Journal for Lab2

Hongyan Yi **yih@oregonstate.edu**

Main Goal

Notes for what I learned from two parts
1) Lab2 assignments
2) Lab2 out of assignments (Class or Codecademy)

Lab2 Assignments

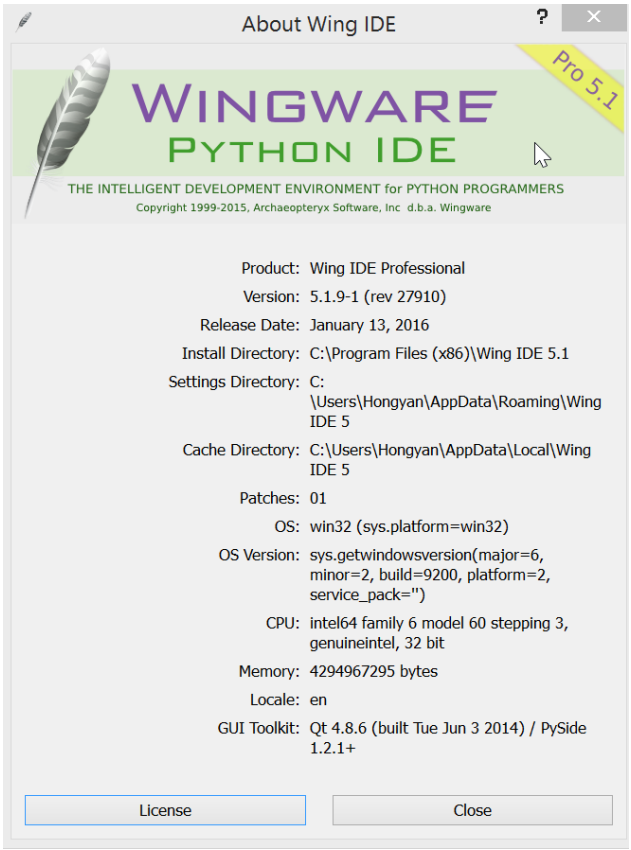
Section 1: Get set up with Wingware

1.1 import sys

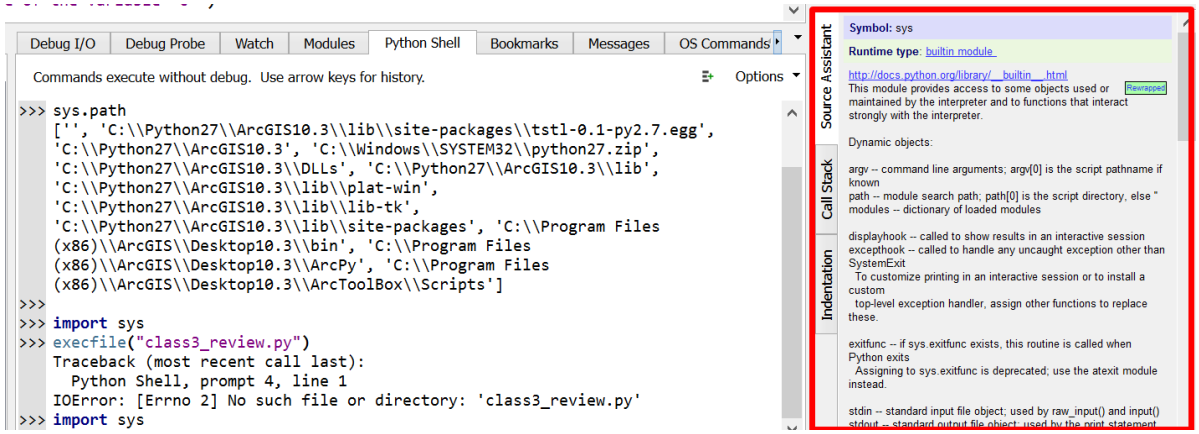
(1pt) What was happening internally in the interpreter's brains when you typed "import sys"?

