



# Introdução à lógica de programação com Python

## O QUE É PYLADIES?

PyLadies é um grupo internacional de mentoría 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:

- ensinar a alguém o caminho  
de casa para o colégio
  - receita de bolo

## O QUE É ALGORITMO



Programar um computador é escrever instruções em qualquer linguagem que o computador entenda.

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

Python é uma destas linguagens.

Python é uma linguagem de alto nível, o que significa que é muito próximo à linguagem humana.

## 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



- 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



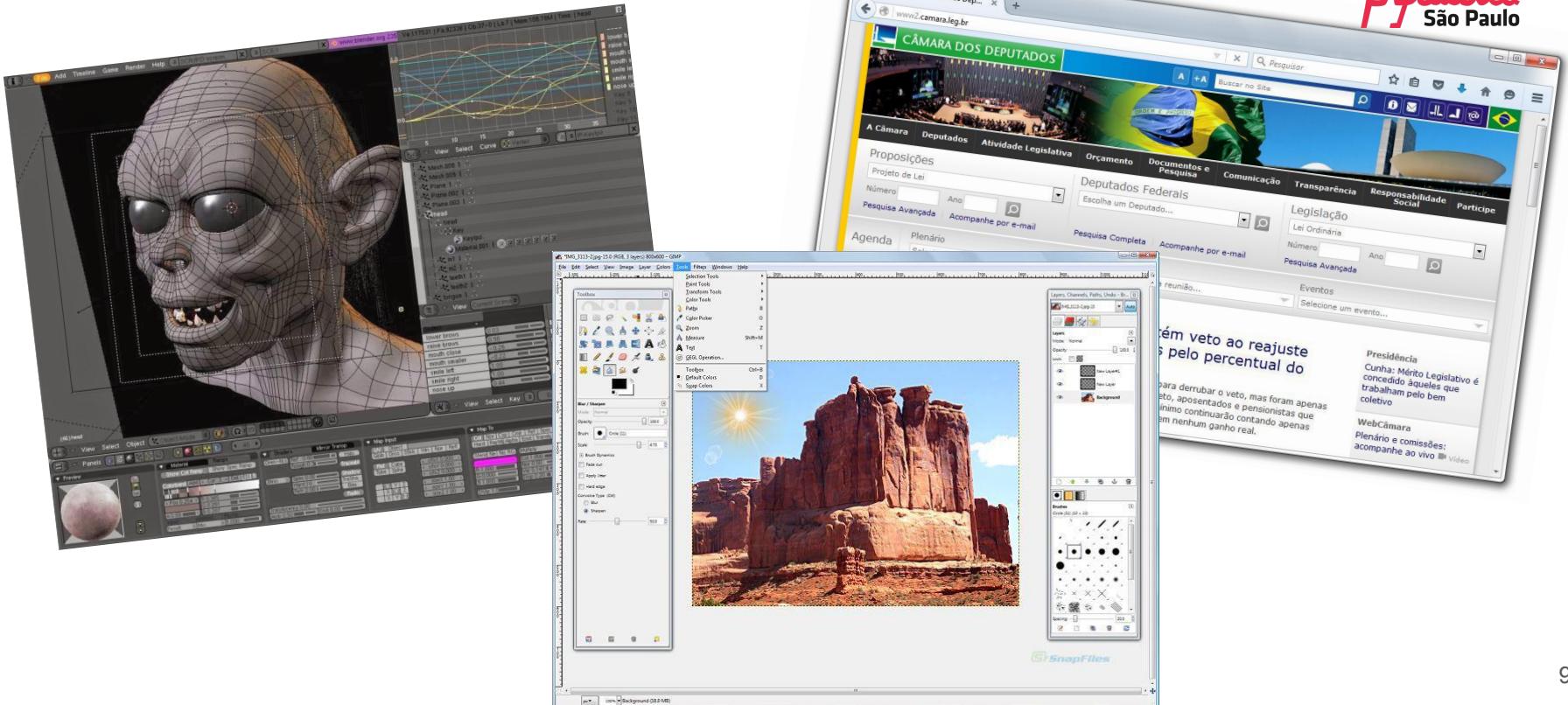
Foi criada em 1989  
por Guido Van Rossum



