



Bem-vindxs ao curso iniciante  
de Python para mulheres

Apoio:



Sob licença CC-BY-NC-ND

## O QUE É PYLADIES?

PyLadies é um grupo internacional de mentoria com foco em ajudar mais mulheres a tornarem-se participantes ativas e líderes da comunidade Python.



#souPyLadiesSP

Um algoritmo é uma sequência finita de instruções.

Por exemplo:

- ir da sala até o quarto
- trocar a lâmpada da sala

## O QUE É ALGORITMO



A lógica expressa pelo algoritmo é a base para tudo em qualquer linguagem de programação.

É a **essência**.

Essa sequência pode ser executada por um humano ou um computador. Logo, programação é a arte de fazer com que o computador execute uma **sequência de instruções definidas**.

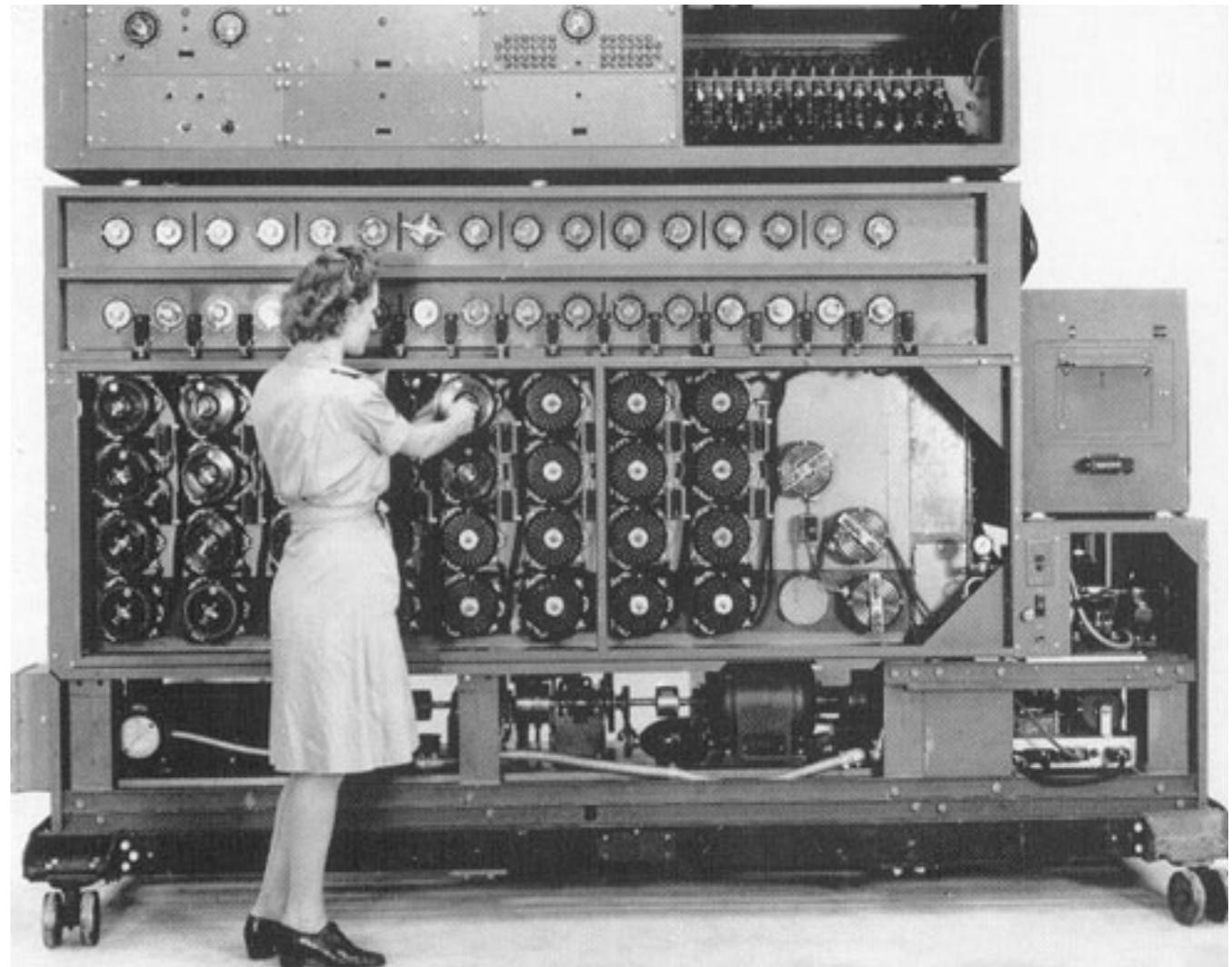
# O QUE É ALGORITMO



Máquina de Cálculos do Babbage - Ada Lovelace



Máquina de Turing



## O QUE É ALGORITMO



Precisamos **pensar pelo computador** e dizer os passos que queremos que ele dê para fazer o que queremos que ele faça.

Cada pessoa pode pensar em uma maneira diferente de instruir o computador a realizar algo, ou seja, cada pessoa irá pensar em um programa de um jeito. Mas existem **meios mais fáceis que outros**.

## O QUE É E ONDE SE USA PYTHON



Python é uma linguagem de programação com código aberto, de alto nível, tipicamente usada para aplicações web ou linguagens de scripts para administração de sistemas.



# O QUE É E ONDE SE USA PYTHON



Foi criada em 1989  
por Guido Van Rossum



O nome Python foi inspirado no seriado britânico Monty Python



## O QUE É E ONDE SE USA PYTHON



- ▶ No Brasil:

**magazineluiza**

**LOCAWEB**

**Embratel**

**globo.com**

**SERPRO**  
Serviço Federal de Processamento de Dados

# O QUE É E ONDE SE USA PYTHON



► Em educação:



UNIVERSIDADE  
FEDERAL RURAL  
DE PERNAMBUCO



IME - Instituto de  
Matemática e Estatística



UFSC



ESCOLA  
POLITÉCNICA  
DA USP



Prof. Jessen Vidal



Massachusetts  
Institute of  
Technology

# O QUE É E ONDE SE USA PYTHON

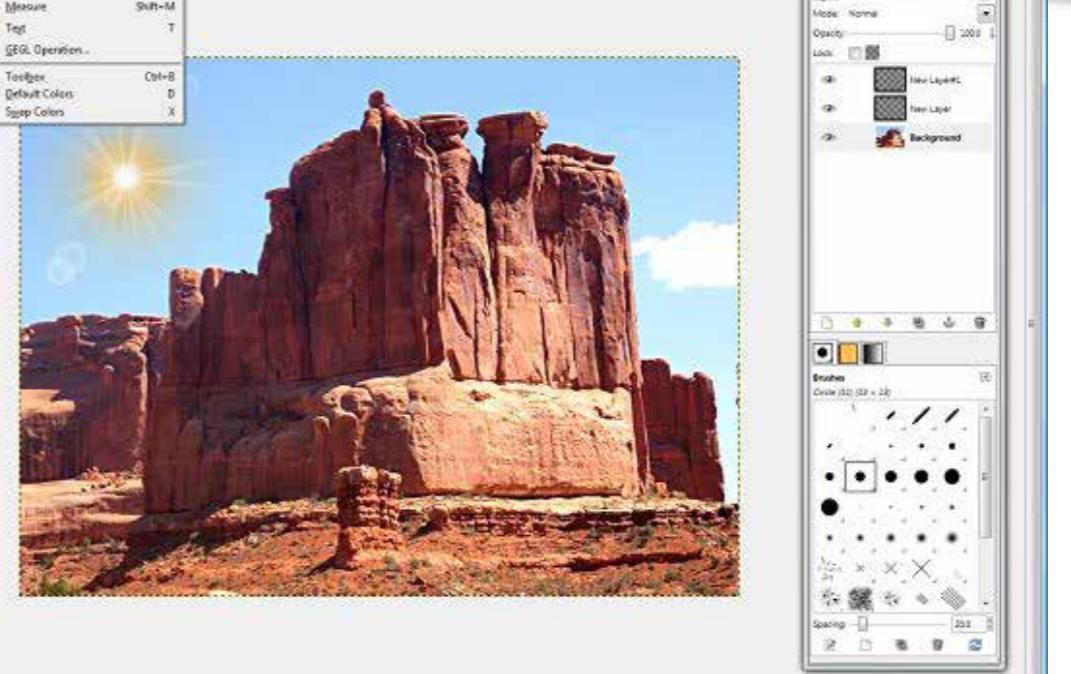
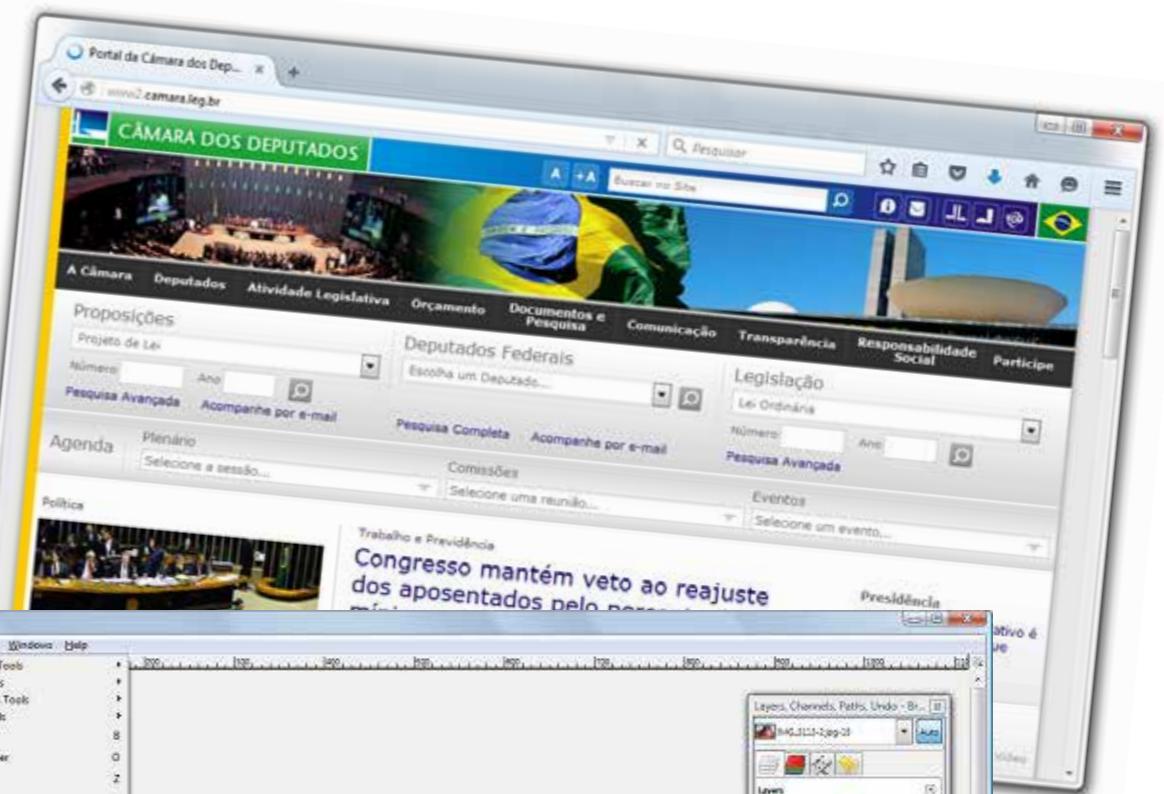