Since I worked in my laptop, setup and debug pictures might be a little bit different, but for code I've double check in lab that they could run correctly.

wing IDE in mylaptop



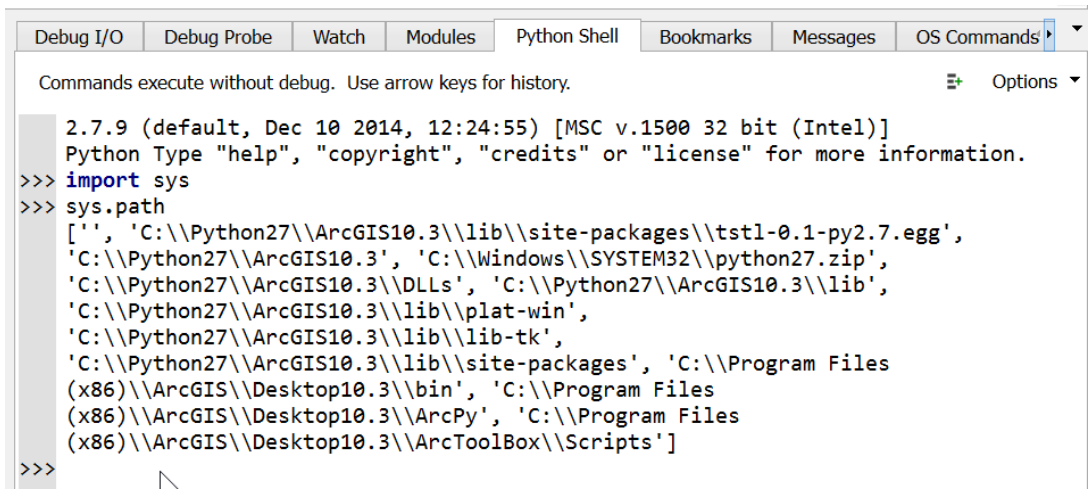
wing IDE in Lab



when I type import sys, there is some auto hint for users, so very convenient.

1.2 check sys.path in python shell

(1pt) What type of variable is sys.path?



Hongyan@HongyanYi /x/class/GIS 578/Lab2/week2
$ python
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', 'C:\\Python27\\ArcGIS10.3\\lib\\site-packages\\tstl-0.1-py2.7.egg',
'C:\\Python27\\ArcGIS10.3', 'C:\\Windows\\SYSTEM32\\python27.zip',
'C:\\Python27\\ArcGIS10.3\\DLLs',
'C:\\Python27\\ArcGIS10.3\\lib',
'C:\\Python27\\ArcGIS10.3\\lib\\plat-win',
'C:\\Python27\\ArcGIS10.3\\lib\\lib-tk',
'C:\\Python27\\ArcGIS10.3\\lib\\site-packages',
'C:\\Program Files (x86)\\ArcGIS\\Desktop10.3\\bin',
'C:\\Program Files (x86)\\ArcGIS\\Desktop10.3\\ArcPy',
'C:\\Program Files (x86)\\ArcGIS\\Desktop10.3\\ArcToolBox\\Scripts']
>>>

1.3 locals()

(2pt) Once you've run 'class3_review.py', use the "locals()" command in the console to SEE THE OBJECTS THAT ARE CURRENTLY IN MEMORY. Do you see the variables that I assigned in class3_review.py? Why or why not?

Yes, I could see all the variables the you assigned.
For a,b,d are normal, but c it only keep the latest value.

```
Hongyan@HongyanYi /cygdrive/x/class/GIS 578/Lab2/week2
$ python
Python 2.7.10 (default, Jun  1 2015, 18:05:38)
[GCC 4.9.2] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> execfile("class3_review.py")
I am now going to print the value of the variable 'c'

How'd that go?
Okay, trying again.
poppoppoppoppoppoppoppoppoppop
Better?
>>> locals()
{'a': -10, 'c': 'poppoppoppoppoppoppoppoppoppop', 'b': 'pop', 'd': 10, 'ty': <ty
pe 'str'>, '__builtins__': <module '__builtin__' (built-in)>, '__package__': Non
e, '__name__': '__main__', '__doc__': None}
>>>
```

>>> locals()

