Programação com Python

PESSOAS > TECNOLOGIA

Quem sou eu?

- Bacharelanda de Ciência da Computação - UFPI
- Estagiária Full Stack - Quiploy
- Técnica em Desenvolvimento de Software - IFPI
- Desenvolvedora no projeto Open Source - Colaboradados

Como me encontrar?

anapaula.ds.mendes@gmail.com

@anapauladsmendes

/anapauladsmendes

@ananoterminal

Zen do Python, por Tim Peters

Bonito é melhor que feio.

Explícito é melhor que implícito.

Simples é melhor que complexo.

Complexo é melhor que complicado.

Linear é melhor do que aninhado.

Esparso é melhor que denso.

Legibilidade conta.

Casos especiais não são especiais o bastante para quebrar as regras.

Ainda que praticidade vença a pureza.

Erros nunca devem passar silenciosamente.

A menos que sejam explicitamente silenciados.

Diante da ambiguidade, recuse a tentação de adivinhar.

Deveria haver um — e preferencialmente só um — modo óbvio para fazer algo.

Embora esse modo possa não ser óbvio a princípio a menos que você seja holandês.

Agora é melhor que nunca.

Embora nunca frequentemente seja melhor que já.

Se a implementação é difícil de explicar, é uma má idéia.

Se a implementação é fácil de explicar, pode ser uma boa idéia.

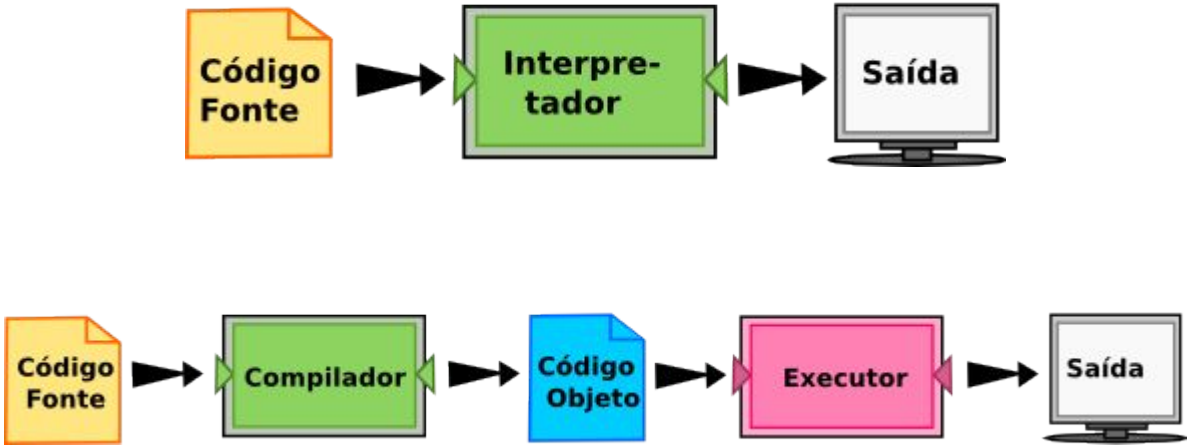
Namespaces são uma grande ideia — vamos ter mais dessas!

Vamos aos conceitos

A linguagem de programação Python

- ➔ Criada pelo Guido Van Rossum;
- ➔ Open Source;
- ➔ Desenvolvimento Web;
- ➔ Ciência de Dados;
- ➔ Linguagem de alto nível;
- ➔ Interpretada;

Interpretadores x Compiladores



Depuração

É encontrar e corrigir erros no programa. Também conhecido como debugging.

Embora possa ser frustrante, depurar é uma das partes intelectualmente mais ricas, desafiadoras e interessantes da programação.

Erros

De Sintaxe

Se refere à estrutura de um programa e as regras sobre essa estrutura.

De Semântica

Se refere a lógica do programa.

De Execução

Também conhecido como runtime errors ou exceções e só aparecem quando executa o programa.

