Aprendendo Python em 10 minutos

Stavros Korokithakis, tradução Osvaldo Vargas Jaques

Considerações preliminares

Então, você quer aprender a linguagem de programação Python, mas não consegue encontrar um tutorial conciso e ainda completo. Este tutorial tentará ensinar-lhe Python em 10 minutos. Provavelmente não é um tutorial, pois é um cruzamento entre um tutorial e lista de dicas, então ele apenas mostrará alguns conceitos básicos para começar. Obviamente, se você quiser realmente aprender uma linguagem, você precisa programar nela por um tempo. Assumirei que você já está familiarizado com a programação e, portanto, ignorará a maioria das coisas que não são específicas da linguagem. As palavras-chave importantes serão destacadas para que você possa identificá-las facilmente. Além disso, preste atenção porque, devido ao estilo deste tutorial, algumas coisas serão introduzidas diretamente no código e apenas comentadas brevemente.

Propiedades

Python é fortemente tipado dinamicamente, tipicamente (por exemplo, você não precisa declarar variáveis), sensível a maiúsculas e minúsculas (isto é, var e VAR são duas variáveis diferentes) e orientado a objetos (ou seja, tudo é um objeto) .

Conseguindo Ajuda

Help in Python is always available right in the interpreter. If you want to know how an object works, all you have to do is call `help(<object>)`! Also useful are `dir()`, which shows you all the object's methods, and `<object>.__doc__`, which shows you its documentation string:

A ajuda no Python está sempre disponível no intérpretador ou prompt. Se você quer saber como um objeto funciona, tudo o que você precisa fazer é digitar `help(<object>)`! Também são úteis `dir()`, que mostra todos os métodos do objeto e `<object>.__doc__`, que mostra sua seqüência de documentação:

```
>>> help(5)
Help on int object:
(etc etc)

>>> dir(5)
['__abs__', '__add__', ...]

>>> abs.__doc__
'abs(number) -> number

Return the absolute value of the argument.
```

Syntaxe

O Python **não possui caracteres de término de declaração obrigatórios** e os **blocos são especificados por indentação.** Recuar para começar um bloco, e consequente fechar o anterior. Declarações que englobam níveis de indentação termina em dois pontos (:). Os **comentários** começam com o sinal de cerquilha (#) e são de uma única linha, **multiplas linhas de comentário**

utilizam """ como iníncio e finalizam com """ (reparem que começam e terminam com três aspas ou três apóstrofos e se começar com aspas termina com aspa e assim sucessivamente). Os **valores são atribuídos** (na verdade, os objetos são **vinculados** aos nomes) com o sinal de igual ("="), e o **teste de igualdade** é feito usando dois sinais iguais ("=="). Você pode incrementar / diminuir os valores usando os operadores + = e - =, respectivamente. Isso funciona em muitos tipos de dados, strings incluídas. Você também pode usar várias variáveis em uma linha. Por exemplo:

```
>>> myvar = 3
>>> myvar += 2
>>> myvar
5
>>> myvar -= 1
>>> myvar
4
"""This is a multiline comment.
The following lines concatenate the two strings."""
>>> mystring = "Hello"
>>> mystring += " world."
>>> print(mystring)
Hello world.
# This swaps the variables in one line(!).
# It doesn't violate strong typing because values aren't
# actually being assigned, but new objects are bound to
# the old names.
>>> myvar, mystring = mystring, myvar
```

Tipos de dados

As estruturas de dados disponíveis no python são **listas, tuplas e dicionários**. Os conjuntos estão disponíveis na biblioteca de conjuntos (mas são integrados no Python 2.5 e posterior). As listas são como arrays unidimensionais (mas você também pode ter listas de outras listas), os dicionários são matrizes associativas (tabelas de hash) e as tuplas são matrizes unidimensionais imutáveis (os "arrays" de Python podem ser de qualquer tipo, para que você possa misturar, por exemplo, inteiros, string, etc. em listas / dicionários / tuplas). O índice do primeiro item em todos os tipos de array é 0. Os números negativos contam desde o final para o início, -1 é o último item. As variáveis podem apontar para funções. O uso é o seguinte:

```
>>> sample = [1, ["another", "list"], ("a", "tuple")]
>>> mylist = ["List item 1", 2, 3.14]
>>> mylist[0] = "List item 1 again" # We're changing the item.
>>> mylist[-1] = 3.21 # Here, we refer to the last item.
>>> mydict = {"Key 1": "Value 1", 2: 3, "pi": 3.14}
>>> mydict["pi"] = 3.15 # This is how you change dictionary values.
>>> mytuple = (1, 2, 3)
>>> myfunction = len
>>> print(myfunction(mylist))
3
```

