em outros módulos, ou seja, reusar um código já pronto, sem fazer o "copia" e "cola", sem duplicar.

Um módulo pode usar funções e/ou declarações definidas em outro módulo.

Para isso, usam-se as seguintes sintaxes:

```
import <nome_modulo>

from <nome_modulo> import <nome_funcao_ou_
declaracao>, <nome_funcao_ou_declaracao>,
...
```

MÓDULOS

- calculadora_simples.py: contém 4 operações
- calculadora_financeira.py: contém as 4 operações + operações financeiras

```python
# em calculadora_simples.py
def soma(a, b):
    return a + b

def multiplicacao(a, b):
    return a * b
```

```python
# em calculadora_financeira.py
from calculadora_simples import
soma, divisao, subtracao,
multiplicacao
def valor_futuro(...):
…

def juros():
    multiplicacao(...)
...
```

```python
# em calculadora_financeira2.py
import calculadora_simples

def valor_futuro(...):
    …

def juros():
    return calculadora_simples.
multiplicacao(...)
```

PROGRAMANDO EM PYTHON

MÓDULOS


```
# no arquivo dicionarios.py
```

```
def mostra_chaves(d):
    for item in d.keys():
        print(item)

def mostra_valores(d):
    for item in d.values():
        print(item)

def mostra_chaves_valores(d):
    for k, v in d.items():
        print(k)
        print(v)
        print('---')
```

são idênticos

```
cat = {'nome': 'Filoca', 'mãe':
True, 'filhotes': 9, 'idade':
4, 'raça': 'indefinida', 'cor':
['preta', 'branco'], 'localização':
(15,20)}
mostra_chaves(cat)
mostra_valores(cat)
mostra_chaves_valores(cat)
```

```
# no arquivo coordenadas.py
coordenadas = {(1,1):'cafeteria',
(300,990): 'pet shop', (2,4):
{'1 andar': None, '2 andar':
['restaurante', 'bar', 'banheiros'],
'3 andar': 'estacionamento'}, (5,
100, 400000): 'topo da colina'}
mostra_chaves(coordenadas)
mostra_valores(coordenadas)
mostra_chaves_valores(coordenadas)
```

PROGRAMANDO EM PYTHON

MÓDULOS

```
# no arquivo dicionarios.py

def mostra_chaves(d):
    for item in d.keys():
        print(item)

def mostra_valores(d):
    for item in d.values():
        print(item)

def mostra_chaves_valores(d):
    for k, v in d.items():
        print(k)
        print(v)
        print('---')

cat = {'nome': 'Filoca', 'mãe':
True, 'filhotes': 9, 'idade':
4, 'raça': 'indefinida', 'cor':
['preta', 'branco'], 'localização':
(15,20)}
mostra_chaves(cat)
mostra_valores(cat)
mostra_chaves_valores(cat)
```

```
# arquivo gatos.py

import dicionarios

cat = {'nome': 'Filoca', 'mãe':
True, 'filhotes': 9, 'idade': 4,
'raça': 'indefinida', 'cor': ['preta',
'branco'], 'localização': (15,20)}
dicionarios.mostra_chaves(cat)
dicionarios.mostra_valores(cat)
dicionarios.mostra_chaves_valores(cat)
```

MÓDULOS

```
# no arquivo dicionarios.py

def mostra_chaves(d):
    for item in d.keys():
        print(item)

def mostra_valores(d):
    for item in d.values():
        print(item)

def mostra_chaves_valores(d):
    for k, v in d.items():
        print(k)
        print(v)
        print('---')

cat = {'nome': 'Filoca', 'mãe':
True, 'filhotes': 9, 'idade':
4, 'raça': 'indefinida', 'cor':
['preta', 'branco'], 'localização':
(15,20)}
mostra_chaves(cat)
mostra_valores(cat)
mostra_chaves_valores(cat)
```

```
# no arquivo coordenadas.py

from dicionarios import mostra_
chaves, mostra_valores, mostra_
chaves_valores

coordenadas = {(1,1):'cafeteria',
(300,990): 'pet shop', (2,4):
{'1 andar': None, '2 andar':
['restaurante', 'bar', 'banheiros'],
'3 andar': 'estacionamento'}, (5,
100, 400000): 'topo da colina'}

mostra_chaves(coordenadas)
mostra_valores(coordenadas)
mostra_chaves_valores(coordenadas)
```

PACOTES

Pacotes são um conjunto de módulos organizados hierarquicamente.

