Python for Scientific Data Analysis

Basic Python

Symbols

Python has a lot of symbols. Understanding how a language works and then being able to look up detailed syntax is nice. But you should commit some things to your own personal *"RAM"*.

Keywords

Keyword	Description	Example
`and`	Logical and.	True and False == False
`as`	Part of the with-as statement.	with X as Y: pass
`assert`	Assert (ensure) that something is true.	assert x == "hello" # (if x is not actually "hello" then you get an error)
break	Stop this loop right now.	while True: break
class	Define a class.	class Person(object)
continue	Don't process more of the loop, do it again.	while True: continue
def	Define a function.	def X(): pass
del	Delete from dictionary.	del X[Y]
elif	Else if condition.	if: X; elif: Y; else: J
else	Else condition.	if: X; elif: Y; else: J
except	If an exception happens, do this.	except ValueError, e: print e

exec	Run a string as Python.	exec 'print "hello"'
finally	Exceptions or not, finally do this no matter what.	finally: pass
for	Loop over a collection of things.	for X in Y: pass
from	Importing specific parts of a module.	from x import Y
global	Declare that you want a global variable.	global X
if	If condition.	if: X; elif: Y; else: J
import	Import a module into this one to use.	import os
in	Part of for-loops. Also a test of X in Y.	for X in Y: pass also 1 in [1] == True; or for i in range(0,5):
is	Like == to test equality.	1 is 1 == True
lambda	Create a short anonymous function.	s = lambda y: y ** y; s(3)
not	Logical not.	not True == False
or	Logical or.	True or False == True
pass	This block is empty.	def empty(): pass
print	Print this string.	print('this string')
raise	Raise an exception when things go wrong.	raise ValueError("No")
return	Exit the function with a return value. def X(): return Y	
try	Try this block, and if exception, go to except.	try: pass

while	While loop.	while X: pass
with	With an expression as a variable do.	with X as Y: pass
yield	Pause here and return to caller.	def X(): yield Y; X().next()

Data Types

For data types, write out what makes up each one. For example, with strings write out how you create a string. For numbers write out a few numbers.

Type	Description	Example
True	True boolean value.	True or False == True
False	False boolean value.	False and True == False
None	Represents "nothing" or "no value".	x = None
strings	Stores textual information.	x = "hello"
numbers	Stores integers.	i = 100
floats	Stores decimals.	i = 10.389 #note: floating point is in double precision
lists	Stores a list of things.	j = [1,2,3,4]
dicts	Stores a key=value mapping of things.	e = {'x': 1, 'y': 2}

String Escape Sequences

For string escape sequences, use them in strings to make sure they do what you think they do.

Escape	Description	Example
\	Backslash	print("how\dy!"); how\dy!
\'	Single-quote	print("how\'dy!"); how'dy!
"	Double-quote	print("how"dy!"); how"dy!
\a	Bell #makes a sound	print("howdy\a!"); makes sound
\b	Backspace	print("how\by!"); hoy!
\f	Formfeed	print("how\fdy!"); how; dy! #where 'd' is just a line down from where it would otherwise be
\n	Newline	print("how\ndy"); how; dy!; #'h' and 'd' are aligned
\r	Carriage	print("h\rowdy!"); owdy!
\t	Tab	print("how\tdy!"); how dy!
\v	Vertical tab	print("howd\vy!"); howd; y!

String Formats

String formatting is highly annoying. Even if you cannot commit these to memory, consider them as a cheat sheet.

Escape	Description	Example
%d Decimal integers (not floating point).	"%d" % 45 == '45'	
%i Same as %d.	"%i" % 45 == '45'	
%o Octal number.	"%o" % 1000 == '1750'	
%u Unsigned decimal.	"%u" % -1000 == '-1000'	
%x Hexadecimal lowercase.	"%x" % 1000 == '3e8'	
%X Hexadecimal uppercase.	"%X" % 1000 == '3E8'	
%e Exponential notation, lowercase 'e'.	"%e" % 1000 == '1.000000e+03'	
%E Exponential notation, uppercase 'E'.	"%E" % 1000 == '1.000000E+03'	
%f Floating point real number.	"%f" % 10.34 == '10.340000'	
%F Same as %f.	"%F" % 10.34 == '10.340000'	
%g Either %f or %e, whichever is shorter.	"%g" % 10.34 == '10.34'	
%G Same as %g but uppercase.	"%G" % 10.34 == '10.34'	
%c Character format.	"%c" % 34 == '"'	
%r Repr format (debugging format).	"%r" % int == ""	
%s String format.	"%s there" % 'hi' == 'hi there'	
%% A percent sign.	"%g%%" % 10.34 == '10.34%'	

Operators

Some of these may be unfamiliar to you, but look them up anyway. Find out what they do, and if you still can't figure it out, save it for later.

Operator	Description	Example
+	Addition	2 + 4 == 6
-	Subtraction	2 - 4 == -2

*	Multiplication	2 * 4 == 8
**	Power of	2 ** 4 == 16
/	Division	2 / 4.0 == 0.5
//	Floor division	2 // 4.0 == 0.0
%	String interpolate or modulus	2 % 4 == 2
<	Less than	4 < 4 == False
>	Greater than	4 > 4 == False
<=	Less than equal	4 <= 4 == True
>=	Greater than equal	4 >= 4 == True
==	Equal	4 == 5 == False
!=	Not equal	4 != 5 == True
( )	Parenthesis	len('hi') == 2
[ ]	List brackets	[1,3,4]
{ }	Dict curly braces	{'x': 5, 'y': 10}
@	At (decorators)	@classmethod
,	Comma	range(0, 10)
:	Colon	def X():
.	Dot	self.x = 10
=	Assign equal	x = 10
;	semi-colon	print("hi"); print("there")
+=	Add and assign	x = 1; x += 2
-=	Subtract and assign	x = 1; x -= 2
*=	Multiply and assign	x = 1; x *= 2
/=	Divide and assign	x = 1; x /= 2

//=	Floor divide and assign	x = 1; x //= 2
%=	Modulus assign	x = 1; x %= 2
**=	Power assign	x = 11; x **= 2; x=121