Introduction to Python
Functions, Booleans, Modules

Christopher Barker

UW Continuing Education

October 8, 2013

Table of Contents

Review of Previous Class

- Values and Types
- Expressions
- Intro to functions

Lightning Talks

Lightning talks today:

Jo-Anne Antoun

Omer Onen

Ryan Small

Catherine Warren

Homework review

Homework Questions?

To loop or not to loop?

Build up strings, then print...

My Solution

Stuff brought up by homework

Bytecode and `*.pyc`

Please send me code:

- Enclosed in an email
- With your name at the beginning of the filename: `chris_problem1.py`

PEP 8

Repeating variable names in nested loops

Basics

It turns out you can't really do much at all without at least a container type, conditionals and looping...

if

Making Decisions...

```
if a:
    print 'a'
elif b:
    print 'b'
elif c:
    print 'c'
else:
    print 'that was unexpected'
```

if

Making Decisions...

```
if a:
    print 'a'
elif b:
    print 'b'

## versus...

if a:
    print 'a'
if b:
    print 'b'
```

switch?

No switch/case in Python

use `if..elif..elif..else`

(or a dictionary, or subclassing....)

lists

A way to store a bunch of stuff in order

"array" in other languages

```
a_list = [2,3,5,9]

a_list_of_strings = ['this', 'that', 'the', 'other']
```

tuples

Another way to store an ordered list of things

```
a_tuple = (2,3,4,5)

a_tuple_of_strings = ('this', 'that', 'the', 'other')
```

Often interchangeable with lists, but not always...

for

When you need to do something to everything in a sequence

```
>> a_list = [2,3,5,9]

>> for item in a_list:
>>     print item
2
3
5
9
```

range() and for

When you need to do something a set number of times

```
>>> range(4)
[0, 1, 2, 3]
>>> for i in range(6):
...     print "*",
...
* * * * * *
>>>
```

intricacies

This is enough to get you started.

Each of these have intricacies special to python

We'll get to those over the next couple classes

Functions: review

Defining a function:

```
def fun(x, y):
    z = x+y
    return z
```

x, y, z are local names

Functions: local vs. global

```
x = 32
def fun(y, z):
    print x, y, z

fun(3,4)
```

```
32 3 4
```

x is global, y and z local

Use global variables mostly for constants

Recursion

Recursion is calling a function from itself.

Max stack depth, function call overhead.

Because of these two(?), recursion isn't used **that** often in Python.

(demo: factorial)

Tuple Unpacking

Remember: `x,y = 3,4` ?

Really "tuple unpacking": `(x, y) = (3, 4)`

This works in function arguments, too:

```
>>> def a_fun( (a, b), (c, d) ):
...     print a, b, c, d
...
>>> t, u = (3,4), (5,6)
>>>
>>> a_fun(t, u)
3 4 5 6
```

(demo)

Lab: more with functions

Write a function that:

- computes the distance between two points:
  dist = sqrt((x1-x2)**2 + (y1-y2)**2)
  using tuple unpacking...
- Take some code with functions, add this to each function:
  `print locals()`
- Computes the Fibonacci series with a recursive function:
  f(0) = 0; f(1) = 1
  f(n) = f(n-1) + f(n-2)
  0, 1, 1, 2, 3, 5, 8, 13, 21, ...
  (If time: a non-recursive version)

Lightning Talks

Lightning Talks:

Jo-Anne Antoun

Omer Onen

Truthiness

What is true or false in Python?

- The Booleans: `True` and `False`
- "Something or Nothing"

`http://mail.python.org/pipermail/python-dev/2002-April/022107.html`

Truthiness

Determining Truthiness:

```
bool(something)
```

Boolean Expressions

False

- None
- False
- zero of any numeric type, for example, 0, 0L, 0.0, 0j.
- any empty sequence, for example, '', (), [] .
- any empty mapping, for example, {}.
- instances of user-defined classes, if the class defines a __nonzero__() or __len__() method, when that method returns the integer zero or bool value False.

http://docs.python.org/library/stdtypes.html

Boolean Expressions

Avoid:

```
if xx == True:
```

Use:

```
if xx:
```

Boolean Expressions

"Shortcutting"

```
                        if x is false,
x or y                     return y,
                           else return x

                        if x is false,
x and y                     return  x
                            else return y

                        if x is false,
not x                     return True,
                          else return False
```

Boolean Expressions

Stringing them together

```
a or b or c or d
```

```
a and b and c and d
```

The first value that defines the result is returned

(demo)

