Com S 127
Lab 2

Checkpoint 0

Please open the CS 127 Blackboard page and click on Groups in the menu at left. Sign up for the group corresponding to the lab section you are attending.

Also, if you haven't done so already, you should first complete Checkpoint 3 from last week's lab (submitting an assignment and checking your submission history). See
http://web.cs.iastate.edu/~smkautz/cs127f16/labs/lab1/page14.html

1. A review of some Python functions

For the first two parts of the lab, start up Wing 101 and use the Python shell window to try everything out.

We have seen all of these built-in functions:

```
type(expression)              # returns the type of an expression
len(string)                   # returns the length of a string
float(number or string)       # converts int or string to a float value
int(number or string)         # converts float or string to an int value
str(number)                   # converts a number to a string
pow(number, number)           # computes a number to a power
```

Here some examples of calls to the built-in functions. Start by defining some variables in the shell:

```
>>> s = "steve"
>>> x = 25
>>> t = "42"
```

And then try entering the following function calls to see what value each function returns. (Remember the shell will attempt to evaluate each expression you type and display its value, so when using the shell this way you don't actually have to type a `print` statement.)

```
>>> len(s)
>>> str(10 * 25)
>>> len(str(10 * 25))
>>> int(t) + 1
>>> int(2.7)
>>> round(2.7)
```

There is also a Python function `pow` for computing powers. `pow(x, y)` returns x^y (x raised to the power y.) Try evaluating these expressions in the shell:

```
>>> pow(5, 2)
>>> pow(2, len(s))
```

Other mathematical functions

What about the rest of the stuff you find on a calculator, like trig functions and logarithms and roots? They're there, but require an extra step. The functions above are all "built in" to the language. We can just use them. Python also has many additional libraries of functions available, called *modules*. One of these modules is called **`math`**, and includes all the functions from your calculator. As an example, consider the square root function, called **`sqrt`**, from the **`math`** module. If you just type this in the shell,

```
>>> sqrt(25)
```

you'll get an error. In order to use a module, we have to *import* it. Then to use a function from the module, we have to include the module's name. So the usage of the square root function looks like this:

```
>>> import math
>>> math.sqrt(25)
>>> 5.0
```

To see all the functions from the **`math`** module, you would look at the Python library documentation:

https://docs.python.org/3/library/math.html

2. Console input and string concatenation

You may have noticed by now that you can print multiple values with one call to the **`print`** function if you separate them with commas. Assuming you still have your variable **`x`** defined from above, try this in the shell:

```
>>>print("The answer is", x)
```

This also puts a space between items. This is convenient for quickly printing output, but sometimes you don't want a space. For example, try

```
>>>print("The price is $", x)
```

Here the output looks odd because you don't want a space after the dollar sign. In a case like this, you can use string *concatenation*. This is a fancy word meaning that you stick two strings together to make a longer string. You use the '+' symbol. Try this:

```
>>>first = "Hello"
>>>second = "World!"
>>>print(first + second)
```

Notice that no space is automatically added after "Hello ", so in practice you'd have to add the space yourself, for example,

```
>>>print(first + " " + second)
```

When combining numbers with text, you have to convert the number to a string before concatenating, using the **`str()`** function:

```
>>>print("The price is $" + str(x))
```

You have also seen the **`input()`** function for reading input. Go ahead and download the example input_test.py. You can find it here

http://web.cs.iastate.edu/~smkautz/cs127f15/examples/week2/input_test.py

You can download the file into your cs127 directory and open it in Wing 101, or you can create a new file in Wing 101 and copy/paste the code. Read it and try it out. Remember that the **`input()`** function always returns a string, so to do arithmetic with input we have to convert to a number using the **`int()`** or **`float()`** functions.

3. Writing interactive programs

In this part, you'll create two programs using the Wing 101 text editor.

a) Write a short interactive program **`pets.py`** for helping the user to select the perfect pet. It should look like this, where the items in **bold** are just examples of values entered by the user:

```
What's your favorite color? purple
What's your favorite animal? giraffe
What's your favorite number? 7
Ok, I guess your ideal pet would be a purple giraffe with 7 legs!
```

Use string concatenation to get the spacing for the output exactly as shown in the example above.

b) Next, write a short program **`donuts.py`** to calculate the total cost of an order of donuts. Suppose that donuts are .75 each or 8.00 per dozen. Prompt the user to enter a number of donuts, and print a brief message along with the total price.

Before writing the code, do a concrete example, e.g., how much should it cost for 50 donuts? How about 100 donuts? After writing the code, test it on your sample values. (Don't worry about how to format the output to show exactly two decimal places, we'll come back to that later.)

Checkpoint 1

Show the TA your completed programs **`pets.py`** and **`donuts.py`**, along with your example calculation for donuts.py.

4. Turtle graphics

Work through the first two sections of Chapter 4 ("Python Turtle Graphics") of the textbook. Complete the exercises in the second section, including the "Extend this program..." that occurs about halfway down the page.

Errors in activecode windows: If you have an error in your turtle code in an activecode window, you may have to close the browser to recover. In general it's easier to write code using turtles in Wing 101, see note below.

In Wing 101, write a program that uses the `turtle` module to draw a red 5-pointed star whose size is entered by the user. (Note: the angle at each of the points is 36 degrees. You can start with one corner at the origin and make the first line going east.)