O nome Python  
foi inspirado no  
seriado britânico  
Monty Python

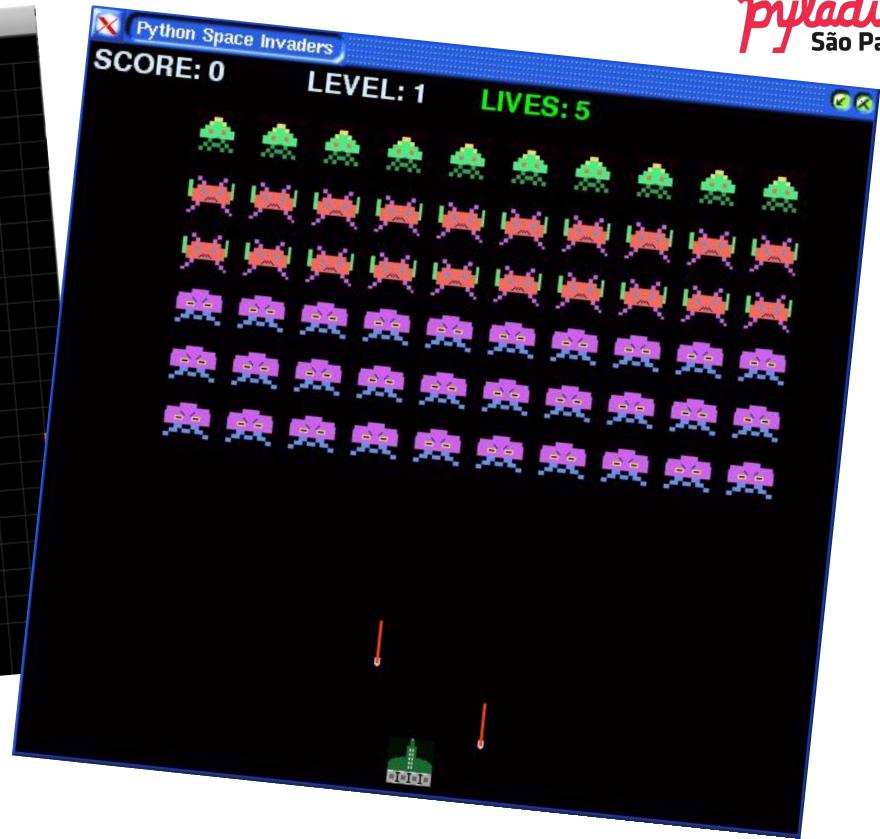
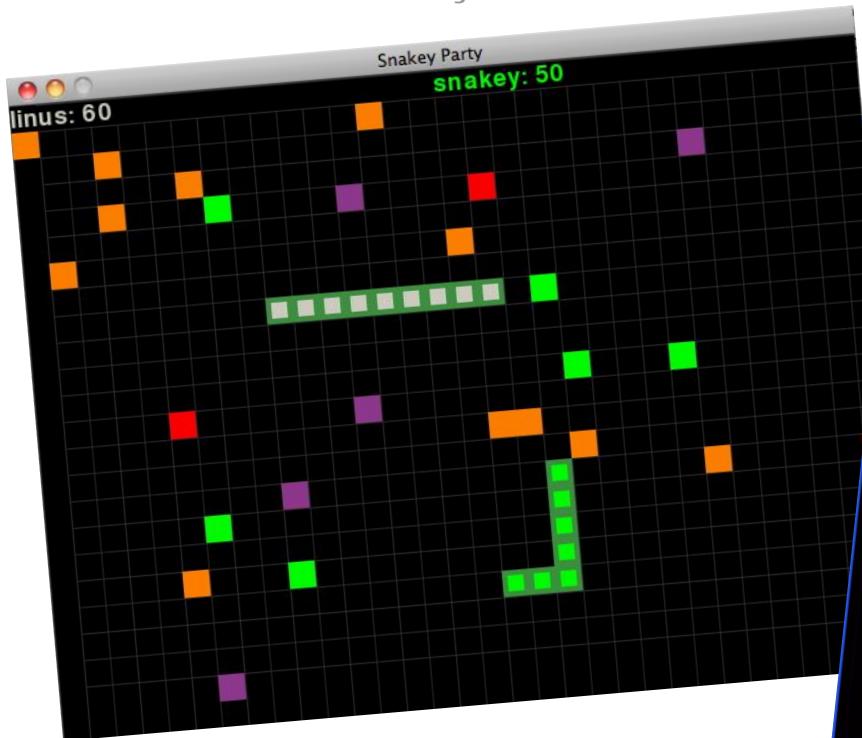
# O QUE É E ONDE SE USA PYTHON

## EXEMPLOS DE UTILIZAÇÃO



# O QUE É E ONDE SE USA PYTHON

## EXEMPLOS DE UTILIZAÇÃO



# 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 }
```

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.
```

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 }
9 }
```

Python

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

**"Simples é melhor  
que complexo"  
(Tim Peters)**

# PROGRAMANDO EM PYTHON

## ESCREVER NA TELA



A linha de comando é indicada pelas setas **>>>**  
É nela que você deve dar os comandos.

Sua primeira tarefa é executar um **print**, o comando  
para imprimir uma mensagem na tela.

O resultado da execução, deve sair logo abaixo, dessa forma:

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

# PROGRAMANDO EM PYTHON

## VARIÁVEIS



De forma simplificada, variável é um nome que associamos a um valor ou expressão (informação).

Somos nós que damos nome às variáveis (em geral, nomes que nos façam lembrar o que ela armazena).

Quando chamamos este nome, o computador retorna o que está guardado naquela variável.

Por exemplo:

```
>>> doce = 'brigadeiro'
```



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

```
id(<variável>)
```

- **Exemplo:**

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

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

- **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 de palavras-chave:

```
>>> import keyword  
>>> print(keyword.kwlist)
```

# PROGRAMANDO EM PYTHON

## VARIÁVEIS



- **Sintaxe:**

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

- **Exemplo:**

```
comida = 'hamburguer'
```

```
nota = 8.7
```



- **Tipos de variáveis:**

Uma variável possui tipos diferentes que as caracterizam.

- **Numéricos:** armazenam números. São: inteiros (`int`), de ponto flutuante (`float`) e 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