{'a': -10,

'c': 'poppoppoppoppoppoppoppoppoppop',

'b': 'pop',

'd': 10,

'ty': <type 'str'>,

'__builtins__': <module '__builtin__' (built-in)>,

'__package__': None,

'__name__': '__main__',

'__doc__': None}

>>>

1.4 Syntax Error

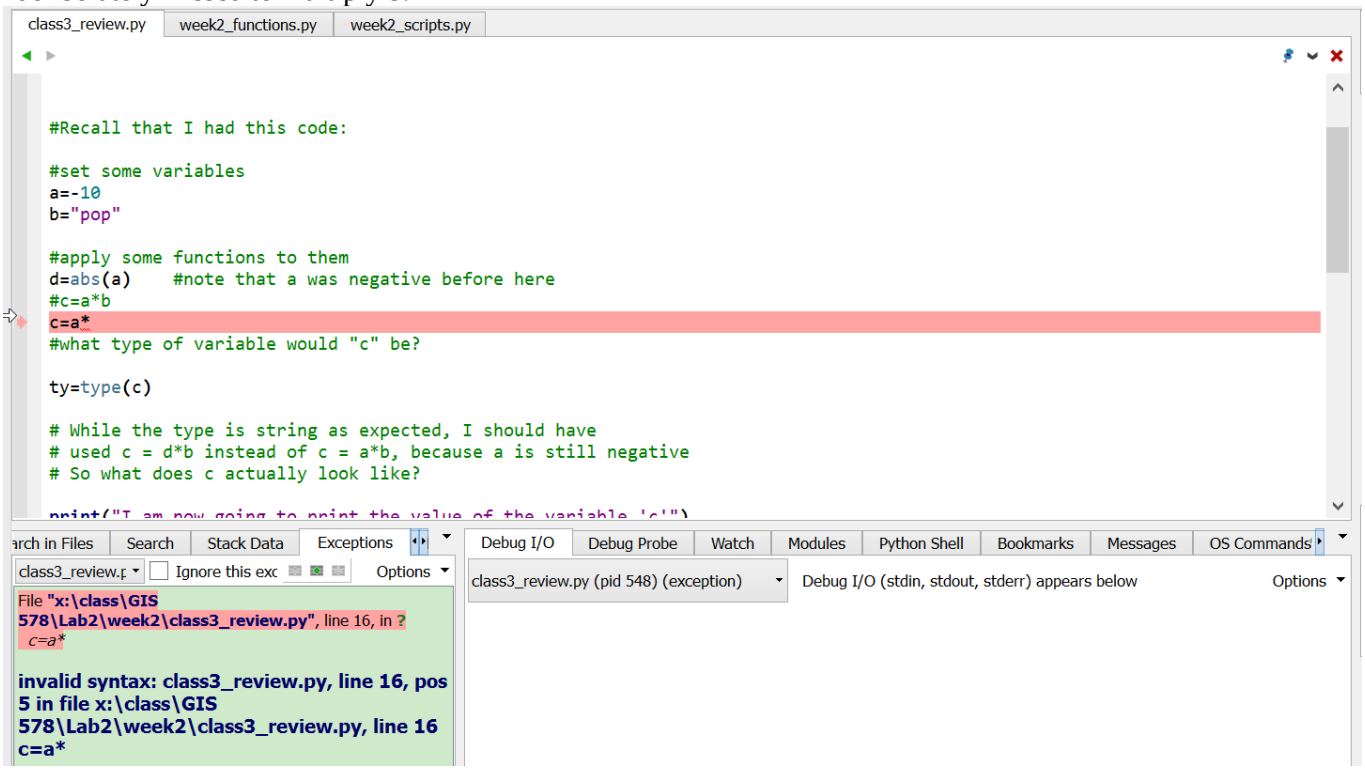
(2pt) Modify the code in class3_review.py to create A SYNTAX ERROR somewhere.

a. Yes, I'm asking you to mess up your code

b. What does Wingware do?