- Aplicações web, desktop e mobile
- Cálculos científicos
- Computação gráfica
- Automação de sistema
- Mineração de dados
- Big Data
- Machine learning
- Processamento de textos
- Tratamento e reconhecimento de imagens
- Animações 3D

# O QUE É E ONDE SE USA PYTHON



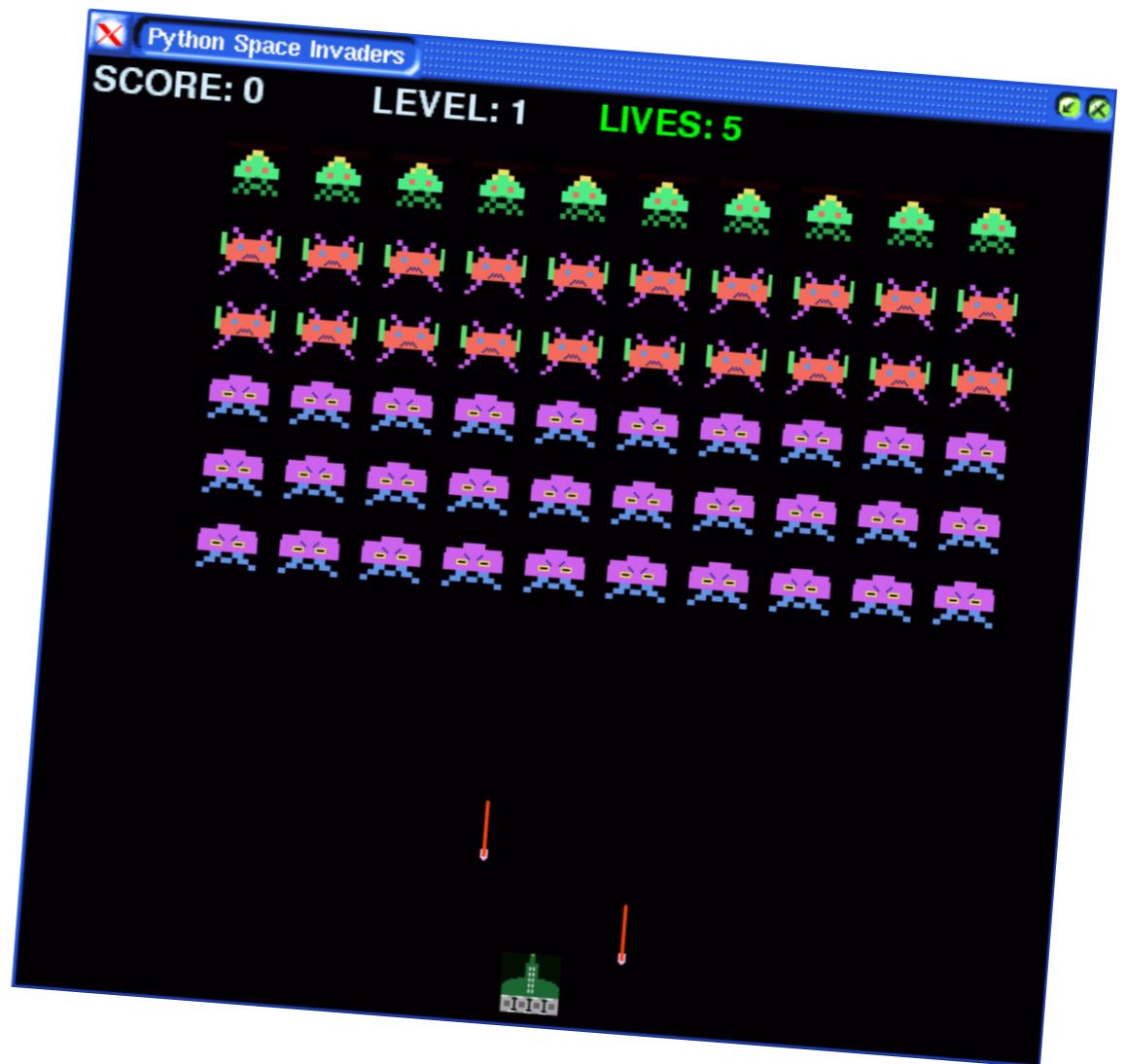
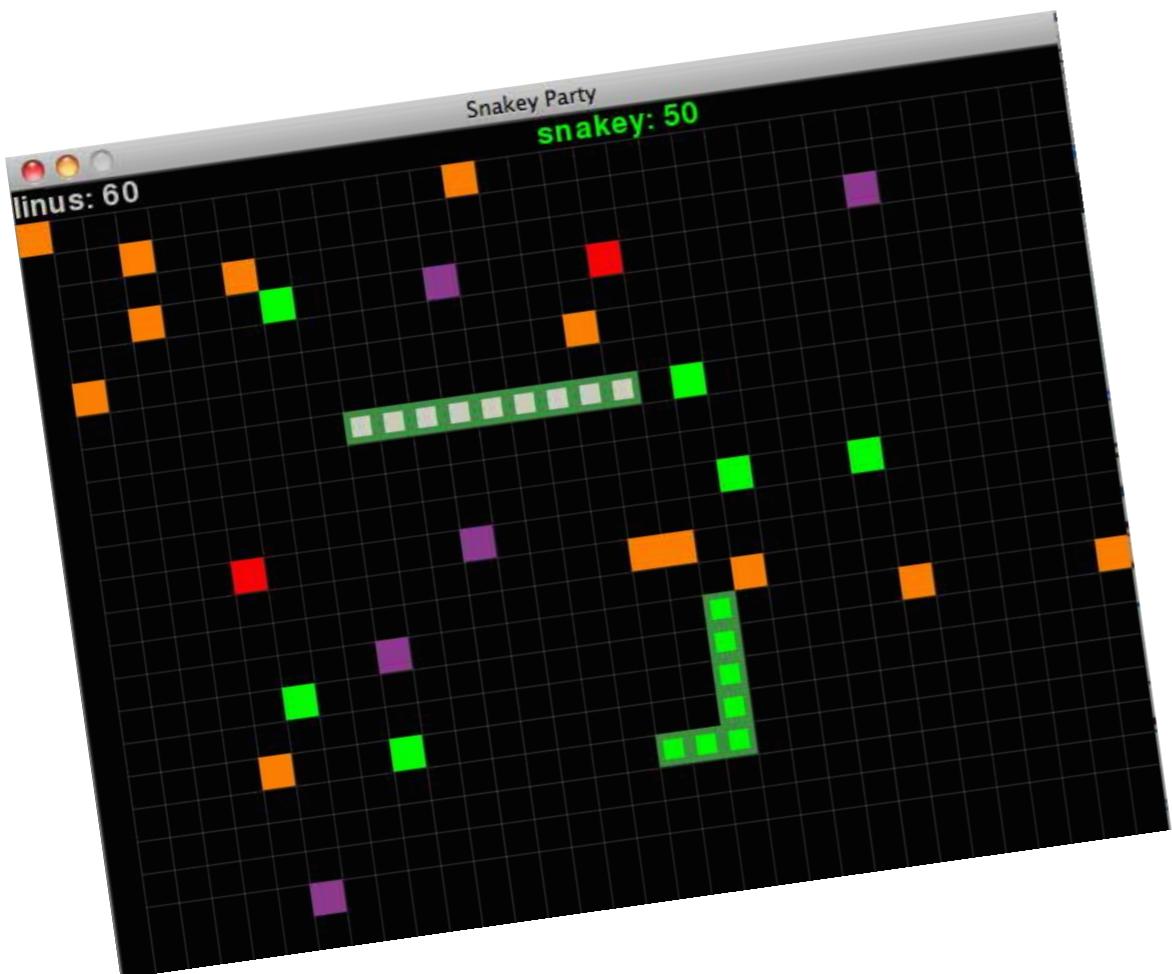
► Exemplos de utilização de Python:



# O QUE É E ONDE SE USA PYTHON



- ▶ Jogos feitos em Python:



Exemplo do mesmo programa  
em várias linguagens:

## Java

```
1 public class Hello
2 {
3     public static void main(String args[]) {
4         java.util.Scanner s = new java.util.Scanner(System.in);
5         System.out.print("Digite seu nome:");
6         String nome = s.nextLine();
7         System.out.println("Olá, " + nome);
8     }
9 }
```

# POR QUE PYTHON?



## Exemplo do mesmo programa em várias linguagens:

Java

```
1 public class Hello
2 {
3     public static void main(String args[]) {
4         java.util.Scanner s = new java.util.Scanner(System.in);
5         System.out.print("Digite seu nome:");
6         String nome = s.nextLine();
7         System.out.println("Olá, " + nome);
8     }
9 }
```

C

```
1 #include <stdio.h>
2 int main()
3 {
4     char nome[200];
5     printf("Digite seu nome:");
6     scanf("%s", nome);
7     printf("Olá, %s\n", nome);
8     return 0;
9 }
```

Fonte: <https://pythonhelp.wordpress.com/por-que-python/>

# POR QUE PYTHON?



## Exemplo do mesmo programa em várias linguagens:

Java

```
1 public class Hello
2 {
3     public static void main(String args[]) {
4         java.util.Scanner s = new java.util.Scanner(System.in);
5         System.out.print("Digite seu nome:");
6         String nome = s.nextLine();
7         System.out.println("Olá, " + nome);
8     }
9 }
```

C

```
1 #include <stdio.h>
2 int main()
3 {
4     char nome[200];
5     printf("Digite seu nome:");
6     scanf("%s", nome);
7     printf("Olá, %s\n", nome);
8     return 0;
9 }
```

## Pascal

```
1 program HelloWorld(output);
2 var
3     nome: string;
4 begin
5     writeln('Digite seu nome:');
6     read(nome);
7     writeln('Olá, ', nome);
8 end.
```

Fonte: <https://pythonhelp.wordpress.com/por-que-python/>

# POR QUE PYTHON?



## Exemplo do mesmo programa em várias linguagens:

Java

```
1 public class Hello
2 {
3     public static void main(String args[]) {
4         java.util.Scanner s = new java.util.Scanner(System.in);
5         System.out.print("Digite seu nome:");
6         String nome = s.nextLine();
7         System.out.println("Olá, " + nome);
8     }
9 }
```

C

```
1 #include <stdio.h>
2 int main()
3 {
4     char nome[200];
5     printf("Digite seu nome:");
6     scanf("%s", nome);
7     printf("Olá, %s\n", nome);
8     return 0;
9 }
```

Pascal

```
1 program HelloWorld(output);
2 var
3     nome: string;
4 begin
5     writeln('Digite seu nome:');
6     read(nome);
7     writeln('Olá, ', nome);
8 end.
```