Variáveis e tipos de dados

- ➔ int
- ➔ float
- ➔ str
- ➔ bool
- ➔ type( )

- ➔ int( )
- ➔ float( )
- ➔ str( )
- ➔ bool( )

Palavras reservadas

and - as - assert - break - class - continue - def - del - elif - else - except - exec - finally - for - from - global - if - import - in - is - lambda - nonlocal - not - or - pass - raise - return - try - while - with - yield - True - False - None

Comando

É uma instrução que o interpretador Python pode executar.

Ex.: while, if, for, import.

Expressão

É uma combinação de valores, variáveis, operadores e chamadas de funções e necessitam ser calculadas.

Ex.: x = x + 1

Operadores aritméticos

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
**	Potenciação
//	Divisão Inteira
%	Resto

Operadores relacionais

>	Maior que
>=	Maior igual que
<	Menor que
<=	Menor igual que
==	Igual
!=	Diferente

Operadores lógicos

and	Conjunção
or	União
not	Negação

Input

- nome = input(“Digite seu nome: ”)
- idade = int(input(“Digite sua idade: ”))
- altura = float(input(“Digite sua altura: ”))

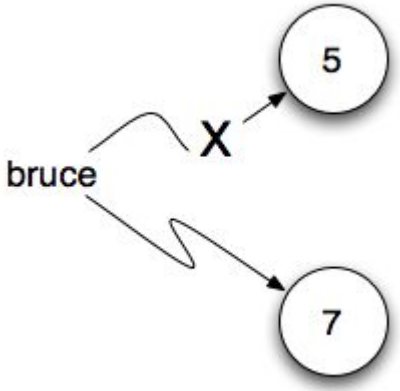
Print

- ➔ print(“Hello World”)
- ➔ print(nome)
- ➔ print(idade)
- ➔ print(altura)

Precedência

- ➔ Parênteses têm a mais alta precedência e podem ser usados para forçar que uma expressão seja calculada na ordem que você deseja.
- ➔ Exponenciação tem a segunda precedência mais alta.
- ➔ Multiplicação e ambas as divisões têm a mesma precedência, que são mais altas que adição e subtração, que também têm a mesma precedência.

Reatribuição de valores



Atualização de variáveis

- ➔ Incrementação.
- ➔ Decrementação.

Hora de praticar!

Você vai precisar criar um cadastro para estudantes.
Solicite sua matricula, nome, idade, telefone, e-mail e curso.
Então imprima os dados de estudante.

Listas

Valores em uma lista

- Lista vazia:
  - vazia = [ ]
- Lista com itens:
  - frutas = [‘banana’, ‘melancia’, ‘abacate’]

Métodos de lista

- Para adicionar itens:
  - frutas.append(‘uva’)
- Para ordenar itens:
  - frutas.sort( )
- Tamanho da lista:
  - len(frutas)
- Acessar itens:
  - frutas[n]

Operações com lista

- Concatenação de listas:
  - [1, 2, 3] + [4, 5, 6]
- Repetição de listas:
  - [1, 2, 3] * 3

Operações com lista

- Fatiamento de listas:
  - lista = [0, 1, 2, 3, 4, 5]
  - lista[:4]
  - lista[3:]
  - lista[1:3]
- Remoção em listas:
  - del lista[n]

Hora de praticar!

Escreva uma lista de frutas com 3 frutas.
Adicione “kiwi” nesta lista.
Escreva uma lista de vegetais com 5 items.
Apague o item do meio.
Junte as duas listas em uma lista só, chamada “feira”.

Tuplas

Valores em uma tupla

- Tupla vazia
  - vazia = ( )
- Tupla com um item
  - tupla = 1,
  - tupla = (1,)
- Tupla com items
  - tupla = 1, 2, 3, 4, 5
  - tupla = (1, 2, 3, 4, 5)

Métodos de tupla

- ➔ Acessa items como na lista
- ➔ O fatiamento é como na lista
- ➔ É imutável, portanto não é possível reatribuir valores
- ➔ É possível adicionar items concatenando tuplas

Utilidade púlica

➔ Sem tupla

```
a = 1
b = 2

temp = a
a = b
b = a
```

➔ Com tupla

```
a = 1
b = 2

a, b = b, a
```

Hora de praticar!

