AICT Computing Series Spring 2011 Workshop

Practical Python 2

charlene.nielsen@ualberta.ca
Monday, May 2nd, 2011
1:00 to 4:00 p.m.
ETLC 5-013

Through hands-on exercises, you will get a taste of some of the many standard library modules while learning functions, error handling, and debugging in Python - an interpreted, object-oriented, high-level programming language that is used in a wide variety of application domains (www.python.org).

This is a continuation of the winter introductory workshop, but attendance at the previous workshop is not required if you have any programming experience or read through the archived materials (www.ualberta.ca/AICT/RESEARCH/Courses/archivecourses.html). For a more engaging learning experience, you will use a variety of sample applications while developing complete programs from start to finish.

"If you're using a programming language named after a sketch comedy troupe, you had better have a sense of humor." www.python.org

Tasks

Computer access in ETLC 5-013

1. Click the middle button to choose session number **5** "WINDOWS XP"
2. Click the CHANGE SESSION button
3. Log-in with your CCID and password
   (and click ACCEPT to the warning)

4. To access the internet: AUTHENTICATE using your CCID and password

Windows XP

1. In a web browser, download the following to C:\TEMP: **bit.ly/pp2zip**
2. Right-click the **START** button to choose EXPLORE

3. Navigate to the TEMP directory on the C drive
4. Extract all/Unzip **pp2.zip** in to C:\Temp\pp2
5. Open **pp2_tasks_s2011.pdf** (this document)

IDLE for Python in ETLC 5-013

1. Click START > PROGRAMS > ACTIVESTATE ACTIVEPYTHON 2.6 (32-bit) > IDLE (PYTHON GUI)
2. Click HELP > IDLE HELP for a quick recap on the integrated development environment application that installs with Python; close when finished reading
3. Click FILE > NEW WINDOW to open the file editor where you will type the workshop's coding samples and scripts as shown below as plain text and screen captures from IDLE
4. Type a few meaningful comments at the top of your script file, similar to below (choose your preference for commenting using the '#' character OR the triple-quoted string):

```
# Date: yyyymmdd
# By: your.name@ualberta.ca
# Description: This script is helping me learn Python
```

5. Click FILE >>> SAVE as C:\Temp\pp2\scripts\scriptname.py (do NOT forget the **.py**)
6. Click RUN >>> RUN MODULE (or press the F5 key). Click OK to SAVE. View the results in the interactive window, cross-referencing with the code in the script window.
7. You are strongly encouraged to access Python's built-in and online **HELP**.

Tip: Remember the useful trick of highlighting lines of your program and then using the FORMAT menu to COMMENT OUT REGION and UNCOMMENT REGION. The INDENT REGION and DEDENT REGION will come in handy when you start using loops and other constructs requiring blocks of code.

Script samples

Start a new file for each of the scripts below. Make notes (comments in the code will do!) if you think the instructor says something important during this hands-on demonstrations of practical Python programming. Especially note the built-in modules highlighted from the standard library.

Various bits of the Python language are highlighted, along with the thought process of how to end up with a simple program that does what you want. Regions of code have been commented out. The instructor will guide you through typing via trail an error, along with tips and tricks, to help you realize the complete program, plus with occasional variations. The samples alone are not complete lessons. Tip: Zoom in on the text to see it better (small font optimizes page-fitting).

list_rename_files.py

```
# list files of specific type in an input folder and rename
# http://docs.python.org/release/2.6.6/library/glob.html
# http://www.doughellmann.com/PyMOTW/glob

import glob
import os

# variables
input_folder = 'C:\\Temp\\pp2\\files'
old_text = '20'
new_text = ''

# get listing of only TIFF file types
file_type = '\\*.tif'
data_folder = input_folder + file_type
data_list = glob.glob(data_folder)

for each in data_list:
    print each
    new_file = each.replace(old_text, new_text)
    print new_file
    os.rename(each, new_file)
```

NOTE: The code samples may not work as is! They are meant to be used in guided hands-on instruction.