## Python

```
1 nome = input('Digite seu nome:')
2 print ('Olá,', nome)
```

**"Simples é melhor  
que complexo"**

**(Tim Peters)**

# INSTALAÇÃO DO PYTHON



Tutorial de instalação  
do Python 3.4

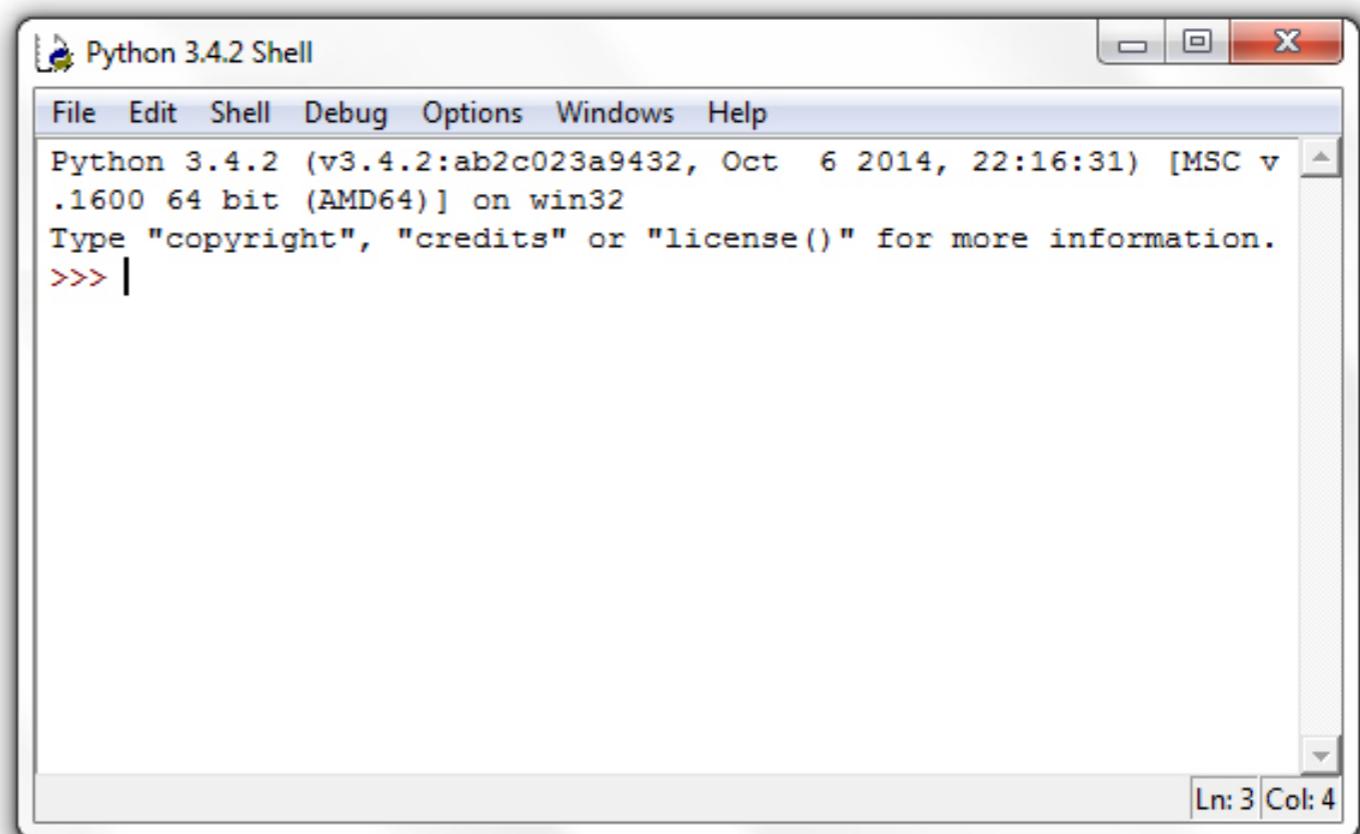
Selecione seu sistema operacional



[Sob licença CC-BY-NC-ND](#)

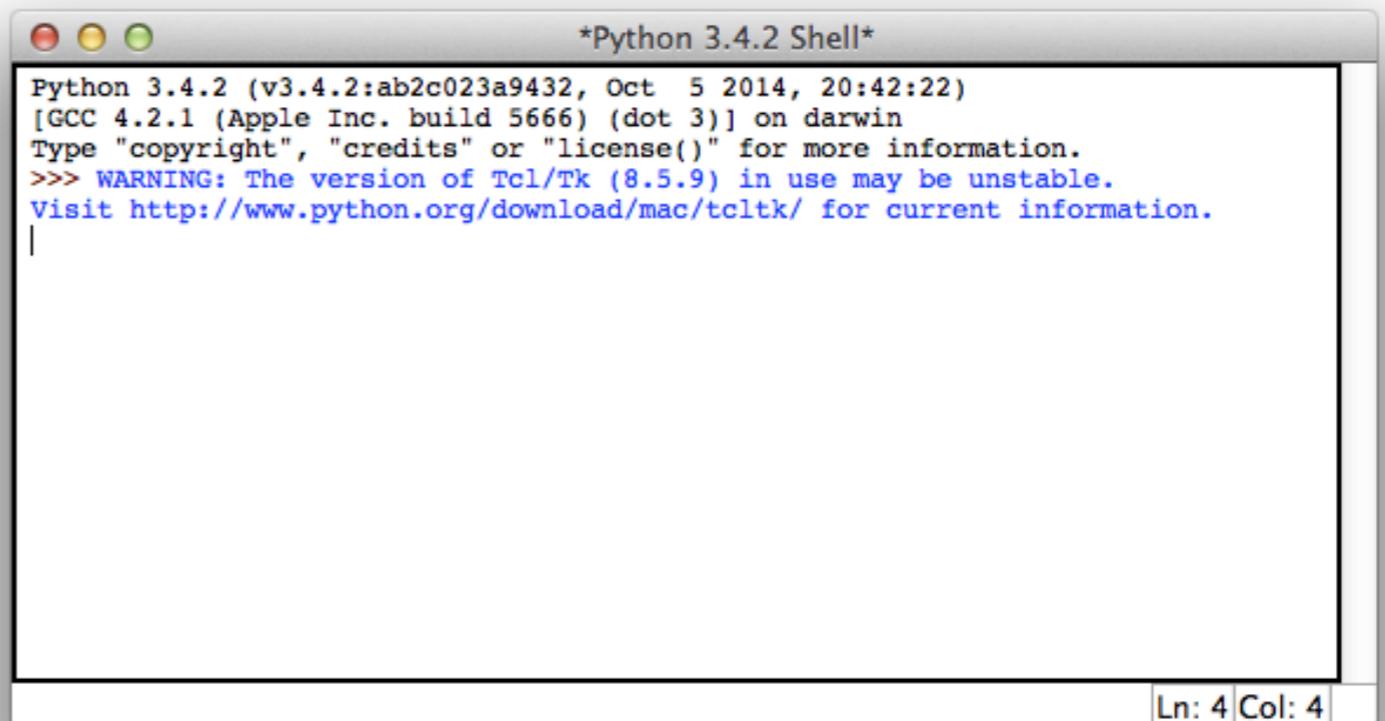
## IDLE no Windows

- **pelo Prompt de Comando** – acesse o Prompt de Comando e digite: python (se houver apenas essa versão instalada)
- **pelo Aplicativo** – no aplicativo IDLE (Python 3.4 GUI)



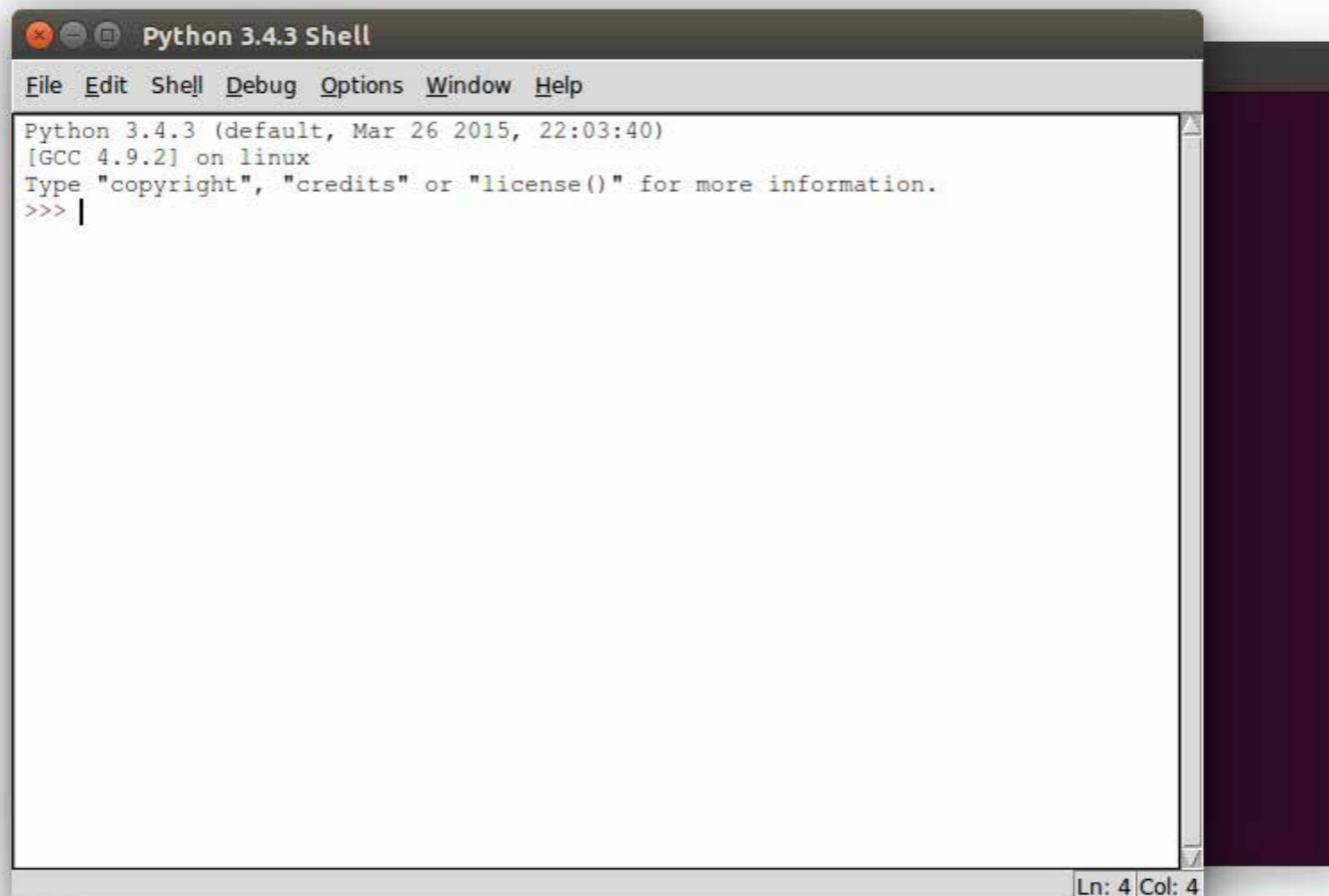
## IDLE no Mac OSx

- **pelo Terminal** – acesse o Terminal e digite: python3
- **pelo Aplicativo** – dentro de Applications (ou Aplicativos), encontre a pasta Python 3.4. Dê um duplo clique em IDLE.app



## IDLE no Ubuntu

- **pelo Terminal** – acesse o Terminal e digite: idle-python3.4



- A linha de comando é indicada pelas setas `>>>` e, como o nome diz, é nela que você deve dar os comandos sobre o que a linguagem deve fazer. Sua primeira tarefa é executar um `print`, o comando para imprimir uma mensagem na tela. O retorno, isto é, o resultado da execução, deve sair logo abaixo.

```
>>> print('Hello, Ladies!')
Hello, Ladies
```

- Variável é uma posição na memória RAM à qual podemos associar um valor ou expressão (informação).

Ela é criada no momento de execução do programa (não existe antes de executar o programa).

Ela possui um identificador dentro da memória que é o seu nome (e também um número que é seu endereço). Somos nós que damos nome às variáveis (em geral, nomes que nos façam lembrar o que ela armazena). Chamando este nome, o computador retorna o que está guardado naquela posição (ou variável).

