Python Cheatsheet

Brad Sturt

September 9, 2011

1 Introduction

The following is meant to be used as reference for individuals who already have some experience coding with Python Programming Language. It is organized by common themes, but there is plenty more that could be added to each theme. Please visit `www.python.org` for more detailed documentation. Additionally, this reference is based heavily on "Learning Python" by Mark Lutz. I highly recommend it to anyone who is interested in learning Python throughly.

Python is a programming language that deserves a pretty good introduction. But in the interest of time, I am skipping that and going right to the info you want to reference. The sections are ordered by **Object Types**, **Loops**, **Functions**, and **Classes**.

2 Object Types

There are 6 primitives (objects built into the language) that should be mentioned. They are:

- Integers
- Strings
- Lists
- Dictionaries
- Tuples
- Files

2.1 Integers

Numeric Literals:

Integers:	-5, 24, 0
Floating Points:	1.23, 1.23e-20, -1.23E210
Octal:	0177, 0011
Hexadecimal:	0xFF, 0x3E89
Complex:	3+4j, 0+.4j

Converting Types Manually:

int(3.141)	becomes 3
float(3)	becomes 3.0
oct(64)	becomes 0100
hex(64)	becomes 0x40

math Module

math.pi	math.sqrt()	math.e
math.sin()	math.cos()	math.sin()

random Module:

random.random()	returns a random number in [0, 1)
random.randint(1, 10)	returns a random number in [1, 10)
random.choice([1, Hello Math 198 Student", 'Goodbye': 42])	random item in list

Operations:

Operation	What it returns
123 + 122	145
1.5 * 4	6.0
2 ** 4	16
4 / 3	1 (note the rounding)
3.0 / 2	1.5

2.2 Strings

String Examples

Type These In	This Returns	This Is Called
s = “spam’s”		Double quotes
’Hello’ + ’ world’	’Hello world’	Concatenation
’Hello’ * 3	’HelloHelloHello’	Repeat
s = ’Spam’		
s[2]	’a’	Returns the character at index 2
s[-1]	’m’	Returns the character at (-1 + 4) = 3rd index
s[1:4]	’pam’	Slice (returns characters at indexes [1, 4)
s[2:]	’am’	Slice (returns characters at index 2 and on)
s[:2]	’Sp’	Slice (returns characters from inexes [0, 2)
s[0:3:2]	’Sa’	Slice (returns every other (2) letters from indexes [0, 3))
len(s)	4	Length
word = “Hypergraphics”		String Formatting
“Hello %s” % word”	’Hello Hypergraphics’	

String Methods

S.capitalize()	S.center(width)	S.count(sub [, start [, end]])
S.encode([encoding [,errors]])	S.endswith(suffix [, start [, end]])	S.expandtabs([tabsize])
S.find(sub [, start [, end]])	S.index(sub [, start [, end]])	S.isalnum()
S.isalpha()	S.isdigit()	S.islower()
S.isspace()	S.istitle()	S.isupper()
S.join(seq)	S.ljust(width)	S.lower()
S.lstrip()	S.replace(old, new [, maxsplit])	S.rfind(sub [, start [, end]])
S.rindex(sub [, start [, end]])	S.rjust(width)	S.rstrip()
S.split([sep [, maxsplit]])	S.splitlines([keepends])	S.startswith(prefix [, start [, end]])
S.strip()	S.swapcase()	S.title()
S.translate(table [, delchars])	S.upper()	

String Backslash Characters

\newline	Ignored (continuation)
\\	Backslash (keeps a \)
\’	Single quote (keeps ’)
\”	Double quote (keeps ”)
\a	Bell
\b	Backspace
\f	Formfeed
\n	Newline (linefeed)
\r	Carriage Return
\t	Horizontal tab
\v	Verticle tab
\xhh	Hex digits value
\ooo	Octal digits value
\0	Null (doesn’t end string)

2.3 Lists

List Examples

Types These In	This Returns	This Is Called
l1 = []		Empty List
l2 = ['Francis', [42, 'Hypergraphics']]		Nested lists
l2[0]	'Francis'	Returns item at index 0
l2[1]	[42, 'Hypergraphics']	Returns item at index 1
l2[1][0]	'Hypergraphics'	Returns item at index 1's 0th index
l2[0, 2]	['Francis', [42, 'Hypergraphics']]	Slice list from indexes [0, 2)
L2 = L2 + [1]	['Francis', [42, 'Hypergraphics'], 198, 1]	Concatanate lists
L1 = [1, 2, 3]		
L1 * 3	[1, 2, 3, 1, 2, 3, 1, 2, 3]	Repeat

