
python
TM

Silvio Luiz

Formação e experiência

- Cursando Pós Graduação em Projetos e Desenvolvimento de Aplicações WEB;
- Graduado em Análise e Desenvolvimento de Sistemas (Fatec Guaratinguetá);
- Proprietário da Orions Sites e Host;

Projetos

- Python Girls
- Code Python
- Projeto i-Train (automação de maquete) na Fatec Guaratinguetá);

Mini cursos ministrados:

Aprendendo Python. 2014
Desenvolvendo Formulários de Contato para sites com validação
em PHP. 2013
Desenvolvimento de sites em HTML e CSS - Padrão W3C. 2012

Facebook
https://www.facebook.com/silviolleite

Twitter
@silvioorions

Linkedin
https://br.linkedin.com/pub/silvio-luis-pereira-leite/83/18/3b8

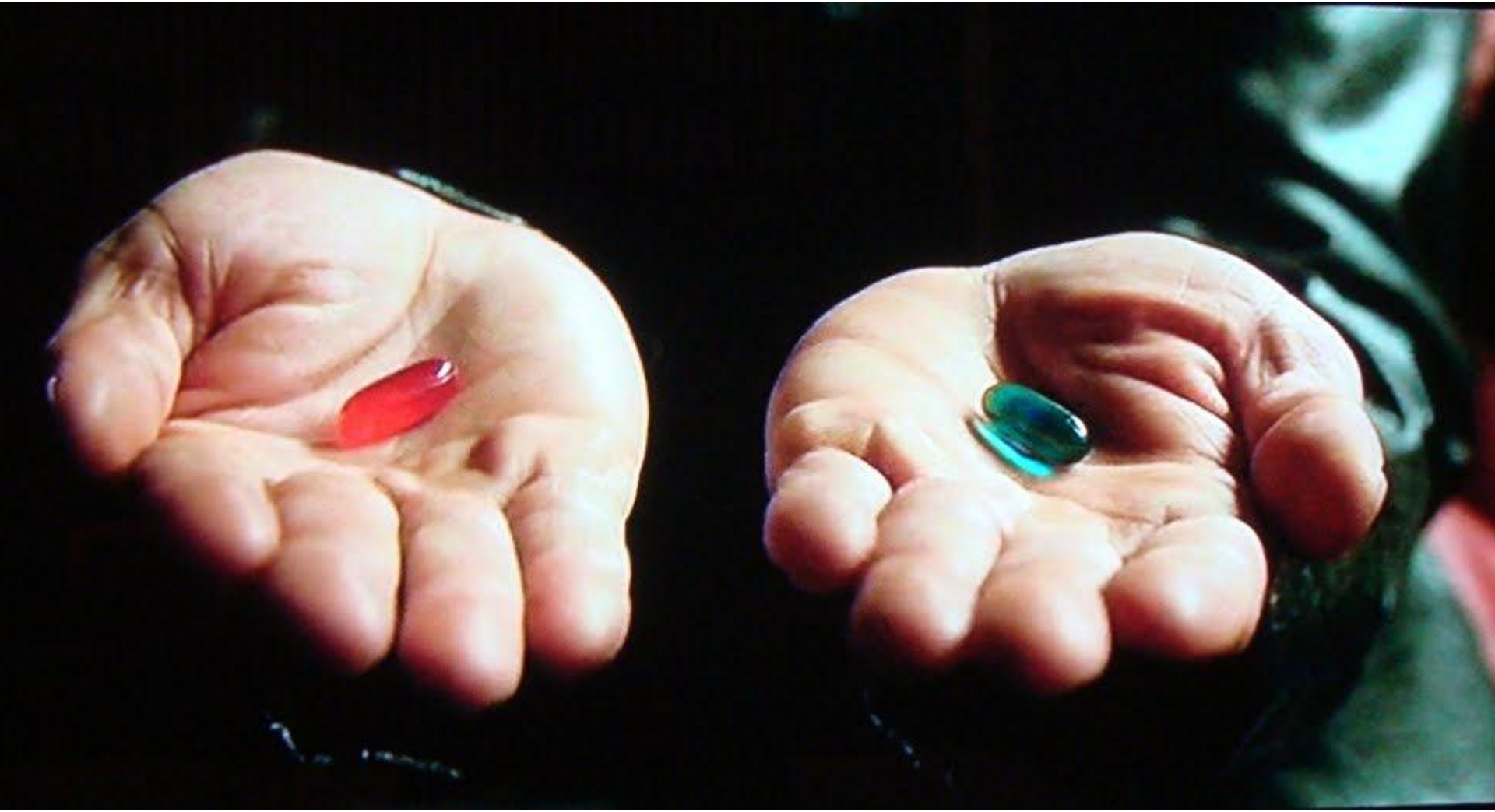
Gplus
https://plus.google.com/u/0/110949568861807162142/

Skype
silviolleite

GitHub
https://github.com/silviolleite

Hangout
silvio@orions.com.br

www.orions.com.br



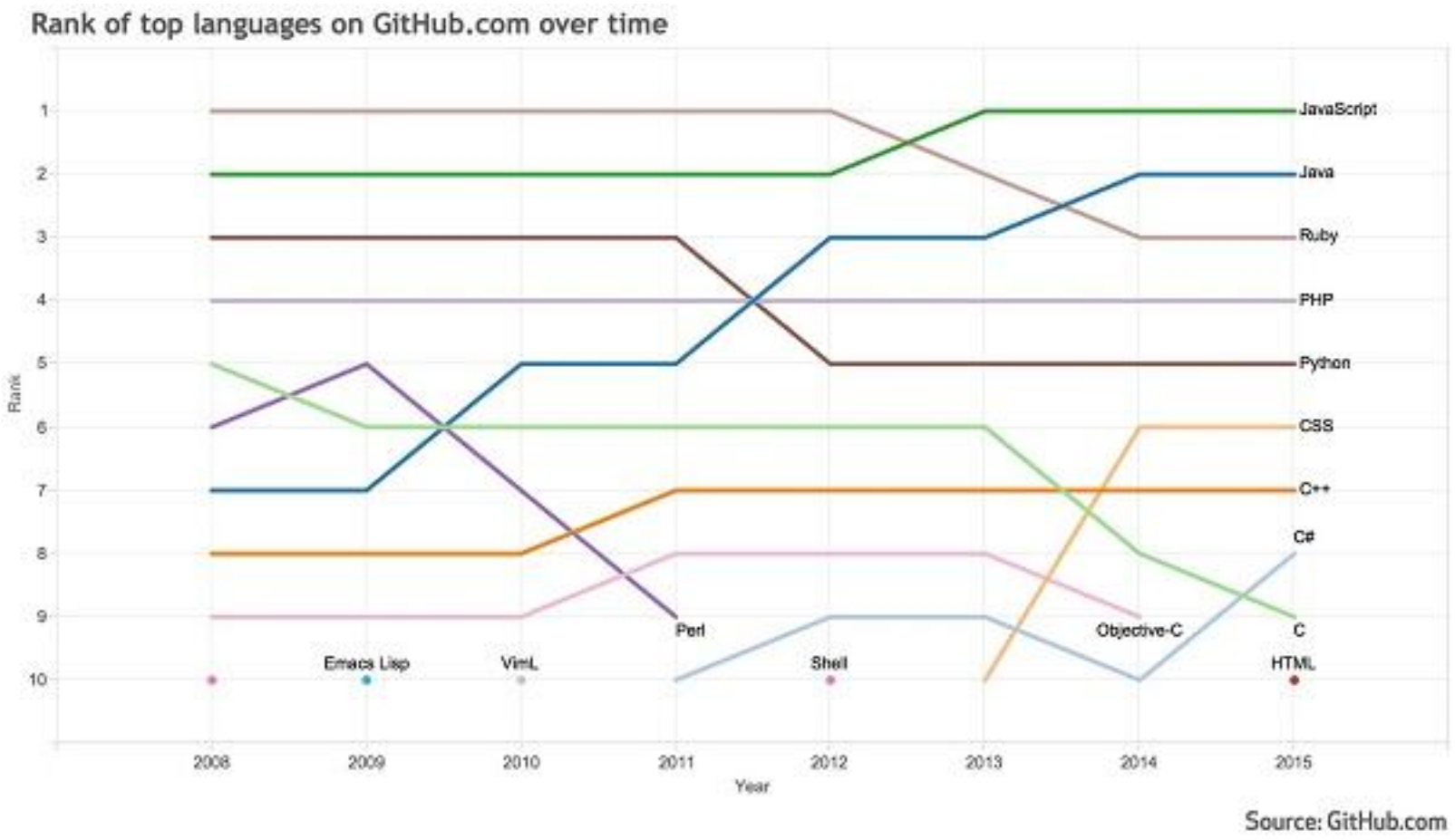
PÍLULA AZUL
VOCÊ VAI PARA A CASA E ESTÁ TUDO ACABADO