- **Endereçamento:** retorna o endereço na memória onde uma variável está armazenada

- **Sintaxe:**

```
id(<variável>)
```

- **Exemplo:**

```
>>> a = 5
>>> id(a)
4297370816
>>> id(5)
4297370816
```

- **Regras:**

- podem ser usados **algarismos, letras ou \_**
- nunca devem começar com um algarismo
- não podemos usar palavras-chave naturais ao Python, por exemplo if, while, etc. (lista completa: import keyword e print keyword.kwlist).

- **Sintaxe:**

<nome da variável> = <valor que quero armazenar>

<> será usado para indicar que você deve inserir alguma informação ali

- **Tipos de variáveis:**

Uma variável possui tipos diferentes que as caracterizam.

- **Numéricos:** armazenam números. São: inteiros (`int`), de ponto flutuante (`float`), complexos (`complex`)
- **Literais:** Armazemam caracteres (qualquer um do teclado) ou sequências de caracteres. São strings (`string`)
- **Lógicos:** Armazemam verdadeiro ou falso. São: booleanos (`bool`)

- **Numéricos:**

- Tipo inteiro (**int**): números inteiros

```
>>> a = 2  
>>> b = 4  
>>> c = 0
```

- Tipo ponto flutuante (**float**): números racionais (em que aparece o ponto da casa decimal ou em notação científica)

```
>>> d = 2.0  
>>> e = 4E-2  
>>> f = 0.0
```

- **Numéricos (continua):**

- **Tipo complexo (complex):** comprehende todos os números complexos. São representados por dois números de ponto flutuante (um para a parte real e um para a parte imaginária, junto com o j)

```
>>> g = 2+3j  
>>> h = 4+1j  
>>> i = 2.3j
```

## VARIÁVEIS



- **Literais:**

- Tipo string (str): comprehende todos os caracteres dentro de aspas (simples ou duplas)

```
>>> j = '2'  
>>> k = '2.0'  
>>> l = '12@#nikukyu'
```

- **Lógicos:**

- Tipo lógico (bool): comprehende respostas do tipo True ou False

```
>>> m = True  
>>> n = False
```



Tá, como eu sei se o Python entendeu que a variável m é do tipo lógico e não uma string?

Existe uma função em Python que testa o tipo de variável.

- **Sintaxe:**

```
type(<nome da variável>)
```

- **Exemplo:**

```
>>>type(m)
<type 'bool'>
```

**AGORA É  
COM VOCÊ**

**2)** Tente definir `m = true`.

O que acontece? Por que dava certo quando era `True`?

Tente `m = "true"` e verifique o tipo de `m`. O que aconteceu?

**1)** Teste algumas variáveis definidas anteriormente.

Perdeu alguma variável pelo caminho?

```
a = 2
b = 4
c = 0
d = 2.0
e = 4E-2
f = 0.0
g = 2+3j
h = 4+1j
i = 2.3j
j = '2'
k = '2.0'
l = '12@#nikukyu'
m = True
n = False
```



## RESPOSTAS

1) a, b, c: int;  
d, e, f: float;  
g, h, i: complex;  
j, k, l: string;  
m, n: bool

Por exemplo:

```
>>> type(a)  
<class 'int'>
```

```
>>> type(d)  
<class 'float'>
```

```
>>> type(g)  
<class 'complex'>
```

```
>>> type(j)  
<class 'str'>
```

## RESPOSTAS

```
2) >>> m = true
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    m = true
NameError: name 'true' is not defined

>>> m = 'true'
>>> type(m)
<class 'str'>
```

- **Atribuição Múltipla:** Permite definir diversas variáveis ao mesmo tempo e inclusive trocar os valores entre elas.
- **Sintaxe:**

<variável1>, <variável2> = <valor da variável1>, <valor da variável2>

- **Exemplo:**

<pre>&gt;&gt;&gt; a, b, c = 2, 3, 4</pre> <pre>&gt;&gt;&gt; a, b, c</pre> <pre>(2, 3, 4)</pre>	<pre>&gt;&gt;&gt; a, b = b, c</pre> <pre>&gt;&gt;&gt; a, b, c</pre> <pre>(3, 4, 4)</pre>
--	--

Como vimos, em Python podemos mudar o tipo de uma variável ao longo do programa, por isso chamamos Python de uma linguagem dinamicamente tipada.

No exemplo anterior, a variável m era do tipo lógica e depois mudou para tipo string. É possível fazer isso com muitas variáveis ao mesmo tempo.

```
>>> a, b, c = 'Jet'  
>>> a, b, c  
('J', 'e', 't')  
>>>  
>>> a,b = b,a  
>>> a,b,c  
('e','J','t')
```



Outra forma de transformar um tipo de variável em outro é usando os comandos:

`int()` ou `float()` ou `complex()` ou `str()`

**Exemplos:**

<code>&gt;&gt;&gt; a = 2</code>	<code>&gt;&gt;&gt; d = str(a)</code>
<code>&gt;&gt;&gt; type(a)</code>	<code>&gt;&gt;&gt; d</code>
<code>&lt;class 'int'&gt;</code>	<code>'2'</code>
<code>&gt;&gt;&gt; b = float(a)</code>	<code>&gt;&gt;&gt; e = int(d)</code>
<code>&gt;&gt;&gt; type(b)</code>	<code>&gt;&gt;&gt; e</code>
<code>&lt;class 'float'&gt;</code>	<code>2</code>
<code>&gt;&gt;&gt; c = complex(a)</code>	<code>&gt;&gt;&gt; type(e)</code>
<code>&gt;&gt;&gt; c</code>	<code>&lt;class 'int'&gt;</code>
<code>(2+0j)</code>	
<code>&gt;&gt;&gt; type(c)</code>	
<code>&lt;class 'complex'&gt;</code>	

Podemos fazer operações em Python. Tanto numéricas, quanto lógicas.

- Operadores numéricos básicos:

**Adição:** +

**Subtração:** -

**Divisão:** /

**Multiplicação:** \*

**Potenciação:** \*\*

**Resto de uma divisão:** %

### Exemplos:

```
>>> a = 2
>>> b = 3
>>> c = (a + b - 2 * a + a ** a) / 5
>>> c
1.0
```

### Tipos de divisão:

```
>>> 10 / 8 (divisão de números inteiros)
1.25
>>> 10 % 8 (resto da divisão entre números inteiros)
2
```

### Algumas diferenças entre versões anteriores à 3.x

Em versões do Python anteriores à 3.x, a divisão retornava o seguinte resultado:

```
>>> 5 / 2
```

**2** apenas o resultado íntero da divisão

esse x é  
qualquer  
versão acima  
do 3.0

O Python 2.x imaginava que desejávamos um resultado inteiro. Para obter o resultado da divisão de forma decimal seria necessário informar da seguinte forma:

```
>>> 5.0 / 2  
2.5
```

```
>>> 5 / 2.  
2.5
```

```
>>> 5 / 2.0  
2.5
```

```
>>> 5. / 2.  
2.5
```

```
>>> 5.0 / 2  
2.5
```

Ou seja, informando qualquer um dos valores como decimal seria retornado o valor decimal.

Caso queria apenas o resultado inteiro de uma divisão, nas versões 3.x, utilize da seguinte forma:

```
>>> 5 // 2
```

```
2
```

apenas o resultado íntero da divisão

## OPERADORES

- **Operadores lógicos ou relacionais:** servem para fazer perguntas que possam ser respondidas com True ou False (verdadeiro/falso)

**Maior que:** >

**Menor que:** <

**Maior ou igual a:** >=

**Menor ou igual a:** <=

**Idêntico:** ==

**Diferente de:** !=

**Não:** not

**E:** and (exige que todas as condições sejam satisfeitas)

**Ou:** or (apenas uma das condições precisa ser satisfeita)

### Exemplos:

```
>>> a = 2
```

```
>>> b = 3
```

```
>>> a > b
```

```
False
```

```
>>> b > a
```

```
True
```

```
>>> a == b
```

```
False
```

```
>>> a != b
```

```
True
```

```
>>> a > b and b > a
```

```
False
```

```
>>> a > b or b > a
```

```
True
```

```
>>> not (a != b) == False
```

```
True
```

### TABELA VERDADE

A	B	A and B	A or B	not (A)	not (B)
True	True	True	True	False	False
True	False	False	True	False	True
False	True	False	True	True	False
False	False	False	False	True	True

**AGORA É  
COM VOCÊ**



Se  $a = 2$ ,  $b = 3$ ,  $c = 2.0$  e  $d = '2.0'$ ,  
faça as operações a seguir:

$a + b$

$b ** a$

$a + c$

E teste as condições:

$a + d$

$a == c$

$a <= b$

$a < b \text{ and } b < c$

$a < b \text{ or } b < c$

$a > c \text{ or } a >= c$

$\text{not}(a != b \text{ and } b <= (a**2) - 1)$

### RESPOSTAS

```
>>> a = 2          >>> a + b
>>> b = 3          5
>>> c = 2.0        >>> b ** a
>>> d = '2.0'       9
                           >>> a + c
                           4.0
                           >>> a + d
                           Traceback (most recent call last):
                           File "<pyshell#7>", line 1, in <module>
                                 a + d
                           TypeError: unsupported operand type(s) for
                           +: 'int' and 'str'
```

## RESPOSTAS

```
>>> a == c  
True  
>>> a <= b  
True  
>>> a < b and b < c  
False  
>>> a < b or b < c  
True  
>>> not(a != b and b <= (a**2)-1)  
False
```

- **Concatenação de caracteres/strings:** a concatenação é uma cola para trechos de textos.

- **Sintaxe:**

```
'<string>' + '<string>'
```

- **Exemplos:**

```
>>> 'a' + 'b'  
'ab'
```

```
>>> 'PyLadies ' + 'São Paulo'  
'PyLadies São Paulo'
```

## OPERAÇÕES COM STRINGS

- **Usando a barra invertida (\):** É possível adicionar dentro da string algum comando não imprimível.
- Exemplos:

```
>>> print('Jet \nBits') (\n =  
Jet nova linha)  
Bits
```



```
>>> print('Jet \tBits') (\t = tab)  
Jet    Bits
```

Outros:  
\b = backspace

\\" = \

Para pular linha dentro de uma string é só usar \n no fim de cada linha  
OU

usar 3 aspas duplas:

```
>>> print("""  
Jet  
vs  
Bits""")  
Jet  
vs  
Bits
```

- **Uppercase:** todas as letras de uma string em maiúscula (pode operar sobre uma variável do tipo texto também)

- **Sintaxe:**

```
'<texto>'.upper()
```

- **Exemplo:**

```
>>> 'Flocos'.upper()  
'FLOCOS'
```



- **Lowercase:** todas as letras de uma string em letra minúscula

- **Sintaxe:**

```
'<texto>'.lower()
```

- **Exemplo:**

```
>>> 'PyLadies'.lower()  
'pyladies'
```

- **Capitalize**: coloca apenas o primeiro caractere em letra maiúscula (se for um número, ele não faz nada)

- **Sintaxe**:

```
'<texto>'.capitalize()
```

- **Exemplo**:

```
>>> 'dia de banho'.capitalize()  
'Dia de banho'
```



- **Title:** coloca a primeira letra de cada palavra em maiúscula

- **Sintaxe:**

```
"<texto>".title()
```

- **Exemplo:**

```
>>> 'flocos e pudim'.title()
```

```
'Flocos E Pudim'
```



## MAIS SOBRE STRINGS

- **Índice em uma string:** número que indica a posição de cada caractere na string

- **Sintaxe:**

<variável tipo string> [número]

O número será 0 (zero)  
para o primeiro caractere na  
string, 1 para o segundo,  
2 para o terceiro, etc.

