Python Para Programadores

Versão 0.0.1

Carlos Maniero

set 27, 2017

Conteúdo

Você já programa em outra linguagem?

Aqui vamos mostrar um pouco do que você precisa saber da linguagem. A ideia não é ensinar programação, pelo contrário! Esse material é para quem já tem vivencia prévia com alguma linguagem e quer iniciar no mundo Python.

Esse é um conteúdo bastante sintetizado e cheio de referências externas onde você poderá aprender mais sobre a linguagem. É uma excelente fonte para quem já programa em Python e deseja entender mais a fundo a linguagem.

CAPÍTULO 1

Prefácio

Após algum tempo programando em Python, tive a oportunidades de trabalhar em diferentes projetos em diferentes empresas e, portanto, com pessoas diferentes. Embora Python não seja uma linguagem nova, o mercado de trabalho está começando a aquecer agora no Brasil. Portanto, é muito difícil você encontrar programadores Python com anos de experiência.

Por outro lado, Python é uma linguagem muito simples e em pouco tempo podemos ver bons programadores de outras linguagens programando perfeitamente em Python. E é isso que pretende esse livro, transformar programadores de outras linguagens em programadores Pythonicos (guarde esse adjetivo, você o verá muito aqui).

Motivação

Entre os projetos que eu participei e dei manutenção, percebi que uma das grandes dificudades de outros programadores é perder o sotaque da sua linguagem principal.

Algumas vezes eu acho engraçado a forma como programadores de outras linguagens pensam ao programar em Python - outras me fazem querer chorar -, muitas vezes chego até a me identificar e lembro que quando eu estava começando no mundo Python eu também queria trazer conceitos de outras linguagens, quando eu poderia resolver de forma muito mais simples se pensasse de forma Pythonica.

Para quem já programa Python, este livro pode ajudar a sanar dúvidas conceituais da linguagem e apresentar-lhes boas práticas para deixar o seu código Python ainda mais Pythonico.

Escolhi o GitHub integrado ao RTFD para que seja um conteúdo dinâmico. Capítulos podem surgir e deixar de existir a medida que issues e pull requests forem criados, então contribua. :)

Agradecimentos

Agradeço à minha noiva por ter acabado de fazer bolo de chocolate e café quente para me ajudar no processo de criação desse livro.

CAPÍTULO 2

Um conto de Natal

Era uma vez um jovem chamado Guido van Rossum [1], apenas um Holandês comum que, como qualquer ser humano comum, ficou extremamente entediado pois seu escritório estaria fechado durante as festas de fim de ano e decidiu criar uma linguagem de programação [2].

Afinal, quem nunca criou uma linguagem durante as festas de fim de ano só para matar o tédio?

O nome Python não está ligado ao réptil, mas sim ao grupo de humor Monty Python. Se você não conhece esse grupo, recomendo fortemente que assista pelo menos um vídeo deles. [3].

Guido era bastante ativo na criação de uma outra linguagem chamada ABC [4], cujo o intuito era ser simples e fácil de aprender. O Python herdou essa filosofia e surgiu com essa missão de ser uma linguagem democrática. De fato, funcionou, Python não é utilizada só por programadores, mas por biólogos [5], químicos, matemáticos e por muitos outros profissionais de diferentes áreas.

Foi assim que nasceu o Python, uma linguagem interpretada, de alto nível e, acima de tudo, simples.

[1] Site pessoal (https://gvanrossum.github.io/) com algumas curiosidades, como um áudio com a pronúncia do seu próprio nome.
[2] Foreword for "Programming Python" (1st ed.) (https://www.python.org/doc/essays/foreword/) (em inglês)
[3] Monty Python - A piada mais engraçada do mundo (https://www.youtube.com/watch?v=St5DY7h19tQ)
[4] ABC Handbook (http://homepages.cwi.nl/~steven/abc/programmers/handbook.html) (em inglês)
[5] Python For Biologists (http://pythonforbiologists.com/)

CAPÍTULO 3

Perguntas Frequentes

Quem vem de outra linguagem, já tem muito bem definido os processos e ferramentas que utilizam para desenvolver software. Quando eu digo que programo em Python, as reações são sempre muito similiares. Tudo começa com um sonoro:

```
- Nossa! Em Python?
```

E em seguida começa uma série de perguntas sobre a linguagem. Eu listei algumas delas aqui.