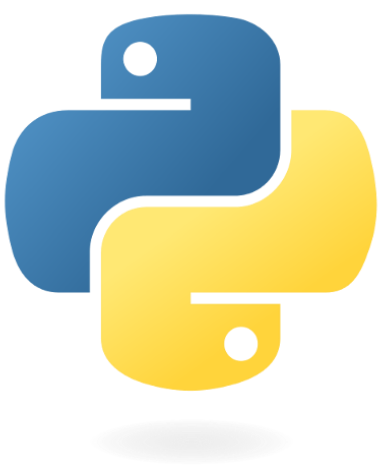
PÍLULA VERMELHA
VOCÊ IRÁ PERCEBER QUE É SÓ O COMEÇO


AUTONOMIA

O GitHub listou as 9 linguagens de programação mais populares do mundo

Fonte: http://imasters.com.br/noticia/ranking-aponta-as-principais-linguagens-de-programacao-de-2016/





python™

```
print ("Vamos começar")
```

Instalação

- Python 3.5.2
- Link: https://www.python.org/ftp/python/3.5.2/python-3.5.2.exe

Como começou?

A linguagem foi criada em 1990 por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI) e tinha originalmente foco em usuários como físicos e engenheiros. O Python foi concebido a partir de outra linguagem existente na época, chamada ABC.

Completamente Open Source

(sim Java e C# eu estou olhando pra vocês)

Características da Linguagem?

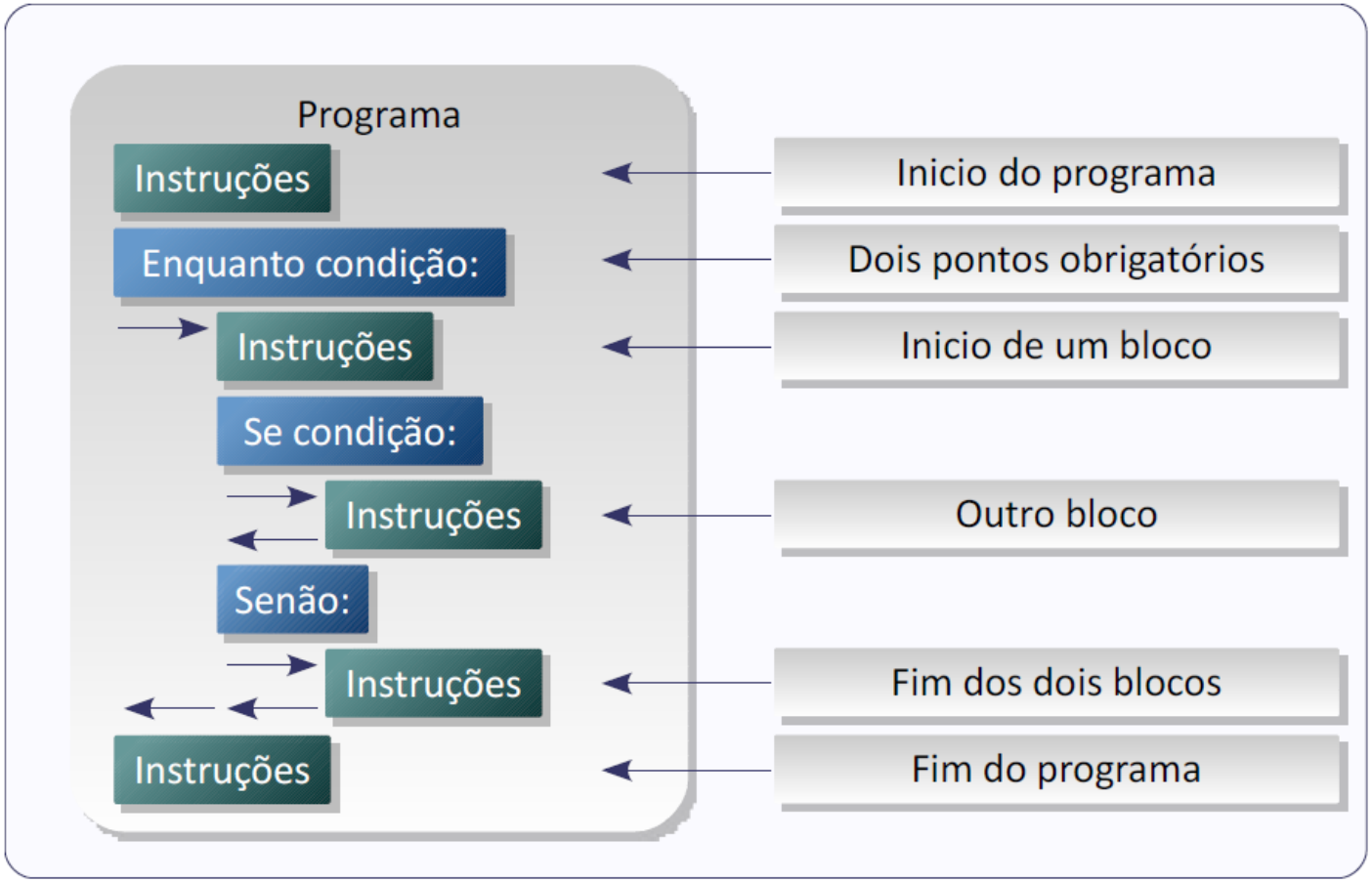
- Python é uma linguagem de programação poderosa e fácil de aprender;
- Interpretada e pseudo-compilada;
- Identação define Características escopo (sem necessidade dos famigerados {});
- Tipagem dinâmica (os tipos das variáveis não precisam ser explicitados, podendo variar durante a execução ou compilação);
- Tipagem forte (a variável mantém seu tipo até sofrer uma nova atribuição);

Características da Linguagem?