Boolean returns

From CodingBat

```
def sleep_in(weekday, vacation):
    if weekday == True and vacation == False:
        return False
    else:
        return True
```

Boolean returns

From CodingBat

```
def sleep_in(weekday, vacation):
    return not (weekday == True and vacation == False)
```

or

```
def sleep_in(weekday, vacation):
    return (not weekday) or vacation
```

bools are ints?

bool types are subclasses of integer

```
In [1]: True == 1
Out[1]: True

In [2]: False == 0
Out[2]: True
```

It gets weirder!

```
In [6]: 3 + True
Out[6]: 4
```

(demo)

Conditional expression

A common idiom:

```
if something:
    x = a_value
else:
    x = another_value
```

Also, other languages have a "ternary operator"
(C family: `result = a > b ? x : y ;`)

```
y = 5 if x > 2 else 3
```

PEP 308: (http://www.python.org/dev/peps/pep-0308/)

LAB

- Look up the % operator. What do these do?

```
10 % 7 == 3
14 % 7 == 0
```

- Write a program that prints the numbers from 1 to 100 inclusive. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz" instead.
- Re-write a couple CodingBat exercises, using a conditional expression
- Re-write a couple CodingBat exercises, returning the direct boolean results

(use whichever you like, or the ones in: `code/codingbat.rst`)

Lightning Talks

Lightning Talks:

Ryan Small

Catherine Warren

Code Structure

Python is all about namespaces – the "dots"

`name.another_name`

The "dot" indicates looking for a name in the namespace of the given object. It could be:

- name in a module
- module in a package
- attribute of an object
- method of an object

indenting and blocks

Indenting determines blocks of code

```
something:
    some code
    some more code
    another block:
        code in
        that block
```

But you need the colon too...

indenting and blocks

You can put a one-liner after the colon:

```
In [167]: x = 12

In [168]: if x > 4: print x
12
```

Only do this if it makes it more readable...

Spaces and Tabs

An indent can be:

- Any number of spaces
- A tab
- tabs and spaces:
  - A tab is eight spaces (always!)
  - Are they eight in your editor?

Always use four spaces – really!

(PEP 8)

Spaces Elsewhere

Other than indenting – space doesn't matter

```
x = 3*4+12/func(x,y,z)
```

```
x = 3*4 + 12 /   func (x,   y, z)
```

Choose based on readability/coding style

PEP 8

Various Brackets

Bracket types:

- parentheses `( )`
  - tuple literal: `(1,2,3)`
  - function call: `fun( arg1, arg2 )`
  - grouping: `(a + b) * c`
- square brackets `[ ]`
  - list literal: `[1,2,3]`
  - sequence indexing: `a_string[4]`
- curly brackets `{ }`
  - dictionary literal: `{"this":3, "that":6}`
  - (we'll get to those...)

modules and packages

A module is simply a namespace

A package is a module with other modules in it

The code in the module is run when it is imported

importing modules

```
import modulename

from modulename import this, that

import modulename as a_new_name
```

(demo)

importing from packages

```
import packagename.modulename

from packagename.modulename import this, that

from package import modulename
```

(demo)

http://effbot.org/zone/import-confusion.htm

importing from packages

```
from modulename import *
```

Don't do this!

("Namespaces are one honking great idea...")

(wxPython and numpy example...)

Except *maybe* math module

(demo)

import

If you dont know the module name before execution.

```
__import__(module)
```

where `module` is a Python string.

modules and packages

The code in a module is NOT re-run when imported again – it must be explicitly reloaded to be re-run

```
import modulename
```

```
reload(modulename)
```

(demo)

```
import sys
print sys.modules
```

(demo)

LAB

Experiment with importing different ways:

```
import math
dir(math) # or, in ipython -- math.<tab>
math.sqrt(4)

import math as m
m.sqrt(4)

from math import *
sqrt(4)
```

LAB

Experiment with importing different ways:

```
import sys
print sys.path

import os
print os.path
```

You wouldn't want to import * those – check out

```
os.path.split()
os.path.join()
```

Homework

Recommended Reading:

- Think Python: Chapters 8, 9, 10, 11, 12
- String methods: `http://docs.python.org/library/stdtypes.html#string-methods`
- Dive Into Python: Chapter 3

Do:

- Six more CodingBat exercises.
- LPTHW: for extra practice with the concepts – some of:

  strings: ex5, ex6, ex7, ex8, ex9, ex10

  raw_input(), sys.argv: ex12, ex13, ex14 (needed for files)