Introdução à lógica de programação com Python

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.

pyladies
São Paulo

#souPyLadiesSP

O QUE É ALGORITMO

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.

O QUE É ALGORITMO

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

pyladies São Paulo

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

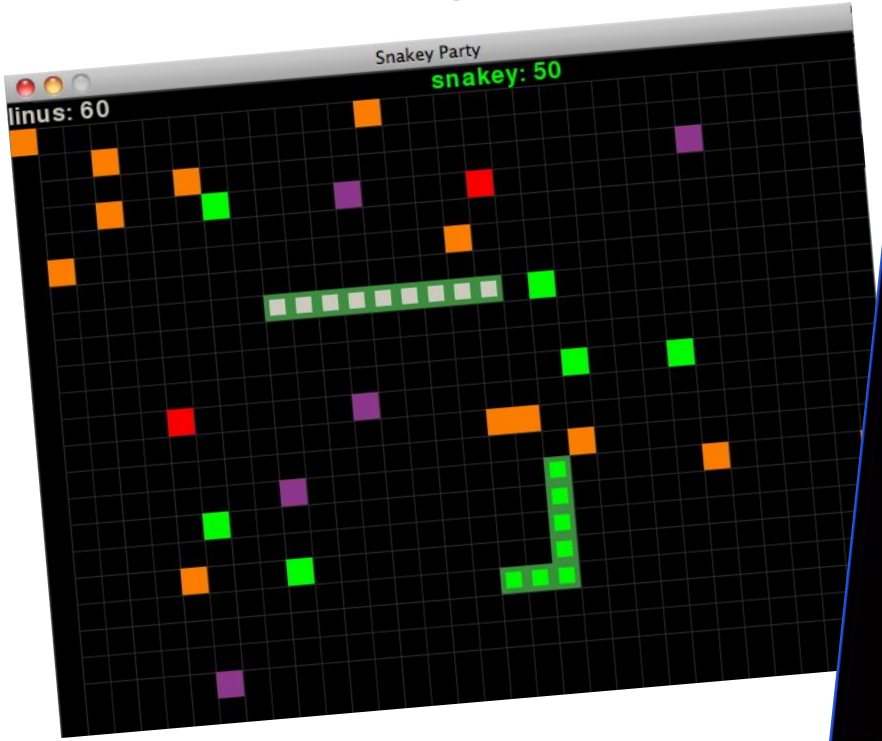
O nome Python
foi inspirado no
seriado britânico
Monty Python