1º Simples
2º Elegante
3º Poderosa
4º Escalável
5º Dinâmica
6º Multiplataforma
7º Multi paradigmas
8º Alto nível
9º Funcional
10º Open Source

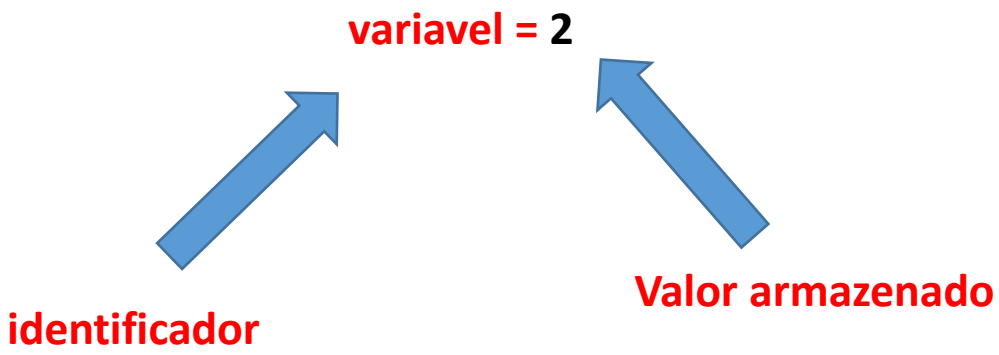
Quem usa Python?

Blocos


Programa
Instruções
Enquanto condição:
Instruções
Se condição:
Instruções
Senão:
Instruções
Instruções
Inicio do programa
Dois pontos obrigatórios
Inicio de um bloco
Outro bloco
Fim dos dois blocos
Fim do programa

Introdução a linguagem Python

Uma variável é um identificador que armazena um único valor.
Como declarar uma variável em Python?



Tipos Primitivos

Inteiro	**variavel = 2**
Float	**pi = 3.14**
Booleano	**variavel = True**
No complexo	**comp = 1 + 0.8j**
String	**nome = “Python”**

Input

```
num = int(input("Digite um número: "))
```

Operadores Aritméticos

+ → soma
– → subtração
* → multiplicação
/ → divisão
// → divisão de inteiros
** → potenciação
% → módulo (resto da divisão)

Operadores Lógicos

> → maior

>= → maior ou igual

<= → menor ou igual

== → igual

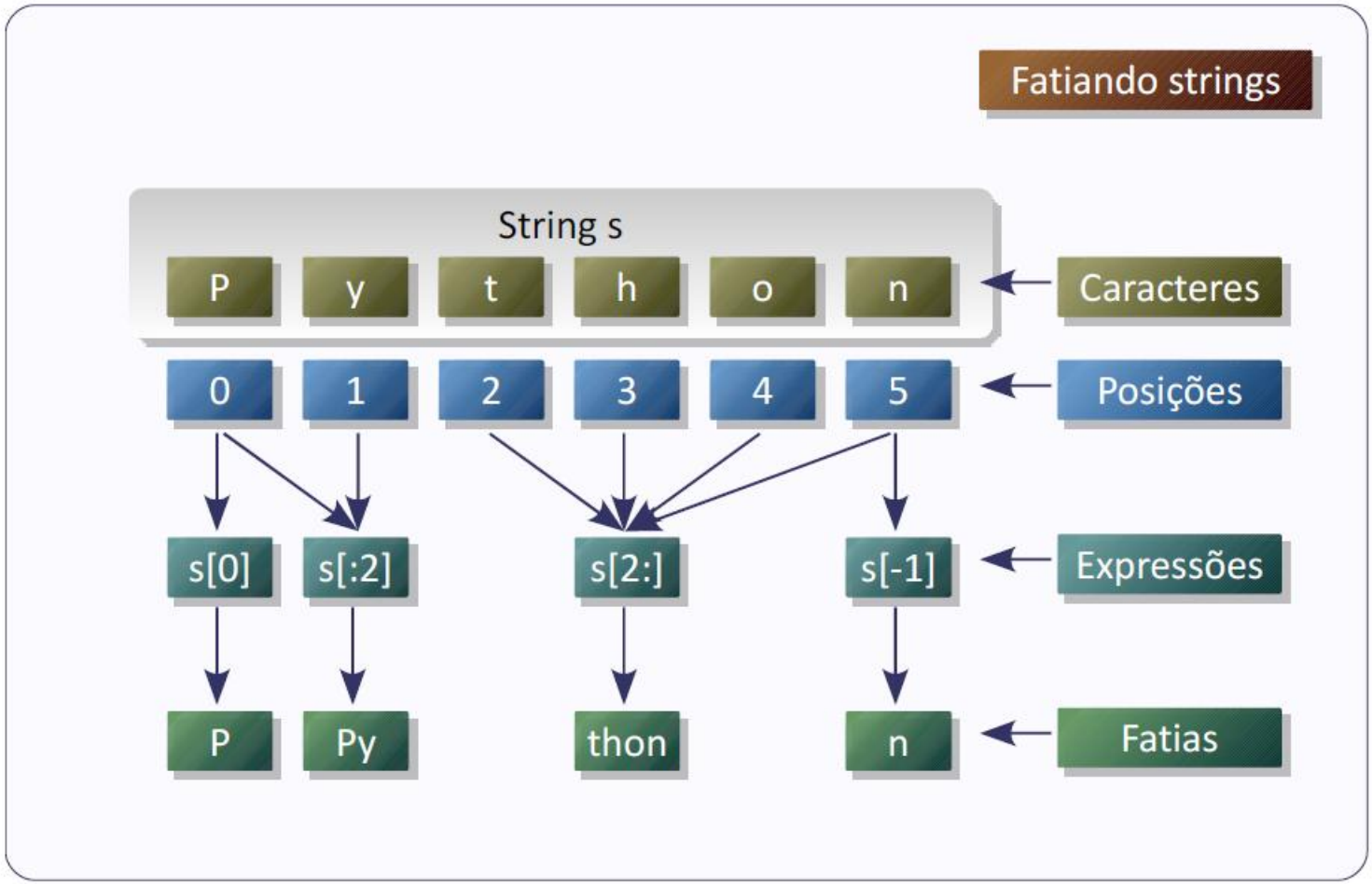
!= → diferente

not → Operador lógico que representa a negação (inverso) da variável atual. Se ela for verdade, torna-se falsa, e vice-versa.

and → Operador lógico onde a resposta da operação é verdade se ambas as variáveis de entrada forem verdade.

or → Operador lógico onde a resposta da operação é verdade se e somente se pelo menos uma das variáveis de entrada for verdade.