List Methods

Types These In	This Returns	This Is Called
myList = [1, 2]		
len(myList)	2	Number of entries in list
myList.append('1a2b')		Sets myList to [1, 2, '1a2b']
myList.extend([3, 4])		Sets myList to [1, 2, '1a2b', 3, 4]
myList.sort()		Sorts L2 by ASCII characters
myList.insert(2, 'Interrupting')		Inserts 'Interrupting' into the 2nd index. Everything in the list after gets pushed back
myList.index(2)	'1a2b'	another way of saying L2[2]
myList.reverse()		Reverse list
myList.pop()	4	Removes last item and returns it
myList.remove('1a2b')		Removes '1a2b' from list

2.4 Tuples

Tuples are virtually identitical to lists, with one major difference. Tuples cannot be changed. No items can be removed or added to the tuple. If you want to do so, you must make a new tuple that will store the new changes. So why do they exist? Since they are *immutable*, they require less resources of your computer so they can be created faster and in greater numbers. Also, if you are just trying to store information without changing that information, tuples just make more sense. Tuples use () instead of list's [].

2.5 Dictionaries

Dictionary Examples

Type These In	This Returns	This Is Called
myDict = {'foo': 4, 'bar': 2}		A dictionary!
myDict['foo']	4	Fetch value with key 'foo'
myDict['Francis'] = ['Math', 'Programmer']		Add key:value to dictionary
len(myDict)	3	Length
myDict.has_key('Brad')	False	does myDict have key 'Brad'
myDict.keys()	['foo', 'bar', 'Francis']	returns list of keys
myDict.values()	[4, 2, ['Math', 'Programmer']]	returns list of values
myDict.items()	[('foo', 4), ('bar', 2), ('Francis', ['Math', 'Programmer'])]	returns dictionary as a list of tuples for each item
myDict.get('foo')	4	same as myDict['foo']

2.6 Files

If you have used C/C++ for working with files, you will find the syntax here to be very similar. If you haven't, no worries.

1. Write to a new file:

`f = open('myfile.txt', 'w')`	open file to write
`f.write('Hello \n')`	write a string to buffer
`f.writelines(['Hello', 'Programmer'])`	write lines of strings from a list to buffer
f.close()	flush buffer to disk, or, in English, *save the file*

2. Read from an existing file:

`f = open('myfile.txt', 'r')`	open file to read
myString = f.read()	read entire file to myString
myString = f.read(10)	read next 10 characters
myString = f.readline()	read next line
myList = f.readlines()	read entire file into list of strings

3. pickle Module:

   What is it: pickle is a module that writes objects to a file and retrieve it from the file kinda easily

   (a) Pickle object to file:

`myFile = open('myfile.txt', 'w')`	open file to write
import pickle	import pickle module
pickle.dump(someObject, myFile)	pickle someObject to myFile
pickle.close()	flush buffer to disk

   (b) Load object from file

`myFile = open('myfile.txt', 'r')`	open file to write
import pickle	import pickle module
obj = pickle.load(myFile)	retrieve someObject from myFile and store it to obj
pickle.close()	flush buffer to disk

What are the alternatives: Check out the shelve module for a pickle-like parsing-alternative with indexing. For databases, check out ZODB module. Python also can script with SQL.

3 Loops and Other Statements

These loops and statements are found commonly in code. In here are:

- for Loops
- if/elif/else Loops
- Miscellaneous

3.1 for Loops

```
for <target> in <object>: # for each <target> in <object>
something # note the indent

Example:
for i in range(10): # for each i in [0, 1, ..., 9]
print i ** 2

Output:
0
1
4
9
16
25
36
49
64
81

What did this do? It did
i = 0: print i ** 2
i  = 1: print i ** 2
...
i = 9: print i ** 2

Example:
for char in 'I love Hypergraphics':
print char, # the comma means not to skip a line
print ' ',

Output:
I   l o v e   H y p e r g r a p h i c s
```

3.2 if/elif/else Loops

```
if <True or False expression>: # if expression in <> is true
something # do what is indented below it

Example:
if 1 < 2:
```

```
print 3

Output:
3

Example:
if 1>2:
print 3
else:
print 4

Output:
4

Example:
x = 1
y = 2
if x < y:
print 'x < y'
elif x == y: # else, if x has the same value as y...
print 'x equals y'
else:
print 'x > y'

Output:
x < y
```

3.3 Miscellaneous

x or y	x and y	not x
zip(['a', 'b', 'c'], [1, 2, 3])	returns [('a', 1), ('b', 2), ('c', 3)]	zip function
dict([('a', 1), ('b', 2), ('c', 3)])	returns {'a':1, 'b':2, 'c':3}	dict function
range(5)	returns [0, 1, 2, 3, 4]	range function