Você pode acessar uma **sequência de array** usando dois pontos (:). Deixar o índice inicial vazio assume o primeiro item, deixando o índice final assume o último item. A indexação é exclusiva, portanto, especificar [2:10] retornará os itens [2] (o terceiro item, por causa da indexação de 0) para [9] (o décimo item), inclusivo (8 itens). Os índices negativos contam do último item para trás (portanto, -1 é o último item), assim:

```
>>> mylist = ["List item 1", 2, 3.14]
```

```
>>> print(mylist[:])
['List item 1', 2, 3.1400000000000001]
>>> print(mylist[0:2])
['List item 1', 2]
>>> print(mylist[-3:-1])
['List item 1', 2]
>>> print(mylist[1:])
[2, 3.14]
# Adding a third parameter, "step" will have Python step in
# N item increments, rather than 1.
# E.g., this will return the first item, then go to the third and
# return that (so, items 0 and 2 in 0-indexing).
>>> print(mylist[::2])
['List item 1', 3.14]
```

Strings

Strings podem usar **aspas simples (apóstrofos) ou duplas** e você pode ter aspas de um tipo dentro de uma seqüência de caracteres que usa o outro tipo (ou seja, "Ele disse 'Olá'." É válido). As citações com múltiplas línhas são delimitadas por três aspas ou apóstrofos. Python **suporta Unicode** fora dos apóstrofos ou aspas, usando a sintaxe `u"Esta é uma string unicode"`. Para **preencher uma string com valores**, você usa o %(formato) operador e uma tupla. Cada % é substituído por um item da tupla, da esquerda para a direita e você também pode usar substituições de dicionário, assim:

```
>>>print("Name: %s\
Number: %s\
String: %s" % (myclass.name, 3, 3 * "-"))
Name: Poromenos
Number: 3
String: ---

strString = """This is
a multiline
string."""

# WARNING: Watch out for the trailing s in "%(key)s".
>>> print("This %(verb)s a %(noun)s." % {"noun": "test", "verb": "is"})
This is a test.
```

Controle de fluxo

As instruções de controle de fluxo são `if`, `for` e `while`. Não há `switch`; em vez disso, use `if`. Use `for` para enumerar os membros de uma lista. Para obter uma lista de números, use o `range(<number>)`. A sintaxe dessas declarações é:

```
rangelist = range(10)
>>> print(rangelist)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for number in rangelist:
    # Check if number is one of
    # the numbers in the tuple.
    if number in (3, 4, 7, 9):
        # "Break" terminates a for without
        # executing the "else" clause.
        break
```

```
    else:
        # "Continue" starts the next iteration
        # of the loop. It's rather useless here,
        # as it's the last statement of the loop.
        continue
else:
    # The "else" clause is optional and is
    # executed only if the loop didn't "break".
    pass # Do nothing

if rangelist[1] == 2:
    print("The second item (lists are 0-based) is 2")
elif rangelist[1] == 3:
    print("The second item (lists are 0-based) is 3")
else:
    print("Dunno")

while rangelist[1] == 1:
    pass
```

Funções

As funções são **declaradas com** a palavra-chave **def**. Os a**rgumentos opcionais** são definidos na declaração de função após os argumentos obrigatórios ao serem atribuídos um valor padrão. Para argumentos **nomeados,** o nome do argumento é associado a um valor. As funções podem retornar uma tupla (e usando a tupla você pode efetivamente retornar **múltiplos valores**). As **funções lambda** são funções ad hoc que são compostas por uma única declaração. Os parâmetros são passados **por referência**, mas os tipos imutáveis (tuplas, ints, strings, etc.) *não podem ser alterados pelo destinatário*. Isso ocorre porque **apenas o local da memória do item é passado** e **vincula outro objeto a uma variável descarta o antigo**, então os tipos imutáveis são substituídos. Por exemplo:

```
# Same as def funcvar(x): return x + 1
funcvar = lambda x: x + 1
>>> print(funcvar(1))
2

# an_int and a_string are optional, they have default values
# if one is not passed (2 and "A default string", respectively).
def passing_example(a_list, an_int=2, a_string="A default string"):
    a_list.append("A new item")
    an_int = 4
    return a_list, an_int, a_string

>>> my_list = [1, 2, 3]
>>> my_int = 10
>>> print(passing_example(my_list, my_int))
([1, 2, 3, 'A new item'], 4, "A default string")
>>> my_list
[1, 2, 3, 'A new item']
>>> my_int
10
```

Classes

O Python suporta uma forma limitada de **herança múltipla** nas classes. **Variáveis privadas e métodos** podem ser declarados (por convenção, isto não é aplicado pela linguagem), adicionando