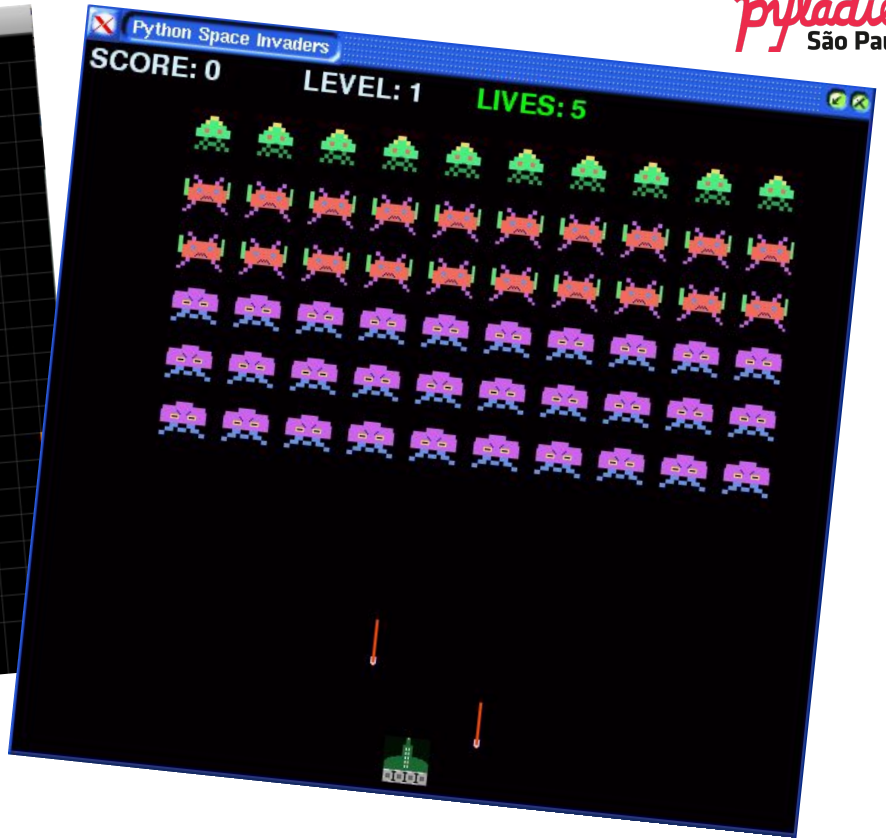
O QUE É E ONDE SE USA PYTHON

EXEMPLOS DE UTILIZAÇÃO

O QUE É E ONDE SE USA PYTHON

EXEMPLOS DE UTILIZAÇÃO





pyladies São Paulo

POR QUE PYTHON?

Exemplo do mesmo programa em várias linguagens:

Java

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

Pascal

```pascal
program HelloWorld(output);
var
    nome: string;
begin
    writeln('Digite seu nome:');
    read(nome);
    writeln('Olá, ', nome);
end.
```

C

```c
#include <stdio.h>
int main()
{
    char nome[200];
    printf("Digite seu nome:");
    scanf("%s", nome);
    printf("Olá, %s\n", nome);
    return 0;
}
```

Python

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

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

POR QUE PYTHON?

pyladies São Paulo

"Simples é melhor que complexo"
(Tim Peters)

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!
```

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'
```

VARIÁVEIS

- **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

VARIÁVEIS

- **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)
```

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: Armazenam caracteres (qualquer um do teclado) ou sequências de caracteres. São strings (string)

  - Lógicos: Armazenam verdadeiro ou falso. São: booleanos (bool)

VARIÁVEIS

- **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

- **Numéricos (continua):**

  - Tipo complexo (**complex**): compreende 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**): compreende todos os caracteres dentro de aspas (simples ou duplas)

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

- **Lógicos:**

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

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

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

VARIÁVEIS

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'>
```

pyladies São Paulo

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?

- **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'>
```

OPERADORES

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

Expoentes

Multiplicação

Divisão

Adição

Subtração

OPERADORES

- **Exemplos:**

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

- **Tipos de divisão:**

```
>>> 10 / 8
1.25
```

(divisão de números inteiros)

```
>>> 10 % 8
2
```

(resto da divisão entre números inteiros)

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 (todas as condições precisam ser satisfeitas)
  Ou: or (apenas uma das condições precisa ser satisfeita)

OPERADORES

- **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
```

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	Flase	False	True	True

OPERADORES

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
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)

OPERAÇÕES COM STRINGS

- **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'
```

OPERAÇÕES COM STRINGS

- **Exemplo 2:**

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

MAIS SOBRE STRINGS

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

- **Sintaxe:**

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

- **Exemplo:**

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

MAIS SOBRE STRINGS

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

- **Sintaxe:**

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

- **Exemplo:**

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

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'
```

