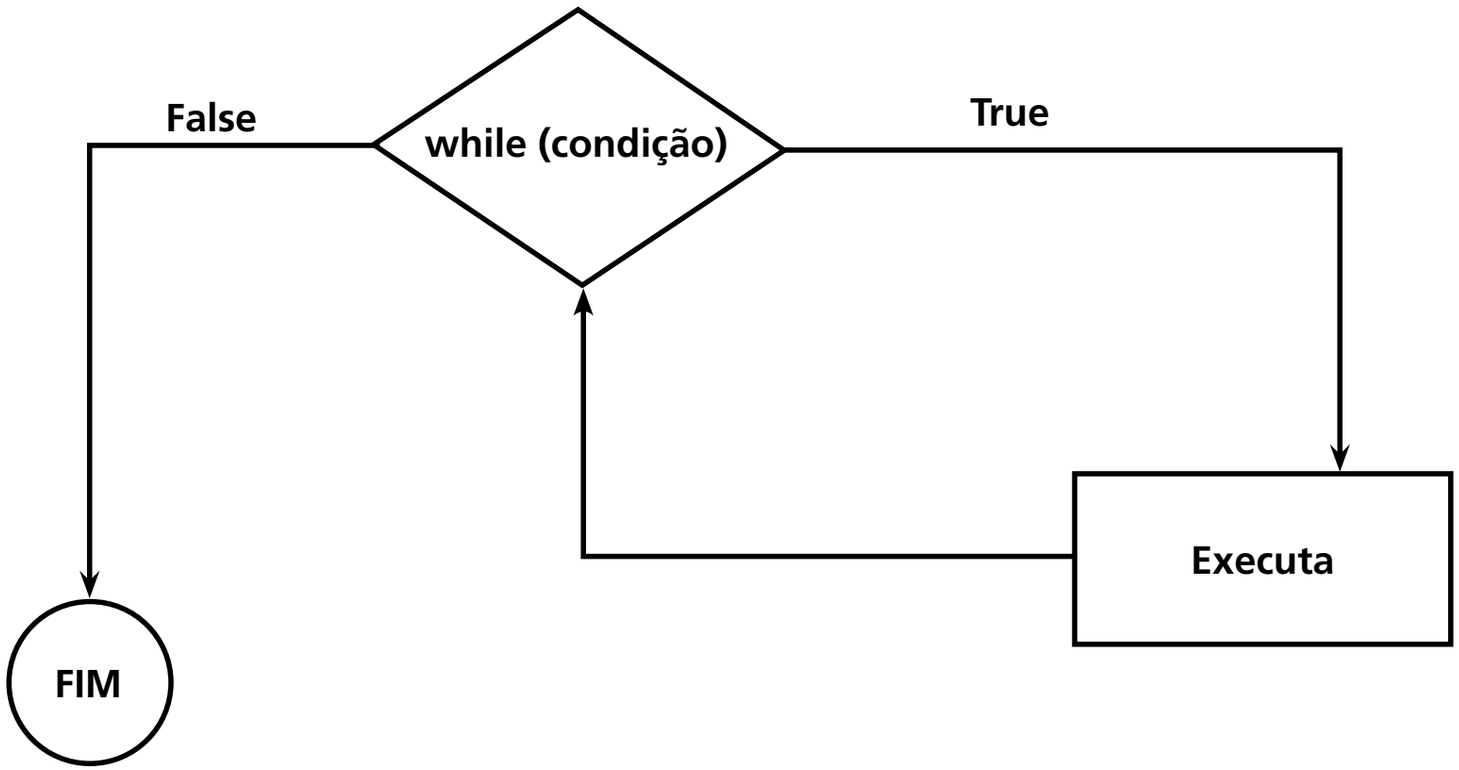
Workshop intermediário I de Python

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: soma de 3 números dados pelo usuário.

```
contador = 1
soma = 0
while contador <= 3:
    nums = int(input('entre com um número: '))
    soma = soma + nums
    contador = contador + 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:**

  Criar um código onde o usuário entre com um número e obtenha a tabuada do mesmo.