Python é compilada?

Não!

Python é uma linguagem interpretada. Você pode também escrever bibliotecas Python utilizando C ou C++ [1], isso não é muito comum, mas necessário em alguns casos.

Provavelmente, se você está acostumado com linguagens compiladas, uma das preocupações que você terá será em como distribuir a sua aplicação sem ceder o código-fonte.

Acho que essa não é uma preocupaçãoa muito relevante, a não ser que você esteja fazendo alguma aplicação para desktop, que será distribuída de forma orgânica e que você não terá controle sobre quem irá instalar a sua aplicação, você consegue resolver isso de forma jurídica, através de contratos.

Se sua aplicação, rodar em um servidor - uma aplicações web -, por exemplo, você pode limitar o acesso do seu cliente aos servidores. Hoje em dia, boa parte das linguagens modernas são interpretadas. Não vejo isso como um grande problema.

Qual IDE utilizar?

Digo mais uma vez que Python é uma linguagem extremamente simples e isso facilita um bocado na hora de programar e dar manutenção nos projetos.

[1] Extending Python with C or C++ (https://docs.python.org/2/extending/extending.html)

Por isso, não há uma real necessidade de uma IDE complexa e cheia de recursos para programar em Python. Na verdade, você ode programar em Python utilizando qualquer dispositívo em que seja possível bater meia dúzia de teclas.

Certa vez, esqueci o meu editor de texto aberto, minha gatinha passou sobre o teclado e quando fui ver, ela havia feito um "Hello World" em Python.

Eu particularmente, utilizo o Vim [2], há uma infinidade de editores bons. Vejo muitos programadores Python utilizando o Atom e o Sublime.

Existem também IDEs comerciais e cheia de recursos para quem está acostumado como Visual Studio (que inclusive tem uma versão que suporta Python [3]), Netbeans [4] e Eclipse [5], como o PyCharm [6], que acredito eu, seja a IDE mais completa para desenvolvimento Python.

Só roda no Linux?

Não!

Eu não sei onde nem quando surgiu esse mito de que Python é uma linguagem exclusiva para ambientes Linux, mas é apenas lenda urbana.

Até um tempo atrás, dependendo do que você fosse fazer, era muito melhor utilizar um sistema operacional Unix-like. Instalar uma biblioteca compilada no Windows era um suplício. Mas até esse tipo de biblioteca já está mais fácil de instalar hoje em dia.

Temos também ferramentas como o Docker, que utilizando o conceito de containers, permite que tenhamos um ambiente Linux dentro do Windows, Mac e o próprio Linux de forma muito mais simples do que se utilizassemos virtualização. É uma excelente prática para manter a compatibilidade com o ambiente em que o seu software irá rodar.

Nota: Docker é uma ferramenta poderosíssima e tem ganhado bastante destaque nas comunidades open sources.

Recomendo muito que estudem e apliquem em seus projetos. Existem excelentes ferramentas de orquestração como o Docker Compose (https://docs.docker.com/compose/) que facilitam e muito essa tarefa.

Dá pra desenvolver Web em Python?

Dá e como dá! Python é excelente para desenvolver Web, dos projetos que eu participei utilizando Python, eram em sua grande maioria, projetos web.