MAIS SOBRE STRINGS

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

- **Sintaxe:**

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

- **Exemplo:**

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

MAIS SOBRE STRINGS

- **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>.startswith('elemento que procuro')
```

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

MAIS SOBRE STRINGS

pyladies São Paulo

- **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

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

pyladies São Paulo

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).

MAIS SOBRE STRINGS

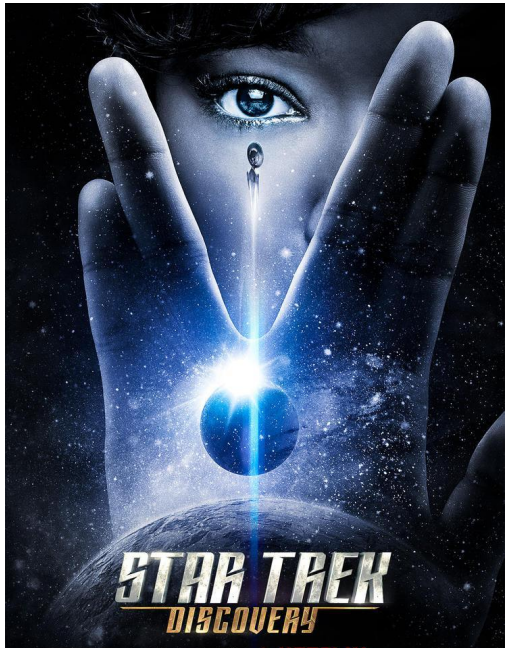
- **Tamanho da string:** conta quantos caracteres tem uma string

- **Sintaxe:**

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

- **Exemplo:**

```
>>> novaserie = 'Star Trek Discovery'
>>> len(novaserie)
19
```



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:**

```
>>> abertura = 'Espaço: a fronteira final… audaciosamente
indo onde ninguém jamais esteve.'
>>> abertura.find('t')
14
>>> abertura.find('a', 13)
18
>>> abertura.find('!')
-1
```

retornou a posição da primeira ocorrência a partir do início

retorna a posição da primeira ocorrência a partir da posição 13

retorna -1 se não encontra o caracter procurado

MAIS SOBRE STRINGS

- **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

- **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

pyladies São Paulo

- **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

- **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']
```

pyladies São Paulo

- **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 != 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:**

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

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

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:**

```
<variável> = (<elemento>,)
```

TUPLAS != LISTAS

- **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

pyladies São Paulo

AGORA É COM VOCÊ

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

1) Qual o tamanho da string?

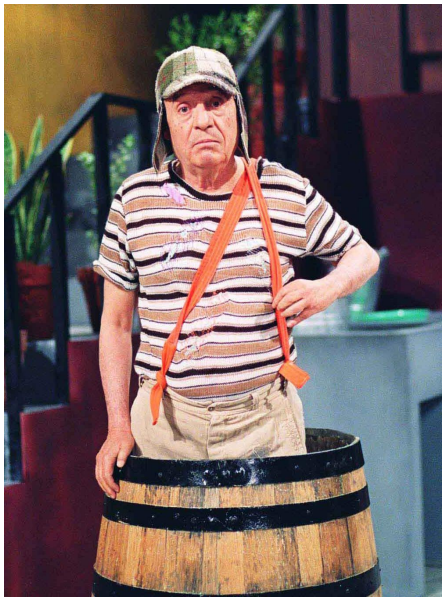
2) Verifique se começa com 'p'?

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'.



- **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 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:**

```
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).

ENTRADAS

- **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:**

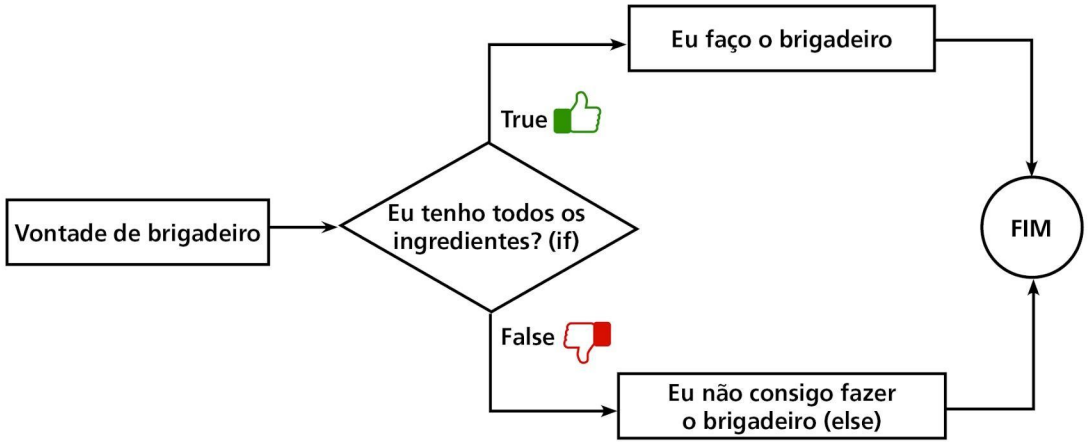
```
idade = int(input('Quantos anos você tem? '))
Quantos anos você tem? |
```

- **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 contínuo 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>
```

