Programação para Data Science e Data Analytics

Daniel Teófilo

Contatos

DANIEL TEÓFILO VASCONCELOS

WhatsApp: 85996517653

E-mail: daniel_teofilo@yahoo.com.br

Linkedin: https://www.linkedin.com/in/daniel-teófilo/

Instagram: https://www.instagram.com/daniel_teofilos/

9 razões para aprender programação

1. Faça a diferença no mercado de trabalho hoje e garanta a sua presença nele amanhã

2. Aprender a programar te ajuda a pensar

3. Programação é a nova alfabetização

4. Aprenda várias disciplinas ao mesmo tempo

5. Desenvolve a habilidade de resolver problemas

6. Não é só uma questão de máquinas, nem números, tem a ver com humanidade

7. Trabalha a persistência e a capacidade de superação

8. Estimula a criatividade

9. Todo mundo é capaz de aprender

A inteligência está
nas máquinas ou nos
desenvolvedores?

O que é a Linguagem Python?

Conhecendo Python

Lançada em 1991, na Holanda, por Guido Van Rossum

Linguagem interpretada

Orientada a Objetos

Portável

Comunidade Ativa

Por que Python?

Por que Python?

É fácil

É poderoso

É divertido

Por que é fácil

- Semelhança com pseudo-código.

```
x = 5

if x == 5:

    y = 0
    while (y <=10):
        print("Valor de y é: ", y)
        y = y + 1

    for i in [5,6,7,8,9]:
        print(i)

    for i in range(100):
        print(i)
```

Por que é fácil

- Uso de identação para marcar bloco.

```
x = 5

if x == 5:

    y = 0
    while (y <=10):
        print("Valor de y é: ", y)
        y = y + 1

    for i in [5,6,7,8,9]:
        print(i)

    for i in range(100):
        print(i)
```

Por que é fácil

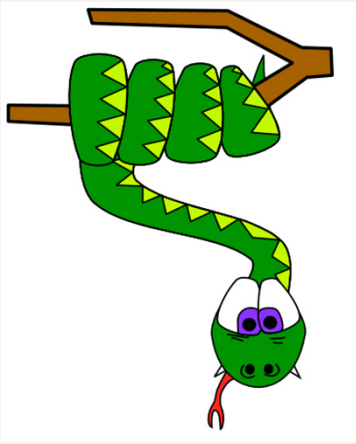
- Orientação à objetos
- Biblioteca padrão completa
- Multi-paradigma
- Multi-plataforma
- Facilmente extensível
- Free Software (GPL)

Por que é poderoso

- Python para Web
- Python para gestão empresarial
- Python para dispositivos móveis
- Python para ciência
- Python para educação
- Python para games

Por que é divertido

Porque é poderoso e fácil ao mesmo tempo



Por onde começo ?

... Criando nosso primeiro Hello World !

Hello World

... ‘hello world’ - Python X {Java, C, PHP, Pascal}

```
program helloworld;
begin
    writeln<'Hello World!'>;
end
```

```
class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

```
#include <stdio.h>
int main (void)
{
    printf("Hello World");
    return 0;
}
```

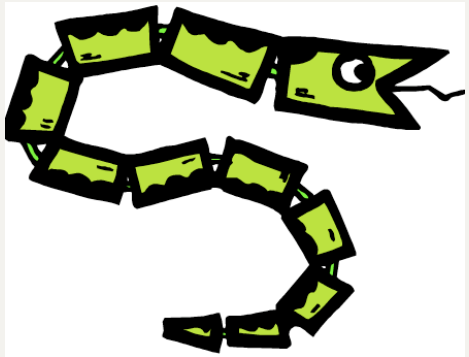
```
<?php
    echo "Hello World";
?>
```

... em Python ...

```
print ("Hello World")
```

Tipos e operações

Vamos ver um trecho de código em Python!



Código Base

```
x = 34 - 23
y = "Hello"
z = 3.45
if (x == 3.45 or y == "Hello"):
    x = x + 1
    y = y + " World"

print(x)
print(y)
```

... entendendo o código...

- Atribuição utiliza “=“ e comparação utiliza “==“

```
x = 34 - 23
y = "Hello"
z = 3.45
if (x == 3.45 or y == "Hello"):
    x = x + 1
    y = y + " World"

print(x)
print(y)
```

... entendendo o código...

- Números: + - * / % tem suas funções características
- + pode ser usado como concatenação de Strings;
- % pode ser usado para formatar Strings (assim como em C).