descriptive_stats.py

```
# calculate descriptive statistics

from math import sqrt

# list of values
numbers = [235, 689, 568, 654, 639, 779, 301, 655, 299, 443]

# calculate using built-in functions
minval = min(numbers)
maxval = max(numbers)
n = len(numbers)
summed = sum(numbers)
mean = sum(numbers) / n
##mean = sum(numbers) / float(n)

### calculate standard deviation - should be 192.02343607
sumsq = 0
for number in numbers:
    diffsq = (number - mean) ** 2
##    print 'diffsq', diffsq
    # sum the difference of square
    sumsq += diffsq
##    print 'sumsq', sumsq
# complete calculation
stdev = sqrt(sumsq / (n - 1))
print stdev

print 'Minimum:', minval
print 'Maximum:', maxval
print 'Sum:', summed
print 'Mean:', mean
##print 'Standard Deviation', stdev
```

random_coordinates.py

```
# make random coordinates within rectangle extent and zip to tuples
# http://docs.python.org/release/2.6.6/library/random.html
# http://www.doughellmann.com/PyMOTW/random

import random

# variables
number = 100
xmin = 170844
xmax = 865133
```

NOTE: The code samples may not work as is! They are meant to be used in guided hands-on instruction.

```
ymin = 5425575
ymax = 6659344

# initialize lists
xlist = []
ylist = []

# generate values using a loop
for i in xrange(1,number):
    # demo this without random. module prefix
    x = randint(xmin, xmax)
    y = randint(ymin, ymax)
##    x = random.randint(xmin, xmax)
##    y = random.randint(ymin, ymax)
    xlist.append(x)
    ylist.append(y)

print xlist
print ylist

### zip to tuple
##coords = zip(xlist, ylist)
##print coords
### loop through tuple and print
##for each in coords:
##    print 'X:' + str(each[0])
##    print 'Y:' + str(each[1])
##
##print len(coords)
### why aren't there 100? fix it in the code above
```

locations_calc.py

```
# calculate new values while reading and writing text files
# this example multiples longitude values for western hemisphere
# http://docs.python.org/release/2.6.6/library/os.html
# http://docs.python.org/release/2.6.6/library/os.path.html

import os

# assign table to a variable
in_table = 'C:\\Temp\\pp2\\files\\locations.txt'

# create new file for writing calculation results to
table_part = os.path.split(in_table)
print table_part
out_dir = table_part[0]
file_name = table_part[1]
print out_dir
```

NOTE: The code samples may not work as is! They are meant to be used in guided hands-on instruction.

```
out_file = out_dir + os.path.sep + 'new_' + file_name
print out_file

# open files
in_file = open(in_table, 'r')
new_file = open(out_file, 'w')

# loop through each line
for line in in_file:
    # view each line as stored in the original table
##    print 'Original line:', line
    # split each line into its items
##    print 'Original items:', line.split(',')
##    # remove newline
##    print 'Remove newline:', line.replace('\n','')
##    line = line.replace('\n','')
    # convert each line to list object
    columns = line.split(',')
    oid = columns[0]
    longitude = columns[1]
    latitude = columns[2]
##    print longitude

    # test if first value is a number and not a text heading
    if longitude[0].isdigit():
        # convert longitude to float for multiplication
        # and convert back to string for writing text
        new_long = str(float(longitude) * -1)
    else:
        # not a number, so no calculation
        new_long = longitude
    print new_long

    # concatenate expression to write to new file
    new_line = oid + ',' + new_long + ',' + latitude
##    new_line = oid + ',' + new_long + ',' + latitude + '\n'
    print new_line
    new_file.write(new_line)

# close both files
in_file.close()
new_file.close()

# demo newline and the 'w' versus 'a' mode for new_file
```

NOTE: The code samples may not work as is! They are meant to be used in guided hands-on instruction.

Debugging, errors, and exceptions

Debugging is what is done to fix mistakes that break your code (or cause it to misbehave). The two types of errors that may need to be debugged are syntax errors and exceptions.