[2] Configurações do meu Vim (https://github.com/carlosmaniero/vim/)
[3] Ferramentas Python para Visual Studio (https://www.visualstudio.com/pt-br/features/python-vs.aspx)
[4] Python support in NetBeans (http://wiki.netbeans.org/Python) (em inglês)
[5] PyDev (http://www.pydev.org/) (em inglês)
[6] PyCharm (https://www.jetbrains.com/pycharm/) (em inglês)

Existe um set imenso de Frameworks [7] que nos ajudam nessa tarefa. Entre os mais populares estão o Django [8] e o Flask [9].

Padrão de Codificação

Assim como a maioria das linguagens, Python também possui um padrão de codigicação ela está é descrina na PEP8 [10]. Haverá um capítulo desse livro para falar a respeito disso.

Existem algumas coisas que para programadores de outras linguagens como Java podem aparecer absurdas. Mas como tempo, você não só se acostuma, mas consegue ver que aquele padrão torna seu código muito mais legível.

[7] Web Frameworks for Python (https://wiki.python.org/moin/WebFrameworks) (em inglês)
[8] Django (https://www.djangoproject.com/) - The web framework for perfectionists with deadlines. (em inglês)
[9] Flask (http://flask.pocoo.org/) Web development, one drop at a time. (em inglês)
[10] PEP 8 – Style Guide for Python Code (https://www.python.org/dev/peps/pep-0008/)

CAPÍTULO 4

Tipo de Dados

Python é uma linguagem de tipagem forte e dinâmica.

Listing 4.1: ex04-1

```
1 name = 'Eleven'      # a string
2 age = 12             # a int
3 power = 42.0         # a float
4
5 type_format = '{} is a {}'
6
7 print(type_format.format(name, type(name)))
8 print(type_format.format(age, type(age)))
9 print(type_format.format(power, type(power)))
```

Saída do programa:

```
Eleven is a <type 'str'>
12 is a <type 'int'>
42.0 is a <type 'float'>
```

Note que não é preciso em momento algum definir o tipo do dado. O Python detecta automáticamente o tipo da variável e atribui o tipo à mesma.

Para definir uma string basta colocar ela entre aspas simples (‘) ou duplas (”).

Dinâmicamente tipada

Você pode alterar o tipo de uma variável durante a execução do código e o Python não lançará nenhuma *Exception*.

Listing 4.2: ex04-2

```
1 name = 'Eleven'      # a string
2 age = 12             # a int
3 power = 42.0         # a float
4
```

```
5  type_format = '{} is a {}'
6
7  print(type_format.format(name, type(name)))
8  print(type_format.format(age, type(age)))
9  print(type_format.format(power, type(power)))
10
11 name = 11            # now name is a int
12
13 print(type_format.format(name, type(name)))
14 print('This is a stranger thing!')
```

Agora o name é do tipo **int** como podemos ver abaixo:

```
Eleven is a <type 'str'>
12 is a <type 'int'>
42.0 is a <type 'float'>
11 is a <type 'int'>
This is a stranger thing!
```

Nota: Alterar o tipo de uma variável, nunca é uma boa prática.

Fortemente tipada

Linguagens de tipagem fraca, permite que você faça operações algumas operações, sem a necessidade da realização do *cast*. Como por exemplo:

Listing 4.3: ex04-3

```
1  name = '11'
2
3  print(1 + name)
```

Isso não é o caso do Python, ao realizar isso. Ele retorna o seguinte *traceback*:

```
Traceback (most recent call last):
  File "type_error.py", line 3, in <module>
      print(1 + name)
      TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Deixando claro que não é possivel somar um inteiro à uma *string*.

Dica: Para fazer esses testes rápidos da linguagem sem precisar ficar criando arquivos, você pode simplesmente abrir o interpretador do python, digitando "python" no terminal (ou abrir o interpretador Python se estiver no Windows).

Uma excelente alternativa é o ipython [1], com ele é possível autocompletar como se tivesse utilizando uma IDE. Para instalar é simples:

```
$ pip install ipython
```

Feito isso, basta digitar ipython no terminal e você poderá utilizar o shell interativo do Python de forma muito mais dinâmica.

Vetores, Matrizes e Conjuntos

Python possui dois tipos de listas a do tipo *list* e a do tipo *tuple*. E qual a diferença entre um e outro?

Uma tupla é uma lista imutável. Então, você nunca consegue alterar o seu conteúdo.

Essa diferença fica clara quando você olha os métodos que uma lista e uma tupla possui.

list:

```
append, count, extend, index, insert, pop, remove, reverse, sort.
```

tuple:

```
count, index.
```

List

Em Python, o tamanho da lista é dinâmico bem como seu tipo. Então você não precisa definir o seu tamanho. Vejamos um exemplo de utilização de listas.

Listing 4.4: ex04-4

```
1  # Create a fruits lists
2  fruits = ['banana', 'apple']
3
4  print('Size of fruits:', len(fruits))
5  print('Minions loves', fruits[0])
6  print('Newton loves', fruits[1])
7
8  # Add new item to fruits list
9  fruits.append('orange')
10
11 print('Size of fruits:', len(fruits))
12 print(fruits[2], 'is the new black')
13
14 # Remove the first element
```

[1] Site oficial: <https://ipython.org>

```
15 print('I don\'t like of', fruits.pop(0))
16 print('Now minions loves', fruits[0])
```

```
Size of fruits: 2
Minions loves banana
Newton loves apple
Size of fruits: 3
orange is the new black
I don't like of banana
Now minions loves apple
```

Nota: Para alterar o valor de um elemento do uma lista, basta utilizar a seguinte sintaxe:

```
>>> fruits[0] = 'kiwi'
>>> print(fruits[0])
kiwi
```

Dica: Para excluir a referência de um objeto você pode usar o statement *del*. Por exemplo:

```
>>> a = 1
>>> del a
>>> print(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  NameError: name 'a' is not defined
```

Você pode utilizar também em listas.

```
>>> fruits = ['banana', 'apple']
>>> del fruits[0]
>>> print(fruits)
['apple']
```

A diferença entre o del e o pop do list, é que o pop irá retornar o elemento excluído.

Garbage Collector

Quando você exclui uma variável utilizando o *del*, você não está apagando o objeto, mas sim a sua referência.

Se você tiver duas variáveis apontando para o mesmo objeto, o objeto só vai sair da memória quando a última referência for excluída, assim o Garbage collector irá detectar que não há referências para o objeto e irá removê-lo da memória.

Tuple

Como dito anteriormente, a principal diferença entre tuplas e listas é que tuplas são imutáveis.

Vamos reescrever o exemplo *ex04-4* utilizando tuples.

Listing 4.5: ex04-5

```
1  # Create a fruits lists
2  fruits = ('banana', 'apple')
3
4  print('Size of fruits:', len(fruits))
5  print('Minions loves', fruits[0])
6  print('Newton loves', fruits[1])
7
8  # You will need create a new tuple
9  new_fruits = fruits + ('orange',)
10
11 print('Size of fruits:', len(fruits))
12 print('Size of new_fruits:', len(new_fruits))
13 print(new_fruits[2], 'is the new black')
14
15 # Remove the first element
16 print('I don\'t like of', fruits[0], 'but i can\'t remove this')
17 print('minions contine to love', fruits[0])
```

Saída do programa:

```
Size of fruits: 2
Minions loves banana
Newton loves apple
Size of fruits: 2
Size of new_fruits: 3
orange is the new black
I don't like of banana but i can't remove this
minions contine to love banana
```

Como podemos ver, na **linha 9** a única forma de adicionar um objeto à uma tupla é fazendo a união entre duas listas. Já na **linha 11**, podemos ver que a tupla original continua inalterada.

Nota: Caso você tente utilizar o statment *del* para tentar excluir um elemento de uma tupla, você receberá um sonoro exception. No próximo capítulo, veremos como funcionam os *slices* em Python que poderá nos ajudar nessa tarefa.