Escreva uma tupla com 5 números ímpares.
Escreva uma tupla com 5 números pares.
Adicione os números pares junto dos ímpares.

Dicionários

Valores em um dicionário

- Dicionário vazio
  - vazio = { }
- Dicionário com items
  - estudante = {‘nome’: ‘Sam’, ‘idade’: 20}

Métodos de dicionário

- Para adicionar items
  - estudante[‘matricula’] = ‘2019000001’
- Para ordenar items
  - sorted(estudante.items( ))
- Tamanho do dicionário
  - len(estudante)
- Para acessar items
  - estudante[‘key’]

Métodos de dicionário

- Acessar as chaves
  - estudante.keys( )
- Acessar os valores
  - estudante.values( )
- Acessar os items
  - estudante.items( )
- Remoção de items
  - del estudante[“key”]

Hora de praticar!

Agora você vai fazer um cadastro de estudante adicionando os dados de estudante de acordo com cada chave. Os campos são: nome, idade, matricula, e-mail, curso.
Organize o dicionário por items.
As entradas devem ser dadas pelo usuário.

Condicionais

Tabela verdade

E		
Falso	Falso	Falso
Falso	Verdadeiro	Falso
Verdadeiro	Falso	Falso
Verdadeiro	Verdadeiro	Verdadeiro

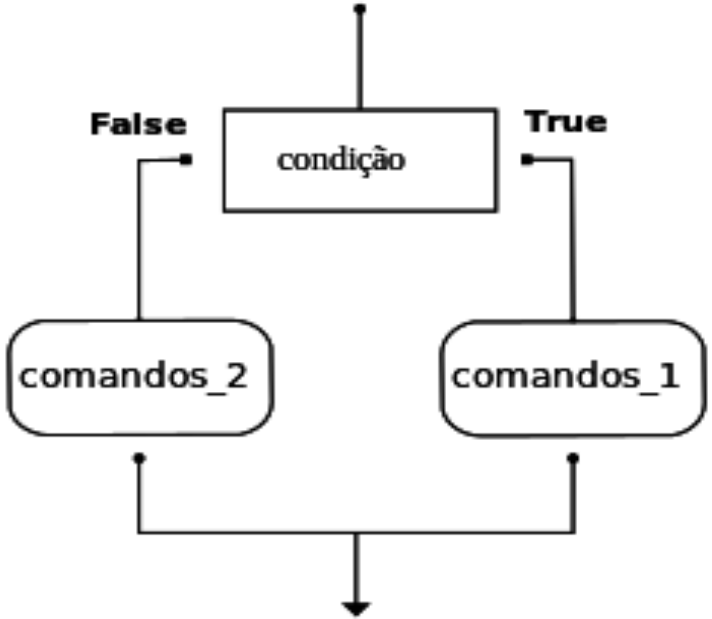
OU		
Falso	Falso	Falso
Falso	Verdadeiro	Verdadeiro
Verdadeiro	Falso	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro

Observações antes de começar

- ➔ Valores booleanos
  - ◆ True
  - ◆ False
- ➔ Expressão booleana
  - ◆ True and True
- ➔ Utiliza dos operadores lógicos e relacionais
- ➔ Precedência importa

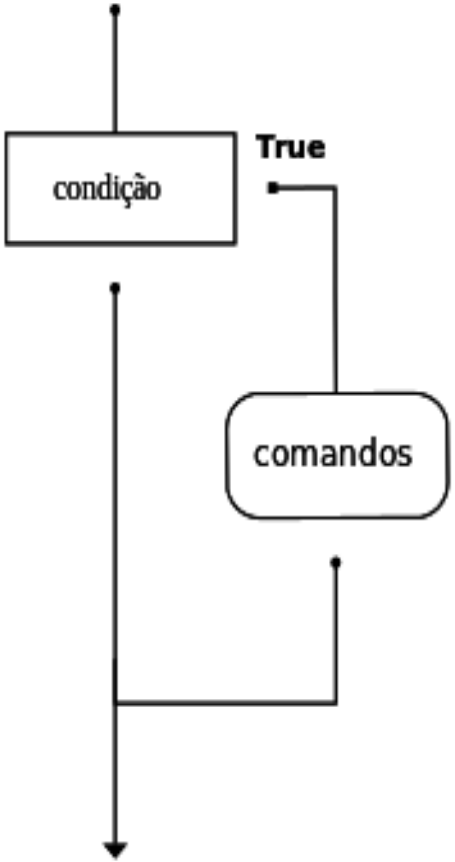
Execução condicional: Seleção binária

```
if EXPRESSÃO BOOLEANA:
        COMANDOS_1
else:
        COMANDOS_2
```