c. UNDO YOUR SYNTAX ERROR for the next step

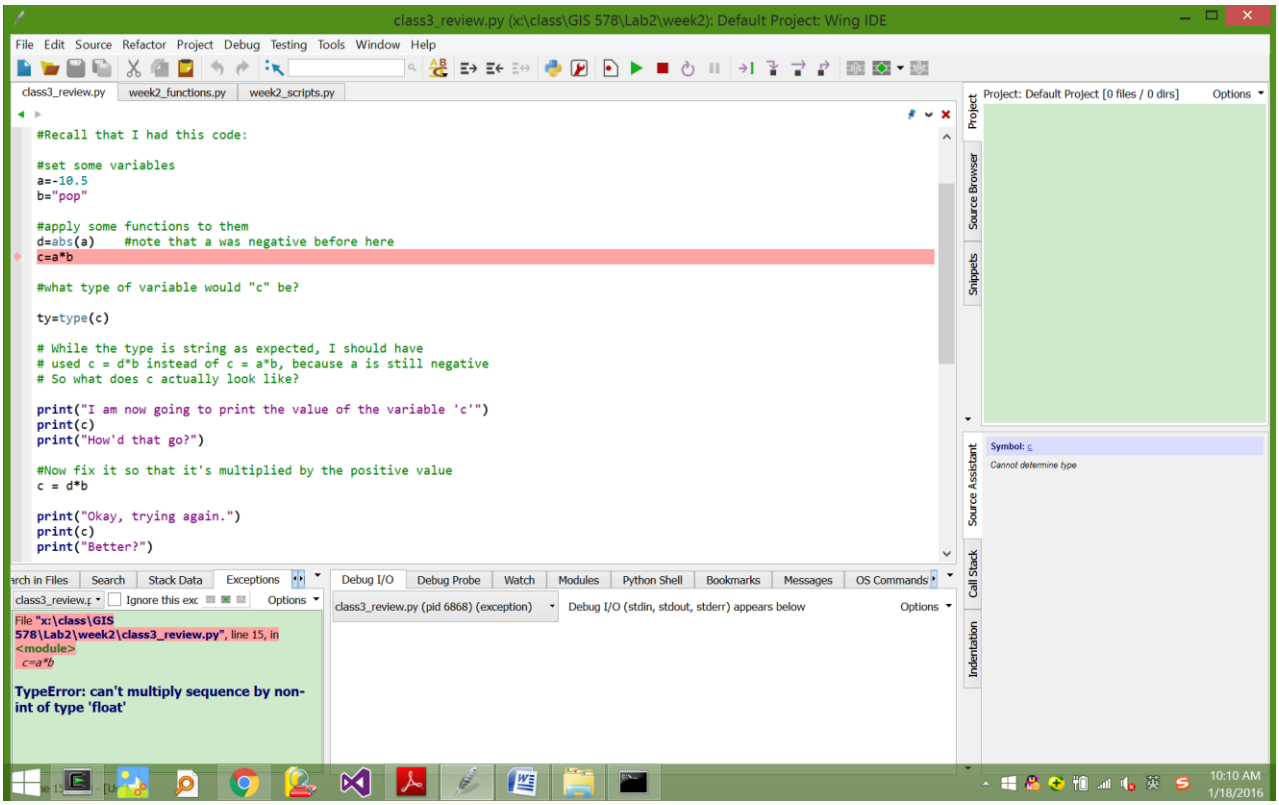
I deliberately missed to multiply b.



1.5 Exception Error

(2pt) In "class3_review.py", try to INSERT AN EXCEPTION ERROR of some sort (your choice!). Recall from class last week the difference between an exception type error and a syntax error, and if necessary Google some examples. Once you've inserted the error, save the file, and then run it again using execfile. Briefly describe what you see. Does the line number where Wingware complains match where you put your exception? What error did you insert?
a. Leave your error!!! We'll need it for the next step.

I change a = -10.5 then wingware show exceptions.

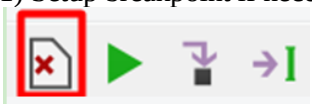


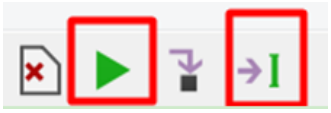
Section 2. Meet the debugger

2.1 debug with stack data

(2pt) Describe how the error you created can be linked logically to the condition of variables whose state you can see in the "Stack Data" tab.

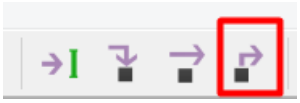
I didn't have the ladybug button in my WingIDE (wingide-5.1.9-1 professional free trial in my laptop) but I could also debugger very well.
1) Setup breakpoint if necessary, and then click F5 or Alt-F5 to run to the breakpoint.