Closing the window if you have errors: When you run a program that uses `turtle`, it brings up a new window for the graphics. If there is an error in your code, you'll see the error messages in the shell in Wing, but you won't be able to directly close the window by clicking on it. To close the window, click the drop-down menu labeled "Options" that is just above the shell in Wing. Select "Restart Shell".

Checkpoint 2

Show the TA the completed exercises from the text, your "Extend this program" from the text, and your 5 pointed star.

NOTE: How to use the turtle module in Wing 101

In order to run turtle in Wing 101, you have to make an adjustment in the Python configuration. You should only have to do this once.

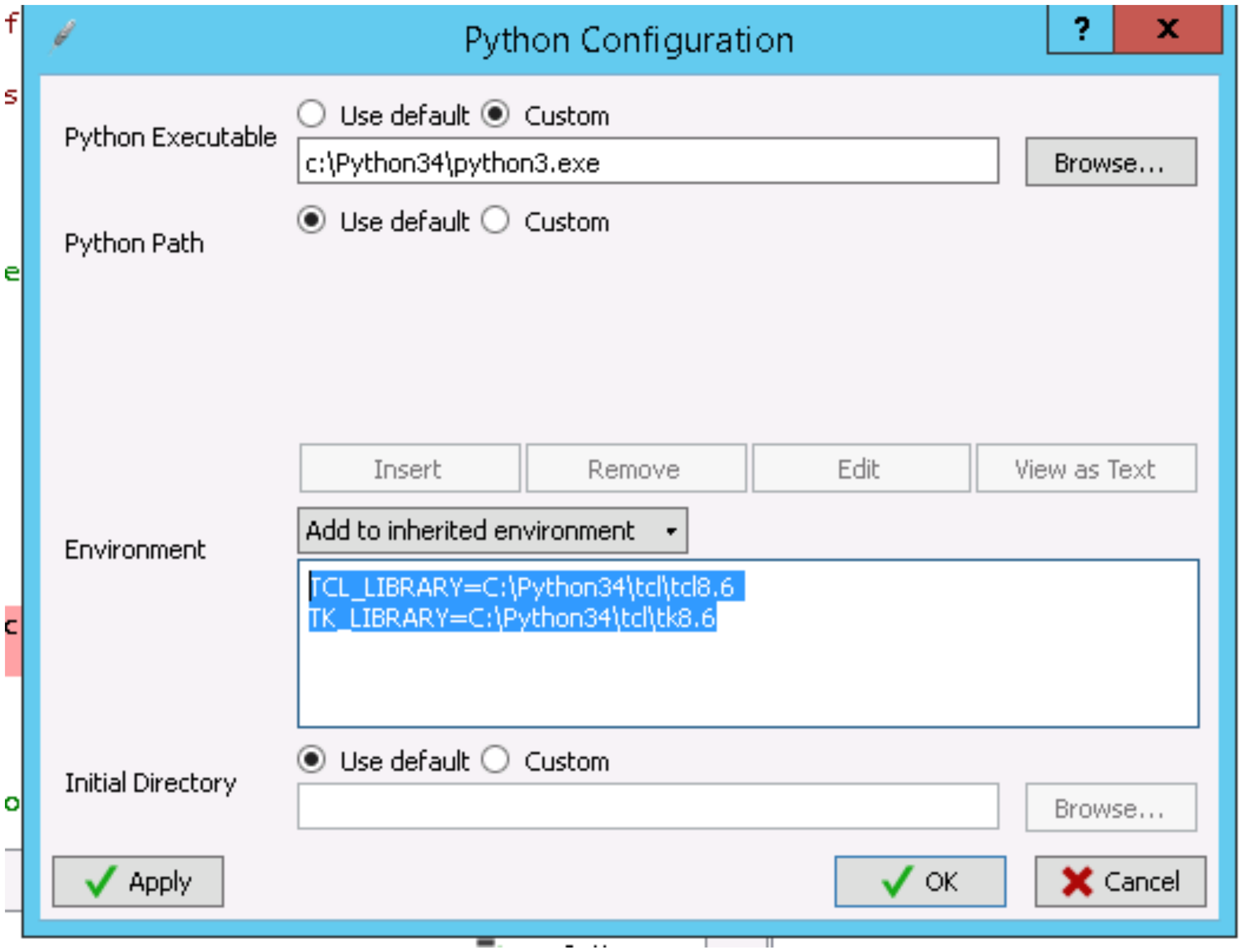
1. Go to Edit -> Configure Python...
2. In the Python Configuration dialog, under Environment, select "Add to inherited environment" from the dropdown
3. Add the following two lines in the text box (you should be able to copy/paste from here)

```
TCL_LIBRARY=C:\Python34\tcl\tcl8.6
TK_LIBRARY=C:\Python34\tcl\tk8.6
```

> (The two paths above may have to be updated when there is a newer installation of Python on the lab machines.)

4. Click OK.

(See the screenshot on the next page.)



Python Executable: Use default [✓ Custom]

c:\Python34\python3.exe

Python Path: [✓ Use default] Custom

Environment: Add to inherited environment

```
TCL_LIBRARY=C:\Python34\tcl\tcl8.6
TK_LIBRARY=C:\Python34\tcl\tk8.6
```

Initial Directory: [✓ Use default] Custom