- **Exemplo:**

```
>>> a = 'RUSSO AZUL'  
>>> a[0]  
'R'  
>>> a[2]  
'S'  
>>> a[5]  
' '  
>>> a[-1]  
'L'
```



Foto: Pixabay/CC

## MAIS SOBRE STRINGS

- **Fatias de uma string:** retorna parte da string, começando pelo primeiro índice e terminando no anterior ao segundo.

- **Sintaxe:**

<variável>[índice1:índice2]

- **Exemplo:**

```
>>> a[6:9]
```

```
'AZU'
```

```
>>> a[4:10]
```

```
'O AZUL'
```

```
>>> a[6:]
```

```
'AZUL'
```

```
>>> a[-1:]
```

```
'L'
```

Quando omitimos um índice, é  
mostrado o caractere do extremo  
correspondente

## MAIS SOBRE STRINGS

- **Incremento de fatia:** permite pegar caracteres alternados, e até mesmo inverter uma string

- **Sintaxe:**

<variável>[<índice1>:<índice2>:<passo>]

*Z Passo é o número  
de strings que  
queremos pular*

- **Exemplo:**

```
>>> a[6:10:2]  
'AU'
```

```
>>> a[::-1]  
'LUZA OSSUR'
```

## MAIS SOBRE STRINGS

**AGORA É  
COM VOCÊ**

Defina a variável

a = 'maine coon' e com  
apenas 1 (um) comando  
sobre a variável:

- 1)** Escreva MAINE COON
- 2)** Escreva Maine Coon
- 3)** Inverta maine coon

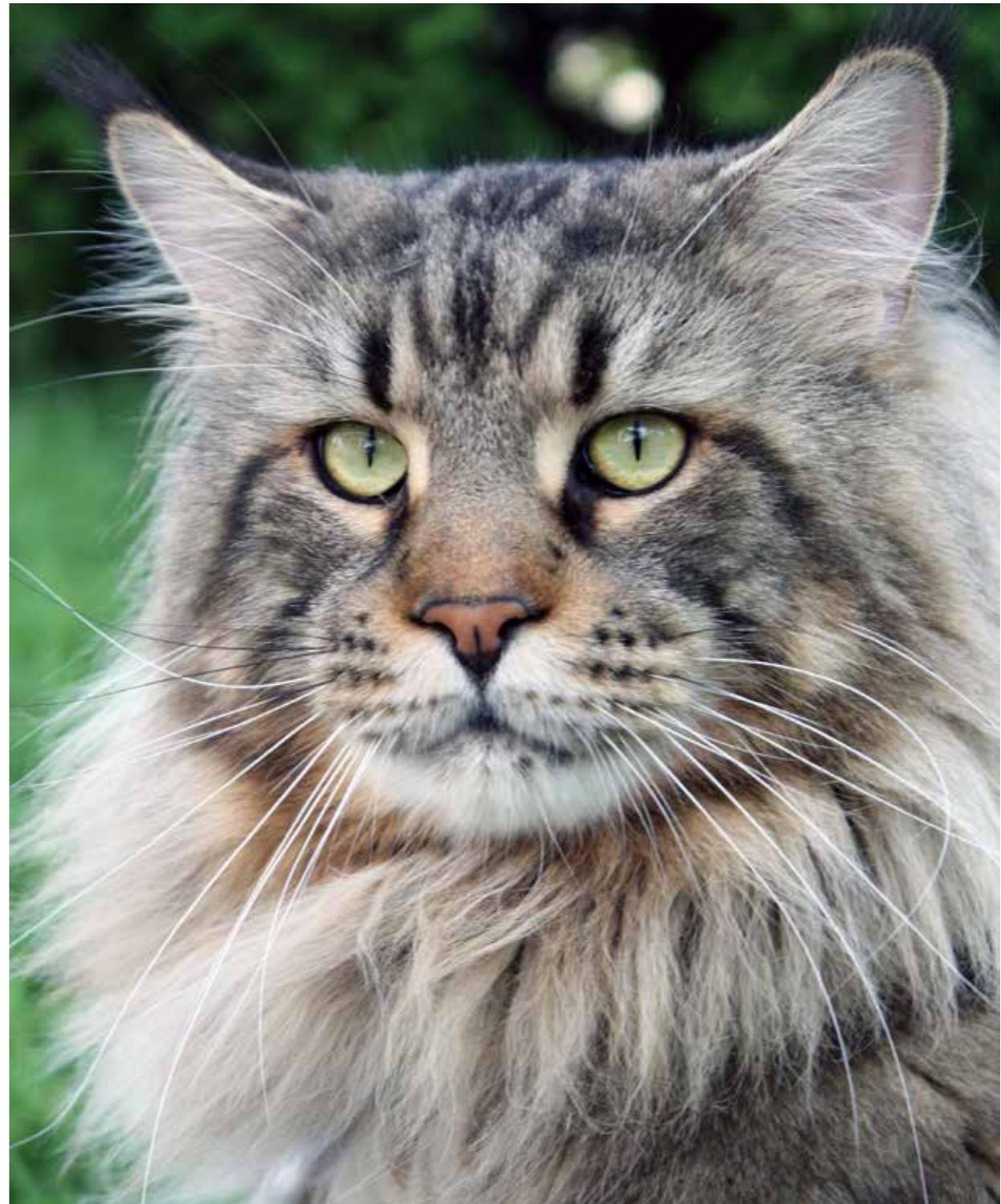


Foto: Pixabay/CC

### RESPOSTAS

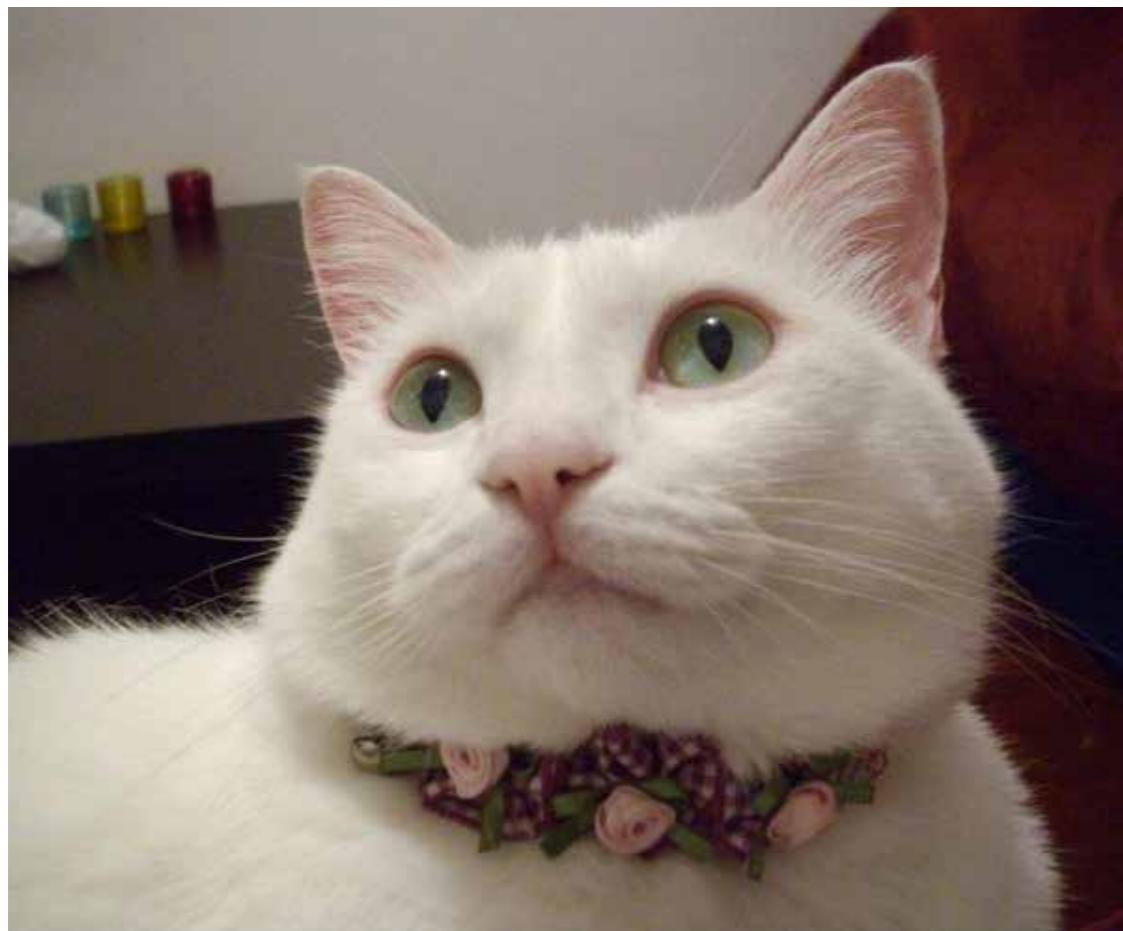
1) `>>> a.upper()`  
`'MAINE COON'`

2) `>>> a.title()`  
`'Maine Coon'`

3) `>>> a[::-1]`  
`'nooc eniam'`

# DESAFIO

Via de regra, os índices de uma string são imutáveis. Crie uma variável e atribua o valor 'Bitcheti'.



Usando as operações com strings que você aprendeu, faça o último 'i' na variável ser transformado em 'e' (sem redefinir a variável).

### POSSÍVEL RESPOSTA DO DESAFIO

```
>>> c = 'Bitcheti'  
>>>  
>>> c[:7] + 'e'  
'Bitchete'
```

- **Tamanho da string:** conta quantos caracteres tem uma string

- **Sintaxe:**

```
len(<string>)
```

- **Exemplo:**

```
>>> a = 'filhote'
```

```
>>> len(a)
```

7



- Comando **Começa com (startswith)**  
ou

**Termina com (endswith)**: testa se um texto começa/termina com um elemento (é um teste lógico)

- **Sintaxe:**

```
<variável a averiguar>.startswith('elemento que procuro')
```

ou

```
<variável a averiguar>.endswith('elemento que procuro')
```

## MAIS SOBRE STRINGS

- Exemplo:

```
>>> a = '''Gatos amam mais  
as pessoas do que elas  
permitiriam. Mas eles têm  
sabedoria suficiente para  
manter isso em segredo.'''
>>> a.startswith('G')
```

True

```
>>> a.startswith('n')
```

False

```
>>> a.endswith('.')
```

True

Python é case sensitive,  
diferencia maiúscula de  
minúscula



Frase de Mary Wilkins

## MAIS SOBRE STRINGS



- **Comando Find:** procura uma string dentro de um texto e retorna a posição de seu primeiro caractere. Se houver mais que uma string igual, ele retorna a posição da primeira. Se não encontrar, ele retorna -1.

- **Sintaxe:**

```
<variável que contém o texto/string>.find('string  
que procuro')
```

ou

```
<variável que contém o texto/string>.find('string  
que procuro', <posição a partir da qual quero  
procurar>)
```

## MAIS SOBRE STRINGS

- **Exemplo:**

```
>>> a = 'Existem duas maneiras de nos refugiarmos das  
misérias da vida: música e gatos.'
```

```
>>> a.find('t')
```

4

retornou a posição da primeira ocorrência a partir do início

```
>>> a.find('a', 11)
```

14

```
>>> a.find('!')
```

-1

retorna -1 se não encontra o caracter

retornou a posição da primeira ocorrência a partir da posição 11



Frase de Albert Schweizer

- **Comando Replace:** troca uma string por outra dentro de um texto.

