Python in half an hour

by Jinkyu Koo

In what follows in this tutorial, the result of a statement is expressed in the form of #result#, unless otherwise mentioned.

- Basic arithmetic operations

```
# Comments start with a hash sign (#).

1 + 2   # Addition
1 / 2   #0# Division
1 / 2.  #0.5# If either one of the numbers in a division is a float, so does the result.
3 % 2   #1# Modulus
2 ** 3  #8# Power. Equivalent to pow(2,3).
```

- Variables

```
# A valid name of a variable is an alpha-numeric string
# consisting of one or more letters, digits or underscore characters,
# just like in C.

x = 3           # Assignment
x + 1           #4#
x = None        # None is is a built-in constant that represents the absence of a value.
x, y = 1, 2     # Multiple assignments at a time
```

- Import modules

```
import math
math.sqrt(4)

# If you do not want to write the module name each time you call the function
from math import *
sqrt(4)

# If you are sure that you only need sqrt() in the math module
from math import sqrt
sqrt(4)

# Module alias
import math as m
m.sqrt(4)

# Function alias
from math import sqrt as sq
sq(4)
```

- Strings

```
# Strings can be expressed with double quotes (") or single quotes (').
# There is no difference bettwen double quotes and single quotes.
x = "abc"
x = 'abc'   # Exactly the same as above
y = "ab'cd'ef"  # 'cd' is part of the string.
z = 'ab"cd"ef'  # "cd" is part of the string. The strings y and z are not the same.

# String formatting
# Use C-like conversion specifiers.
'test %s %d %.1f' % ('abc', 10, 10.33)   #'test abc 10 10.3'#

# String methods
'  /sys/fs  '.strip()    #'/sys/fs'# Remove whitespaces on the left and the right.
'/sys/fs'.split('/')     #['', 'sys', 'fs']#
```

- Lists

```
# Lists start with [ and end with ].
# Elements are separated by a comma (,).
x = [1, 2, 3]

# A list can hold different types of elements.
y = [1, 2, "abc", [5,6]]

# List methods
z = ['b', 'a']
z.append('c')    #['b','a','c']#
z.index('b')     #0# Returns the index of the first occurrence of a value
z.sort()         #['a','b','c']#
z.remove('a')    # Removes an element that appears first.
```

- Tuples

```
# Tuples start with ( and end with ).
# Tuples are just like lists with an exception that they cannot be changed.
x = (1, 2)

# Values separated by commas automatically becomes a tuple.
x = 1, 2     # The same as above.

# A tuple with a single element
x = (1,)     # Here the comma is important. (1) is just 1.
```

- Sequence commons: strings, lists, and tuples are called sequences.

```
x = '123'
y = [1, 2, 3]
z = (1, 2, 3)

# Sequences can be indexed as follows:
x[0]         #'1'#
y[0:2]       #[1,2]# Returns a list with elements such that 0<=index<2.
z[1:]        #(2,3)# Return a tuple with elements such that 1<=index.
x[:2]        #'12'# Returns a string with elements such that index<2.
y[-1]        #3# Returns the last element.
z[-2]        #2# Returns the second last element.
y[:]         #[1, 2, 3]# Returns a list with all elements.

# Addition and multiplication for sequences
# Two objects of the same type can be added.
x + '4'      #'1234'#
y + [4, 5]   #[1, 2, 3, 4, 5]#
z + (4,)     #(1, 2, 3, 4)#

# Multiplication can be understood as multiple additions.
y*3          #[1, 2, 3, 1, 2, 3, 1, 2, 3]# Regard it as y+y+y.

# Pairing elements of two sequences
x = [1, 2, 3]
y = [4, 5, 6]
zip(x, y)    #[(1, 4), (2, 5), (3, 6)]#

# * operator: argument unpacking
z = [x, y]
zip(*z)      #[(1, 4), (2, 5), (3, 6)]# The same as zip(x,y)
```

