CPADS Programming Activity III – Due 9/25

“Now I Know My S,T,O,P’s!”

One very useful and powerful programming construct that makes code both more readable, modular, and flexible is *functions.* Functions are typically used when the same non-trivial operation is done frequently but with different values. A function allows us to define the generic behavior, i.e. *encapsulation*, based on a set of *parameters* and then *call* the function whenever we wish with particular *arguments* (i.e. specific values for the parameters). This technique also helps make our code more readable by replacing a complex sequence of statements with a single meaningfully named function call. Furthermore, if we write functions well, they can be used by other people who only need to know *what* the function does (including what values they need to provide and what type of value they can expect to receive back) without consideration of *how* the computations are performed. Likewise, we can take advantage of functions other people have written without being concerned with the actual underlying code. For example, the square root function (**sqrt()** in Python) finds the square root of a number for us without us knowing how the computation is performed.

In Python, functions are usually defined at the top of the source code file using the following syntax

**def** *func\_name***(***param1, param2, …***):**

**statements**

**return** *return\_value*

where *func\_name* is the name of the function and *param1, param2, …* is the parameter list of variables that will be used *within* the function. **statements** (i.e. the function *body*) are any valid Python code that uses the parameters to perform a desired computation (including declaring *local* variables within the function). Unlike C/Java, because of the dynamic typing in Python, we do not specify the data type for the parameters. If the function will return a *single* value, the **return** statement can be used. Note that in Python, the body of the function (i.e. statements within the function) are denoted by *indentation* (unlike C/Java which denotes the function body with {}).

Once we have defined a function, we may use it anywhere we wish, including in the main program and/or in other functions, by making a function *call* at the location we wish to invoke the function’s behavior using the syntax

*var* **=** *func\_name***(***arg1, arg2, …***)**

where *var* is a variable to store the return value in (if the function returns a value) and *arg1, arg2, …* are *expressions* (including values and variables) whose *value* will be *passed* to the function (i.e. used to set the corresponding *parameters* when the function executes). Hence we must have the same number of arguments in the function call as there are parameters (in the same order), but they need not have the same names. We must be careful to pass arguments of appropriate types to ensure the function performs correctly (later after learning about decisions we will be able to test the arguments prior to passing them to the function).

**1. You make the call.**

The first exercise is similar to the first program from the previous programming activity, but this time using a function to draw a right angle.

* Open IDLE (**Start->All Programs->Python 2.7->IDLE (Python GUI)**).
* Open a new editor window (**File->New Window)** and type the following code (note the indentation is done using <tab>)

**# Load TurtleWorld functions**

**from TurtleWorld import \***

**# Right angle function**

**def right\_ang(t,size):**

**fd(t,size)**

**rt(t,90)**

**fd(t,size)**

**rt(t,90)**

**# Main program function**

**def main():**

**# Create TurtleWorld object**

**world = TurtleWorld()**

**# Create Turtle object**

**turtle = Turtle()**

**turtle.delay = 0.01**

**# Define variables**

**length = 100**

**# Draw graphics**

**right\_ang(turtle,length)**

**# Press enter to exit**

**key = input(‘Press enter to exit’)**

**world.destroy()**

**# Call main program**

**main()**

* After the *call* to **main()**, i.e. at the end of the program, add a second function call to **right\_ang()** passing the same arguments as the call to **right\_ang()** in the **main()** function. What is the output of this program? Explain any error messages that appear.
* Move the call to **right\_ang()** that you added at the end of the program to inside the **main()** function after the existing call to **right\_ang().** Modify the second argument in this new call to make it twice the value of length, i.e. **2\*length**.
* Save the program somewhere in your documents folder (e.g. Documents/CS100/programs) with the filename **rightfunc.py**
* Run the program by **Run->Run Module** (or using <F5>)

Sketch the output produced in the turtle graphics window.