```
numero = int(input('Qual tabuada você quer calcular? '))
print('Tabuada do {}' .format(numero))
multiplicado_por = 1
while multiplicado_por <= 10:
    print('{} x {} = {}' .format(numero, multiplicado_por,
numero*multiplicado_por))
    multiplicado_por = multiplicado_por+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

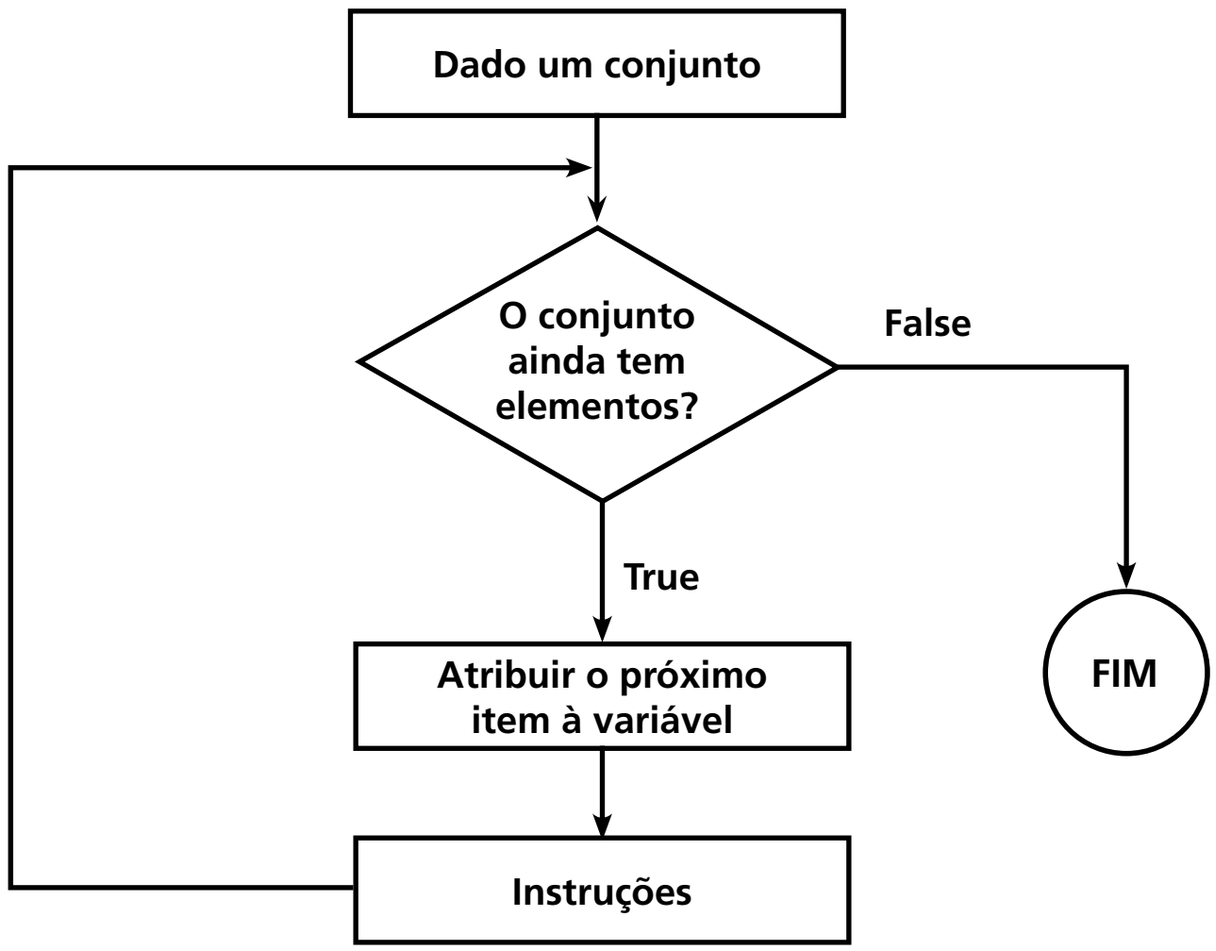
```
soma = 0
while True:
    numero = int(input('Digite o número: '))
    if numero == 0:
        break
    soma = soma + numero
print('Soma: {}' .format(soma))
```

```
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.



PROGRAMANDO EM PYTHON

FOR (PARA)

- **Sintaxe**:

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

- **Exemplo 1**:

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

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

FOR (PARA)

AGORA É COM VOCÊ

Dada a Lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], percorra essa lista, some apenas os números ímpares e imprima o resultado no final.

FOR (PARA)

pyladies São Paulo

RESPOSTA