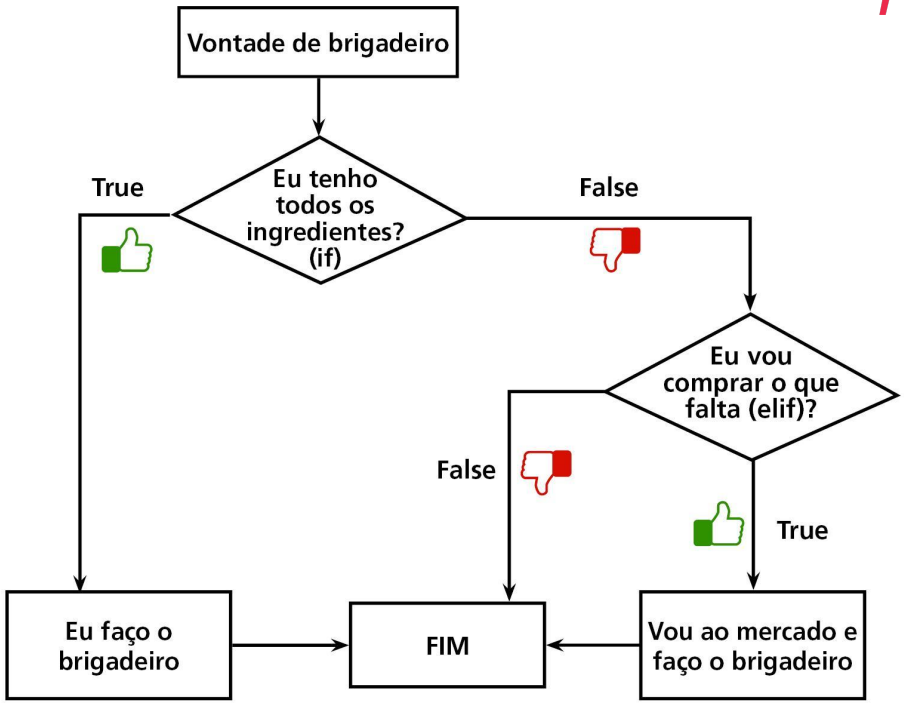
pyladies São Paulo

- **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
>>>
```

CONDICIONAIS

- Mas eu posso ir ao mercado e comprar o que falta 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>
```

pyladies São Paulo

- **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
>>> |
```

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.

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.

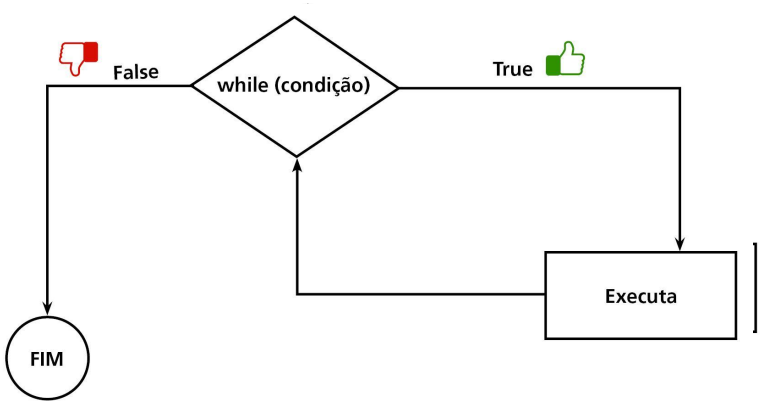
- **Acumuladores:** Contadores que incrementam valores diferentes.

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