**2. “Give Me a P!”**

# Load TurtleWorld functions

from TurtleWorld import \*

# Right angle function

def right\_ang(t,size):

fd(t,size)

rt(t,90)

fd(t,size)

rt(t,90)

**# Draw P function**

**def draw\_P(t,height):**

**# Put pen down before drawing the letter**

**pd(t)**

# Main program function

def main():

# Create TurtleWorld object

world = TurtleWorld()

# Create Turtle object

turtle = Turtle()

turtle.delay = 0.01

# Define variables

length = 100

# Draw graphics

**draw\_P(turtle,length)**

# Press enter to exit

key = input(‘Press enter to exit’)

world.destroy()

# Call main program

main()

Using the skeleton code above (based on the previous program), write the function **draw\_P()** that takes two parameters, **t** representing the turtle used to draw and **height** representing the height of the letter such that:

* The function draws a block **P** where the **height is twice the width** and assumes the turtle cursor *starts* in the *upper-left* corner and is facing *right.*
* **The turtle cursor MUST *end* in the *upper-right* corner and be facing *right.***
* *All* drawing **MUST** be done using the **right\_ang()** function (which ***MUST NOT*** be modified). In the **draw\_P()** function, the turtle can be repositioned without drawing, i.e. when the pen is up, using any of the turtle graphics drawing commands.

Save the program as **drawP.py** and show the instructor your output.

**3. STOP POP TOPS!**

The greatest advantage to using functions is the ability to perform the same operations at different points in our programs. Using your **drawP.py** program as a reference and the following skeleton code

**# Load TurtleWorld functions**

**from TurtleWorld import \***

**# Right angle function**

**def right\_ang(t,size):**

**fd(t,size)**

**rt(t,90)**

**fd(t,size)**

**rt(t,90)**

**# TODO: Draw P**

**def draw\_P(t,height):**

**pd(t)**

**# TODO: Draw O**

**def draw\_O(t,height):**

**pd(t)**

**# TODO: Draw Space**

**def draw\_Space(t,height):**

**pu(t)**

**# TODO: Other drawing functions**

**# Main program function**

**def main():**

**# Create TurtleWorld object**

**world = TurtleWorld()**

**# Create Turtle object**

**turtle = Turtle()**

**turtle.delay = 0.01**

**# Define variables**

**letter\_height = 75**

**# TODO: Draw graphics**

**# Create Inspector Turtle to show the center of the TurtleWorld window**

**inspector = Turtle()**

**# Press enter to exit**

**key = input(‘Press enter to exit’)**

**world.destroy()**

**# Call main program**

**main()**

**AS YOU WRITE EACH FUNCTION,** add a function call in the **main()** function to *test* your new function for correct operation using **turtle** and **letter\_height** as arguments. Refer to the example screenshots on the next page for the shape of each letter.

* Write the function **draw\_O()** that takes two parameters, **t** representing the turtle used to draw the letter and **height** representing the height of the letter such that:
  + The function draws a block **O** where the **height is twice the width** and assumes the turtle cursor *starts* in the *upper-left* corner and is facing *right.*
  + **The turtle cursor MUST *end* in the *upper-right* corner and be facing *right.***
  + Consider using the **right\_ang()** function when appropriate, but you may draw some segments of the **O** without the **right\_ang()** function.
* Write the function **draw\_Space()**such that:
  + It moves the turtle forward 1/5 the height of the letters without drawing
* Call the **draw\_P()**, **draw\_O()**, and **draw\_Space()** functions in **main()** function to create the word **POP** in the **center** of the turtle graphics window.
  + The center of the **O** in **POP** should be in the middle of the screen (where the turtle initially starts indicated by the inspector turtle)
* Write the functions **draw\_S()** and **draw\_T()** following the same guidelines listed above for the **draw\_S()** function. Again, assume the turtle starts at the upper-left corner of the letter and must end at the upper-right corner.
* Call the various drawing functions to write the word **STOP** above the word **POP** in the turtle graphics window.
  + Before writing the word **STOP** you will need to reposition the turtle at the start of the **S** in **STOP**
  + Make the spacing between *lines* 1/4 the height of the letters.
  + The word **STOP** must also be centered horizontally in the turtle graphics window.
* Call the various drawing functions to write the word **TOPS** below the word **POP** in the turtle graphics window.
  + Before writing the word **TOPS** you will need to reposition the turtle at the start of the **T** in **TOPS**
  + Make the spacing between *lines* 1/4 the height of the letters.
  + The word **TOPS** must also be centered horizontally in the turtle graphics window.
* Changing the **letter\_height** variable in **main()** should scale the entire output (maintaining the same centered position).
* Save the file as **stoppoptops.py**, print out and attach a copy to this activity, and submit your source file through Marmoset (**https://cs.ycp.edu/marmoset/** ) in **program02**.

*Hints*:

* **THINK BEFORE YOU CODE.** A little thoughtful planning with paper and pencil will pay large dividends when it comes time to write code.
* The screenshot on the left uses a value of 75 while the one on the right uses a value of 25 without any other modifications to the program.