Strings


Fatiando strings
String s
P
y
t
h
o
n
Caracteres
0
1
2
3
4
5
Posições
s[0]
s[:2]
s[2:]
s[-1]
Expressões
P
Py
thon
n
Fatias

Manipulando Strings

```
>>> test = 'eai beleza!!'
>>> test.title()
'Eai Beleza!!'
```

Esse método é bem lega. Ele troca todas as primeiras letras da frase de minuscula por maiúsculas.

```
>>> texto = "eai beleza!!"
>>> texto.swapcase()
'EAI BELEZA!!'
```

Esse método troca as letras da string para maiúsculas.

```
>>> texto = 'Python'
>>> len(texto)
6
```

A função len retorna o número de caracteres de uma string:

```
>>> texto = "Python"
>>> texto.startswith("Py")
True
```

A função verifica se a string começa com a string passado por parâmetro.
Também podemos usar o string.endswith() para verificar o final da string.

Manipulando Strings

```
>>> grupo = "Meninas"
>>> grupo.replace("a","o")
'Meninos'
>>> grupo
'Meninas'
```

Este método faz a troca dos valores. O primeiro parâmetro será trocado pelo segundo.

```
>>> texto = "contato@python.org"
>>> texto.split("@")
['contato', 'python.org']
```

Esse método retorna uma lista onde os itens são as partes que foram separadas pelo marcador passado como parâmetro

```
>>> site = "www.example.com"
>>> site.strip("cmow.")
'example'
```

Retornar uma cópia do string com os caracteres esquerda e à direita removidos.

```
>>> nome = "python"
>>> nome.upper()
'PYTHON'
```

Retornar o texto com upercase (caixa alta)

Formatando Strings

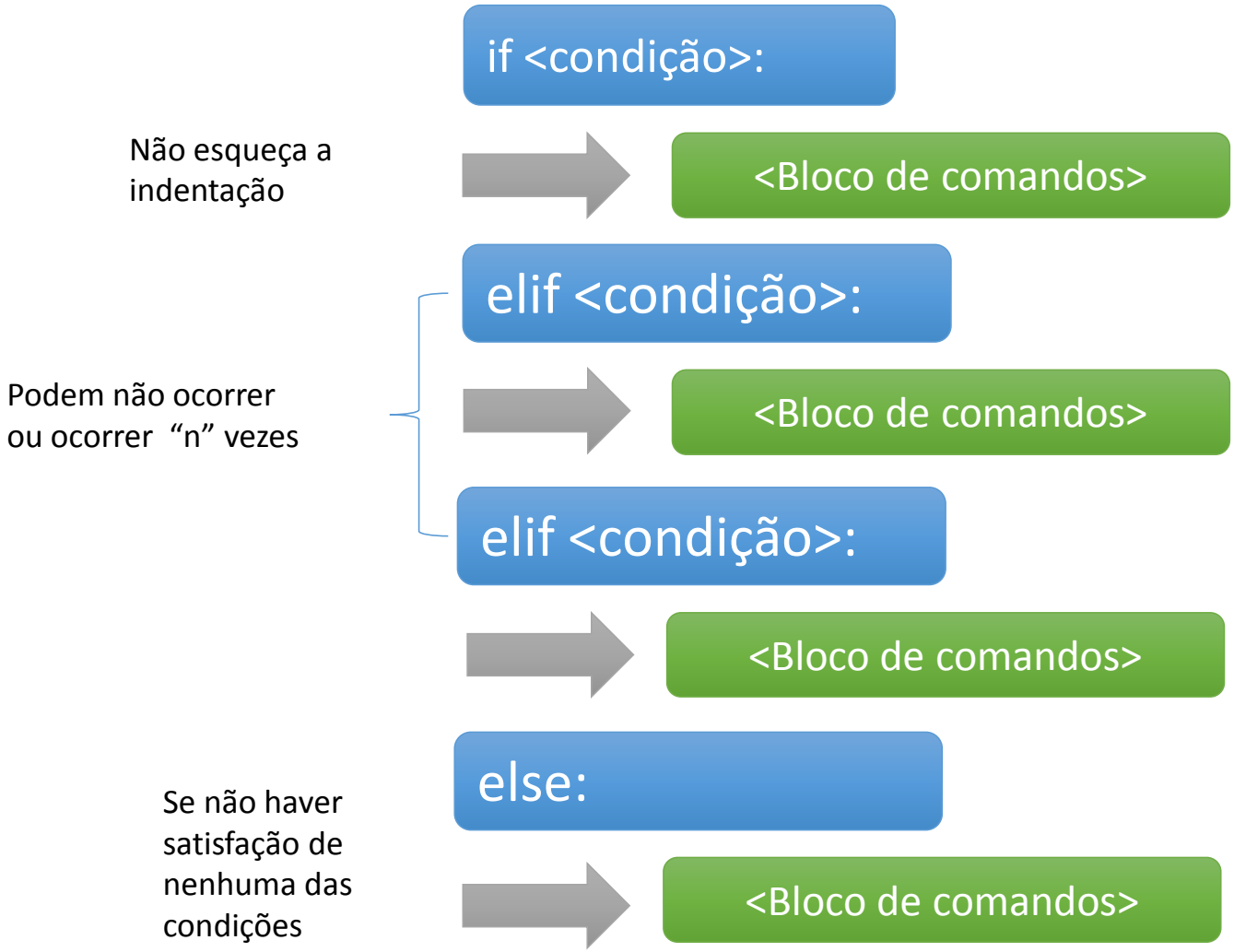
s → string
d → decimal
f → ponto flutuante
b → converte para binário
c → converte para a tabela ASCII
o → converte para octal
x → converte para hexadecimal
n → mesmo que “d”

