AICT Computing Series Winter 2011 Workshop

Practical Python Programming

charlene.nielsen@ualberta.ca
Thursday, February 24th, 2011
9:00 a.m. to 12:00 noon
ETLC 5-013

This is an informal introduction to the basic concepts and features of Python - an interpreted, object-oriented, high-level programming language that is used in a wide variety of application domains (www.python.org) - but really it is "a programming language that knows how to stay out of your way when you write your programs. It enables you to implement the functionality you want without any hassle, and lets you write programs that are clear and readable" (Magnus Lie Hetland). Python is:

- powerful... and fast
- plays well with others
- runs everywhere
- is friendly... and easy to learn
- is Open

Through hands-on exercises, you will learn basic calculations, work with coordinate data, and get a taste of just some of the many library modules.

“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

The workshop's coding samples and scripts are shown below as plain text and screen captures from IDLE, so you know exactly what you should be typing in yourself. Please see the companion slides for more information.

Note: This document is named **ppp1_tasks_w2011.pdf** and the companion slides are named **ppp1_slides_w2011.pdf**. The companion slides have information on the following background information, programming concepts, and tutorial theory:

1. Who, when, what, where, why, how?
2. Easy environments
3. Programming primer
4. Coding conventions
5. Fussing with the flow
6. Excellent e-resources

Environments and Primer

Tip: Make note of what happens with each of the lines of code.

The following text – **except do NOT type the '>>>' prompt** – can be typed directly in to the interactive window (also called the Python Shell window):

```
>>> import this

>>> help()

>>> import

>>> topics

>>> NUMBERS

>>> strings          # note the lower case

>>> quit

>>> help(keywords)  # now retype with quotes around 'keywords'

>>> help('print')

>>> print 'what is a blue moon?'

>>> abluemoon = 'second to last full moon in a four-moon season'

>>> print abluemoon

>>> for once in abluemoon:
        print once       # we'll learn more about looping later
```

Tip: Type Alt-P a couple of times (and try Alt-N)

```
>>> 1 + 2 + 3

>>> _

>>> _ - 4

>>> a = 1

>>> b, c = 2, 3          # we’ll do more variables later

>>> A + b * c            # then type a + b * c

>>> 'stuff'

>>> _
```

Note: The default result variable ‘_’ is available only in the interactive window.

```
>>> # calculator expressions

>>> 1 / 2

>>> 1 / 2.0

>>> 1.0 / 2

>>> -3 ** 2

>>> (-3) ** 2

>>> 'p' + 'p' + 'p'

>>> 'p' * 3

>>> 'p' + 3

>>> 'Ha, ' * 5 + 'Ha!'
```

Note: Try a few calculations on your own.

```
>>> # commenting and docstrings

>>> # dear interpreter, please ignore all comments, thank you

>>> """ Geography is an important antidote to the "infantile"
habit of thinking the world is a laboratory in which we can
carry out all kinds of experiments, or a huge rubbish heap where
we can get rid of all our trash. """
```

```
>>> """ Type something
and type some more
and more
and then some more again
and maybe
put yourself out of your misery
and finish """

>>> print _

>>> # print statements

>>> print 'And now for something completely different!'

>>> print 'And now for something\ncompletely different!'

>>> print 'Green' + 'eggs' + 'and' + 'ham'

>>> print 'Green', 'eggs', 'and', 'ham'

>>> print 'Today is', 24, 'February', 2011

>>> print "This is my pretend long string \

that I just don't want to fit on one line"

>>> print "Let's go!"

>>> print 'Let\'s go!'

>>> print "She said, \"Hello\""

>>> print 'C:\new folder'

>>> print 'C:\temp'

>>> print 'C:\\newfolder', 'C:\\temp'

>>> # sequences (lists and strings)

>>> mylist = [123, 456, 'seven ate nine']

>>> mylist[0]

>>> mylist[1]

>>> mylist[2]

>>> mylist[-1]
>>> 'mylist'[0]

>>> 'mylist'[2]

>>> list('mylist')

>>> 'm' in mylist

>>> 'm' in 'mylist'
```

Note: An entire day could be devoted to a workshop on lists, strings, other sequences such as tuples, and all kinds of indexing, formatting, operations, slicing, and methods! The examples here are modestly minor. An excellent quick reference for many topics is available here: http://rgruet.free.fr/PQR26/PQR2.6.html.