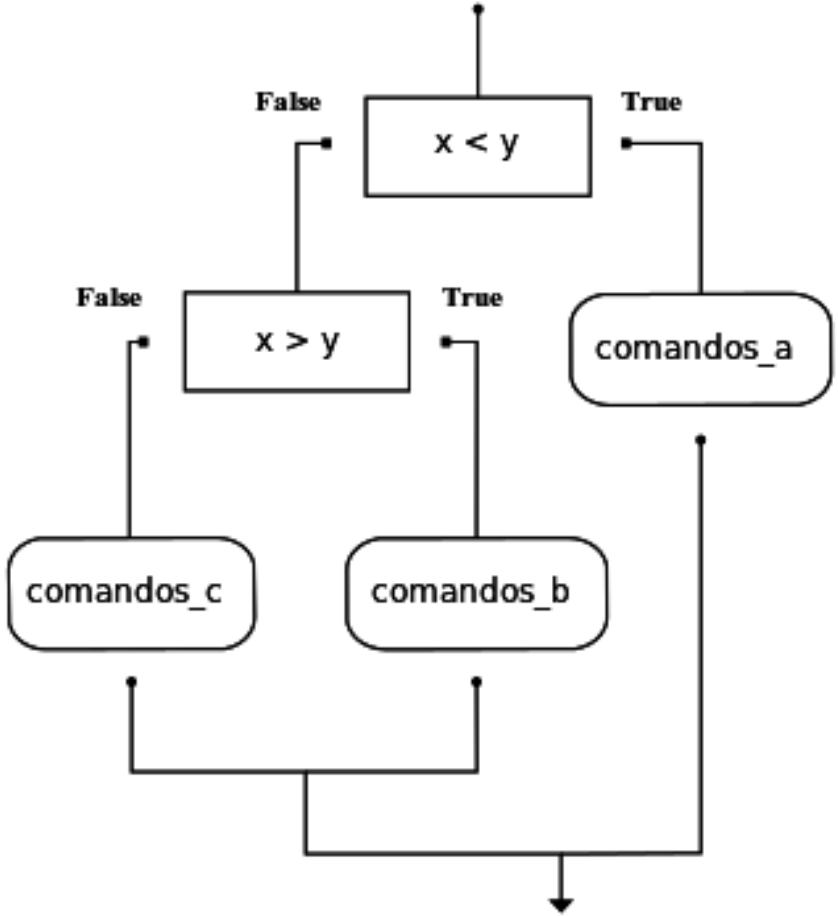
Seleção unária: omissão do else

```
if EXPRESSÃO BOOLEANA:
        COMANDOS_1
COMANDOS_2
```