Dê nomes significativos aos pacotes e módulos de forma a identificar o que cada um faz sem precisar abri-los

https://docs.python.org/3/tutorial/modules.html#packages

PACOTES

```
movimentos/
   __init__.py
   seres_vivos/
      __init__.py
      voam/
         __init__.py
         insetos_que_voam.py
         aves_que_voam.py
      terrestres/
         bipedes.py
         insetos_que_nao_voam.py
         quadrupedes.py
         rastejar.py
   aquaticos/
            seres_que_nadam.py
objetos_inanimados/
   __init__.py
   roda.py
   bola.py
   cubo.py
objetos_motorizados/
    carro.py
    aviao.py
```

Pacote do nível mais alto

Inicializa o pacote movimentos

subpacote para movimentos de seres vivos

```
# estando em objetos_inanimados
from . import bola
from .. import seres_vivos
from ..objetos_motorizados import aviao
```

subpacote para movimentos de objetos inanimados

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

1. da biblioteca padrão do Python, ou seja, já está instalado 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.

MÓDULOS E PACOTES

A **PEP8** (*pep eight*) (www.python.org/dev/peps/pep-0008/) – um guia de estilo de programação para Python – recomenda a seguinte ordem de importação:

1. Bibliotecas padrão
2. Bibliotecas de terceiros
3. Pacotes e módulos locais

Cada grupo separado por uma linha

```
import os

from bs4 import BeautifulSoup
from PIL import Image

import dicionarios
import gatos
import coordenadas
```

ARQUIVOS

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

```
import os
```

Para saber onde está, em que caminho:

- **Sintaxe:** `os.getcwd()`
- **Exemplo:**

```
>>> os.getcwd()
'C:\\Users\\Roberta'
```

Obs.: No Windows verá as barras \\.
No Linux será /.

Para funcionar tanto em Windows como em Linux, use /

ARQUIVOS

Para listar os arquivos de uma pasta:

- **Sintaxe:** `os.listdir(<caminho da pasta>)`

- **Exemplo:**

```
>>> os.listdir('/users/')
['Administrator', 'All Users', 'Default', 'Default
User', 'Default.migrated', 'DefaultAppPool', 'desktop.
ini', 'Public', 'Roberta']
```

Para indicar o próprio local onde está, use:
os.listdir('.')

Para indicar uma pasta logo abaixo onde está:
os.listdir('./roberta')

ARQUIVOS

Para criar pastas:

- **Sintaxe:**

```
os.makedirs(<caminho completo com várias pastas>)
```

- **Exemplo:**

```
>>> os.makedirs('/users/roberta/pyladies/2016/curso-iniciante')
```

ARQUIVOS

Para saber se um arquivo existe:

- **Sintaxe:**

```
os.path.isfile(<caminho do arquivo>)
```

- **Exemplo:**

```
>>> os.path.isfile('/users/roberta/pyladies/2016/curso-iniciante')

False
```

ARQUIVOS

Para saber se uma pasta existe:

- **Sintaxe:**

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

- **Exemplo:**

```
>>> os.path.isdir('/users/roberta/pyladies/2016/curso-iniciante')

True
```

ARQUIVOS

Para obter o caminho de uma pasta ou arquivo:

- **Sintaxe:**

```
os.path.dirname(<caminho do arquivo>)
```

- **Exemplo:**

```
>>> os.path.dirname('/users/roberta/pyladies/2016/curso-iniciante')

/users/roberta/pyladies/2016
```

ARQUIVOS

Para obter o nome de uma pasta ou arquivo:

- **Sintaxe:**

```
os.path.basename(<caminho do arquivo>)
```

- **Exemplo:**

```
>>> os.path.basename('/users/roberta/pyladies/2016/
curso-iniciante')

curso-iniciante
```

ARQUIVOS

```
>>> import os
>>> os.getcwd()
c:\\users\roberta
>>> os.path.exists('./pyladies/curso/intermediario1')
False
>>> os.makedirs('./pyladies/curso/intermediario1')
>>> os.path.isdir('./pyladies/curso/intermediario1')
True
>>> os.path.isfile('./pyladies')
False
>>> os.path.dirname('./pyladies/curso/intermediario')
'./pyladies/curso'
>>> os.path.basename('./pyladies/curso/intermediario')
'intermediario'
```

ARQUIVOS

Crie em pyladies/curso/intermediario1, um arquivo com o nome zen.txt e com o conteúdo do quadro ao lado, isto é, copiando e colando em um editor de texto.