pelo menos dois sobretraços (por exemplo, `__spam`). Também podemos vincular **nomes arbitrários** a instâncias de classe. Um exemplo segue:

```
class MyClass(object):
    common = 10
    def __init__(self):
        self.myvariable = 3
    def myfunction(self, arg1, arg2):
        return self.myvariable

    # This is the class instantiation
>>> classinstance = MyClass()
>>> classinstance.myfunction(1, 2)
3
# This variable is shared by all instances.
>>> classinstance2 = MyClass()
>>> classinstance.common
10
>>> classinstance2.common
10
# Note how we use the class name
# instead of the instance.
>>> MyClass.common = 30
>>> classinstance.common
30
>>> classinstance2.common
30
# This will not update the variable on the class,
# instead it will bind a new object to the old
# variable name.
>>> classinstance.common = 10
>>> classinstance.common
10
>>> classinstance2.common
30
>>> MyClass.common = 50
# This has not changed, because "common" is
# now an instance variable.
>>> classinstance.common
10
>>> classinstance2.common
50

# This class inherits from MyClass. The example
# class above inherits from "object", which makes
# it what's called a "new-style class".
# Multiple inheritance is declared as:
# class OtherClass(MyClass1, MyClass2, MyClassN)
class OtherClass(MyClass):
    # The "self" argument is passed automatically
    # and refers to the class instance, so you can set
    # instance variables as above, but from inside the class.
    def __init__(self, arg1):
        self.myvariable = 3
        print(arg1)

>>> classinstance = OtherClass("hello")
hello
>>> classinstance.myfunction(1, 2)
3
# This class doesn't have a .test member, but
# we can add one to the instance anyway. Note
```

```
# that this will only be a member of classinstance.
>>> classinstance.test = 10
>>> classinstance.test
10
```

Exceções

Exceções em Python são manipuladas com os blocos **try-except [exceptionname]** :

```
def some_function():
    try:
        # Division by zero raises an exception
        10 / 0
    except ZeroDivisionError:
        print("Oops, invalid.")
    else:
        # Exception didn't occur, we're good.
        pass
    finally:
        # This is executed after the code block is run
        # and all exceptions have been handled, even
        # if a new exception is raised while handling.
        print("We're done with that.")

>>> some_function()
Oops, invalid.
We're done with that.
```

Importação de pacotes e bibliotecas

Bibliotecas externas são usados com `import [nome da biblioteca]`. Também podemos usar `from [libname] import [funcname]` para funções individuais. Aqui está um exemplo:

```
import random
from time import clock

randomint = random.randint(1, 100)
>>> print(randomint)
64
```

Entrada e Saída (File I/O)

O Python possui uma grande variedade de bibliotecas incorporadas. Como exemplo, temos a **serialização** (conversão de estruturas de dados em strings usando o pacote `pickle`) onde a E / S de arquivos é usada:

```
import pickle
mylist = ["This", "is", 4, 13327]
# Open the file C:\\binary.dat for writing. The letter r before the
# filename string is used to prevent backslash escaping.
myfile = open(r"C:\\binary.dat", "w")
pickle.dump(mylist, myfile)
myfile.close()

myfile = open(r"C:\\text.txt", "w")
```

```
myfile.write("This is a sample string")
myfile.close()

myfile = open(r"C:\\text.txt")
>>> print(myfile.read())
'This is a sample string'
myfile.close()

# Open the file for reading.
myfile = open(r"C:\\binary.dat")
loadedlist = pickle.load(myfile)
myfile.close()
>>> print(loadedlist)
['This', 'is', 4, 13327]
```

Miscelâneas

- **Condicionais que podem ser usadas**: `1 < a < 3` verifica se a é menor que 3 e maior que 1.
- Podemos usar `del` para **apagar variáveis ou elementos em arrays**.
- **List comprehensions** fornecem uma maneira poderosa de criar e manipular listas. Eles consistem em uma expressão seguida por uma cláusula `for` seguida de zero ou mais cláusulas `if` ou `for` , tais como

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [3, 4, 5]
>>> print([x * y for x in lst1 for y in lst2])
[3, 4, 5, 6, 8, 10, 9, 12, 15]
>>> print([x for x in lst1 if 4 > x > 1])
[2, 3]
# Check if a condition is true for any items.
# "any" returns true if any item in the list is true.
>>> any([i % 3 for i in [3, 3, 4, 4, 3]])
True
# This is because 4 % 3 = 1, and 1 is true, so any()
# returns True.

# Check for how many items a condition is true.
>>> sum(1 for i in [3, 3, 4, 4, 3] if i == 4)
2
>>> del lst1[0]
>>> print(lst1)
[2, 3]
>>> del lst1
```

- **Variáveis Global** são declaradas fora das funções e pode ser lidas sem nenhuma declaração especial, mas se quisermos alterá-las devemos declará-las no início da função com a palavra-chave `global`, caso contrário, o Python irá vincular esse objeto a uma nova variável local (Tenhamos cuidado com isso, podemos nos dar mal). Por exemplo:

```
number = 5

def myfunc():
    # This will print 5.
    print(number)
```

```
def anotherfunc():
    # This raises an exception because the variable has not
    # been bound before printing. Python knows that it an
    # object will be bound to it later and creates a new, local
    # object instead of accessing the global one.
    print(number)
    number = 3

def yetanotherfunc():
    global number
    # This will correctly change the global.
    number = 3
```

Epílogo

Este tutorial não pretende ser uma lista exaustiva de tudo (ou mesmo de um subconjunto) de Python. Python possui uma vasta gama de bibliotecas e muito mais funcionalidades que você terá que descobrir por outros meios, como o excelente livro Dive into Python. Espero ter facilitado a sua transição em Python.