```
>>> pasteis = 2
>>> coxinhas = 0
>>> amigos_no_face = 523
```

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

```
>>> altura = 2.0
>>> preço = 4.5
>>> numero = 4E-2
```

use ponto ao  
invés de vírgula

# PROGRAMANDO EM PYTHON

## VARIÁVEIS



- **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
```



- **Literais:**

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

```
>>> heroína = 'Mulher Maravilha'  
>>> telefone = '2145-8970'  
>>> email = 'saopaulo@pyladies.com'
```

- **Lógicos:**

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

```
>>> aprovada = True
```



```
>>> aprovada = False
```



Tá, como eu sei se o Python entendeu que a variável aprovada 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(aprovada)
<type 'bool'>
```

**AGORA É  
COM VOCÊ**

**1)** Teste as seguintes variáveis:

```
a = 2
nota = 8.7
nome = 'Aninha'
m = True
```

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

O que acontece? Por que dava certo antes?

## RESPOSTAS

1) a: int;

```
>>> type(a)
```

nota: float;

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

nome: string;

```
>>> type(nota)
```

m: bool

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

```
>>> type(nome)
```

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

```
>>> type(m)
```

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

## 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 'bool'>`

- **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:**

```
>>> nome1, nome2, nome3 = 'Ana', 'Luisa', 'Julia'  
>>> nome1, nome2, nome3  
('Ana', 'Luisa', 'Julia')
```

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

Podemos transformar um tipo de variável em outro usando os comandos:  
`int( )` ou `float( )` ou `str( )`

# PROGRAMANDO EM PYTHON

## VARIÁVEIS



- Exemplos:

```
>>> a = 2
>>> type(a)
<class 'int'>
>>> b = float(a)
>>> type(b)
<class 'float'>
>>> b
2.0
>>> c = str(a)
>>> c
'2'
>>> d = int(c)
>>> d
2
>>> type(d)
<class 'int'>
```

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

- Operadores numéricos básicos:
  - Adição:** `+`
  - Subtração:** `-`
  - Divisão:** `/`
  - Multiplicação:** `*`
  - Potenciação:** `**`
  - Resto de uma divisão:** `%`

**Lembre-se do**

**PEMDAS**

**Parênteses**

**Exponentes**

**Multiplicação**

**Divisão**

**Adição**

**Subtração**

# PROGRAMANDO EM PYTHON

## OPERADORES



- **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
```

- **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 (todas as condições precisam ser satisfeitas)

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

- Exemplos:

```
>>> a = 2          >>> a != b
                  True
>>> b = 3
                  False
>>> a > b        >>> a > b and b > a
                  True           False
False
>>> b > a        >>> a > b or b > a
                  True
True
>>> a == b        >>> not (a != b) == False
                  True
False
```

# PROGRAMANDO EM PYTHON

## OPERADORES



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    |

# PROGRAMANDO EM PYTHON

## OPERADORES

**AGORA É  
COM VOCÊ**

Defina as variáveis como

```
a = 2  
b = 3  
c = 2.0  
d = '2.0'
```

e faça as operações a seguir:

```
a + b  
b ** a  
a + c  
a + d
```

### EXTRA

Teste as condições:

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



## RESPOSTAS

```
>>> a = 2  
>>> b = 3  
>>> c = 2.0  
>>> d = '2.0'  
>>> a + b  
5  
>>> b ** a  
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 DOS EXTRAS

```
>>> 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
```

# PROGRAMANDO EM PYTHON

## OPERAÇÕES COM STRINGS



- Exemplo 2:

```
>>> nome = 'Penny...'  
>>> print(nome * 5)  
>>> 'Penny...Penny...Penny...Penny...Penny...'
```



- **Uppercase:** toda a string em maiúscula

- **Sintaxe:**

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

- **Exemplo:**

```
>>> 'Batata frita'.upper()  
'BATATA FRITA'
```



- **Lowercase:** toda a string em letra minúscula



- **Sintaxe:**

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

- **Exemplo:**

```
>>> 'Coxinha'.lower()  
'coxinha'
```

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

- **Sintaxe:**

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

- **Exemplo 1:**

```
>>> 'a' + 'b'  
'ab'  
>>> 'PyLadies ' + 'São Paulo'  
'PyLadies São Paulo'
```

# PROGRAMANDO EM PYTHON

## MAIS SOBRE STRINGS



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

- **Sintaxe:**

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

- **Exemplo:**

```
>>> 'pão de queijo'.capitalize()  
'Pão de queijo'
```





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

- **Sintaxe:**

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

- **Exemplo:**

```
>>> 'bolacha ou biscoito'.title()  
'Bolacha Ou Biscoito'
```

- **Começa com (startswith)**

ou

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

- **Sintaxe:**

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

ou

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

# PROGRAMANDO EM PYTHON

## MAIS SOBRE STRINGS



- Exemplo:

```
>>> saudacao = 'Vida longa e próspera!'
```

```
>>> saudacao.startswith('V')
```

```
True
```

```
>>> saudacao.startswith('v')
```

```
False
```

```
>>> saudacao.endswith('!')
```

```
True
```



# PROGRAMANDO EM PYTHON

## MAIS SOBRE STRINGS



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

<variável tipo string>[número]



- **Exemplo:**

```
>>> serie = 'DOCTOR WHO'  
>>> serie[0]  
'D'  
>>> serie[2]  
'C'  
>>> serie[6]  
' '  
>>> serie[-1]  
'O'
```

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

# PROGRAMANDO EM PYTHON

## 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:**

```
>>> serie[6:9]
```

```
' WH'
```

```
>>> serie[4:10]
```

```
'OR WHO'
```

```
>>> serie[7: ]
```

```
'WHO'
```

```
>>> serie[-1: ]
```

```
'O'
```



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

# PROGRAMANDO EM PYTHON

## MAIS SOBRE STRINGS



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

- **Sintaxe:**

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

- **Exemplo:**

```
>>> serie[4:10:2]
```

```
'O H'
```

```
>>> serie[::-1]
```

```
'OHW ROTCOD'
```

Passo é o número  
de strings que  
queremos pular



# PROGRAMANDO EM PYTHON

## MAIS SOBRE STRINGS

**AGORA É  
COM VOCÊ**

Defina a variável

expectativa = 'stranger things'  
e com apenas um comando sobre a variável:

- 1)** Escreva STRANGER THINGS
- 2)** Escreva Stranger Things
- 3)** Inverta stranger things



# PROGRAMANDO EM PYTHON

## MAIS SOBRE STRINGS



## RESPOSTAS

- 1)**

```
>>> expectativa.upper()  
'STRANGER THINGS'
```
- 2)**

```
>>> expectativa.title()  
'Stranger Things'
```
- 3)**

```
>>> expectativa[::-1]  
'sgnicht regnarts'
```

# PROGRAMANDO EM PYTHON

## MAIS SOBRE STRINGS

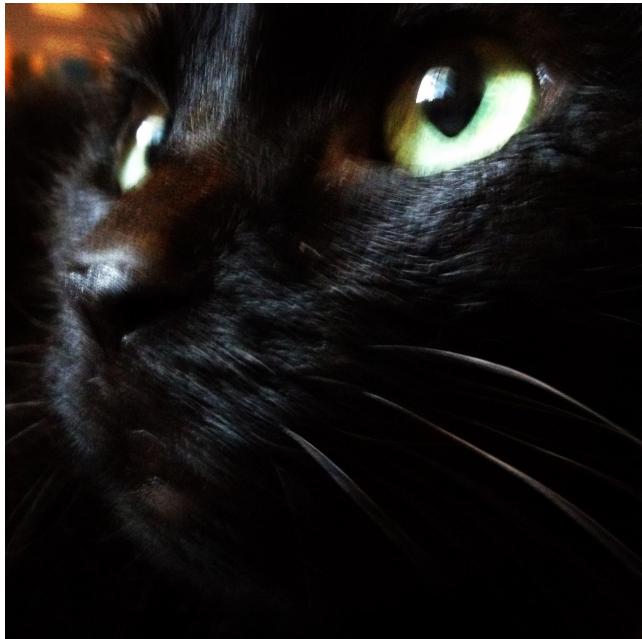
### **DESAFIO**

Via de regra, os índices de uma string são imutáveis.

Crie uma variável e atribua o valor 'Gatinho'.

Usando as operações com strings que você aprendeu, faça a letra 'o' na variável ser transformada em 'a' (sem redefinir a variável).





### POSSÍVEL RESPOSTA

```
>>> c = 'Gatinho'  
  
>>> c[:6] + 'a'  
'Gatinha'
```

# PROGRAMANDO EM PYTHON

## MAIS SOBRE STRINGS



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

- **Sintaxe:**

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

- **Exemplo:**

```
>>> novaserie = 'Star Trek Discovery'
```

```
>>> len(novaserie)
```

```
19
```



- **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>)
```

# PROGRAMANDO EM PYTHON

## MAIS SOBRE STRINGS



- **Exemplo:**

```
>>> abertura = 'Espaço: a fronteira final... audaciosamente  
indo onde ninguém jamais esteve.'
```

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

14

retornou a posição da primeira  
ocorrência a partir do início

```
>>> abertura.find('a', 13)
```

18

retorna a posição

da primeira

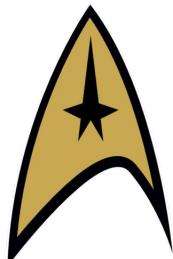
ocorrência a partir

da posição 13

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

-1

retorna -1 se não encontra o  
caracter procurado



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

- **Sintaxe:**

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

- **Exemplo:**

```
>>> spock = 'Fascinante, capitão Kirk'  
>>> spock.replace('Fascinante' , 'Incrível')  
'Incrível, capitão Kirk'
```

- **Listas:** permitem armazenar várias informações diferentes (número, string, lógico) em uma mesma variável.