Formatando Strings

Exemplo

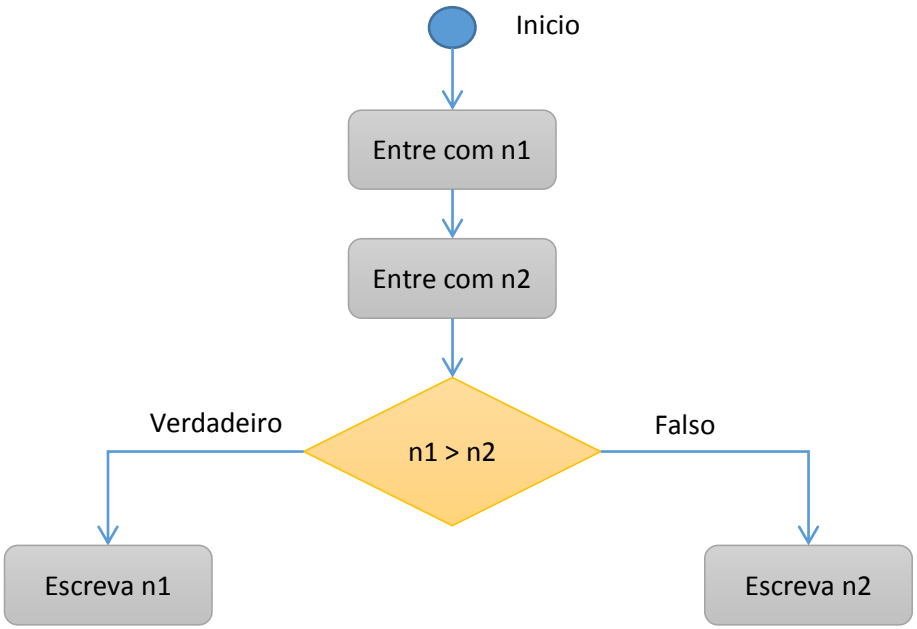
```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'
>>> coord = (3, 5)
>>> 'X: {0[0]};  Y: {0[1]}'.format(coord)
'X: 3;  Y: 5'
>>> "int: {0:d};  hex: {0:x};  oct: {0:o};  bin: {0:b}".format(42)
'int: 42;  hex: 2a;  oct: 52;  bin: 101010'
>>> #com prefixo
>>> "int: {0:d};  hex: {0:#x};  oct: {0:#o};  bin: {0:#b}".format(42)
'int: 42;  hex: 0x2a;  oct: 0o52;  bin: 0b101010'
>>> #com separador
>>> '{:,}'.format(1234567890)
'1,234,567,890'
```

Estrutura de decisão



Estrutura de decisão

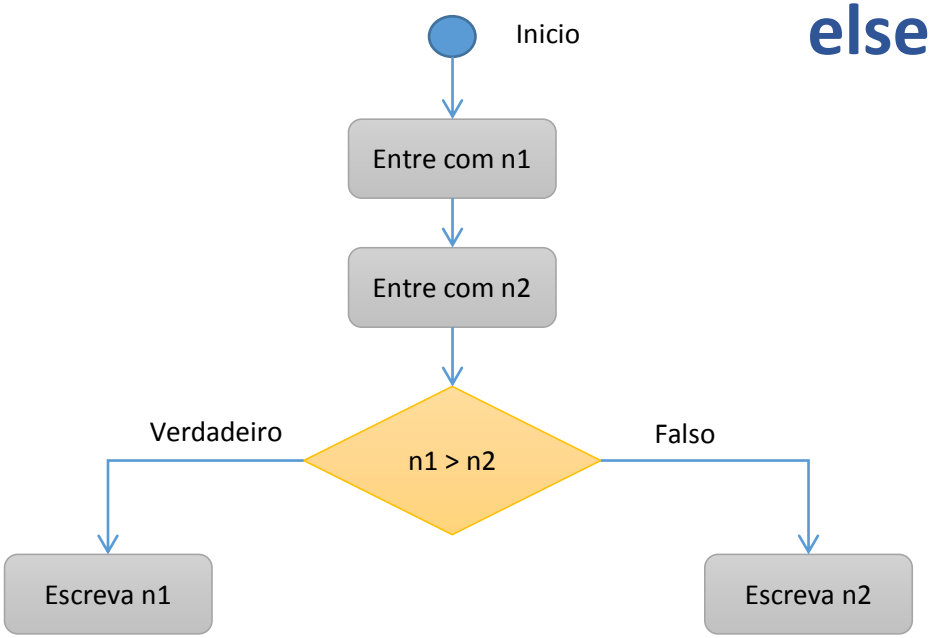
Fica a critério do programador ou da exigência da lógica de condição utilizar ou não o “()”. Ex: if (n1 < n2):



Estrutura de decisão

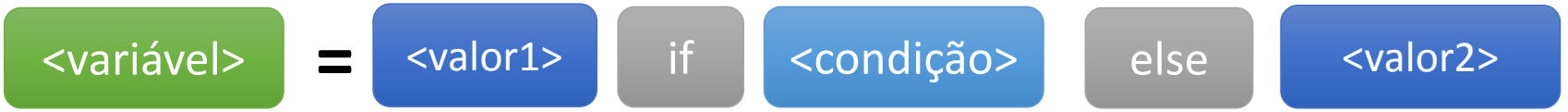
Fica a critério do programador ou da exigência da lógica de condição utilizar ou não o "()". Ex: if (n1 < n2):

```
n1 = int(input("Entre com n1: "))
n2 = int(input("Entre com n2: "))
if n1 > n2:
        print(n1)
else:
        print(n2)
```



Estrutura de decisão simplificada

A variável recebe o valor1 se a condição for verdadeira senão receberá o valor2



Sintaxe

```
logado = True
saudacao = “Olá Username” if logado else “faça Login”
Print(saudacao)
```

python™

Vamos Praticar?

1 - Faça um Programa que leia três números e mostre o maior e o menor deles.

2 - Faça um Programa que pergunte em que turno você estuda. Peça para digitar M-matutino ou V-Vespertino ou N- Noturno. Imprima a mensagem "Bom Dia!", "Boa Tarde!" ou "Boa Noite!" ou "Valor Inválido!", conforme o caso.

3 - Faça um Programa que leia um número e exiba o dia correspondente da semana. (1-Domingo, 2- Segunda, etc.), se digitar outro valor deve aparecer valor inválido.

Laço de Repetição

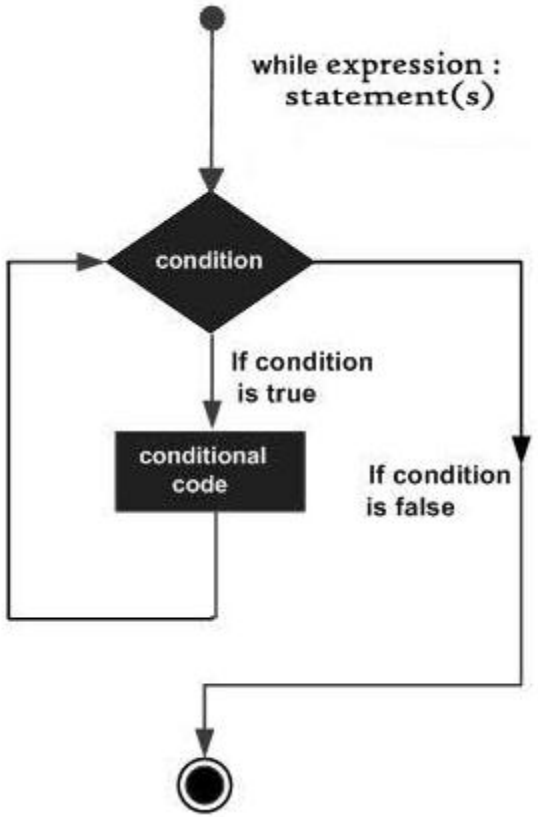
While (enquanto)

```
while (condicao) :
    codigos
```

