PROGNIT

Guia Python para Programadores

Novembro – 2016

Versão 1.0

Índice

Apresentação

Nosso curso visa atender a todas as pessoas que já programam em alguma linguagem de programação e desejam conhecer o lado prático da Análise de Sentimentos. Assim, todos os códigos do curso foram implementados de forma **simples, objetiva e sem elegâncias**. Entretanto, em um primeiro contato, a linguagem Python pode dificultar a compreensão das implementações realizadas. Dessa forma, este guia foi criado com o objetivo de orientar e auxiliar os/as alunos/alunas que não conhecem e/ou nunca programaram em Python. Ele contém as definições mais básicas da linguagem e **não é recomendado para pessoas que não conhecem nenhuma linguagem de programação.**

Introdução

Python é uma linguagem de programação interpretada, livre e que roda nos principais sistemas operacionais. Foi criada por Guido van Rossum e lançada em 1991. O nome da linguagem se refere ao grupo de comédia britânico Monty Python. Atualmente possui as versões 2.x e 3.x, onde a 3.x é a mais recente. Pode ser baixada pelo site https://www.python.org/.

Algumas de suas principais características são:

- A linguagem é case sensitive
- Os comandos não são finalizados com ponto e vírgula (;)
- A linguagem não possui marcadores de bloco (como as chaves {}, por exemplo).
- A identação é obrigatória
- Variáveis não precisam ser declaradas

Algumas das organizações que adotam Python são Google, Yahoo, Blender 3D, Walt Disney, ABN AMRO Bank, NASA, Nokia e IBM. Em [4] você pode conferir uma lista mais completa. Em [3] você tem acesso a algumas das aplicações de sucesso do Python.

O Guia Python

Selecionamos para este guia os principais aspectos e definições da linguagem Python para auxiliar na compreensão das implementações do nosso analisador de sentimentos. Este material teve como referência a documentação Python 3.x [1]. Os códigos aqui apresentados podem ser testados tanto no ambiente IDLE (Ambiente de desenvolvimento integrado do Python que acompanha a instalação básica) como no Anaconda/Spyder.

1. A função `print()`

A função print() é utilizada para apresentar valores na tela. Exemplo:

```
>>> print('Hello World')
>>> #O resultado das operações serão apresentados em azul
Hello World
```

2. Comentários

Comentários são feitos utilizando o caracter '#'. Exemplo:

```
#Isto é um comentário
```

3. Variáveis

Python utiliza tipagem dinâmica, logo as variáveis não precisam ser declaradas. Elas são iniciadas no primeiro momento em que são utilizadas. A convenção é iniciar o nome das variáveis com letras minúsculas.

3.1. Tipos de dados mais comuns

Os tipos de dados mais comuns são apresentados na Tabela 1:

Tipo de Dado	Exemplo
`int`	-2, 7, 102, -3, 54
`float`	-10.25, -1.0, 0.8, 7.5, 65.3
`String`	'Bom Dia', 'Hello!', 'PROGNIT Cursos'

Tabela 1: Tipos de dados mais comuns em Python

Exemplos:

```
>>> frase = 'Hello World'
>>> print(frase)
Hello World
```

```
>>> numero1 = 2016
>>> print(numero1)
2016
```

```
>>> numero2 = 3.56
>>> print(numero2)
3.56
```

4. Conversão de Tipos

Dado os três tipos básicos de variáveis, para conversão as funções utilizadas são `str()`, `int()` ou `float()`. Exemplos:

```
>>> #Converte um int para String
>>> str(3)
'3'
>>> #Converte um float para String
>>> str(5.76)
'5.76'
>>> #Converte um String para int
>>> int('10')
10
>>> #Converte um String para float
>>> float('8.65')
8.65
>>> #Converte um float para int
>>> int(34.76)
34
>>> #Converte um int para float
>>> float(48)
48.0
```