- **Sintaxe:**

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

- **Exemplo:**

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

- **Fatiando listas** - para obter apenas um trecho da lista:
- **Exemplo 1:**

```
>>> meubicho = ['Gato', 9, True]  
>>> meubicho[0]  
'Gato'
```

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

- Exemplo 2:

```
>>> meubicho = ['Gato', 9, True, ['preto', 'branco']]  
>>> meubicho[3]  
['preto', 'branco']
```

O elemento 3 da lista  
meubicho é uma outra lista

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

# PROGRAMANDO EM PYTHON

## LISTAS



- **Comando Append:** acrescenta dados ao final de uma lista.

- **Sintaxe:**

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

- **Exemplo:**

```
>>> nomedaserie = ['Gotham', 'A', 'Dark']
>>> nomedaserie.append('Knight')
>>> print(nomedaserie)
['Gotham', 'A', 'Dark', 'Knight']
```



# PROGRAMANDO EM PYTHON

## LISTAS



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

*Funciona apenas com strings*

- **Sintaxe:**

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

- **Exemplo:**

```
>>> herois = ['Flash', 'Arrow', 'Supergirl']
>>> ' e '.join(herois)
'Flash e Arrow e Supergirl'
```



- **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:** 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
```

# PROGRAMANDO EM PYTHON

## TUPLAS != LISTAS



- Exemplo:

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

```
>>> a
```

```
(3,5,8)
```

```
>>> b = 3,5,8
```

```
>>> b
```

```
(3,5,8)
```

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

```
>>> True
```

```
>>> type(b)
```

```
<class 'tuple'>
```

- **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:**

```
<variável> = (<elemento>, )
```

- Exemplos:

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

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

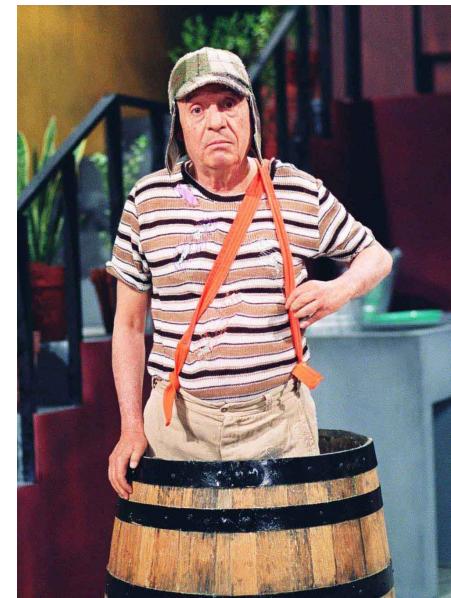
os exemplos parecem iguais,  
mas são reconhecidos de  
forma diferente pelo Python

**AGORA É  
COM VOCÊ**

Dada a variável

`chaves = 'Eu prefiro morrer do que perder a vida.'`

- 1)** Qual o tamanho da string?
- 2)** Verifique se começa com 'p'?
- 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'.



## RESPOSTAS

1) `>>> len(chaves)`  
39

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

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

4) `>>> chaves.find(' , ')`  
-1

### RESPOSTAS (continua)

- 5) 

```
>>> chaves.replace('. ', '!!')
'Eu prefiro morrer do que perder a vida!'
```
- 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']
```

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

Para inteiro:

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

Para float:

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

- Exemplos (continua):

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

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

- Sintaxe:

```
dir(<variável>)
```

- Exemplo:

```
>>> dir('Felino')
['__add__', '__class__', '__contains__', '__delattr__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattribute__', '__getitem__', '__getnewargs__',
 '__gt__', '__hash__', '__init__', '__init_subclass__',
 '__iter__', '__le__', '__len__', '__lt__', '__mod__',
 '__mul__', '__ne__', '__new__', '__reduce__']
```

- **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

    Return a copy of S converted to uppercase.
```

- **Passos para a entrada de dados:**

- O ideal é estar no modo edição do IDLE para poder salvar o que vai escrever
- Criar uma variável e fazer a entrada dada pelo usuário via teclado coincidir com ela. É possível exibir 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:**

```
nome = input('Qual o seu nome? ')  
Qual o seu nome? |
```

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 1:**

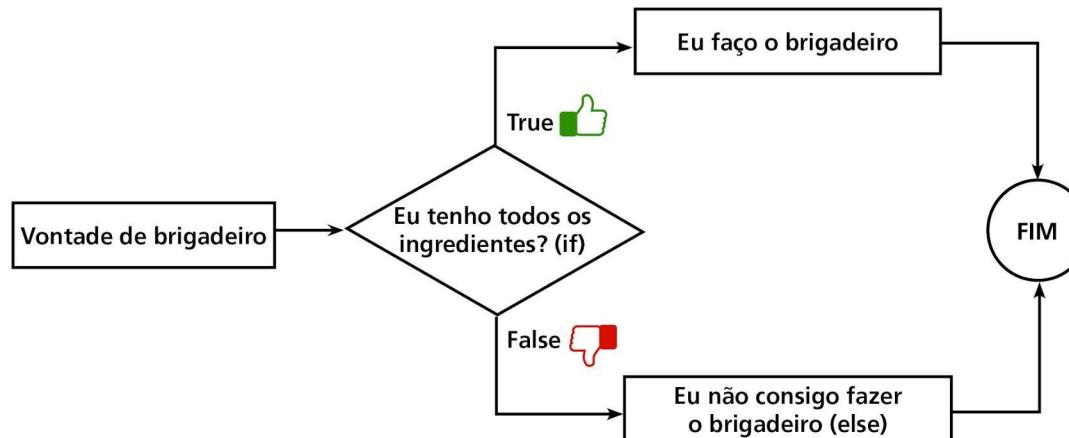
```
idade = int(input('Quantos anos você tem? '))  
Quantos anos você tem? |
```

- **Exemplo 2:**