```
total = 0
num = int(input('Entre com um número: ')) 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 (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:** 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)
```

contador: soma um valor fixo

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

pyladies São Paulo

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.

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.

WHILE (ENQUANTO): interrompendo a repetição

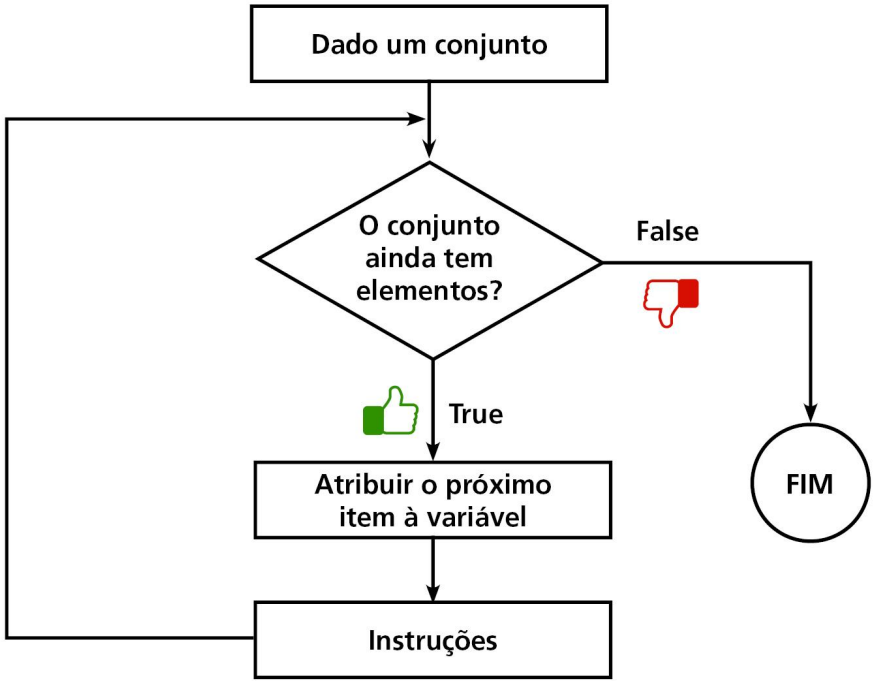
- **Exemplo:** soma de valores (com centavos) até ser digitado zero

```
soma = 0
while True:
    x = float(input('Digite o valor gasto: '))
    if x == 0:
        break
    soma = soma + x
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
>>>
```

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.



FOR (PARA)

- **Sintaxe:**

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

- **Exemplo 1:**

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

- **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]
somaPar = 0
for numero in lista:
    if (numero % 2) == 0:
        somaPar = somaPar + numero
print('A soma dos pares de toda lista é {}' .format(somaPar))
```

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.

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 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 :)

- **Exemplo:**

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

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

chamada da função soma

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
>>>
```

pyladies São Paulo

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:

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

pyladies São Paulo

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

ONDE ESTUDAR ONLINE

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

REFERÊNCIAS

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

REFERÊNCIAS

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

MÃO NA MASSA

pyladies São Paulo

Procurem trabalhar em grupo e trocar informações.

Tendo dúvidas, estamos à disposição

PyLadiesSP

PyLadiesSãoPaulo

PyLadiesSP

@PyLadiesSP

Meetup PyLadiesSP

saopaulo@pyladies.com

Mulheres que amam programar e ensinar Python