5. Operadores Aritméticos

Os operadores aritméticos são apresentados na Tabela 2:

Operador	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão Inteira
%	Módulo
**	Exponenciação

Tabela 2: Operadores Aritméticos em Python

Exemplo:

```
>>> #Atribuição de valores para as variáveis n1 e n2
>>> n1 = 5
>>> n2 = 3
>>> #Adição de valores
>>> r = n1 + n2
>>> print(r)
8
```

```
>>> #Subtração de valores
>>> r = n1 - n2
>>> print(r)
2
>>> #Multiplicação de valores
>>> r = n1 * n2
>>> print(r)
15
>>> #Divisão de valores
>>> r = n1 / n2
>>> print(r)
1.6666666666666667
>>> #Divisão inteira de valores
>>> r = n1 // n2
>>> print(r)
1
>>> #Exponenciação
>>> r = n1 ** n2
>>> print(r)
125
```

Se o operador + estiver entre duas Strings, ocorrerá a concatenação. Exemplo:

```
>>> s1 = 'PROGNIT '
>>> s2 = 'Cursos'
>>> print(s1 + s2)
PROGNIT Cursos
```

6. Entrada e Saída (I/O)

A função `print()` é utilizada para realizar a saída de dados na tela. A função `input()` é utilizada para receber dados do teclado, isto é, entrada de dados. Exemplo:

```
>>> print('Informe o nome de uma comida:')
Informe o nome de uma comida:
>>> comida = input()
#Valor que o usuário informou
feijoada
>>> print(comida)
>>> #Valor apresentado pela função print()
feijoada
```

7. Estruturas de Decisão

Em Python, a estrutura de decisão `if/else` ocorre na forma:

```
if condicao:
```

```
   bloco de código
```

Python não possui marcadores de bloco como outras linguagens ({}, por exemplo). Aqui a delimitação de bloco é determinada pela **identação**. Quanto mais interna a identação, mais interno é o bloco. Assim, a identação em Python é obrigatória.

Exemplo:

```
>>> n = 10
>>> if n > 0:
       #A identaçao do print indica que ele pertence ao bloco do if
       print('Valor positivo')

Valor positivo
```

Para uma segunda verificação de condição (`else if`), o comando em Python é `elif`. Para o `senão sem condição`, o comando é `else`. Após a condição deve-se utilizar dois pontos (`:`). Exemplo:

```
>>> if n > 0:
       print('Valor positivo')
    #elif é o else if de outras linguagens
    elif(n < 0):
       print('Valor negativo')
    else:
       print('Valor zero')

Valor positivo
```

7.1. Operadores Lógicos e Relacionais:

A Tabela 3 apresenta os operadores relacionais em Python:

Operador	Operação
==	Igualdade
! =	Diferença
<	Menor que
>	Maior que
<=	Menor ou igual
>=	Maior ou igual

Tabela 3: Operadores relacionais em Python

A Tabela 4 apresenta os operadores lógicos:

Operador	Significado
&&	E (and)
||	Ou (or)
!	Negação

Tabela 4: Operadores lógicos em Python

8. Estruturas de Repetição

Python possui os laços `while` e `for`. Um exemplo para o laço `while`:

```
>>> cont = 0
>>> #Após a condição utilizar dois pontos (:)
>>> while cont < 5:
        print('Dia de Estudos')
        cont = cont + 1

Dia de Estudos
Dia de Estudos
Dia de Estudos
Dia de Estudos
Dia de Estudos
```

O laço `for` é acompanhado da função `range()`. Esta função determina o intervalo de contagem do laço. Exemplo para fazer um laço com 5 iterações:

```
>>> #Utilizar os dois pontos (:) após a condição
>>> for c in range(5):
        print('Dia de Estudos')

Dia de Estudos
Dia de Estudos
Dia de Estudos
Dia de Estudos
Dia de Estudos
```