- **Sintaxe:**

```
<variável>.replace('string que quero mudar', 'nova  
string')
```

- **Exemplo:**

```
>>> a = 'Existem duas maneiras de nos refugiarmos  
das misérias da vida: música e gatos.'  
>>> a.replace('misérias', 'mazelas')  
'Existem duas maneiras de nos refugiarmos das  
mazelas da vida: música e gatos.'
```

## LISTAS



- **Listas:** permitem armazenar várias informações diferentes (número, string, lógico) em uma mesma variável. São divididas em duas categorias, aquelas que podem ser editadas (chamadas apenas de **listas**) e as que não são editáveis (chamadas de **tuplas**).
- **Sintaxe (listas editáveis ou simplesmente LISTAS):**

```
<variável> = [info1, info2, info3]
```

## LISTAS

- Exemplo 1:

```
>>> a = ['Gato', 9, True]
```

```
>>> print(a[0])
```

Gato

Para chamar um dos elementos, uso o índice da mesma entre colchetes como faço com strings

- Exemplo 2:

```
>>> a = ['Gato', 9, True, [5, 'Rapper
```

Bits']]

```
>>> a[3]
```

[5, 'Rapper Bits']

O elemento 3 da lista a é uma outra lista

Exemplo de lista com um string, um inteiro, um booleano e uma lista

- **Comando Append:** acrescenta dados a uma lista.

- **Sintaxe:**

```
<variável1>.append(<variável2>)
```

- **Exemplo:**

```
>>> a = ['Gato', 'de']
>>> a.append('Botas')
>>> print(a)
['Gato', 'de', 'Botas']
```



- **Comando Join:** gruda os elementos de uma sequência de strings, usando um parâmetro fornecido
- **Sintaxe:**

```
'<parâmetro que quero usar>'.join(<nome da sequência>)
```

- **Exemplo:**

```
>>> chas = ['Camomila', 'Hortela', 'Cidreira']
>>> '/'.join(chas)
'Camomila/Hortela/Cidreira'
```

Atenção: esse comando não muda a variável; se precisar usar o resultado depois, é necessário criar uma nova variável

- **Comando Split:** separa uma string em pontos onde existam separadores de texto (espaço, tab, enter, '/', '+', etc.), criando uma lista de strings.

- **Sintaxe:**

```
'<string>'.split('<separador>')
```

- **Exemplo:**

```
>>> '1,2,3,4'.split(',')
['1', '2', '3', '4']
```

## TUPLAS != LISTAS



- **Tuplas:** são similares às listas, mas imutáveis. Não podemos adicionar ou modificar nenhum de seus elementos.
- **Sintaxe:**

```
<variável> = (info1, info2, info3)
```

ou

```
<variável> = info1, info2, info3
```

## TUPLAS != LISTAS

- Exemplo 1:

```
>>> a = (3, 5, 8)  
>>> a  
(3, 5, 8)  
>>> b = 3, 5, 8  
>>> b  
(3, 5, 8)
```

```
>>> a == b  
True  
>>> type(b)  
<class 'tuple'>
```

Exemplo de uma tupla com um string, um inteiro, um booleano, uma lista e outra tupla

- Exemplo 2:

```
>>> c = ('Gato', 9, True, [10, 'fofs'], ('Leloo',  
'Leloo', 'Leloo', 'Leloo'))  
>>> c[3]  
[10, 'fofs']
```

O elemento 4 da tupla 'c' é uma lista

## TUPLAS != LISTAS



- **CASO ESPECIAL:** para criar uma tupla de um único elemento, não funciona colocar o elemento entre parênteses. O Python fica em dúvida se é um elemento entre parênteses ou uma tupla.

- **Sintaxe para tupla de um só elemento:**

<variável> = (<elemento>, )

- **Exemplos:**

```
>>> a = (1,)  
>>> a  
(1,)  
>>> type(a)  
<class 'tuple'>
```

```
>>> a = (1)  
>>> a  
1  
>>> type(a)  
<class 'int'>
```

Os exemplos parecem iguais, mas são reconhecidos de forma diferente pelo Python

- **Resumindo:** elementos de listas e tuplas são chamados da mesma forma, usando o índice do elemento entre colchetes, como fazemos ao chamar o caractere de uma string. A diferença? Na lista posso mudar, mas na tupla, não. Assim, os comandos que funcionam para ambas são em geral os mesmos, com exceção dos comandos que estão ligados à edição de uma lista, pois estes não valerão para tuplas.

## TUPLAS != LISTAS

- Mas, pra que existe tupla se ela não poder ser mudada?

- Muito bem, Leeloo!  
A resposta é que,  
quando vamos criar uma  
variável que pode ser  
mudada posteriormente,  
precisamos de mais  
espaço (bits) para  
armazená-la. Então  
quando usamos uma tupla, estamos economizando espaço.  
Obviamente, que se formos usar uma variável que queremos  
aumentar ao longo do código, é melhor usar listas mutáveis.



## MAIS SOBRE STRINGS

**AGORA É  
COM VOCÊ**

Dada a variável

```
var = 'Para medir o calor do dia, olhem  
para o comprimento do gato que dorme.'
```

- 1)** Qual o tamanho da string?
- 2)** Verifique se começa com 'p'?  
Justifique sua resposta
- 3)** Verifique se termina com ':'
- 4)** Verifique a posição do caractere ','
- 5)** Troque o caractere '.' por '!'.
- 6)** Dada a lista mercado = ['1 kg de banana',  
'12 ovos', '1kg de farinha'], acrescente a  
string 'fermento em pó'.



## MAIS SOBRE STRINGS

### RESPOSTAS

1) `>>> len(var)`  
`70`

2) `>>> var.startswith('p')`  
`False`

3) `>>> var.endswith('.')`  
`True`

4) `>>> var.find(',',')`  
`25`

5) `>>> var.replace('.','!!')`  
`'Para medir o calor do dia, olhem para o comprimento do gato que dorme!'`

6) `>>> mercado = ['1 kg de banana', '12 ovos', '1 kg de farinha']`  
`>>> mercado.append('fermento em pó')`  
`>>> mercado`  
`['1 kg de banana', '12 ovos', '1 kg de farinha',`  
`'fermento em pó']`

Python diferencia o p  
minúsculo do P maiúsculo

- **Marcadores de variáveis:** servem para referenciar uma variável dentro de um print.

Nas versões do Python anteriores a 3.x eram utilizados da seguinte forma:

- **Exemplos:**

Para inteiro: %d

```
>>> a = 3.0112
>>> print('Resultado = %d' %a)
Resultado = 3
```

Para float: %f

```
>>> b = 3.0112
>>> print('Resultado = %f' %b)
Resultado = 3.011200
```

Para string: %s

```
>>> c = '3.0112'
>>> print('Resultado = %s' %c)
Resultado = 3.0112
```

```
>>> d = 3.0112
```

```
>>> print('Resultado = %.2f' %d)
Resultado = 3.01
```

mostra duas casas  
após a vírgula

- A partir das versões 3.x foi apresentado um novo método para realizar a formatação através do .format()
- **Exemplos:**

Para inteiro:

```
>>> a = 3.0112
>>> print('Resultado = {:.d}'.format(int(a)))
Resultado = 3
```

Obs: a marcação de {:d} nesse caso acho inútil pois ele não aceita passar o {:d} para um ponto flutuante, apenas inteiros, e ao converter em int(a) é desnecessária a marcação de {:d}

```
>>> a = 3.0112
>>> print('Resultado = {}'.format(int(a)))
Resultado = 3
```

Para float:

```
>>> b = 3.0112
>>> print('Resultado = {}'.format(b))
Resultado = 3.0112
```

Para string:

```
>>> c = '3.0112'
>>> print('Resultado = {}'.format(c))
Resultado = 3.0112
```

Para quantidade de casas decimais:

```
>>> d = 3.0112
>>> print('Resultado = {:.2f}'.format(d))
Resultado = 3.01
```

*mostra duas casas após a vírgula*

Para data:

```
>>> from datetime import datetime
>>> '{:%Y-%m-%d %H:%M}'.format(datetime(2015, 9, 7, 21, 0))
'2015-09-07 21:00'
```

Para mais exemplos e possibilidades veja: <https://pyformat.info/>

**DICA:** Use o **dir** e o **help** para saber os comandos válidos para um tipo de variável

- **Comando dir:** lista os comandos válidos para a variável
- **Sintaxe:**

`dir(<variável>)`

- **Exemplo:**

`>>> dir('Felino')`

*Lista os comandos válidos  
para a string 'Felino'*

```
'__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',  
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__  
getnewargs__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__  
len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', __  
reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__',  
['__str__', '__subclasshook__', 'capitalize', 'casifold', 'center', 'count',  
'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index',  
'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower',  
'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust',  
'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex',  
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith',  
'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

- **Comando help:** mostra o que o comando faz com a variável

- **Sintaxe:**

```
help(<comando aplicado à variável>)
```

- **Exemplo:**

```
>>> help('PyLadies'.upper)
```

```
Help on built-in function upper:
```

```
upper(...) method of builtins.str instance
S.upper() -> str
```

Mostra o que o comando upper faz na string 'PyLadies'

Return a copy of S converted to uppercase.

- **Passos para a entrada de dados:**
  - O ideal é estar no modo edição para poder salvar o que vai escrever
  - Crio uma variável e faço a entrada dada pelo usuário via teclado coincidir com ela. Posso deixar uma mensagem para o usuário saber o que fazer.

A entrada de dados em Python é feita por meio do comando **input()**

- **Sintaxe 1:**

```
<variável> = input(<"mensagem para o usuário entrar com o  
dado">")
```

- **Exemplo:**

```
hq = input('Entre com o nome de seu personagem de  
quadrinho favorito: ')
```

```
Entre com o nome de seu personagem de quadrinho  
favorito: |
```

Mas o **input()** sempre retorna uma string (ele entende que o usuário digitou um texto). Se quero a entrada de um número, então preciso transformar a variável que foi lida como string em uma variável tipo numérico (usando o **int( )**, por exemplo).

- **Sintaxe 2:**