```
x = 34 - 23
y = "Hello"
z = 3.45
if (x == 3.45 or y == "Hello"):
    x = x + 1
    y = y + " World"

print(x)
print(y)
```

... entendendo o código...

- Operadores lógicos são palavras e não símbolos (||, &&)
- and, or, not

```
x = 34 - 23
y = "Hello"
z = 3.45
if (x == 3.45 or y == "Hello"):
    x = x + 1
    y = y + " World"

print(x)
print(y)
```

... entendendo o código...

- print é o comando básico para “impressão” na tela

```
x = 34 - 23
y = "Hello"
z = 3.45
if (x == 3.45 or y == "Hello"):
    x = x + 1
    y = y + " World"

print(x)
print(y)
```

Tipos Básicos

- Inteiros (padrão para números)
  - Divisão entre inteiros, resposta um inteiro!
- Inteiros Longos
  - L ou l no final. (Convertido automaticamente com precisão de inteiros > 32 bits)
- Floats (ponto flutuante)
  - 1.23, 3.4e-10
- Complexas
  - >> 2 + 3j
- Operações válidas: +, *, >> , ** , pow, abs, etc.

Tipos Básicos

- Representação numérica
  - Representação de dígitos com/sem formatação de string
- Divisão clássica / base
  - Uso dos operadores // e /
- Operações em nível de bit
  - 1 << 2 , 1| 2 , 1&2
- Notações hexadecimal / octal
  - 2 , 0x10 , 0100 , oct(64), hex(255) , int('200'), int('0100',8) , int('0x40',16)
- Operações válidas: +, *, >> , ** , pow, abs, round, etc.

Tipos Básicos

Operação	Resultado
x + y	Soma dos valores x e y
x - y	Subtração de x por y
x * y	Multiplicação de x por y
x / y	Divisão de x por y
x // y	Divisão de x por y, obs.: Pegando o piso.
x % y	Resto da divisão de x por y
+x	Não altera nada
-x	Inverte o sinal de x
abs(x)	Valor absoluto de x
int(x)	x convertido em inteiro
long(x)	x convertido em long
float(x)	x convertido em float
complex(re, im)	Um número complexo com parte real re e imaginária im
x ** y	x elevado a y
pow(x, y)	x elevado a y

Obs.: as operações citadas acima valem para todos os tipos numéricos exceto números complexos

Tipos Básicos

- Strings
  - “abc” ou ‘abc’
- Operadores de expressão de Python e sua precedência
- http://docs.python.org/reference/expressions.html#summary

Símbolo	ação comparativa
“<”	Menor que
“<=”	menor ou igual
“>”	maior que
“>=”	maior ou igual
“==”	igual (objeto ->referência)
“!=”	diferente
“<>”	diferente
“is”	igualdade de objetos
“is not”	diferença de objetos

```
>>>True > False
<Qual seria o resultado???>
```

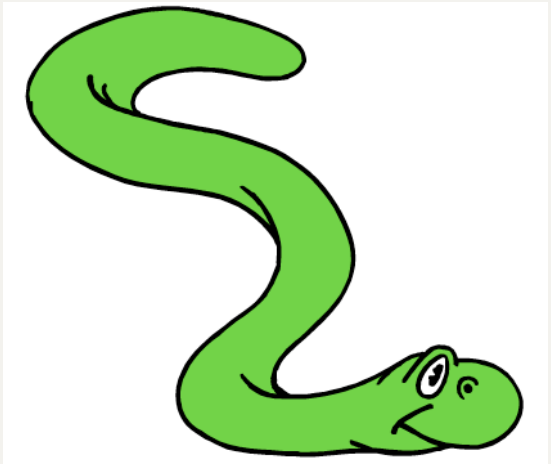
Comandos básicos

- Alguns comandos básicos que podem ajudar no ínicio!
  - dir(element) - todos os atributos e métodos que estão associados ao elemento.
  - type(element) - Descobrir o tipo do objeto!
  - import - importe módulos para uso no seu código!