Repare bem onde vai salvá-lo, pois usará no próximo exercício.

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one – and preferably only one – obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!

ARQUIVOS: LER E ESCREVER

Todo programa trabalha com entrada e saída de dados.

Arquivos são uma das formas de fornecer dados de entrada para os programas e de guardar dados de saída dos programas.

- **Exemplos:**

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

  Saída: programa guarda um relatório em arquivo.

ARQUIVOS: LEITURA

- **Sintaxe:**

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

```
with open(<caminho do arquivo>, 'r') as <arquivo>:
    for linha in <arquivo>:
        print(linha)
```

ARQUIVOS: LEITURA

- **Exemplo:**

```
with open('zen.txt', 'r') as f:
    conteudo = f.read()
    print(conteudo)

with open('zen.txt', 'r') as f:
    for linha in f:
        print(linha)
```

ARQUIVOS: ESCRITA

- **Sintaxe:**

```
with open(<caminho do arquivo>, 'w') as <arquivo>:
    <arquivo>.write('blbababaabb\n')
    <arquivo>.write('123\n445')
```

- **Exemplo:**

```
with open('zen2.txt', 'w') as f:
    f.write('super zen')
```

ARQUIVOS: ESCRITA

Use `'a'` para escrever ao final do arquivo, ou seja, sem apagar o que já existe.

- **Sintaxe:**

```
with open(<caminho do arquivo>, 'a') as <arquivo>:
    <arquivo>.write('blbababaabb\n')
    <arquivo>.write('123\n445')
```

- **Exemplo:**

```
with open('zen2.txt', 'a') as f:
    f.write('mais um super zen')
```

ARQUIVOS

Crie um módulo chamado arquivos.py. Nele, crie uma função chamada escreve_novo_zen que leia zen.txt, escreva em zen2.txt somente as linhas que contiverem uma dada palavra. Neste módulo, execute a função e, em seguida, leia e mostre (print) o conteúdo de zen2.txt.

- **Dicas:**
  - -Dada palavra é um parâmetro da função
  - -Ler um arquivo linha a linha
  - -Testar se na linha existe a dada palavra
  - -Escrever no arquivo se a condição acima for verdadeira
- **Atenção:** como estão usando DOIS arquivos ao mesmo tempo, use variáveis diferentes para distinguir os arquivos.

ARQUIVOS

```
def escreve_novo_zen(palavra):
    with open('zen.txt', 'r') as f:
        for linha in f:
            if palavra in linha:
                with open('zen2.txt', 'w') as f2:
                    f2.write(linha)

escreve_novo_zen('is')
with open('zen2.txt', 'r') as f:
    print(f.read())
```

ARQUIVOS

Edite o módulo chamado arquivos.py. Nele, crie outra função chamada continua_escrevendo_novo_zen que leia zen.txt e que continue escrevendo sem apagar o que já estava em zen2.txt, somente as linhas que NÃO contiverem uma dada palavra. Neste módulo, execute a função e, em seguida, leia e mostre (print) o conteúdo de zen2.txt.

- **Dicas:**

  -Dada palavra é um parâmetro da função

  -Ler um arquivo linha a linha

  -Testar se na linha **NÃO** existe a dada palavra

  -Escrever no arquivo ('a') se a condição acima for verdadeira

- **Atenção:** como estão usando DOIS arquivos ao mesmo tempo, use variáveis diferentes para distinguir os arquivos.

ARQUIVOS

```
def escreve_novo_zen(palavra):
    with open('zen.txt', 'r') as f:
        for linha in f:
            if palavra in linha:
                with open('zen2.txt', 'w') as f2:
                    f2.write(linha)

def continua_escrevendo_novo_zen(palavra):
    with open('zen.txt', 'r') as f:
        for linha in f:
            if not palavra in linha:
                with open('zen2.txt', 'a') as f2:
                    f2.write(linha)

escreve_novo_zen('is')
with open('zen2.txt','r') as f:
    print(f.read())
continua_escrevendo_novo_zen('better')
with open('zen2.txt','r') as f:
    print(f.read())
```

pyladies São Paulo

Extra: como saber a quantidade de cada palavra que ocorre em um texto?

```
palavras = text.split()
# palavras é uma lista
from collections import Counter
contador_de_palavras = Counter(palavras)
print(contador_de_palavras)
for palavra, qtd in contador_de_palavras.items():
    print(palavra)
    print(qtd)
```

FIM