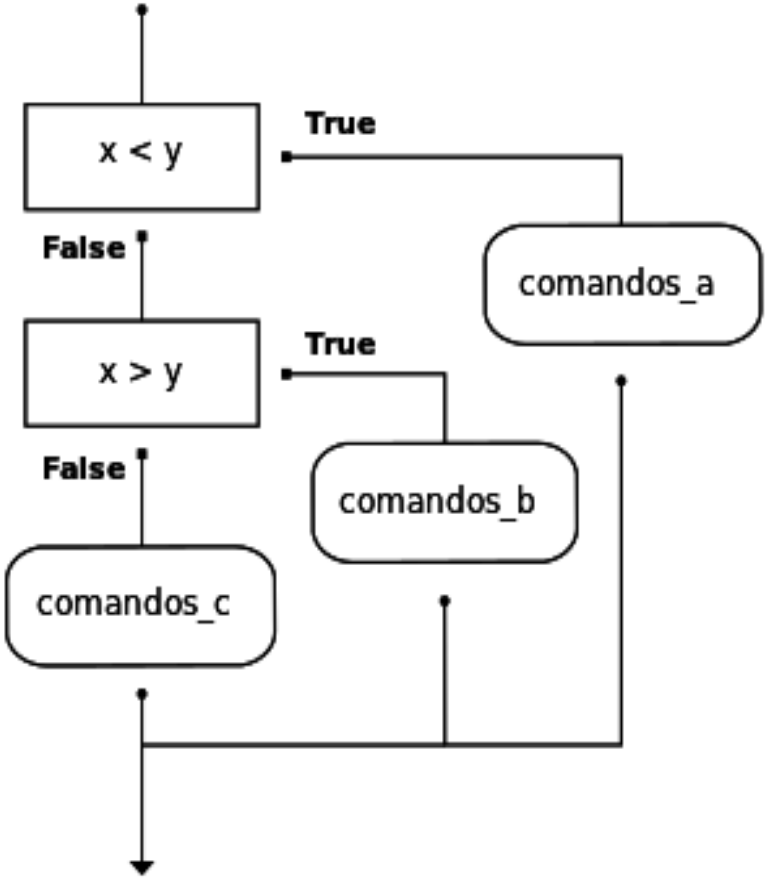
Condicionais Aninhados

```
if EXPRESSÃO BOOLEANA:
        COMANDOS_1
else:
        if EXPRESSÃO BOOLEANA:
        COMANDOS_2
        else:
        COMANDOS_3
```



Condicionais Encadeados

```
if EXPRESSÃO BOOLEANA:
        COMANDOS_1
elif EXPRESSÃO BOOLEANA:
        COMANDOS_2
else:
        COMANDOS_3
```



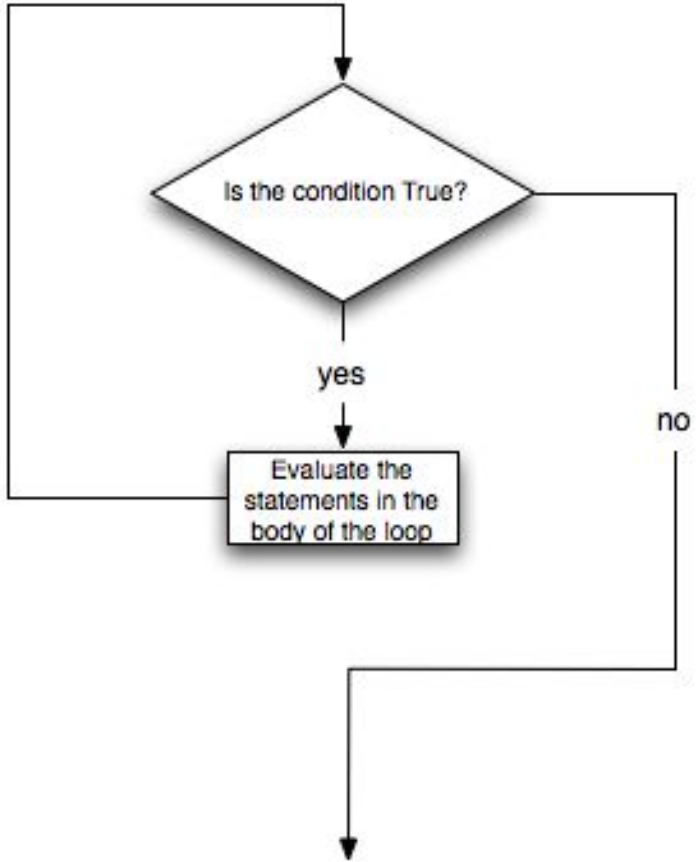
Hora de praticar!

Escreva um programa para determinar se um triângulo é equilátero, isósceles ou escaleno. As entradas devem ser dadas pelo usuário.