First file

In the Python Shell (interactive window), click FILE >>> NEW WINDOW

Tip: It's a good idea to type a few meaningful comments at the top of your script file. Once you have opened a new file editor script window, type the following, including the blank lines for readability (note: 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
```

OR

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

Click FILE >>> SAVE as C:\Temp\ppp\sequences.py (do NOT forget the **.py**)

In the interactive window, type help('SEQUENCES') and give the documentation a quick look through. Similarly, type help('METHODS') , help('string'), and help('list') and examine those topics. Return your focus to the script window and start typing:

```
# the following sequence commands and methods use palindromes

pal1 = "never odd or even"

pal2 = 'kayak'

pal3 = 'a man a plan a canal panama'


# fun with strings - indices point between characters

print pal1.capitalize()

print pal1[0]

print pal1[-1]

print pal1[1:4]

print pal1[0:5]

print pal1[6:]


# type the dot, press ctrl-space, and use arrow key to select
method

print pal2.upper()

print pal2[-3:]

length = len(pal2)

print '# of characters in "' + pal2 + '" =' + length
```

Click RUN >>> RUN MODULE. Click OK to SAVE. Can you figure out what the error message is? Change the '+' to a ',' or apply the str() function.

```
print '# of characters in "' + pal2 + '" =', length
```

OR

```
print '# of characters in "' + pal2 + '" =' + str(length)

print pal2.strip('k')
```

Click RUN >>> RUN MODULE again (or press the F5 key). View the results in the interactive window, cross-referencing with the code in the script window. Type:

```
print pal3.swapcase()

print 'last word: ' + pal3[21:len(pal3)]

print pal3.strip(' panama')

print pal3.lstrip(' panama')

print pal3.rstrip(' panama')

print pal3.replace('a', '@')
```

Click RUN >>> RUN MODULE again (or press the F5 key). Examine.

Now, use the mouse cursor to click and drag to highlight/select all the text from "`# fun with strings`…" to the end of the file. Click FORMAT >>> COMMENT OUT REGION. Click RUN again to see what happens.

Type some more, alternating between running the code and inspecting the results as you go along. Note: Wherever you see blank lines would be a good point to RUN.

```
# fun with lists

list1 = list(pal3)

print list1

print list1.count('a')

list2 = pal3.split()

print list2

list3 = list2

list3.sort()

print list3
print list2[0]

print list2[-2:]


list4 = [10, 20, 30, 40, 50]

print list4

print list4[1]

print len(list4)

list4.append(60)

print list4


# not sequences, but useful functions to know

a = 1

b = 2

c, d = 'three', '100'

print a + b

print str(a) + str(b) + c

print int(d)

print float(d)


goodbye = '\nokay, that\'s pretty good for a first script file!'

print '*' * len(goodbye)