Você também pode determinar o início e fim do intervalo. Exemplo:

```
>>> #Em range(12, 16), 12 representa o início do intervalo e 16 o fim
>>> for c in range(12, 16):
        print('Valor: ', c)

Valor:  12
Valor:  13
Valor:  14
Valor:  15
```

9. Funções

Funções em Python são identificadas pela declaração `def`. Exemplo:

```
>>> #def nomedafuncao():
>>> def funcaoHello():
        print('Hello World')

#Chamada da função
funcaoHello()

Hello World
```

9.1. Parâmetros e argumentos:

Em Python, parâmetros são os nomes utilizados na definição das funções. Os argumentos são os valores que são passados na chamada da função. Parâmetros definem qual tipo de argumentos a função irá aceitar. Exemplo:

```
#msg é o parametro
>>> def funcaoHello(msg):
        print(msg)

#'Hello World' é o argumento na chamada da função
funcaoHello('Hello World')

Hello World
```

9.2. Retorno:

Retorno em Python é definido pela declaração `return`. Exemplo:

```
>>> #n1 e n2 sao os parametros
>>> def calcAdicao(n1, n2):
        #o retorno da função será a adição entre n1 e n2
        return n1 + n2

#5 e 2 sao os argumentos
r = calcAdicao(5, 2)
print('Resultado: ',r)

Resultado:  7
```

9.3 Função x Método:

Assim como outras linguagens, Python diferencia função de método. Ambos são blocos de código que executam uma determinada ação, entretanto os métodos estão sempre vinculados a um valor ou objeto. Exemplo de uma chamada de método para converter todos os caracteres de uma String para caixa baixa (`lower()`):

```
>>> valor = 'ANALISADOR'
>>> #lower() é um método, pois está vinculado ao objeto valor
>>> print(valor.lower())
analisador
```

10. Listas

Listas são tipos de dados que podem conter vários valores. Comparando com outras linguagens de programação, são semelhantes aos *arrays*. Listas são delimitadas por colchetes e seus itens separados por vírgulas. Exemplos:

```
>>> listaNomes = ['Ana', 'Beto', 'Carlos', 'Daniela']
>>> print(listaNomes)
['Ana', 'Beto', 'Carlos', 'Daniela']
```

```
>>> listaIdades = [30, 25, 34, 21]
>>> print(listaIdades)
[30, 25, 34, 21]
```

10.1. Recuperação de itens:

A recuperação de itens específicos de uma lista é feita utilizando colchetes e o índice do item (`[índice]`). Exemplo:

```
>>> print('Item 1 de listaNomes: ', listaNomes[1])
Item 1 de listaNomes:  Beto
>>> print('Item 2 de listaIdades: ', listaIdades[2])
Item 2 de listaIdades:  34
```

Python também aceita índices negativos. Eles representam a recuperação de itens em ordem decrescente. Assim, o índice -1 recupera o último item da lista, -2 o penúltimo, etc. Exemplo:

```
>>> print('Item -1 de listaNomes: ', listaNomes[-1])
Item -1 de listaNomes:  Daniela
```

```
>>> print('Item -2 de listaIdades: ', listaIdades[-2])
Item -2 de listaIdades:  34
```

Python também permite a recuperação de "fatias" da lista (*slices*). Para recuperar, por exemplo, o intervalo do índice 2 ao índice 5 de uma lista, os índices são escritos dentro dos colchetes e separados por dois pontos (`[2:5]`). O primeiro índice indica o início do recorte e o segundo o fim. Exemplo:

```
>>> listaValores = [23, 54, 21, 27, 91, 432, 2, 7, 10]
>>> print(listaValores[2:5])
[21, 27, 91]
```

Quando o intervalo da lista envolve a recuperação a partir do início da lista, o índice (`0`) pode ser omitido. Exemplo para recuperar o intervalo entre 0 e 4:

```
>>> print(listaValores[:4])
[23, 54, 21, 27]
```