2) or directly click F7 to step into the first line of code

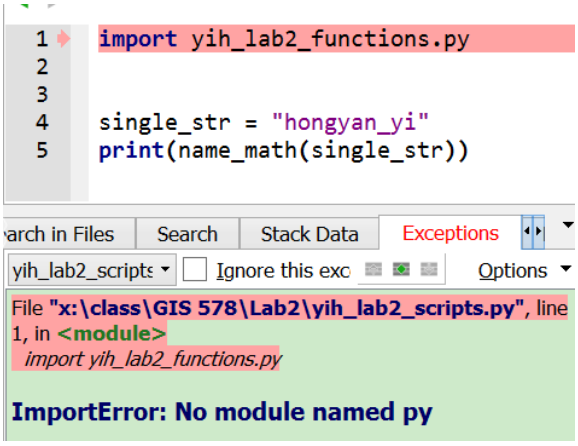
3) click F8 to run each line of code, then I could see the condition of variables in the "Stack Data" tab, so that I could check whether the value of variables change correctly.

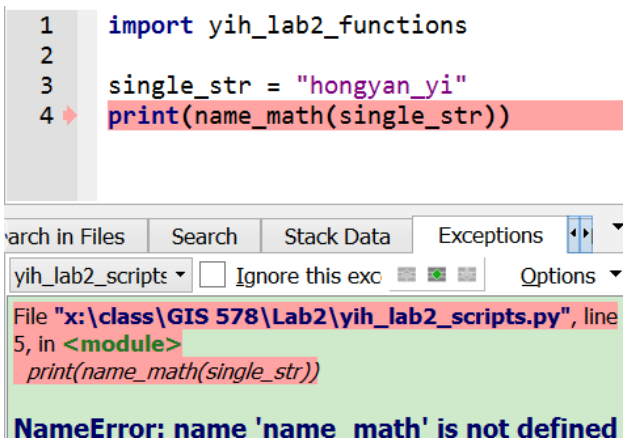


2.2 debug without stack data

(3pt) Pretend you did not know what caused the error you created. Look only at the error and the code (not the stack data), and develop a testable hypothesis about what caused the error that can be resolved by looking at the status of objects in the Stack Data window.
Example using very generic language: "My error is about passing the wrong operand type to function X. As far as I know, that function X requires type Y.
Test: If I look in the stack data at the variable type I am passing to the function and it is not type Y, then I know I need to look earlier in my code to see why that variable is defined with the wrong type. If it is type Y, then there must be something else in the function syntax that I don't know about I'll have to look that up in the function syntax on Python.org."

My error is a TypeError: can't multiply sequence by non-int of type 'float.
Test: I don't see my code but first think about my error. I think in c = a*b, in variables a and b, at least one of them is a float number and the other is a sequence, so I go to Stack data to check the value of a and b, and found a is a float number, so I think I should use c = round(a) * b, but then there is still the same error, so I set x = round(a), c = x * b, and then use Stack data to see x = -11.0, so I know I should use int(round(a)) * b, then the there is no error left.





Section 3. Implement

3.1 (3pts) Name math

a. Write a function that accepts your first and last name as a single string, uses the str.split() function to separate your first and last name, and returns the product of the length of your first and last name.

3.1.1 str.split

I inserted a "_" between my first and last name, and then split the single string using "_", and got a list.

```
split_list = str.split(arg1,"_")        # split str with _, return a list
```

Note: split could not use directly, but have to use str. before it.

str.**split**([*sep*[, *maxsplit*]])

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most `maxsplit+1` elements). If *maxsplit* is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`). The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2', '3']`). Splitting an empty string with a specified separator returns `['']`.

If *sep* is not specified or is `None`, a different splitting algorithm is applied: runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end if the string has leading or trailing whitespace. Consequently, splitting an empty string or a string consisting of just whitespace with a `None` separator returns `[]`.

For example, `' 1 2 3 '.split()` returns `['1', '2', '3']`, and `' 1 2 3 '.split(None, 1)` returns `['1', '2 3 ']`.

3.2 (5pts) Latitude math

a. Write a function that accepts a three element list of [degrees, minutes, seconds]
and converts it to a single decimal degree value.
i. Note that 60 minutes = 1 degree, and 60 seconds = 1 minute
ii. Make sure the user does not enter a value greater than 90 degrees or less than 0 degrees. If they do, let them know gently.

3.2.1 limit input

I used the following structure to limit users to input correctly, otherwise input again.

```
while True:
    if case1:
        break
    else:
        case2
```

3.2.2 list VS array

list in python is like the array in C
list could put 3 variables directly to a list, and use them with index, eg list_nums[0]

```
list_nums = [num_degree,num_minutes,num_seconds] # directly put 3 variables to list.
```

3.3 (4pt) Filename iteration generator:

a. run a bunch of tests that will generate many variants on a starting file name.
Assume the file name has a file extension with three characters following a ".", e.g. "filename.img". Write a function that accepts a single string, and returns a list variable with 10 different strings. Each string will be the starter string, but will have the number from the sequence [100m, 200m, 300m, etc. up to 1000m] interspersed before the filename extension, e.g. "filename100m.img",
"filename200m.img", etc.