```
import modulo
import modulo.algo
import modulo.algo as malg
from modulo import *
from modulo import algo
from modulo import algo.item as ait
```

Atribuição

... Vamos entender como funciona atribuição!



Atribuição

- Atribuição de uma variável em Python significa criar um rótulo para armazenar uma referência para algum objeto.
  - Atribuição cria referências e não cópias!
  - Inferência do tipo da referência baseado no tipo de dado atribuído
- A referência é deletada por meio de Garbage Collection
  - Quando o objeto deixa de ser referenciado por nenhum outro rótulo(variável).

Atribuição

- Você pode inicializar várias variáveis de uma só vez!
  - x = y = z = 2.0
- Rótulos de variáveis são Case Sensitive e não podem iniciar com número. Números, letras e underscores são permitidos!
  - bob bob_2 _bob _2_bob bob_2 BoB
- Não esquecer das palavras reservadas!

```
and, assert, break, class, continue, def, del,
elif, else, except, exec, finally, for, from,
global, if, import, in, is, lambda, not, or, pass,
print, raise, return, try, while
```

Atribuição

- Entendendo manipulação de atribuição de referências
  - x = y não significa que você fez uma cópia de y!
  - x = y o que realmente faz é x referencia ao objeto que y referencia!
- O que realmente acontece por trás dessa simples atribuição:

```
>>> x = 3
>>> x = x + 1
>>> print x
4
```

Atribuição

- Mas e se fizermos isso ?! Qual será o valor de x ?