```
altura = float(input('Qual sua altura em metros? '))  
Qual sua altura em metros? |
```

### Vamos fazer um brigadeiro

- se (**if**) eu tiver todos os ingredientes, eu começo a receita e continuo o processo
- caso contrário (**else**), eu saio para comprar o que falta



- **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:

```
ingredientes = 'sim'

if ingredientes == 'sim':
    print('Colocar tudo na panela e mexer bem')

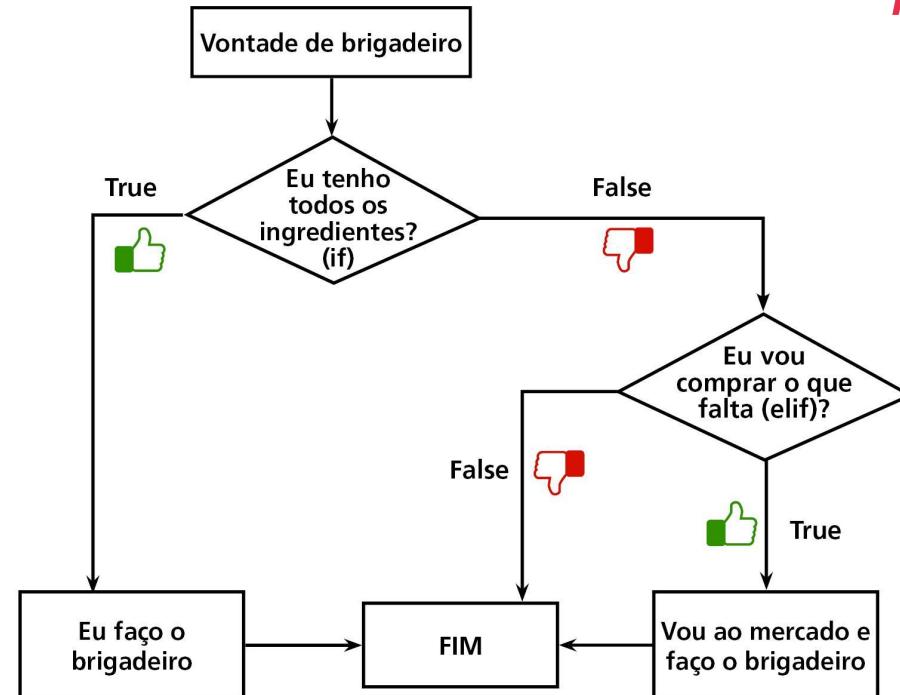
else:
    print('Vou ficar com vontade de brigadeiro')

>>>
Colocar tudo na panela e mexer bem
>>>
```

# PROGRAMANDO EM PYTHON

## CONDICIONAIS

- Mas eu posso ir ao mercado e comprar o que falta e continuo o processo: **elif**



# PROGRAMANDO EM PYTHON

## CONDICIONAIS



- **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:

```
ingredientes = 'não'  
vontade = 'muita'  
if ingredientes == 'sim':  
    print('Colocar tudo na panela e mexer bem')  
elif vontade == 'muita':  
    print('Compro o que falta e faço o brigadeiro')  
else:  
    print('Vou ficar com vontade de brigadeiro')  
=> Compro o que falta e faço o brigadeiro  
=> |
```

# PROGRAMANDO EM PYTHON

## CONDICIONAIS

**AGORA É  
COM VOCÊ**



Crie um código para uma outra pessoa adivinhar um número de 1 a 10.

Escolha o número que ela deve acertar.

Peça para ela digitar um número de 1 a 10.

Avise se ela acertou, se o número que ela chutou era mais baixo ou mais alto e mostre o número certo em qualquer das situações.

## RESPOSTA

```
print('Olá! Vamos brincar de adivinha.')
chute = int(input('Escolha um número de 1 a 10: '))
acerto = 4
if chute == acerto:
    print('Uau! Você acertou!')
elif chute < acerto:
    print('O número é mais alto.')
else:
    print('O número é mais baixo.')
print('O número correto é {}'.format(acerto))
```

### OUTRA RESPOSTA POSSÍVEL

```
print('Olá! Vamos brincar de adivinha.')
chute = int(input('Escolha um número de 1 a 10: '))
acerto = 4