Syntax errors, also known as parsing errors, repeat the offending line indicating where in the line the error was detected. In the File window, choose RUN > CHECK MODULE (Alt+X) to try and view any of these before actually running the script.

Exceptions are errors detected during execution/runtime and result in various messages. Use **try/except** to attempt to perform an operation; if all goes well, great; if not, ask for forgiveness.

A debugger is a program that lets you step through your code one line at a time as Python executes them, showing you how each affects your program. For full introductions to IDLE debugging, with a GUI (DEBUG > DEBUGGER) or a module, see:

- http://inventwithpython.com/chapter7.html
- http://www.dreamincode.net/forums/topic/210537-python-debugging-part-1/
- http://pythonconquerstheuniverse.wordpress.com/2009/09/10/debugging-in-python/

Tip: Other IDEs may have friendlier debugging tools than IDLE (see the **Resources** below).


circle_area.py

```
# calculate area of a circle from user input radius
# try out various debugging/exception and find the intentional errors
# http://docs.python.org/release/2.6.6/library/pdb.html?highlight=pdb
# http://www.doughellmann.com/PyMOTW/pdb/


from math import pi
## import pdb
## pdb.set_trace()

radius = 10

##radius = raw_input('Type a number: ')
##radius = float(raw_input('Type a number: '))
### flow control to test input data type
##if radius.isdigit()
##    print radius
##    area = pi * pow(float(radius), 2)
##else:
##    print 'Value entered is not a number'

area = pi * Radius ** 2
##area = pi * pow(radius, 2)

print 'Area of a circle with radius ' + radius + ' is ' + Area
```

```
# replace last line with the following
try:
    print 'Area of a circle with radius ' + \
          str(radius) + ' is ' + str(area)
except:
    print 'Cannot show area'

# or relocate last line to inside flow control
```

NOTE: The code samples may not work as is! They are meant to be used in guided hands-on instruction.

User-defined functions

A function is a series of statements which returns some value to a caller (it can also be passed zero or more arguments for its use/execution). Built-in functions are objects that can be used by all Python code without the need of an import statement. A user-defined function is a block of organized, reusable code that is used to perform a single related action that you create. They are similar to procedures, subroutines, and functions in other programming languages, but may or may not return a value. The generic syntax requires the keyword def to define the function:

```
def functionname( parameters ):
   """function_docstring"""
   function_suite
   return [expression]
```

For more practice with functions, start at chapter 18 of http://learnpythonthehardway.org.

constant_value_field.py

```
# preliminary coding to add a constant value field to a database table

# function to test if number
def is_float(s):
    """ return Boolean value after trying convert string to float """
    try:
        float(s)
        return True
    except:
        return False

# get user input
field_name = raw_input('Type the new field name: ')
constant_value = raw_input('Type a constant value: ')

# test user input
if is_float(constant_value):
    constant_field_type = 'DOUBLE'
else:
    # assign defaults
    constant_field_type = 'TEXT'

# this is where you would add and calculate the new field
print 'The new field %s has the data type %s' % (field_name,
constant_field_type)

# on your own: insert code above to test that the field_name
# starts with a letter and only contains alphanumeric characters
# homework: combine this with the locations_calc.py code
# ignore the constant_field_type when writing to new_file
```

NOTE: The code samples may not work as is! They are meant to be used in guided hands-on instruction.

Really useful resources

http://www.python.org/dev/peps/pep-0008/

http://docs.python.org/faq/index.html

http://docs.python.org/release/2.6.6/library

http://docs.python.org/release/2.6.6/library/functions.html

http://docs.python.org/release/2.6.6/reference/compound_stmts.html

http://docs.python.org/release/2.6.6/tutorial/errors.html

http://docs.python.org/release/2.6.6/library/pdb.html?highlight=pdb

http://wiki.python.org/moin/IntegratedDevelopmentEnvironments

...and don't forget all the excellent e-resources in the "Practical Python Programming" winter workshop archives.

"Python Foot" graphic by David Day