```
>>> x = “casa”
>>> y = x
>>> x = “fazenda”
>>> print x
```

STRINGS

Strings são usadas em Python para gravar informações em formato de texto, como nomes por exemplo. Strings em Python são na verdade uma sequência de caracteres, o que significa, basicamente, que Python mantém o controle de cada elemento da sequência.

STRINGS

Python entende a string "Olá", como sendo uma sequência de letras em uma ordem específica. Isso significa que você será capaz de usar a indexação para obter um caracter específico (como a primeira letra ou a última letra).

STRINGS

Strings – sequência imutável de caracteres ou apenas 1 caracter

"Essa é uma string"

"a"

STRINGS

Indexando Strings

Já sabemos que Strings são uma sequência. Isso significa que Python pode usar índices para chamar partes da sequência. Vamos aprender como isso funciona.

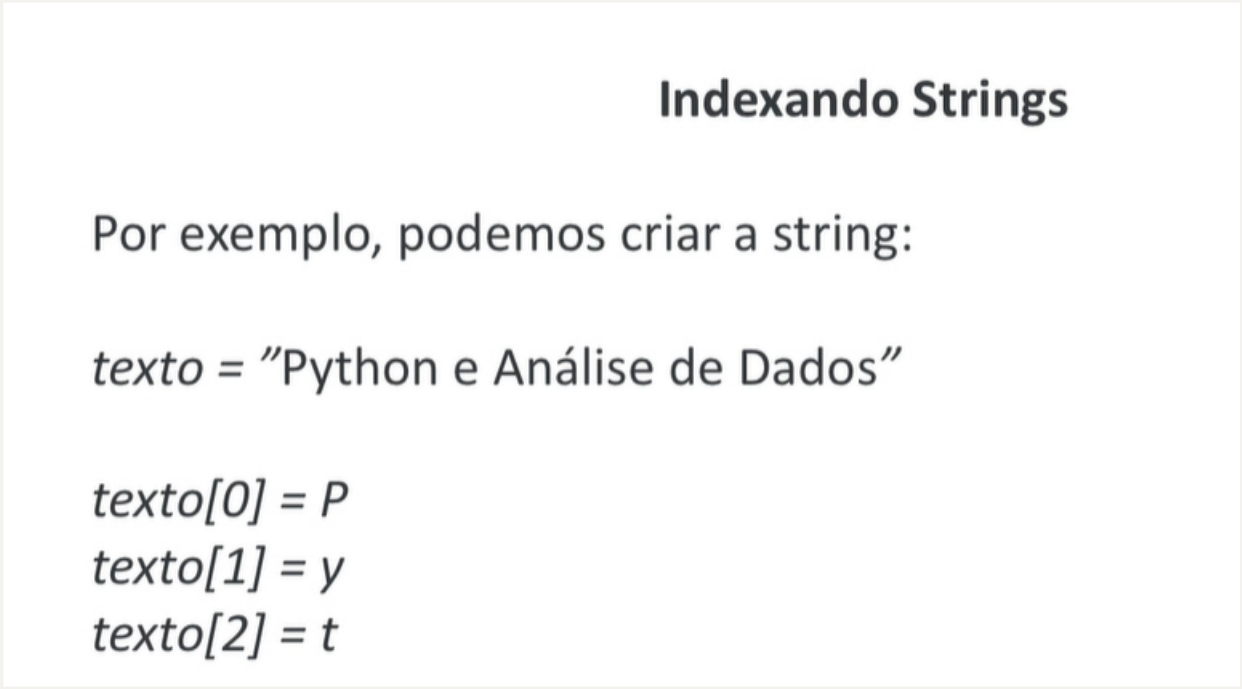
STRINGS

Indexando Strings

Em Python, usamos **colchetes []** para representar o índice de um objeto.

→ *Em Python, a indexação começa por 0.*

STRINGS


Indexando Strings
Por exemplo, podemos criar a string:
texto = ”Python e Análise de Dados”
texto[0] = P
texto[1] = y
texto[2] = t

STRINGS

Funções Built-in de Strings

Python é uma linguagem orientada a objeto, sendo assim as estruturas de dados possuem atributos (propriedades) e métodos (rotinas associadas às propriedades). Tanto os atributos quanto os métodos são acessados usando ponto (.).

STRINGS

Funções Built-in de Strings

Os métodos estão sob a forma:

objeto.atributo
objeto.método()
objeto.método(parâmetros)

LISTAS

As listas são construídas com o uso de colchetes [] e vírgulas separando cada elemento da lista.

lista = [item1, item2, ..., itemz]

LISTAS

Se você estiver familiarizado com outra linguagem de programação, você pode traçar paralelos entre matrizes em outras linguagens e listas em Python. Listas em Python no entanto, tendem a ser mais flexíveis do que as matrizes em outras linguagens por dois bons motivos:

1. Listas não têm tamanho fixo (o que significa que não precisamos especificar quão grande uma lista será)

2. Listas não têm restrições de tipo fixo

LISTAS

Uma grande característica de estruturas de dados em Python é que elas suportam aninhamento. Isto significa que podemos usar estruturas de dados dentro de estruturas de dados.

Dicionários

Até aqui falamos bastante sobre sequências em Python, mas agora vamos mudar um pouco o foco e aprender sobre mapeamentos em Python. Se você estiver familiarizado com outras linguagens de programação, pode imaginar os dicionários como tabelas de hash (hash tables).

Os dicionários são construídos com o uso de chaves {} e vírgulas separando cada elemento do dicionário

dict = {k1:v1, k2:v2, ..., kn:vn}

Dicionários

Então, o que são mapeamentos?

Mapeamentos são uma coleção de objetos que são armazenados por uma chave, ao contrário de uma sequência de objetos armazenados por sua posição relativa.

Dicionários

Então, o que são mapeamentos?

Um dicionário Python consiste de uma chave e, em seguida, um valor associado.

Esse valor pode ser quase qualquer objeto Python.

Dicionários

Dicionários – mapeamento de chaves e valores

{chave1: valor1, chave2: valor2}

TUPLAS

Em Python, tuplas são muito semelhantes às listas, no entanto, ao contrário de listas, tuplas são imutáveis, o que significa que não podem ser alteradas. Você usaria tuplas para apresentar dados que não devem ser alterados, como os dias da semana ou datas em um calendário.

As tuplas são construídas com o uso de **parênteses ()** e vírgulas separando cada elemento da tupla.

tupla = (item1, item2,..., itemz)

TUPLAS

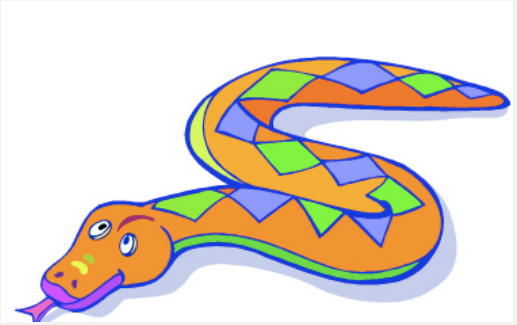
"Por que se preocupar usando tuplas, uma vez que trata-se de um objeto com limitações e um número menor de métodos disponíveis?"

TUPLAS

Tuplas não são utilizadas com frequência, como listas por exemplo, mas são usadas quando é necessário imutabilidade. Se em seu programa você precisa ter certeza de que os dados não sofrerão mudança, então tupla pode ser a sua solução. Ela fornece uma fonte conveniente de integridade de dados.

Booleanos

Expressões lógicas



Expressões lógicas

- *True* e *False* são constantes em Python
  - False : 0, None, [] , {} , 0.0
  - True: Valores Numéricos exceto 0, objeto não vazios
  - Um dicionário pode armazenar diferentes tipos de valores e é **mutável**!
- Operadores de comparação: ==, != , < , <=, etc.
  - X == Y (efetua teste de equivalência de valor)
  - X is Y (Testa a identidade do objeto)

Expressões lógicas

- *None* é similar ao NULL em linguagem C
  - L = [None] * 100 (declara uma lista de 100 items None)
- Operações com or e and
  - not -> inversão lógica (true -> false , false -> true)
  - and e or (&& e II)
    - **Casos especiais: Ele retorna o valor de uma das sub-expressões!
- isinstance(element,type)
  - Verifica se um elemento é do tipo type

Instruções compostas