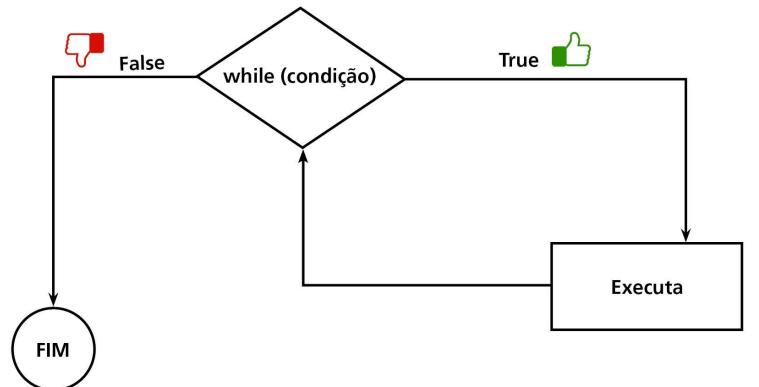
if chute == acerto:
    print('Uau! Você acertou!')
elif chute < acerto:
    print('O número é mais alto. O número correto é {}'.format(acerto))
else:
    print('O número é mais baixo. O número correto é {}'.format(acerto))
```

- **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: ')) 1ª vez
total = total + num
num = int(input('Entre com um número: ')) 2ª vez
total = total + num
num = int(input('Entre com um número: ')) 3ª vez
total = total + num
```

- **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>
```

# PROGRAMANDO EM PYTHON

## WHILE (ENQUANTO)



- Exemplo: 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)
```

contador: soma  
um valor fixo

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

# PROGRAMANDO EM PYTHON

## WHILE (ENQUANTO)



**AGORA É  
COM VOCÊ**

Tomei lanche todos os dias durante o curso.

Faça um código, usando contadores e acumuladores, que calcule quanto gastei durante o evento.

*Dica 1: ninguém gosta de comer a mesma coisa todo dia, então cada dia é um valor diferente (nham!).*

*Dica 2: o valor pode conter centavos.*

## RESPOSTA

```
dia = 1
total = 0
while dia <= 4:
    gasto = float(input('Gastei: R$ '))
    total = total + gasto
    dia = dia + 1
print('Gastei durante o curso R$ ' ,total)
```

# PROGRAMANDO EM PYTHON

## 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**.

Por exemplo, vou fazer compras, mas não sei quantos produtos vou comprar,  
então não há como colocar um marcador.  
Diferente do número de lanches que como na semana  
que já são predeterminados.

# PROGRAMANDO EM PYTHON

WHILE (ENQUANTO): interrompendo a repetição



- Exemplo: soma de valores (com centavos) até ser digitado zero

```
soma = 0
while True:
    valor = float(input('Digite o valor gasto: '))
    if valor == 0:
        break
    soma = soma + valor
print('Soma: {}'.format(soma))
```

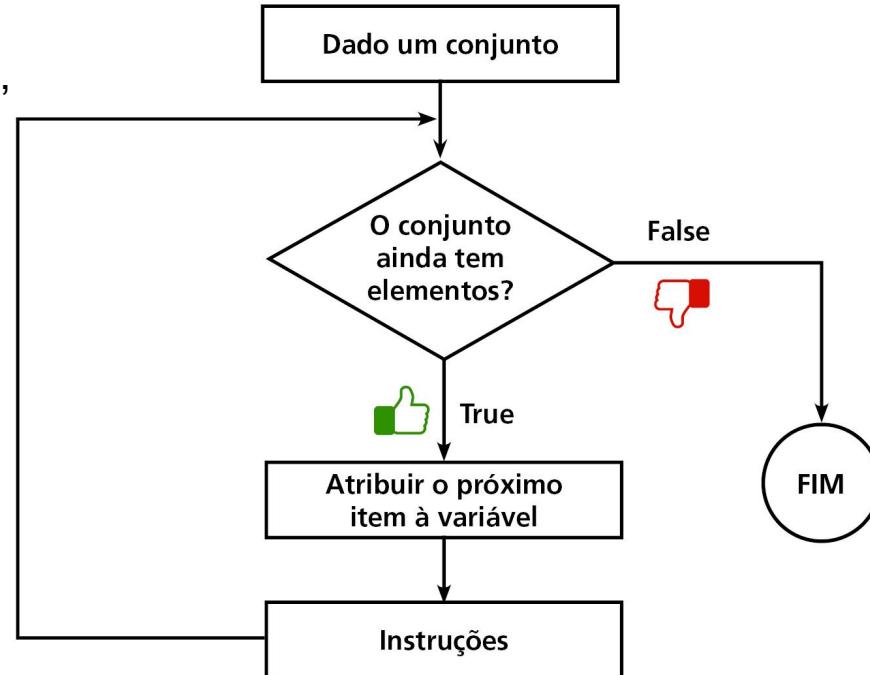
Digite o número: 3.5  
Digite o número: 7.3  
Digite o número: 13.7  
Digite o número: 76.4  
Digite o número: 93  
Digite o número: 10.2  
Digite o número: 0  
Soma: 204.1  
>>>

# PROGRAMANDO EM PYTHON

## FOR (PARA)



O comando **for** opera 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:**

```
alunas = ['Ana', 'Beatriz', 'Caroline', 'Denise', 'Elida',  
          'Fernanda', 'Glaucia']  
for aluna in alunas:  
    if aluna.startswith('C'):  
        print(aluna)
```