print goodbye
```

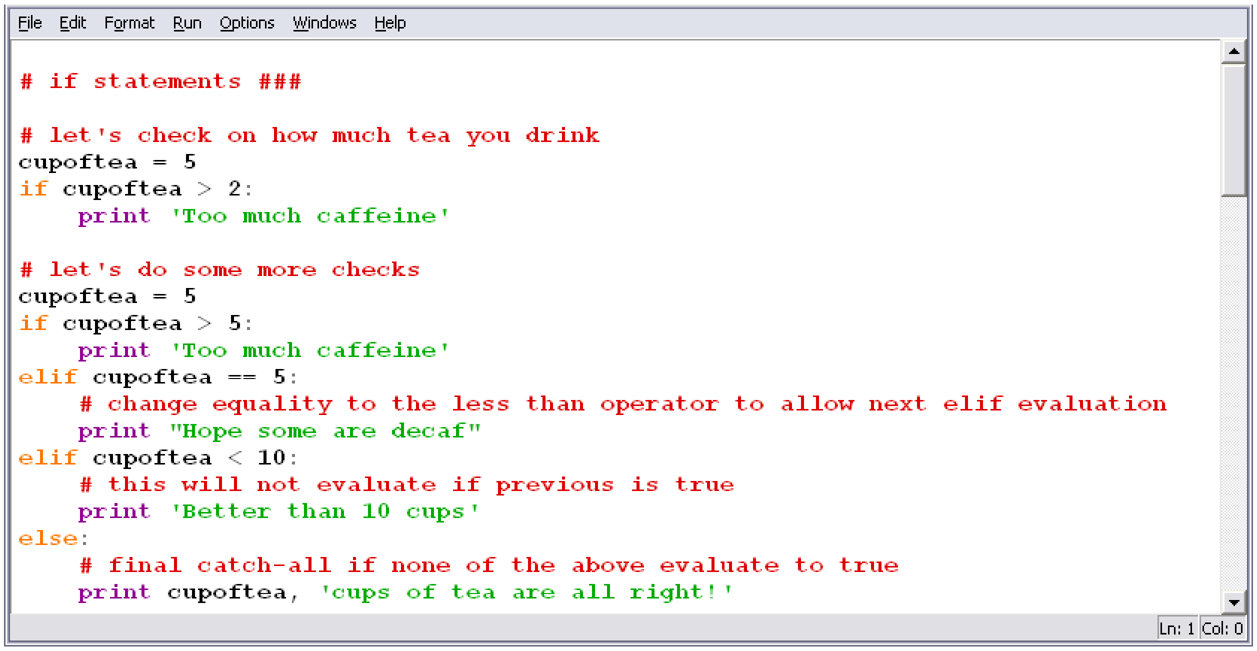
Tip: Remember the useful trick of highlighting lines of your program and then using the FORMAT menu to COMMENT OUT REGION and UNCOMMENT REGION. The INDENT REGION and DEDENT REGION will come in handy when you start using loops and other constructs requiring blocks of code.

Coding conventions

Thoroughly read the Style Guide for Python Code:
http://www.python.org/dev/peps/pep-0008/

Fussing with the flow

Start a new file for each of the scripts below. Give them a meaningful file name with a .py extension! Make notes (comments in the code will do!) if you think the instructor says something important during this hands-on demonstration of programming with control and decision.



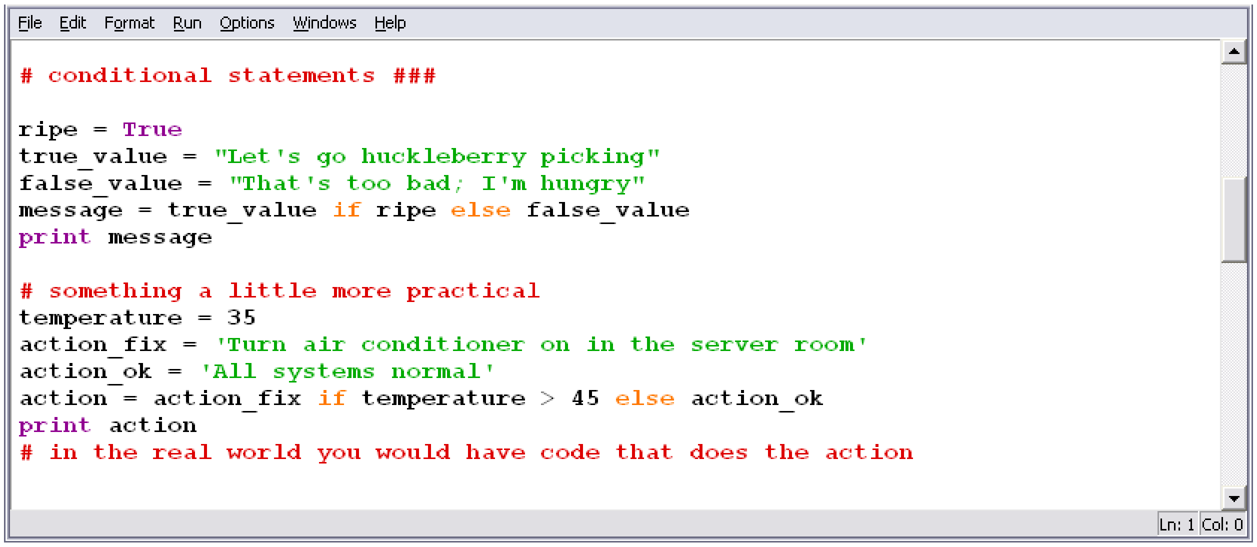
```
# if statements ###