O mesmo para recuperar itens de um determinado início até o final da lista. Exemplo para recuperar o intervalo de 3 a 9:

```
>>> print(listaValores[3:])
[27, 91, 432, 2, 7, 10]
```

10.2. Tamanho da Lista

Para saber o tamanho da lista, utilize a função `len()`. Exemplo:

```
>>> print(len(listaValores))
9
```

10.3. Concatenação de Listas

O operador + entre listas realiza a concatenação. Exemplo:

```
>>> novaLista = listaNomes + listaIdades
>>> print(novaLista)
['Ana', 'Beto', 'Carlos', 'Daniela', 30, 25, 34, 21]
```

10.4. Remover Itens da Lista

A declaração `del` apaga um item da lista. Exemplo:

```
>>> del novaLista[2]
>>> print(novaLista)
['Ana', 'Beto', 'Daniela', 30, 25, 34, 21]
```

10.5. Operações com Métodos em Lista

As operações nas listas também podem ser feitas com métodos. Para adicionar itens a uma lista, pode-se utilizar as opções `append()` ou `insert()`. Com `append()` o novo item é adicionado no final da lista. Exemplo:

```
>>> novaLista.append('Paulo')
>>> print(novaLista)
['Ana', 'Beto', 'Daniela', 30, 25, 34, 21, 'Paulo']
```

Com o `insert()` é possível determinar a posição onde o novo item será adicionado. Para inserir a String 'Maria' na posição 3:

```
>>> #inserir na posiçao 3 a String 'Maria'
>>> novaLista.insert(3, 'Maria')
>>> print(novaLista)
['Ana', 'Beto', 'Daniela', 'Maria', 30, 25, 34, 21, 'Paulo']
```

Com `remove()` é possível remover um determinado item:

```
>>> #remove o item 25
>>> novaLista.remove(25)
>>> print(novaLista)
['Ana', 'Beto', 'Daniela', 'Maria', 30, 34, 21, 'Paulo']
```

Com `sort()` é possível ordenar crescentemente a lista:

```
>>> listaSobrenomes = ['Silva', 'Santos', 'Ferreira', 'Carvalho']
>>> listaSobrenomes.sort()
>>> print(listaSobrenomes)
['Carvalho', 'Ferreira', 'Santos', 'Silva']
```

Para ordenar em ordem decrescente, passe o argumento `reverse=True`:

```
>>> listaSobrenomes.sort(reverse = True)
>>> print(listaSobrenomes)
['Silva', 'Santos', 'Ferreira', 'Carvalho']
```

Utilize `index()` para saber o valor do índice de um determinado item:

```
>>> print(listaSobrenomes.index('Santos'))
1
```

11. Importações

As importações em Python podem ser feitas, basicamente, de duas formas:

```
import <módulo>
```

ou

```
from <módulo> import <objeto, atributo, método>
```

No primeiro caso importa-se todo o conteúdo do módulo e a utilização de qualquer de seus recursos será feita por meio de referência ao nome do módulo. Como exemplo, vamos utilizar a geração de números aleatórios com o módulo `random`. Exemplo:

```
import random

#gera um inteiro aleatório pertencente ao intervalo (20, 100)
print (random.randint(20, 100))
86
```

O mesmo código, porém utilizando o segundo tipo de importação:

```
from random import randint

#não é necessário fazer referência ao módulo random
print (randint(20, 100))
25
```

Conclusão

Neste guia apresentamos as principais definições e características da linguagem Python. O objetivo não foi ensinar Python, mas auxiliar os estudos de alunos e alunas que não programam em Python a compreender as implementações do analisador de sentimentos. Em caso de dúvidas, publique-as no Fórum Guia Python para Programadores.

Referências

[1] https://docs.python.org/3.4/
[2] https://www.python.org/
[3] https://www.python.org/about/success/
[4] https://wiki.python.org/moin/OrganizationsUsingPython