```
print('Soma de números ímpares')
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

somaImpar = 0

for item in lista:
    if (item % 2) != 0:
        somaImpar = somaImpar + item

print('A soma de números ímpares da lista é {}'.format
(somaImpar))
```

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)
```

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):  chamada da função soma
    return a + b
```

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

print(soma('PyLadies', ' São Paulo'))  usando strings como parâmetro
>>>
PyLadies São Paulo
```

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

pyladies São Paulo

RESPOSTA

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

    div = n1 / n2
    print('Total: {}' .format(div))

    div = n1 // n2
    print('Inteiro: {}' .format(div))

    div = n1 % n2
    print('Resto: {}' .format(div))

Divide()
```

```
>>>
Informe o primeiro número: 53
Informe o segundo número: 16
Total: 3.3125
Inteiro: 3
Resto: 5.0
>>>
```

DICIONÁRIOS

Um **dicionário** é uma coleção não ordenada de pares chave-valor.

- **Sintaxe**:

```
variável = {} para dicionário vazio
variável = {'chave 1': <valor 1>, 'chave 2': <valor 2>, …,
'chave n': <valor n>}
```

- **Exemplo:**

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

DICIONÁRIOS

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

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

  - **Exemplo:**

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

- **Para adicionar um item**: basta definir o valor associado à chave
  - **Exemplo:**

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

DICIONÁRIOS

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

```
del <variável>['chave']
```

  - **Exemplo:**

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

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'
```

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'])
```

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

- **Sintaxe:**

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

- **Exemplo:**

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

DICIONÁRIOS

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

- **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
```

DICIONÁRIOS

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

- **Exemplo:**

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

DICIONÁRIOS

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

- **Exemplo:**

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

Dica: a ordenação não funciona se houver tipos diferentes na chave ou valor

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
>>> cat = OrderedDict()
>>> cat['nome'] = 'Gatinho'
>>> cat['cor'] = 'cinza'
>>> cat['idade'] = 2
>>> cat
OrderedDict([('nome', 'Gatinho'), ('cor', 'cinza'),
('idade', 2)])
```

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

```
# em calculadora_simples.py

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

def subtracao(a, b):
    return a - b

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

def divisao(a, b):
    return a / b
```

```
# em calculadora_financeira.py

from calculadora_simples import soma,
divisao, subtracao, multiplicacao
def valor_futuro(...):
…

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

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

PROGRAMANDO EM PYTHON

PACOTES

```
#cachorro.py
def latir():
    print('o cachorro faz au au')

#gato.py
def miar()
    print('o gato faz miau miau')
```

```
#vaca.py
def mugir():
    print('a vaca faz muuuu')
```

```
#__init__.py
from cachorro import latir
from gato import miar
from vaca import mugir

#outro.py ou animais.py?
import Animais

Animais.latir()
Animais.miar()
Animais.mugir()
```

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'
```

PROGRAMANDO EM PYTHON

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)
```

pyladies São Paulo

FIM