Repetição

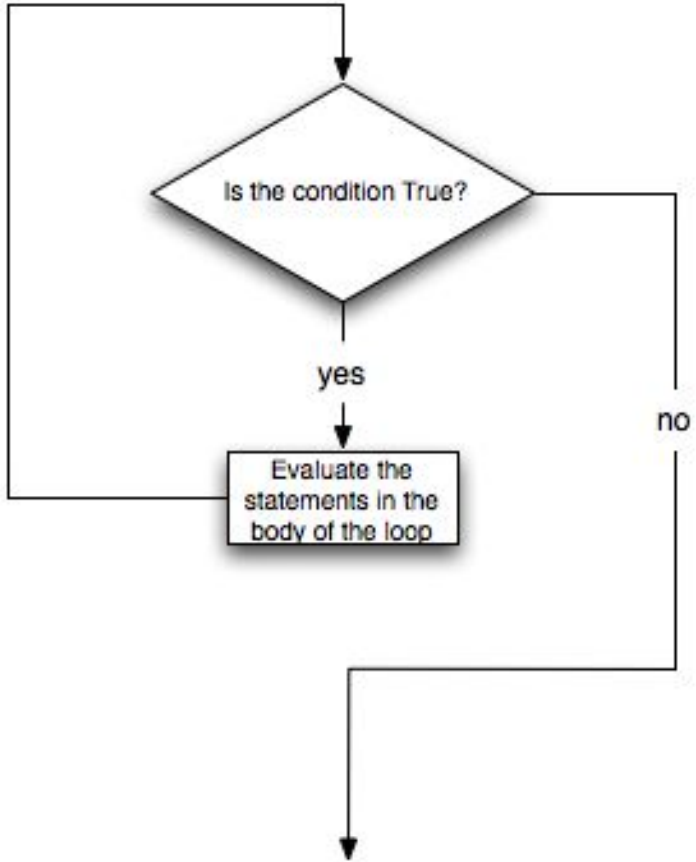
Laço for

for <var> in <lista>:
    <bloco de código identado>



Laço while

while <condição>:
    <bloco de código identado>



Hora de praticar!

Imprima os números pares de 0 a 100. Faça um programa com “for” e outro programa com “while”.

Funções

Construindo uma função

```
def NOME_DA_FUNCAO( LISTA DE PARAMETROS ) :
        COMANDOS
```

Chamando uma função

NOME_DA_FUNCAO(LISTA DE PARAMETROS)

Hora de praticar!

Escreva uma função chamada “sumAll” para somar todos os números de uma lista. A lista deve ser preenchida pelo usuário fora da função e deve ser passada como parâmetro da função.

Classes

Construindo uma classe

```
class NOME_DA_CLASSE:
        def __init__(self, ARGUMENTOS):
                self.ARGUMENTOS
        COMANDOS
```

Chamando uma classe

NOME_DA_CLASSE(LISTA DE PARAMETROS)

Hora de praticar!

Crie uma classe “animal” e instancie um objeto “gato” outro “cachorro” e outro “galinha”. A classe animal deve ter os atributos nome do animal e som. Imprima cada um.

Mini-Projeto__

Crie um programa que simule o funcionamento básico de uma conta.

O programa deverá:

- Ter um menu com as opções:
  - Cadastrar clientes com os campos número da conta, senha e saldo;
  - Fazer saque/depósito/transferir na conta, apenas se o usuário for autenticado, ou seja, existir no cadastro e a conta e senha forem correspondentes ao cadastrado;
  - Imprimir as informações do cliente;

Observações:

- Deve ser verificado se a conta está com o saldo suficiente para saque ou transferência, é proibido o saldo negativo na conta. Se o saldo for insuficiente para saque ou transferência, exibir um alerta.

Referências

- Aprenda Computação com Python's documentation! (https://aprendendo-computacao-com-python.readthedocs.io/en/latest/index.html)
- Como pensar como um cientista da computação (https://panda.ime.usp.br/pensepy/static/pensepy/index.html#listas)

Obrigada!__