# PROGRAMANDO EM PYTHON

## FOR (PARA)



- **Exemplo 2:** somar todos os números pares da lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
soma_par = 0
for numero in lista:
    if (numero % 2) == 0:
        soma_par = soma_par + numero
print('A soma dos pares de toda lista é {}'.format(soma_par))
```

# PROGRAMANDO EM PYTHON

## FOR (PARA)



**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.

# PROGRAMANDO EM PYTHON

## FOR (PARA)



## RESPOSTA

```
linguagens = ['Java', 'JavaScript', 'PHP', 'C', 'Python']
for linguagem in linguagens:
    if linguagem.startswith('P'):
        print(linguagem)
```

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:**

```
def <nome da função> ():
```

*Quando a função não  
recebe parâmetros*

ou

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

*Quando a função  
recebe parâmetros*

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

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

Agora vamos criar a nossa própria função :)

# PROGRAMANDO EM PYTHON

## FUNÇÕES



- Exemplo:

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

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

ou

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

usando como parâmetro strings

- **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 calcular a média de três notas de provas

O cálculo é feito somando as notas e dividindo pela quantidade de nota

$$\frac{\text{nota1} + \text{nota2} + \text{nota3}}{3}$$

## RESPOSTA

```
def calcula_media(nota1, nota2, nota3):  
    return ((nota1 + nota2 + nota3) / 3)  
  
print(calcula_media(7.0, 4.0, 10.0))
```

**AGORA É  
COM VOCÊ**

Faça uma função para calcular o IMC  
(Índice de Massa Corporal).

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

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



## RESPOSTA

```
def imc(peso, altura):  
    return(peso / (altura ** 2))  
  
print(imc(70, 1.62))
```

# DESAFIO

Faça uma função para calcular a média de 3 notas e mostre se a aluna foi aprovada, está de recuperação ou reprovada

| Resultado   | Nota          |
|-------------|---------------|
| Aprovada    | acima de 7,0  |
| Recuperação | 5,0 a 7,0     |
| Reprovada   | abaixo de 5,0 |

## RESPOSTA

```
def calcula_nota():
    nota1 = float(input('Qual foi a sua nota da prova 1: '))
    nota2 = float(input('Qual foi a sua nota da prova 2: '))
    nota3 = float(input('Qual foi a sua nota da prova 3: '))
    media = (nota1 + nota2 + nota3) / 3
    print('A sua média é {:.2f}'.format(media))
    if media < 5.0:
        print('Você está REPROVADA')
    elif 5.0 < media < 7.0:
        print('Você está de RECUPERAÇÃO')
    else:
        print('Você está APROVADA)
calcula_nota()
```

# DESAFIO

Faça uma função para calcular o IMC que pergunte o peso, a altura e mostre se o índice está ideal, acima ou abaixo da tabela da OMS:

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

## RESPOSTA

```
def imc():
    peso = float(input('Entre com seu peso em kg: '))
    altura = float(input('Entre com sua altura em metros: '))
    indice = peso / (altura ** 2)
    print('Seu IMC é {:.2f}'.format(indice))
    if indice < 20:
        print('Você está abaixo do peso')
    elif indice < 24.9:
        print('Você está com o peso normal')
    elif indice < 29.9:
        print('Você está com sobrepeso')
    else:
        print('Você está com obesidade')

imc()
```

## ONDE ESTUDAR ONLINE



- [www.codecademy.com/pt](http://www.codecademy.com/pt)
- [www.sololearn.com/Course/Python](http://www.sololearn.com/Course/Python)
- [pythontutor.com](http://pythontutor.com)
- [www.pycursos.com/python-para-zumbis](http://www.pycursos.com/python-para-zumbis)
- [coursera.org](http://coursera.org)
- [www.urionlinejudge.com.br/judge/pt/login](http://www.urionlinejudge.com.br/judge/pt/login)

# REFERÊNCIAS



- <http://wiki.python.org.br/PrincipiosFuncionais>
- **Curso Python para Zumbis**
- **Curso “An Introduction to Interactive Programming in Python” - Coursera**
- <http://www.peachpit.com/articles/article.aspx?p=1312792&seqNum=6>
- <https://docs.python.org/release/2.3.5/whatsnew/section-slices.html>
- <http://www.bbc.co.uk/education/guides/zqh49j6/revision/3>
- <https://www.youtube.com/watch?v=SYioCdLPmfw>
- [https://pt.wikibooks.org/wiki/Python/Conceitos\\_b%C3%A1sicos/Tipos\\_e\\_operadores](https://pt.wikibooks.org/wiki/Python/Conceitos_b%C3%A1sicos/Tipos_e_operadores)
- <http://www.dcc.ufrj.br/~fabiom/mab225/02tipos.pdf>
- <http://pt.stackoverflow.com/questions/62844/como-se-insere-n%C3%BAmeros-complexosem-python>
- [www.arquivodecodigos.net/principal/dicas\\_truques\\_categoria2.php?linguagem=12&categoria1=1&categoria2=59](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/>

MÃO NA MASSA



**Procurem trabalhar em grupo e  
trocar informações.**

**Tendo dúvidas, estamos à disposição**



PyLadiesSP



PyLadiesSãoPaulo



PyLadiesSP



@PyLadiesSP



PyLadiesSP



saopaulo@pyladies.com



Mulheres que  
amam programar  
e ensinar Python