- Dictionaries

```
# A dictionary starts with { and ends with },
# and it is like a hash table that maps a key to a value.

# An element is defined as a pair of a key and a value.
d = {'key1':'value1', 'key2':'value2'}
d = {}                    # An empty dictionary
d['key3'] = 'value3'      # Adds a key-value pair.
```

```
d['key3']               #'value3'# Returns a value for the key

# The key must be immutable.
d['abc'] = 3          # OK
d[(a,b)] = [1,2]      # OK
d[3] = 'abc'          # OK
d[[1,2,3]] = 2        # Not OK, since a list is mutable.

# Dictionary methods
d.get(key)       # Returns a value corresponding to the key.
d.has_key(key)   # Returns True if the key in the dictionary, and False otherwise.
d.items()        # Returns a list of (key, value) tuple pairs.
d.keys()         # Returns a list of keys.
d.values()       # Returns a list of values.
d.copy()         # Return a dictionary object that has exactly the same contents as d.
```

- Print to screen

```
# Note that what is shown here valid in Python 2.7 only.
# Syntax for print is quite changed in Python 3.4.
x = 1
y = [1,2]
print 3                  #3#
print 3, "abc"           #3 abc#
print "%d %d" % (3, 4)   #3 4#
print x,                 #1# Trailing comma suppresses newline.
print str(y)             #[1, 2]# str() transforms an object into a string.
print repr(y)            #[1, 2]# Most of times, repr() results in the same as str().
```

- Assignment is by reference!

```
# One thing to remember
# when you assign a list or a dictionary (i.e., mutable objects) to another variable:
# Assignment is always by reference, not by copy.
x = [1, 2, 3]
y = x            # Here, y get a reference to what x points to, i.e., [1, 2, 3]
y[0] = 4
x                #[4, 2, 3]# Note that x is changed by y.

# To copy a list
y = x[:]         # Creates a new list that contains all elements of x.
y[0] = 4
x                #[1 ,2 ,3]#

# To copy a dictionary
x = {1: 1}
y = x
z = x.copy()     # Use the copy() method.
y[1] = 2
z[1] = 3
print str(x), str(y), str(z)     #{1: 2} {1: 2} {1: 3}#
```

- Conditionals

```
if x == 5:           # Note that there is a colon (:) at the end.
    # VERY IMPORTANT: all statements in a block must be indented by the same amount.
    # Otherwise, you will see an error.
    x += 1
    y = x + 2
    print x, y
elif x in [4,5,6]:  # True if x is one of 4, 5, or 6.
    if x != 2:      # Conditions can be nested.
        print x
else:
    # Empty blocks are not allowed.
    # Put 'pass' for a placeholder when you want do nothing.
    pass
```

- The for-loops

```
x = [1, 2, 3]
for v in x:          # Iterates the elements of x from the first to the last.
    print v,         #1 2 3#

range(3)             #[0, 1, 2]# Returns a list of integers from 0 to 3-1.
range(1,4)           #[1, 2, 3]# Returns a list of integers from 1 to 4-1.

for i in range(3):
    print x[i],      #1 2 3#
```

- The while-loops

```
x = 5
while x > 0:
    x -= 1
    if x == 3:
        continue     # continue as in C
    if x == 1:
        break        # break as in C
    print x,         #4 2#
```

- List comprehension: making a list from other list

```
x = [i*2 for i in range(5)]
y = [i*2 for i in range(5) if i%2 == 0]
x       #[0, 2, 4, 6, 8]#
y       #[0, 4, 8]#
```

- Iterating over a dictionary

```
d = {1: 'a', 2: 'b', 3: 'c'}
for k, v in d.items():
    print str(k) + ':' + v,    #1:a 2:b 3:c#

# The following is the same as above.
key = d.keys()
value = d.values()
for k, v in zip(key, value):
    print str(k) + ':' + v,    #1:a 2:b 3:c#
```