```
>> a = (1, 2)
>> del a[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: 'tuple' object doesn't support item deletion
```

Matrizes

Uma matriz em Python é uma lista de lista, ou tuplas de tuplas.

Listing 4.6: ex04-6

```
tic_tac_toe = (
    ['o'], ['x'], ['o']
    ['x'], ['o'], ['x']
    [' '], ['x'], ['o']
)

# win!
tic_tac_toe[2][0] = 'o'
```

No nosso exemplo, temos uma tupla de listas. A lista responsável pelas linhas do tic-tac-toe são imutáveis e deve ser sempre as mesmas listas, por isso a utilização de tuplas. As colunas em contra partida devem ser mutáveis, já que é preciso alterar o seu conteúdo.

Set

Conjuntos são similiares às listas, porém, como o nome mesmo diz, são conjuntos, ou seja, não aceita elementos duplicados.

Listing 4.7: ex04-7

```
values = {1, 2}
values.add(2)
values.add(2)
values.add(2)
values.add(2)
values.add(2)
print(values)
```

```
{1, 2}
```

Não importa quantas vezes você adicionar um elemento ao set. Se ele já existir, ele sempre será ignorado.

Nota: Para criar uma lista vazia, você basta abrir e fechar os colchetes **[]**. Seguindo a mesma lógica basta um par de parênteses **()**.

Mas isso não é válido para criar um conjunto vazio. O par de chaves, também é utilizado para criação de um dicionário (Calma! Vamos falar de dicionários disso logo abaixo). Então, para definir um conjunto vazio é necessário fazer de forma literal. Veja um exemplo abaixo:

```
>>> a = {}
>>> print(type(a))
<type 'dict'>
>>> a = set()
```

```
>>> print(type(a))
<type 'set'>
```

O construtor do *set* aceita como parâmetro um iterável [2]. A partir desse iterável será gerado um conjunto.

```
>>> set([1, 1, 2, 3, 3, 4])
{1, 2, 3, 4}
```

Dicionários

Um dict em Python, é uma estrutura de dado chave/valor. É muito simples trabalhar com ele. Veja um exemplo abaixo:

Listing 4.8: ex04-8

```
1  cartels = {
2      'escobar': 'Medellin',
3      'gustavo': 'Medellin',
4      'pacho': 'Cali',
5  }
6
7  print('The cartel of Pablo Escobar is', cartels['escobar'])
8  print('The cartel of Pacho is', cartels['pacho'])
9  print('The cartel of Gustavo is', cartels['gustavo'])
10
11 # History spoiler
12 print('Gustavo was killed')
13 cartels['gustavo'] = 'The Hell'
14
15 print('The cartel of Gustavo is', cartels['gustavo'])
16
17
18 cartels['miguel'] = 'Cali'
19 print('The cartel of Miguel is', cartels['miguel'])
```

```
The cartel of Pablo Escobar is Medellin
The cartel of Pacho is Cali
The cartel of Gustavo is Medellin
Gustavo was killed
The cartel of Gustavo is The Hell
```

Nas **linhas 1-6**, estamos definindo o dicionário, na **linha 14**, estamos mudando o cartel do Gustavo. Já na **linha 16** surge um novo personagem - o Miguel -, ele pertence ao cartel de Cali.

[2] Um iterável é uma lista ou uma tupla, por exemplo. Dedicaremos um capítulo inteiro do livro para explicar o que é um iterável.

Nota: A chave de um dicionário não precisa ser necessariamente uma *string*. Na verdade, a chave pode ser qualquer objeto.

O Python utiliza o sistema de hash (assunto do próximo episódio, digo, capítulo) para identificar se a chave solicitado está no dicionário. Caso não esteja, uma *exception* do tipo KeyError é lançada.

Dica: Assim como em listas, você pode excluir um valor utilizando o del ou pop.

```
>>> del cartes['escobar']
```

ou

```
>>> cartes.pop('escobar')
Medellin
```