```
If python == “cool”:
        print “Oh yeah!”
```

Fluxo de Controle

- Várias expressões Python para controlar o fluxo do programa. Todos eles fazem uso de testes condicionais booleanos.
  - ifs, else
  - loops while, for
  - assert

Instruções if

```
x = 5
if 0 < x and x < 12 or x == 5:
    print u"x é menor que uma dúzia!"

if 0 < x < 12:
    print u"x é menor que uma dúzia!"

if x in [ 1,2,3,4,5,6,7] :
    print u"x pertence da lista!"
elif x == 5 <= 10 > 0 < 1000:
    print u"x não está neste intervalo!"
else:
    print u"nenhuma das condições acima são verdadeiras!"

st = x < 7 and "reprovado" or "aprovado"
```

Instruções while

- Você pode usar o comando break para sair do loop mais próximo que a envolve.
- Você pode usar o comando continue para pular para o ínicio do loop mais próximo que a envolve e pular para a próxima iteração.
- Você pode usar o comando pass quando você não quer que se faça nada (instrução vazia)
- Você pode o o bloco else do loop para quando se quer executar um código quando se sai normalmente do loop (sem ser por comando break)

Instruções while

```
while x < 10:
    print x
    x+=1
    if x == 10:
        break

while not fim:
    fim = fim - 1

while True:
    pass
```

```
t = []
for x in xrange(10):
    if x % 2:
        print u"é par"
        continue
    else: t.append(x)
    print "%d é par" % x
```

Instruções for

- Loops for iteram sobre uma sequência de items (listas, tuplas, string ou quaisquer outros objetos cuja a linguagem considere como um “iterator”)
- Várias maneiras de iterar sobre um conjunto de items!
- Também possui o bloco else quando se sai normalmente do loop (similar ao while)
- Função muito usada nos loops for: range()
  - range() - Retorna uma lista de números que varia de 0 a ao número passado como parâmetro.
  - xrange() - Retorna uma lista como range() só que libera o item quando for requisitado! Mais eficiente, porém apenas com items do mesmo tipo e sem suporte à slicing, repetição e concatenação.

Instruções for

```
for n in [1,2,3,4,5]:
    print n

for m in ["teste","de","for"]:
    print m, len(m)

for s in range(10): print s**s

d = {"gol": 30000,"fusca":2500,"hilux":115000}
for k, v in d.items():
    print u"O preço do %s é %d" % (k, v)
```

```
>>>range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>range(5,25,7)
[5, 12, 19]
>>>range(-10,-50,-15)
[-10, -25, -40]
>>>xrange(10)
xrange(10)
>>>for 'a in xrange(10): print    a,
...
0 1 2 3 4 5 6 7 8 9
```

FIM