3.3.1 for loop with range

set the range ahead of time, with start, stop and step, then in the for loop, could use the index directly.

```
num_list = range(100,1100,100)          # range(start, stop[, step])
for i in num_list:
    filename_variant_list.append(filename + str(i) + tail)
```

```
range(stop)
range(start, stop[, step])
```

This is a versatile function to create lists containing arithmetic progressions. It is most often used in `for` loops. The arguments must be plain integers. If the *step* argument is omitted, it defaults to `1`. If the *start* argument is omitted, it defaults to `0`. The full form returns a list of plain integers `[start, start + step, start + 2 * step, ...]`. If *step* is positive, the last element is the largest `start + i * step` less than *stop*; if *step* is negative, the last element is the smallest `start + i * step` greater than *stop*. *step* must not be zero (or else `ValueError` is raised). Example:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 30, 5)
[0, 5, 10, 15, 20, 25]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> range(0)
[]
>>> range(1, 0)
[]
```

3.3.2 list.append(x)

Add an item to the end of the list; Here the item is a string
more about list method
https://docs.python.org/2/tutorial/datastructures.html

3.4 (4pts) Euclidean distance between two points

a. You can pass more than one argument to a function. To pass more than one, you simply define the function as:
def myfunction(arg1, arg2):
b. Write a function that accepts as arg1 an [x1,y1] list of coordinates and arg2 as an [x2, y2] list of coordinates. Use the math.sqrt module to find the Euclidean distance between the two points.

3.4.1 use list to indicate coordinate

here use 2D points, such as [x1,y1], in 3D would be [x1,y1,z1]

3.4.2 print list

could use str(list) to print list

3.4.3 math.sqrt

```
distance = math.sqrt((x2-x1)**2 + (y2-y1)**2)
```

```
math.sqrt(x)
```

Return the square root of *x*.

more detail about math module:
https://docs.python.org/2/library/math.html

3.5 (bonus 4pts) Random points

a. Write a function that generates a list of random samples within a coordinate box
defined by the user. The function accepts three arguments:
i. arg1: number of samples desired
ii. arg2: the [x,y] pair of the upper left corner of the box
iii. arg3: the [x,y] pair of the lower right corner of the box
b. The function returns a list of [x,y] coordinate list variables (a list of lists), or returns an array object (check out the "numpy" module, which we'll be using later in the term).

3.5.1 for loop without range

when giving specific number, use for i in number: to get for loop

3.5.2 random

how to call random
from random import random

```
p = random()              # a random number between 0 and 1.0
```

3.5.3 points between two points

calculate points between two points
x = p * x1 + (1-p) * x2
y = p * y1 + (1-p) * y2
0<=p<=1

```
for i in range(num_points):
    p = random()            # a random number between 0 and 1.0
    x = p * x1 + (1-p) * x2
    y = p * y1 + (1-p) * y2
    point_list = [x,y]
    coordinate_list.append(point_list)
return coordinate_list
```

Lab2 out of Assignments

1) Control flow

Control flow gives us this ability to choose among outcomes based off what else is happening in the program.

```
def clinic():
    print "You've just entered the clinic!"
    print "Do you take the door on the left or the right?"
    answer = raw_input("Type left or right and hit 'Enter'."
).lower()
    if answer == "left" or answer == "l":
        print "This is the Verbal Abuse Room, you heap of parrot
droppings!"
    elif answer == "right" or answer == "r":
        print "Of course this is the Argument Room, I've told you
that already!"
    else:
        print "You didn't pick left or right! Try again."
        clinic()

clinic()
```

2) lower() after raw_input()

directly put lower() after raw_input()

raw_input("ssddd").lower()

3) Conditional statements

if something:

    case1

elif something:

    case2

else:

    case3

4) Operators Precedence

Python Operators Precedence

The following table lists all operators from highest precedence to lowest. **Operator**	**Description**
**	Exponentiation (raise to the power)
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

5) isalpha()

Built-in String Methods

11	isalpha() Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.