- Functions

```
# Writing your own functions
def foo(arg1):       # Do not forget the colon at the end.
    x = arg1 + 1
    return x         # You can omit the return statement if there is nothing to return.

def bar(arg1, arg2 = 3):     # arg2 gets its value as 3 by default.
    x = arg2 + 1
    return arg1[0], x        # Here, arg1 can be of any type except a literal constant.

q, w = bar('abc')        # You can omit the argument that has its default value.
print q, w               #a 4#
print bar('abc')         #('a', 4)# arg1 is a string.
print bar('abc', 4)      #('a', 5)# Put something to change the default value.
print bar([1, 2])        #(1, 4)# arg1 is a list.
print bar((5, 6))        #(5, 4)# arg1 is a tuple.
print bar(7)             # An error

# Make use of argument unpacking operator (*).
x = (1, 2)
def add(a, b):
    return a+b

print add(*x)               #3# Just add(x) will cause an error.

# A function can have a different name.
```

```
my_add = add
my_add(1,2)                 #3#

# Lambda expressions
add2 = lambda arg1, arg2: arg1+arg2
add2(1,2)                   #3#

# Lambda expressions may be used when you pass an argument that is a function.
a = [(4,2),(1,3)]
sorted(a, key=lambda x:x[0])    #[(1, 3), (4, 2)]# Sort by the first element of tuples
sorted(a, key=lambda x:x[1])    #[(4, 2), (1, 3)]# Sort by the second element of tuples

# To change global variables within a function
x = 1
def foo(a):
    global x        # Without this, x won't be changed.
    x += a
    return
```

- Exceptions

```
# If something bad happens within a try block, an exception is raised.
def foo(x):
    y = 1
    try:
        y = y/x
    except ZeroDivisionError:   # You can specify the type of an exception to deal with.
        print "divided by zero"
    except:                     # All exceptions other than the above are handled here.
        print "something else"

foo(0)                          #divided by zero#
foo('1')                        #something else#

# Raise exceptions and catch exception objects.
import traceback                    # To use print_exc()
try:
    raise Exception("my exception") # Raise an exception with some argument.
except Exception as e:              # The 'Exception' is the base of all exception
                                    # objects, so it can catch all exceptions.
    if e.args == ("my exception",): # You can check what argument is in.
        print "my exception occurs"
    print e.args[0]                 #my exception# Print the argument of the exception.
    # Print exception information by which you can locate the culprit.
    traceback.print_exc()

# The else-clause: executed when there is no problem in the try block.
while True:
    default = '1'
    menu = 1
    try:
        # Take an input from a user
        # Just typing the enter key will cause the default value to be chosen.
        t = raw_input("Enter a number (default="+default+"): ") or default
        menu = int(t) % 10
        print 'You selected ' + str(menu)
    except:
        print 'Invalid input'    # e.g., character inputs will come to here.
    else:
        break       # Executed if no exceptions are raised in the try block.
```

- Reading from and writing to a file

```
fr = open('text.txt', 'r')      # Open text.txt to read.
fw = open('result.txt', 'w')    # Open result.txt to write.
for line in fr.readlines():     # Iterate text.txt line by line.
    fw.write(line + ' some')    # Write a string as a a line
fr.close()                      # Do not forget to close files
fw.close()
```

- Some useful built-in functions

```
x = [1, 0, 2]
len(x)              #3# Returns a length of a sequence.
max(x)              #2$ Returns the largest element.
min(x)              #0# Returns the smallest element.
sorted(x)           # Returns a list of sorted elements of a sequence.
sum(x)              #3# Returns the sum of all elements.

# The 'in' operator returns True if a value is in a sequence and False otherwise.
2 in x              #True#

# To delete an element of a list or a dictionary
del x[1]            # Here, 1 is an index.

d = {1:2, 3:4}
del d[1]            # Here, 1 is a key.

# Convert a string or a number to an integer or an floating number.
int('12')           #12#
int(12.3)           #12#
float("3.3")        #3.3#
```