# let's check on how much tea you drink
cupoftea = 5
if cupoftea > 2:
    print 'Too much caffeine'

# let's do some more checks
cupoftea = 5
if cupoftea > 5:
    print 'Too much caffeine'
elif cupoftea == 5:
    # change equality to the less than operator to allow next elif evaluation
    print "Hope some are decaf"
elif cupoftea < 10:
    # this will not evaluate if previous is true
    print 'Better than 10 cups'
else:
    # final catch-all if none of the above evaluate to true
    print cupoftea, 'cups of tea are all right!'
```

Extra 'if' example:

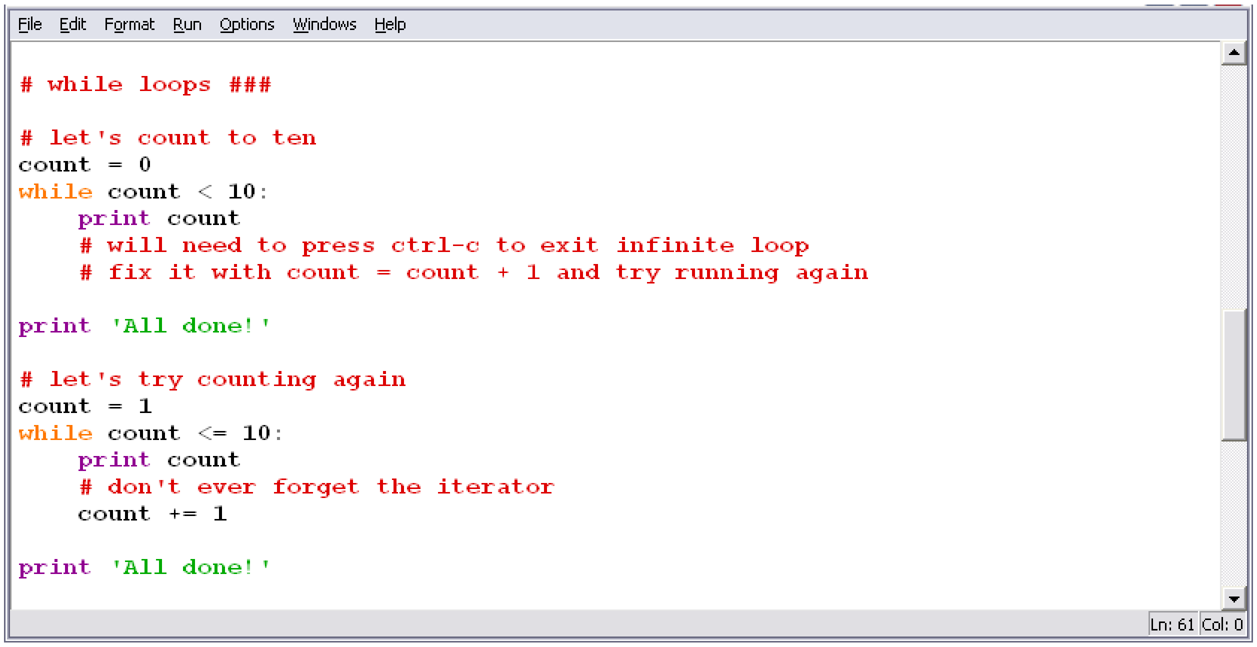
```
# test if number is even or odd
number = input("Tell me a number: ")
if number % 2 == 0:
    print number, "is even."
elif number % 2 == 1:
    print number, "is odd."
else:
    print number, "is not an integer."
```