Laço de Repetição

While (enquanto)

```
while (condicao) :
    codigos
```

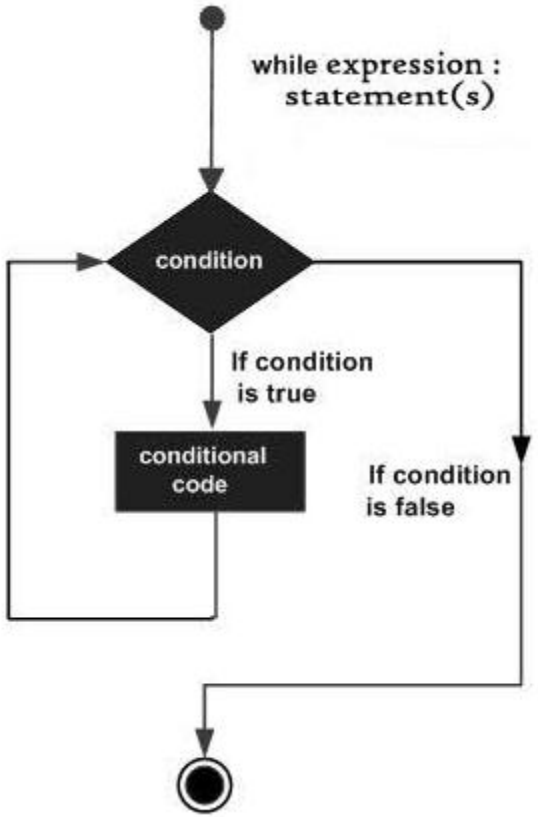


python™

Laço de Repetição

While (enquanto)

```
while (condicao) :
    codigos
```



Exemplo

```
i = 0
while i <= 10:
    print(i, end=", ")
    i += 1
```

Saída

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

python™

Laço de Repetição

For (para)

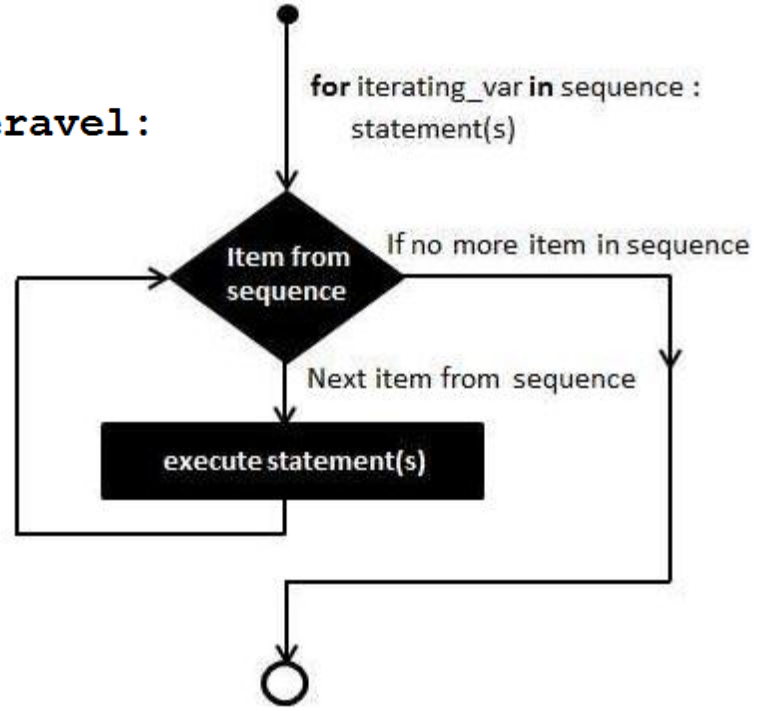
```
for variavel_de_interacao in objeto_interavel:
    codigos
```

python™

Laço de Repetição

For (para)

```
for variavel_de_interacao in objeto_interavel:
    codigos
```

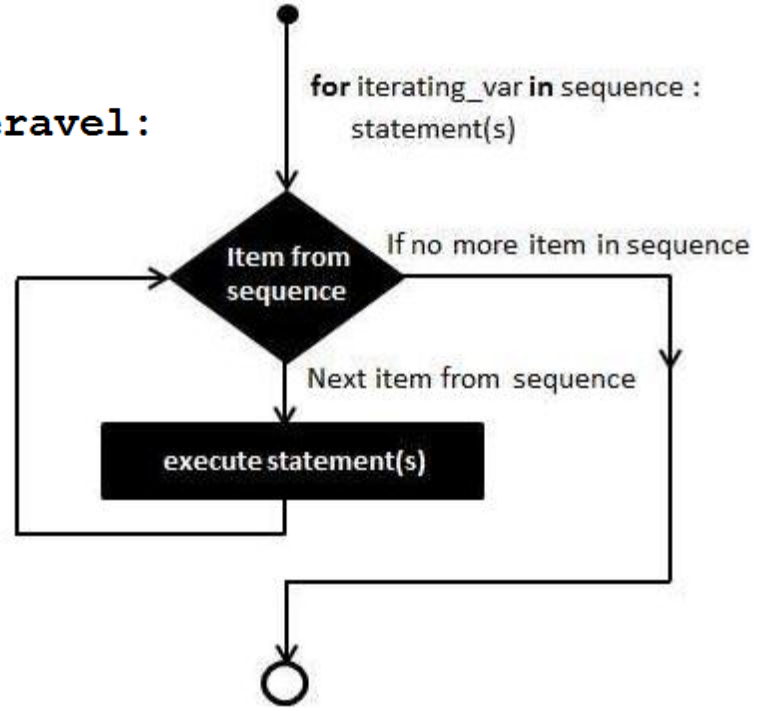


python™

Laço de Repetição

For (para)

```
for variavel_de_interacao in objeto_interavel:
    codigos
```



Exemplo

```
for i in range(11):
    print(i, end=", ")
```

Saída

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
```

python™

Sequência (Arrays) em Python

Em programação, muitas vezes torna-se necessário agrupar variáveis ou dados do mesmo tipo.
Por exemplo, se quisermos fazer um programa para gerir uma turma, temos que criar variáveis para armazenar cada nome dos alunos da turma, depois outras variáveis para cada nota de cada aluno.

Mas isso pode ser simplificado usando sequências. Sequências também facilitam muito no acesso á dados.

Vimos nas aulas anteriores que String é uma sequência de caracteres, agora vamos ver outros tipos de sequências em Python.

Lista

Uma das mais importantes sequências em Python é a lista.

Sintaxe:

```
lista = [dado1,dado2,dado3,....]
```

Exemplo:

```
lista = [“Silvio”,1,”Aprendiz”]
```

python™

Lista

Vamos praticar!