```
<variável> = <tipo de número*>(input("mensagem para o  
usuário entrar com o dado"))
```

*int* para inteiro e  
*float* para ponto  
flutuante

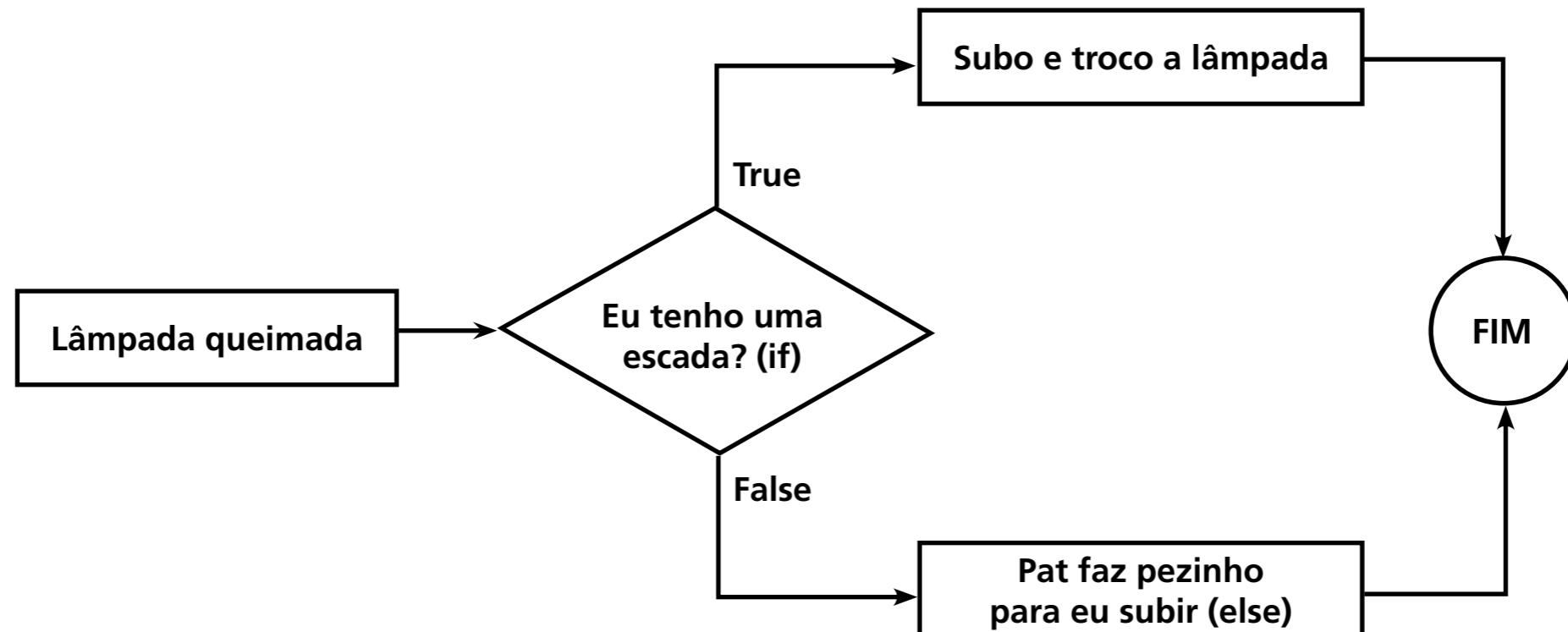
- **Exemplo:**

```
gasto = int(input('Escreva quantos quadrinhos em  
média você compra por semana: '))
```

**Escreva quantos quadrinhos em média você compra  
por semana: |**

### Voltemos ao exemplo da troca da lâmpada...

- se (**if**) o zelador tiver uma escada, eu subo nela e continuo o processo
- caso contrário (**else**), eu chamo a Pat para me dar pezinho, subo e continuo o processo de trocar a lâmpada



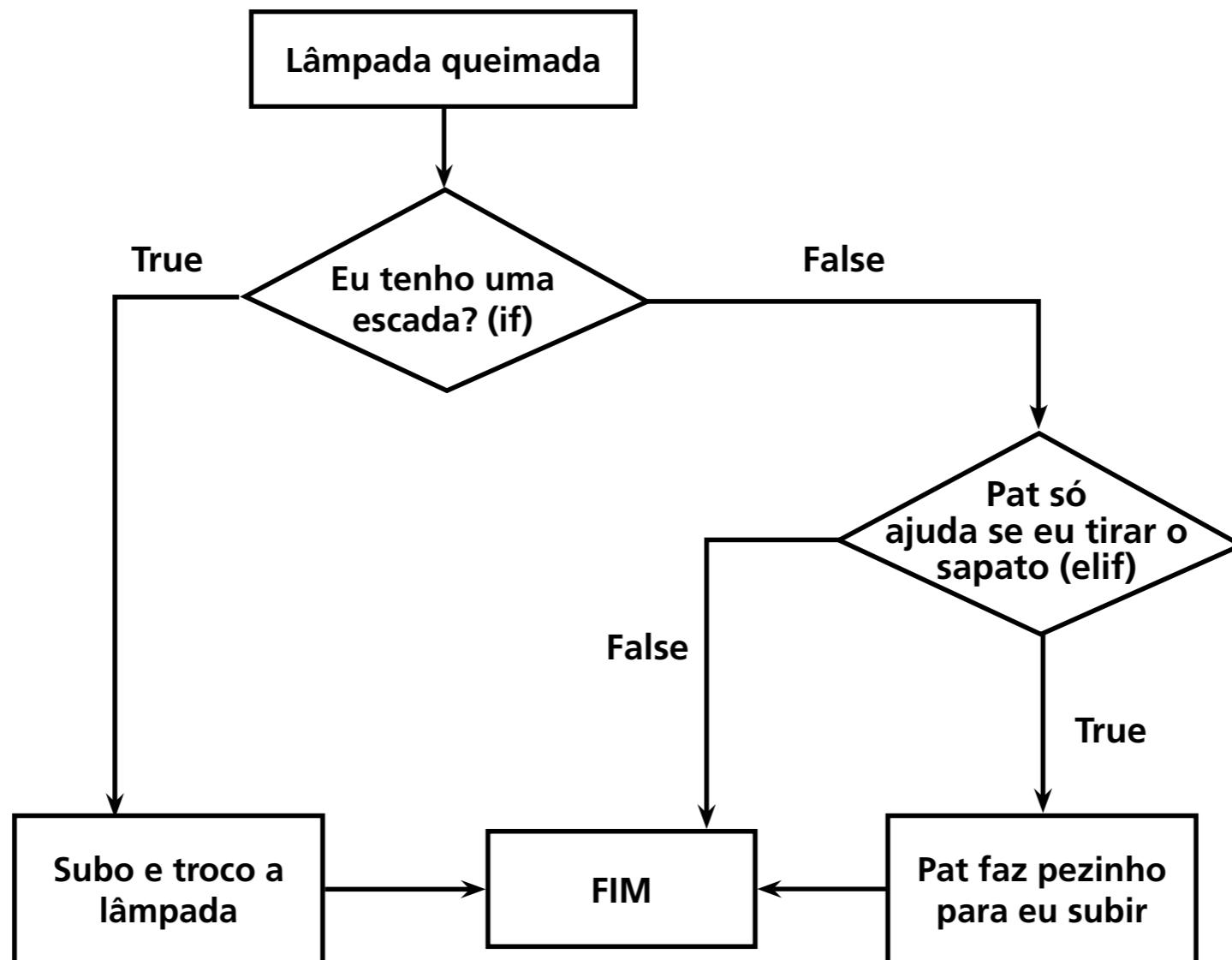
- **Sintaxe:**

```
if ____ <condição dada por operador booleano>____:  
    <o que tenho que fazer, caso a condição seja satisfeita>  
else:  
    <o que tenho que fazer, caso a condição não seja satisfeita>
```

- **Exemplo:**

```
escada = 'sim'  
if escada == 'sim':  
    print('Subir na escada e trocar a lâmpada')  
else:  
    print('Pat faz pezinho para eu subir')  
>>>  
Subir na escada e trocar a lâmpada  
>>>
```

- Mas a Pat pode dizer que só ajuda se eu tirar o sapato. Então, se eu tirar o sapato, eu uso o pezinho da Pat e continuo o processo: **elif**



### Sintaxe:

```
if ____ <condição dada por operador booleano>____:  
    <o que tenho que fazer, caso a condição seja satisfeita>  
(caso a condição anterior não seja satisfeita, tenho mais uma condição para verificar)  
elif ____<condição dada por operador booleano>____:  
    <o que tenho que fazer se a segunda condição for satisfeita>  
else:  
    <o que tenho que fazer, caso nenhuma das condições acima sejam satisfeitas>
```

- Exemplo:

```
escada = 'nao'  
Alini = 'descalça'  
if escada == 'sim':  
    print('Subir na escada e trocar a lâmpada')  
elif Alini == 'descalça':  
    print('Subir pezinho Pat e trocar a lâmpada')  
else:  
    print('Não trocar a lâmpada')  
  
>>>  
Subir pezinho Pat e trocar a lâmpada  
>>> |
```

**AGORA É  
COM VOCÊ**



Crie um código para uma outra PyLady executar. Pergunte seu nome e a convide para tomar um chá, um café ou um suco. Deixe a opção de ela não querer tomar nada; caso ela escolha esta opção escreva um smile triste.

Tempo: 7 minutos.

### RESPOSTA

```
print('Olá! Qual é seu nome?')
print('Você quer beber alguma coisa?')
opcao = int(input('Digite 1 se quer café, 2 se quer chá,
3 se quer um suco e 4 se não quer tomar nada: '))

if opcao == 1:
    print('Que bom, já deve estar na hora do coffee break;
vamos?')
elif opcao == 2:
    print('Que bom, já deve estar na hora do coffee break;
vamos?')
elif opcao == 3:
    print('Que bom, já deve estar na hora do coffee break;
vamos?')
else:
    print(':((')
```

## OUTRA RESPOSTA POSSÍVEL

```
print('Olá! Qual é seu nome?')
print('Você quer beber alguma coisa?')
opcao = int(input('Digite 1 se quer café, 2 se quer chá, 3
se quer um suco e 4 se não quer tomar nada: '))

if opcao == 1 or opcao == 2 or opcao == 3:
    print('Que bom, já deve estar na hora do coffee break;
vamos?')
else:
    print(' :( )
```

### UMA TERCEIRA RESPOSTA POSSÍVEL

```
print('Olá! Qual é seu nome?')
print('Você quer beber alguma coisa?')
opcao = int(input('Digite 1 se quer café, 2 se quer chá, 3
se quer um suco e 4 se não quer tomar nada: '))
i = 3