Practice 1. Drawing a graph

You may need to additionally install numpy and matplotlib modules to plot a graph. In Ubuntu, the easiest way to get them is to type:

```
$ sudo apt-get install python-numpy python-matplotlib
```

```
import numpy as np
import matplotlib.pyplot as plt

x1 = [1, 4, 8]
y1 = [0.5, 2, 4]

x2 = [1, 2, 3]
x2_array = np.array(x2)      # np.array() makes an array object.
y2_array = x2_array ** 2     # The array object enables MATLAB-like element-wise operations

# plt.plot() takes lists or arrays as its data arguments.
plt.plot(x1, y1, x2_array, y2_array)

# The following commands are self-explanatory.
plt.title('title')
plt.xlabel('label for x-axis')
plt.ylabel('label for y-axis', fontsize=15)
plt.grid()
plt.legend(['x/2', 'pow(x,2)'])
plt.xlim(0, 9)
plt.ylim(0, 10)
plt.savefig('fig1.png', format='png')    # Try pdf, eps, ... almost all you can imagine.


# Whenever you want another figure
plt.figure()

plt.subplot(211)
plt.plot(x1, y1, 'r--')

plt.subplot(212)
plt.plot(x2_array, y2_array, 'b.', markersize=30)
plt.xlim(0, 4)
plt.ylim(0, 10)

plt.savefig('fig2.png')
```

Practice 2. 3D plotting

To plot a function in a 3D space, you can mimic the following example. For more detail, visit:
http://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D


x = y = np.arange(-5.0, 5.0, 0.05)
# np.meshgrid() returns coordinate matrices from coordinate vectors.
X, Y = np.meshgrid(x, y)
# An example function that outputs a value corresponding to a 2D vector.
zf = lambda x, y: 3*pow(1-x, 2)*pow(2,-x**2-(y+1)**2) \
    -pow(2,-(x+1)**2-y**2)/3 \
    -10*(x/5-x**3-y**5)*pow(2,-x**2-y**2)
# np.ravel() returns a flattened array.
zs = np.array([zf(x,y) for x,y in zip(np.ravel(X), np.ravel(Y))])
Z = zs.reshape(X.shape)


fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_surface(X, Y, Z)

# Plots a marker at the point (0,2,zf(0,2)).
ax.scatter(0,2,zf(0,2), c='r', marker='o', s=500)

# Labels
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')


fig = plt.figure()

# When you want subfigure features
ax = fig.add_subplot(121, projection='3d')
# Different texture for surface
ax.plot_surface(X, Y, Z, rstride=8, cstride=8, alpha=0.3)

# Plots contours
cset = ax.contour(X, Y, Z, zdir='x', offset=-5, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='y', offset=5, cmap=cm.coolwarm)
cset = ax.contour(X, Y, Z, zdir='z', offset=-20, cmap=cm.coolwarm)


ax = fig.add_subplot(122, projection='3d')
# cmap defines a colormap for the surface patches.
ax.plot_surface(X, Y, Z, cmap=cm.jet)


plt.show()
```

Practice 3. Redirect a text stream to your Python script

Sometimes you may want to process the output of Linux commands within Python scripts by redirecting. Use the following example script in such a case. Say you write the codes in `find_keyword.py` and you want to catch the lines that contain 'kworker' from /proc/kmsg. Then, type:

```
$ sudo cat /proc/kmsg | python find_keyword.py kworker file1.txt
```

```
import sys

if (len(sys.argv) != 3):
    print '\n Usage example:'
    print ' When you want to find lines from /proc/kmsg that contain "kworker",'
    print ' print them to screen, and store them into file1.txt,'
    print ' sudo cat /proc/kmsg | python '+sys.argv[0]+' kworker file1.txt\n'
    sys.exit(1)

fo = open(sys.argv[2], "w")

while True:
    try:
        line = sys.stdin.readline() # Reads a line from stdin
    except KeyboardInterrupt:       # until a user hits ctrl+c
        break

    if not line:                    # or until there is nothing left to read
        break

    if sys.argv[1] in line:
        print line
        fo.write(line + '\n')

fo.close()
```

Practice 4. Regular expressions

A regular expression specifies a set of strings that matches it. You may have to spend non-trivial time to be familiar with all pattern syntaxes of regular expressions. Here, we show just a few of use cases. For more detail, refer to:

https://docs.python.org/2/library/re.html