```
>>> carros = ["Fiat Uno", "Fusca", "Ferrari"]
>>> animais = ["Totó", "Rex"]
>>> aplicacoes = [1.500, 2000, 10000]
>>> pertences = [carros, animais, aplicacoes]
>>> pertences (Veja a saída)
```

python™

Lista

Vamos praticar!

Tal como no caso de Strings, em Python também usamos o operador + para indicar concatenação:

```
>>> carros + animais
['Fiat Uno', 'Fusca', 'Ferrari', 'Totó', 'Rex']
```

A função “len” que vimos em Strings também é válida para listas e devolve o seu tamanho:

```
>>> len(carros)
3
```

python™

Lista

Facilitando a vida!

Métodos	Ação
list.append(x)	Adiciona um item na lista.
list.extend(L)	Adiciona todos os itens da outra lista.
list.insert(i, x)	Insere um item x na posição i.
list.remove(x)	Remove o primeiro elemento com valor igual à x. e retorna um erro se não encontrar.
list.pop([i])	Remove o item do index i e retorna o mesmo. Se o index não for especificado, list.pop() remove e retorna o ultimo elemento da lista.
list.clear()	Remove todos os elementos da lista, equivalente ao del a[:].
list.index(x)	Retorna o index onde está armazenado o valor x. Retorna um erro se não encontrar.
list.count(x)	Retorna o número de ocorrências do valor x na lista.
list.sort()	Ordena a lista.
list.reverse()	Inverte o posicionamento dos valores na lista.
list.copy()	Copia a lista. Equivalente a list[:].

python™

Lista

Entendendo cópias!

```
>>> #copiando uma referência
>>> lista = [1,3]
>>> nova_lista = lista
>>> nova_lista is lista
True
```

python™

Lista

Entendendo cópias!

```
>>> #copiando uma referência
>>> lista = [1,3]
>>> nova_lista = lista
>>> nova_lista is lista
True


>>> #copiando de verdade
>>> nova_lista = lista[:]
>>> nova_lista is lista
False
```

python™

Lista

Entendendo cópias!

```
>>> #copiando uma referência
>>> lista = [1,3]
>>> nova_lista = lista
>>> nova_lista is lista
True


>>> #copiando de verdade
>>> nova_lista = lista[:]
>>> nova_lista is lista
False


>>> #usando o método copy()
>>> nova_lista = lista.copy()
>>> nova_lista is lista
False
```

python™

Tuplas

Vamos ver agora um novo tipo de sequência, um outro tipo de sequência padrão na linguagem:

a tupla (tuple).

Uma tupla consiste em uma sequência de valores separados por vírgulas, podendo ser vista como lista em Python, com a diferença de ser imutável (assim como strings).

Sintaxe:

```
tupla = ()
```

python™

Tuplas

Uma das grandes utilidades das Tuplas é para a representação de valores constante, alem disso elas podem ser usadas de diversas formas: pares ordenados (x, y), registros de funcionário extraídos uma base de dados, etc.

Um ponto interessante é a criação de tuplas contendo 0 ou 1 itens: a sintaxe usa certos truques para acomodar estes casos. No caso das Tuplas vazias, um par de parênteses vazios é o necessário para construí-la; uma tupla unitária é construída por um único valor e uma vírgula entre parênteses (não basta colocar um único valor entre parênteses). Um pouco estranho, mas é assim que funciona:

Tuplas

Uma das grandes utilidades das Tuplas é para a representação de valores constante, alem disso elas podem ser usadas de diversas formas: pares ordenados (x, y), registros de funcionário extraídos uma base de dados, etc.

Um ponto interessante é a criação de tuplas contendo 0 ou 1 itens: a sintaxe usa certos truques para acomodar estes casos. No caso das Tuplas vazias, um par de parênteses vazios é o necessário para construí-la; uma tupla unitária é construída por um único valor e uma vírgula entre parênteses (não basta colocar um único valor entre parênteses). Um pouco estranho, mas é assim que funciona:

```
>>> vazia = ()
>>> tupla = 'hello',
>>> len(vazia)
0
>>> len(tupla)
1
```

python™

Tuplas

As tuplas também suportam acesso aos valores através dos índices e maior parte das operações das listas, como fatiamento. Podemos utilizá-las também com o for:

Tuplas

As tuplas também suportam acesso aos valores através dos índices e maior parte das operações das listas, como fatiamento. Podemos utilizá-las também com o for:

```
>>> numeros = (1, 2, 3, 4)
>>> for num in numeros:
        print (num)


1
2
3
4
```

python™

Tuplas

Python também permite operações chamadas de empacotamento e desempacotamento. O empacotamento acontece como no exemplo: t = 12345, 54321, 'python!'.
Os valores 12345, 54321, 'python!' são empacotados na tupla t.
Já o desempacotamento acontece como neste caso:
a, b = 10, 20.
Para funcionar, é necessário que a lista de variáveis do lado esquerdo tenha o mesmo comprimento da sequência à direita.
Sendo assim, a atribuição múltipla é um caso de empacotamento de tupla e desempacotamento de sequência. Ainda é possível fazer trocas rapidamento dos valores das variáveis:

Tuplas

Python também permite operações chamadas de empacotamento e desempacotamento. O empacotamento acontece como no exemplo: t = 12345, 54321, 'python!'.
Os valores 12345, 54321, 'python!' são empacotados na tupla t.
Já o desempacotamento acontece como neste caso:
a, b = 10, 20.
Para funcionar, é necessário que a lista de variáveis do lado esquerdo tenha o mesmo comprimento da sequência à direita.
Sendo assim, a atribuição múltipla é um caso de empacotamento de tupla e desempacotamento de sequência. Ainda é possível fazer trocas rapidamento dos valores das variáveis:

```
>>> a,b = 20,10
>>> a, b = b, a
>>> a
10
>>> b
20
```

python™

Tuplas

Também podemos criar tuplas a apartir de listas, usando a função

tuple()

Tuplas

Também podemos criar tuplas a apartir de listas, usando a função

tuple()

```
>>> l = [1, 2, 3]
>>> t = tuple(l)
>>> t
(1, 2, 3)
```

python™

Tuplas

Também podemos criar tuplas a apartir de listas, usando a função

tuple()

```
>>> l = [1, 2, 3]
>>> t = tuple(l)
>>> t
(1, 2, 3)
```

Mesmo que não possamos fazer alterações na tupla depois de criá-la, podemos concatená-las... mas saiba que isso gera novas tuplas:

python™

Tuplas

Também podemos criar tuplas a apartir de listas, usando a função

tuple()

```
>>> l = [1, 2, 3]
>>> t = tuple(l)
>>> t
(1, 2, 3)
```

Mesmo que não possamos fazer alterações na tupla depois de criá-la, podemos concatená-las... mas saiba que isso gera novas tuplas:

```
>>> t1 = (1, 2, 3)
>>> t2 = (4, 5, 6)
>>> t1 + t2
(1, 2, 3, 4, 5, 6)
```

python™

Tuplas

IMPORTANTE: Tuplas podem conter objetos que podem ser alterados, mas as alterações nesses objetos não são consideradas mudanças na tupla em si, como no exemplo:

Tuplas

IMPORTANTE: Tuplas podem conter objetos que podem ser alterados, mas as alterações nesses objetos não são consideradas mudanças na tupla em si, como no exemplo:

```
>>> tupla = ("a", ["b", "c", "d"])
>>> tupla
('a', ['b', 'c', 'd'])
>>> len(tupla)
2
>>> tupla[1]
['b', 'c', 'd']
>>> tupla[1].append("e")
>>> tupla
('a', ['b', 'c', 'd', 'e'])
```

python™

Vamos Praticar!

4 - Faça um programa que leia 5 números e informe o maior número.

5 - Desenvolva um gerador de tabuada, capaz de gerar a tabuada de qualquer número inteiro entre 1 a 10. O usuário deve informar de qual numero ele deseja ver a tabuada. A saída deve ser conforme o exemplo abaixo:
Tabuada de 5:
5 X 1 = 5
5 X 2 = 10
...
5 X 10 = 50

Vamos Praticar!

6 - Monte um programa onde o usuário entra com 1 número decimal e o programa imprime a conversão em Binário, Hexadecimal, Octal e Caractere da tabela ASCII .

Entrada:

65

Saída:

“Bin: 1000001, Hex: 41, Oct: 101, Caractere: A”

python™

Dicionários

Outra sequência de dados muito útil embutida em Python é o dicionário, cujo tipo é:

dict

É uma estrutura parecida com a lista, mas possui propriedades de acesso diferentes. Dicionários são delimitados por chaves: {}

Sintaxe:

```
dicionario = { }
```

Dicionários

O dicionário contém uma lista de pares chave:valor separada por vírgulas. Eles são também chamados de “memória associativa” ou “vetor associativo” em outras linguagens.

Dicionários

O dicionário contém uma lista de pares chave:valor separada por vírgulas. Eles são também chamados de “memória associativa” ou “vetor associativo” em outras linguagens.

Exemplo de declaração:

```
>>> mercado = {
        "banana": 1.50,
        "uva": 2.45,
        "laranja":3.50}
```

python™

Dicionários

O dicionário contém uma lista de pares chave:valor separada por vírgulas. Eles são também chamados de “memória associativa” ou “vetor associativo” em outras linguagens.

Exemplo de declaração:

```
>>> mercado = {
        "banana": 1.50,
        "uva": 2.45,
        "laranja":3.50}
```

Se tentar recuperar um valor usando uma chave inexistente, será gerado um erro do tipo KeyError. Como no exemplo:

python™

Dicionários

O dicionário contém uma lista de pares chave:valor separada por vírgulas. Eles são também chamados de “memória associativa” ou “vetor associativo” em outras linguagens.

Exemplo de declaração:

```
>>> mercado = {
        "banana": 1.50,
        "uva": 2.45,
        "laranja":3.50}
```

Se tentar recuperar um valor usando uma chave inexistente, será gerado um erro do tipo KeyError. Como no exemplo:

```
>>> print(mercado["manga"])
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(mercado["manga"])
KeyError: 'manga'
```

python™

Dicionários **Facilitando a vida!**

Métodos	Ação
dict.keys()	Retorna uma lista com as chaves de um dicionário
dict.values()	Retorna a lista de valores de um dicionário
dict.items()	Retorna os dois, na forma de uma lista de tuplas - cada tupla com um par chave-valor
dict.copy(dicts)	Copia para um novo dicionário as chaves e valores.
“key” in dict	Retorna True se a chave existir ou False se não existir.
dict.clear()	Limpa o dicionário deixando ele vazio.
del dict[“chave”]	Deleta o par de chave e valor.

python™

Vamos Praticar!

7 - Crie um dicionário e armazene nele os seus dados: nome, idade, telefone, endereço. Imprima todos os dados usando o padrão chave: valor.

8 – Faça um programa em Python que conta a ocorrência da palavra "tigre" na frase "um tigre, dois tigres, três tigres".

Set

Python também inclui um tipo de dados para conjuntos, chamado set. Um conjunto é uma coleção desordenada de elementos, mas não possui elementos repetidos.

A vantagem desse tipo de dado é quando precisamos garantir que dados não se repitam, funcionando como um verificação eficiente da existência de objetos e a eliminação de itens duplicados.

Conjuntos também suportam operações matemáticas como:
união, interseção, diferença e diferença simétrica.

Set exemplos

```
>>> linguagens = {"python", "php", "groovy", "javascript", "python", "php"}
>>> linguagens
{'php', 'python', 'javascript', 'groovy'}
>>> "python" in linguagens
True
>>> "c#" in linguagens
False
>>> #tirando a duplicidade de letras em strings
>>> letras = set("aprendendo python")
>>> letras
{'n', 't', 'd', 'a', 'y', 'p', ' ', 'h', 'r', 'e', 'o'}
>>> #operações com conjuntos
>>> a = {'a','r','b','c','d'}
>>> b = {'a','l','c','z','m'}
>>> #letras em "a" mas não em "b"
>>> a - b
{'d', 'r', 'b'}
>>> #letras em "a" ou em "b"
>>> a | b
{'b', 'c', 'd', 'a', 'm', 'z', 'r', 'l'}
>>> #letras em "a" e em "b"
>>> a & b
{'a', 'c'}
>>> #letras em "a" ou em "b" mas não em ambos
>>> a ^ b
{'b', 'd', 'm', 'z', 'r', 'l'}
```

Funções

As funções são úteis para empacotar uma tarefa específica em um trecho de código. A vantagem disso é a reutilização do código, a solução criada naquela função poderá ser usada sempre que necessária sem precisar que reescreve-la sempre.

Funções

As funções são úteis para empacotar uma tarefa específica em um trecho de código. A vantagem disso é a reutilização do código, a solução criada naquela função poderá ser usada sempre que necessária sem precisar que reescreve-la sempre.

Sintaxe

```
def funcao(parametros):
    codigos
```

python™

Funções

As funções são úteis para empacotar uma tarefa específica em um trecho de código. A vantagem disso é a reutilização do código, a solução criada naquela função poderá ser usada sempre que necessária sem precisar que reescreve-la sempre.

Sintaxe

```
def funcao(parametros):
    codigos
```

Exemplo

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

print(soma(5,5))
```

Saída

```
10
```

python™

Vamos Praticar!

9 - Faça um programa, com uma função que necessite de três argumentos, e que forneça a soma desses três argumentos.

10 - Faça um programa, com uma função que necessite de um argumento. A função retorna o valor de caractere ‘P’, se seu argumento for positivo, e ‘N’, se seu argumento for zero ou negativo.

11 - Crie uma função numero par que permita verificar um dado número x passado como parâmetro é número par.

Referência

Documentação Python
www.python.org

Livro Python Escreva seus primeiros programas
Felipe Cruz

print(“Obrigado e bons estudos!”)

Link para os arquivos do projeto no GitHub
https://github.com/silviolleite/minicursopython2016