if opcao <= i:
    print('Que bom, já deve estar na hora do coffee break;
vamos?')
else:
    print(' :( )
```

## CONTADORES E ACUMULADORES



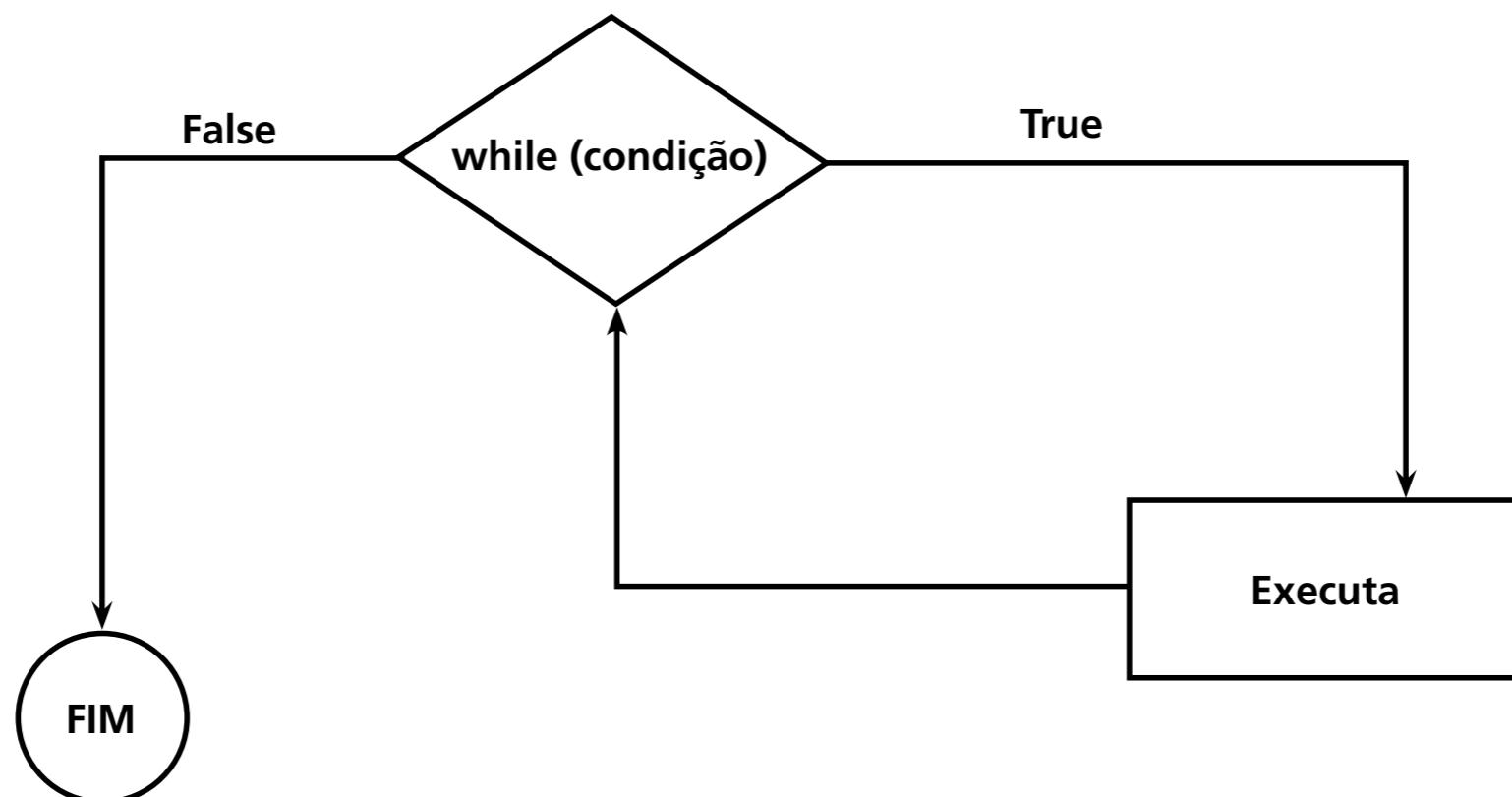
- **Contadores:** variáveis auxiliares que usamos para incrementar valores fixos, como por exemplo somar o valor 1 em uma variável. O valor pode ser qualquer valor positivo ou negativo.
- **Acumuladores:** Contadores que incrementam valores diferentes.

**Exemplo:** soma de 3 números dados pelo usuário.

```
total = 0
num = int(input('entre com um número: '))
total = total + num
num = int(input('entre com um número: '))
total = total + num
num = int(input('entre com um número: '))
total = total + num
```

## 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>
```

**Exemplo 1:** soma de 3 números dados pelo usuário.

```
i = 1
soma = 0
while i <= 3:
    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:**

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

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

**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.

### 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

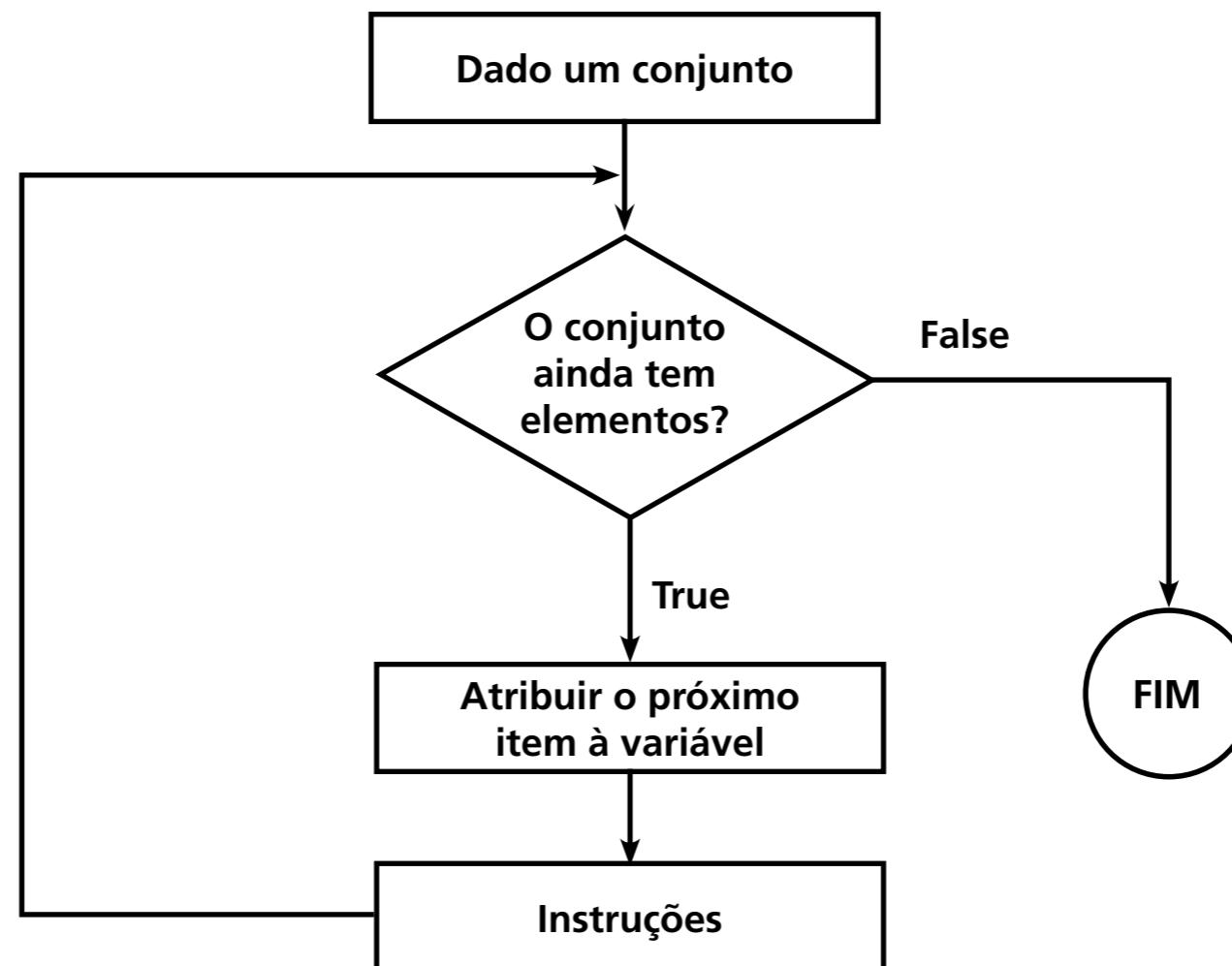
Às vezes quero interromper uma repetição no meio de um processo, dependendo do que o usuário digita ou outro motivo. Nestes casos, posso usar o **Break**.

- **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: {}' .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
>>>
```

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.



- **Sintaxe:**

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

- **Exemplo 1:**

```
a = ['Ariadyne', 'Beatrix', 'Caroline', 'Debora',  
'Élida', 'Jussara', 'Veronica']  
  
for i in a:  
    if i.startswith('C') and i.endswith('e'):  
        print(i)
```

- **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: {}'.format(x))
        print('A soma dos ímpares é: {}'.format(somaImpar))
print('A soma de numeros ímpares da lista é {}'.format(somaImpar))
```

**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.

## RESPOSTA

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

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

PHP

PYTHON

>>>

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** 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.

- **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 :)

- **Exemplo:**

```
def soma(a, b):      chamada da
    return a + b    função soma
```

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

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

- **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  
>>>
```

**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

### 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(int(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
>>>
```

# 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](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-complexos-em-python>
- [www.arquivodecodigos.net/principal/dicas\\_truques\\_categoria2.php?linguagem=12&categoria1=1&categoria2=59](http://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](http://www.arquivodecodigos.net/principal/dicas_truques_categoria2.php?linguagem=12&categoria1=1&categoria2=51)
- [www.dotnetperls.com/lower-python](http://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](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#format-spec>
- <https://www.python.org/dev/peps/pep-3101/>

Procurem trabalhar em grupo e  
trocar informações.

Tendo dúvidas, estamos à disposição

### Cálculo de café para manhã e tarde

Escreva um código que pergunte o número de participantes, qual a porcentagem de pessoas que toma café e quantos ml por pessoa.

Se o curso PyLadies tem 40 alunas, 7 monitoras e 3 mulheres na organização, calcule quantos litros de café são necessários para o coffee break da manhã, sabendo que cerca de 70% dos presentes beberão café e que em média cada um bebe cerca de 100ml.

```
partic = int(input('Qual o número de participantes? '))
percapita = float(input('Quantos ml de café por pessoa? '))
quemtoma = float(input('Qual a porcentagem de pessoas que
toma café? Digite apenas o número. '))
```

```
litros = partic * (percapita * 0.001) * (quemtoma * 0.01)
print('São necessários {:.2f} litros'.format(litros))
```

```
>>>
Qual o número de participantes? 50
Quantos ml de café por pessoa? 100
Qual a porcentagem de pessoas que toma café?
Digite apenas o número. 70
São necessários 3.50 litros
>>>
```

## MÃO NA MASSA

### EXERCÍCIO

#### Tabuada

Imprima as tabuadas de 1 a 10.



$$3 \times 1 = 3$$
$$3 \times 2 = 6$$
$$3 \times 3 = 9$$
$$3 \times 4 = 12$$
$$3 \times 5 = 15$$
$$3 \times 6 = 18$$
$$3 \times 7 = 21$$
$$3 \times 8 = 24$$
$$3 \times 9 = 27$$
$$3 \times 10 = 30$$

```
tabuada = 1
while tabuada <= 10:
    n = 1
    print('Tabuada {}'.format(tabuada))
    while n <= 10:
        print('{ } x { } = {}' .format(tabuada, n,
tabuada*n))
        n = n+1
    tabuada = tabuada + 1
```

**Palíndromos** são palavras grafadas de forma idêntica quando escritas da esquerda para a direita e da direita para a esquerda.

Por exemplo: asa, matam e radar.

Crie um código que peça uma palavra ao usuário e a teste para saber se é um palíndromo.

Mostre ao usuário se ela é igual ou não.

```
palavra = input('Entre com uma palavra: ')
if palavra == palavra[::-1]:
    print('É um palíndromo')
    print('{} é igual a {}'.format(palavra, palavra[::-1]))
else:
    print('Não é um palíndromo')
    print('{} é diferente de {}'.format(palavra,
palavra[::-1]))
```

### Show do Milhão falso

Escreva um programa que mostre na tela uma pergunta e peça que o usuário entre com a alternativa correta para o autor da frase. Se o usuário entrar com a alternativa correta, escreva na tela "Resposta exata!", caso contrário escreva "Resposta errada." e conte ao usuário qual é a resposta correta.

**Pergunta:** Quem é o autor da seguinte frase? "Independência ou morte!"

#### Alternativas:

- a. Jânio Quadros
- b. Floriano Peixoto
- c. Pedro Álvares Cabral
- d. Dom Pedro I

```
print("""Quem é o autor da seguinte frase?  
"Independência ou morte!"  
a. Jânio Quadros  
b. Floriano Peixoto  
c. Pedro Alvares Cabral  
d. Dom Pedro I""")  
  
resposta = input('Entre com a letra da alternativa  
escolhida: ')  
  
if resposta == 'd':  
    print('Resposta exata!')  
else:  
    print('Resposta errada')  
    print('A resposta certa é d')
```