```
import re

text1 = 'Fig. 1: initially there are 60 points in each class.'
text2 = 'My phone number is 123-456-7890.'
text3 = 'MemFree:           50116 kB'
lines = [text1, text2, text3]

# See if text1 contains 'where'.
m1 = re.search('where', text1, flags=0)
print repr(m1)    #None#
if m1:            # None is equivalent to False in a condition.
    print "This won'n be printed."


# See if text1 is a string that contains 'there' somewhere in it.
m2 = re.search('.*there.*', text1, flags=0)
# '.*' means any character of 0 or more occurrences.

if m2:             # If matched, m2 is not None.
    # m2.group() is the whole string that matches the pattern '.*there.*'.
    print m2.group()


# Use parentheses for grouping
m3 = re.search(r'.*([A-Z]).*there.*\s(\d+).*', text1, flags=0)
# Why 'r' before the opening quote? Check raw strings!
# Without the 'r', all backslashes must be twice-typed.
# To avoid confusion, patterns in Python code are usually expressed in raw string notation.
# '[A-Z]' means an alphabet between A and Z,
# '\s' a white space, and
# '\d+' a digit of 1 or more occurrences.

if m3:
    print m3.group()
    print m3.group(1)    #F# The string captured within the first parentheses.
    print m3.group(2)    #60# The string captured within the second parentheses.


# See if text2 contains a phone number
m4 = re.search(r'\d{3}-\d{3}-\d{4}', text2, flags=0)
if m4:
    print m4.group()     #123-456-7890#


# Parse the number that corresponds to Memfree from a list of strings.
for line in lines:
    m5 = re.search(r'MemFree:\s+(\d+)\s+kB', line, flags=0)
    if m5:
        print m5.group(1)     #50116#
```

Practice 5. Classes

Python provides all the standard features of object oriented programming by classes:

```
class Person:
    # Class variables defined in this way
    # are shared by all instances,
    # like static member variables in C++.
    cnt = 0

    # The initializer method, like a constructor in C++.
    def __init__(self, name_):
        # Class variables defined in this way
        # are unique to each instance.
        self.name = name_

        # Access like a static member variable in C++.
        Person.cnt += 1

    # Class methods
    def showCount(self):
        print "The number of Persons are %d." % (Person.cnt)

    def showName(self):
        print "My name is %s." % (self.name)


# Creating instances
p1 = Person('James')
p2 = Person('Matt')
p1.showName()        #My name is James.#
p2.showName()        #My name is Matt.#
p1.showCount()       #The number of Persons are 2.#

# Class members are normally all public in C++ terminology.
print Person.cnt, p1.name, p2.name      #2 James Matt#


# Add or remove attributes of class instances.
p1.age = 10
del p1.age


# Inheritance
class Student(Person):  # Inherits from Persion
    def __init__(self, name_, grade_):
        self.grade = grade_
        Person.__init__(self, name_)
    def showGrade(self):
        print "Hmm.. My grade is %s." % (self.grade)

    # All methods in Python are virtual in C++ terminology:
    # Derived classes override methods of the same name defined in their base classes.
    def showName(self):
        # This is how to extend rather than simply replace
        # the base class method of the same name.
        Person.showName(self)
        print 'And I am a student.'

s = Student('Aaron', 'A+')
s.showName()         #My name is Aaron.\nAnd I am a student.#
s.showGrade()        #Hmm.. My grade is A+.#
```