```
# conditional statements ###

ripe = True
true_value = "Let's go huckleberry picking"
false_value = "That's too bad; I'm hungry"
message = true_value if ripe else false_value
print message

# something a little more practical
temperature = 35
action_fix = 'Turn air conditioner on in the server room'
action_ok = 'All systems normal'
action = action_fix if temperature > 45 else action_ok
print action
# in the real world you would have code that does the action
```



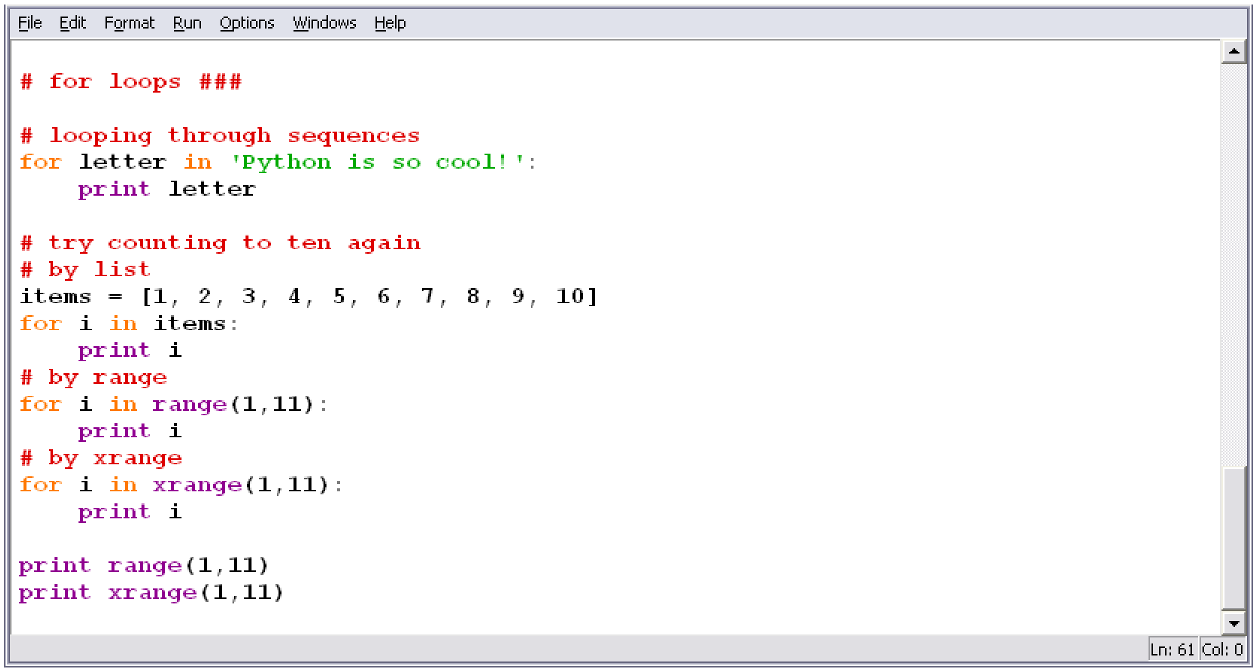
```
# while loops ###

# let's count to ten
count = 0
while count < 10:
    print count
    # will need to press ctrl-c to exit infinite loop
    # fix it with count = count + 1 and try running again

print 'All done!'

# let's try counting again
count = 1
while count <= 10:
    print count
    # don't ever forget the iterator
    count += 1

print 'All done!'
```



```
# for loops ###

# looping through sequences
for letter in 'Python is so cool!':
    print letter

# try counting to ten again
# by list
items = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in items:
    print i
# by range
for i in range(1,11):
    print i
# by xrange
for i in xrange(1,11):
    print i

print range(1,11)
print xrange(1,11)
```

Be sure to see the companion slides
for excellent e-resources to help you learn even more!