# Tutorial - Learn Python in 10 minutes

NOTE: If you would like some Python development done, my company, Stochastic Technologies, is available for consulting.

This tutorial is available as a short ebook. The e-book features extra content from follow-up posts on various Python best practices, all in a convenient, self-contained format. **All future updates are free for people who purchase it.**

## Preliminary fluff

So, you want to learn the Python programming language but can't find a concise and yet full-featured tutorial. This tutorial will attempt to teach you Python in 10 minutes. It's probably not so much a tutorial as it is a cross between a tutorial and a cheatsheet, so it will just show you some basic concepts to start you off. Obviously, if you want to really learn a language you need to program in it for a while. I will assume that you are already familiar with programming and will, therefore, skip most of the non-language-specific stuff. The important keywords will be highlighted so you can easily spot them. Also, pay attention because, due to the terseness of this tutorial, some things will be introduced directly in code and only briefly commented on.

Stavros Korokithakis

Man

Greek. Amateur F1 driver. Technology enthusiast. Single parent. Liar.

Founder of Stochastic Technologies, a software development agency, and creator of various products which you can find in the résumé.

# Properties

Python is strongly typed (i.e. types are enforced), dynamically, implicitly typed (i.e. you don't have to declare variables), case sensitive (i.e. var and VAR are two different variables) and object-oriented (i.e. everything is an object).

# Getting help

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:

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

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

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

Return the absolute value of the argument.'
```

# Syntax

Python has no mandatory statement termination characters and blocks are specified by indentation. Indent to begin a block, dedent to end one. Statements that expect an indentation level end in a colon (:). Comments start with the pound (#) sign and are single-line, multi-line strings are used for multi-line comments. Values are assigned (in fact, objects are bound to names) with the *equals* sign ("="), and equality testing is done using two *equals* signs ("==). You can increment/decrement values

using the += and -= operators respectively by the right-hand amount. This works on many datatypes, strings included. You can also use multiple variables on one line. For example:

```
>>> 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 valu
es aren't
# actually being assigned, but new objects are
bound to
# the old names.
>>> myvar, mystring = mystring, myvar
```

# Data types

The data structures available in python are lists, tuples and dictionaries. Sets are available in the `sets` library (but are built-in in Python 2.5 and later). Lists are like one-dimensional arrays (but you can also have lists of other lists), dictionaries are associative arrays (a.k.a. hash tables) and tuples are immutable one-dimensional arrays (Python "arrays" can be of any type, so you can mix e.g. integers, strings, etc in lists/dictionaries/tuples). The index of the first item in all array types is 0. Negative numbers count from the end towards the beginning, -1 is the last item. Variables can point to functions. The usage is as follows:

```
>>> sample = [1, ["another", "list"], ("a", "tu
```

```
>>> sample = [1, ["another", "list"], ("a", "tu
ple")]
>>> mylist = ["List item 1", 2, 3.14]
>>> mylist[0] = "List item 1 again" # We're cha
nging the item.
>>> mylist[-1] = 3.21 # Here, we refer to the l
ast item.
>>> mydict = {"Key 1": "Value 1", 2: 3, "pi": 3
.14}
>>> mydict["pi"] = 3.15 # This is how you chang
e dictionary values.
>>> mytuple = (1, 2, 3)
>>> myfunction = len
>>> print myfunction(mylist)
3
```

You can access array ranges using a colon (:). Leaving the start index empty assumes the first item, leaving the end index assumes the last item. Negative indexes count from the last item backwards (thus -1 is the last item) like so:

```
>>> 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 Py
thon step in
# N item increments, rather than 1.
# E.g., this will return the first item, then g
o to the third and
# return that (so, items 0 and 2 in 0-indexing)
.
>>> print mylist[::2]
['List item 1', 3.14]
```

## Strings

Its strings can use either single or double quotation marks, and you can have quotation marks of one kind inside a string that uses the other kind (i.e. "He said 'hello'." is valid). Multiline strings are enclosed in *triple double (or*

*single) quotes* (""""). Python supports Unicode out of the box, using the syntax u"This is a unicode string". To fill a string with values, you use the % (modulo) operator and a tuple. Each %s gets replaced with an item from the tuple, left to right, and you can also use dictionary substitutions, like so:

```
>>>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 "%(k
ey)s".
>>> print "This %(verb)s a %(noun)s." % {"noun"
: "test", "verb": "is"}
This is a test.
```

# Flow control statements

Flow control statements are `if`, `for`, and `while`. There is no `switch`; instead, use if. Use for to enumerate through members of a list. To obtain a list of numbers, use `range(<number>)`. These statements' syntax is thus:

```
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 loo
```

```
p.
        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
```

## Functions

Functions are declared with the "def" keyword. Optional arguments are set in the function declaration after the mandatory arguments by being assigned a default value. For named arguments, the name of the argument is assigned a value. Functions can return a tuple (and using tuple unpacking you can effectively return multiple values). Lambda functions are ad hoc functions that are comprised of a single statement. Parameters are passed by reference, but immutable types (tuples, ints, strings, etc) *cannot be changed*. This is because only the memory location of the item is passed, and binding another object to a variable discards the old one, so immutable types are replaced. For example:

```
# 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 d
efault 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

Python supports a limited form of multiple inheritance in classes. Private variables and methods can be declared (by convention, this is not enforced by the language) by adding at least two leading underscores and at most one trailing one (e.g. "__spam"). We can also bind arbitrary names to class instances. An example follows:

```
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 clas
s,
# 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 mak
es
# 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 automatical
ly
    # and refers to the class instance, so you
can set
    # instance variables as above, but from ins
ide 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 classinsta
nce.
>>> classinstance.test = 10
>>> classinstance.test
10
```

# Exceptions

Exceptions in Python are handled with try-except [exceptionname] blocks:

```
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 ha
ndling.
        print "We're done with that."

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

## Importing

External libraries are used with the `import [libname]` keyword. You can also use `from [libname] import [funcname]` for individual functions. Here is an example:

```
import random
from time import clock

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

## File I/O

Python has a wide array of libraries built in. As an example, here is how serializing (converting data structures to strings using the `pickle` library) with file I/O is used:

```
```

```
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]
```

# Miscellaneous

- Conditions can be chained. `1 < a < 3` checks that a is both less than 3 and greater than 1.
- You can use `del` to delete variables or items in arrays.
- List comprehensions provide a powerful way to create and manipulate lists. They consist of an expression followed by a `for` clause followed by zero or more `if` or `for` clauses, like so:

```
>>> 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
```

- Global variables are declared outside of functions and can be read without any special declarations, but if you want to write to them you must declare them at the beginning of the function with the "global" keyword, otherwise Python will bind that object to a new local variable (be careful of that, it's a small catch that can get you if you don't know it). For example:

```
number = 5

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

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

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

# Epilogue

This tutorial is not meant to be an exhaustive list of all (or

even a subset) of Python. Python has a vast array of libraries and much much more functionality which you will have to discover through other means, such as the excellent book Dive into Python. I hope I have made your transition in Python easier. Please leave comments if you believe there is something that could be improved or added or if there is anything else you would like to see (classes, error handling, anything).

By the way, you should follow me on Twitter.



## Subscribe to my mailing list

Did you like what you just read and want to be notified when I post more? Subscribe to my mailing list to get updates on my posts and other random goodies.

Email | First name

Subscribe



## Join the conversation!