Name _____

CPADS Programming Activity IV – Due 11/10

"Smell the Flower!"

Now that we have some basic experience with modularizing our programs using functions, we are ready to introduce another fundamental programming concept – *iteration*, i.e. repeatedly performing a set of instructions. In this activity we will investigate *fixed iteration*, i.e. doing something a predetermined number of times. For this type of iteration, the loop is executed **a set number of times** typically through a range of values tracked with a *loop counter*.

Almost every programming language has looping structures to perform fixed iteration. Python (similar to C/Java) uses the `for` construct which has the following syntax

```
for loop_counter in range(num_loops):
      statements
```

where *loop_counter* is a variable that will cycle through the values from 0 to *num_loops*-1 (for a total of *num_loops* iterations). As with functions, the body of the loop is indicated by *indentation*, otherwise any valid Python code may be used within the loop.

1. "Poly wants a cracker!"

Create a new project named **Activity4**, and a new Python file named **poly.py**. Enter the following skeleton code as a starting point:

```
# Load TurtleWorld functions
from TurtleWorld import *

# Polygon function
def draw_polygon(t, length, n):
      ang = 360.0/n
      for i in range (n):
            fd(t,length)
            rt(t,ang)

# Main program function
def main():
      # Create TurtleWorld object
      world = TurtleWorld()

      # Create turtle object
      turtle = Turtle()

      # Define polygon variables
      num_sides = int(input('Enter the number of sides '))
      side_length = int(input('Enter the length of each side '))

      # Draw graphics
      draw_polygon(turtle, side_length, num_sides)

      # Press enter to exit
      key = input('Press enter to exit')
      world.destroy()

#Call main program
main()
```

Name _____

- Sketch the output of the program for 6 sides with length 100.

- What happens if different values are entered, e.g. 8 sides of length 50?

Name _____

2. "Help me Noah!"

The Turtle graphics package constructs objects using only line segments. To approximate curves (for things like circles), we can use small line segments drawn with a slight angular change between each segment using iteration). Arcs (and circles) are usually specified by a *radius* **`r`** and an *angle* **`theta`**. Thus, we can create an arc similarly to the **`draw_polygon()`** function by first computing the *arc length* **`arc_len`** (fractional circumference of a circle) which is given by the formula

$$arc\_len = (2\pi r)(theta / 360)$$

If each segment has length **`seg_len`**, the number of segments **`num_seg`** (*as an integer*) is given by

$$num\_seg = \text{int}(arc\_len / seg\_len)$$

Finally the angular change between each segment **`dang`** for a total angle **`theta`** is

$$dang = theta / num\_seg$$

Create a new Python file named **`arc.py`** and enter the following skeleton code as a starting point:

```
# Load TurtleWorld functions
from TurtleWorld import *
import math

# TODO: Arc function
def draw_arc(t,r,theta,seg_len):
      pd(t)

# Main program function
def main():
      # TurtleWorld objects
      world = TurtleWorld()
      turtle = Turtle()
      turtle.delay = 0.01

      # Arc variables
      radius = int(input('Enter radius '))
      angle = int(input('Enter angle '))
      line_len = int(input('Enter segment length '))

      # Draw graphics
      draw_arc(turtle,radius,angle,line_len)

      # Press enter to exit
      key = input('Press enter to exit')
      world.destroy()

# Call main program
main()
```

Name _____

- Complete the function named **`draw_arc()`** that takes four parameters – **`t`** for the drawing turtle, **`r`** for the arc radius, **`theta`** for the arc angle, and **`seg_len`** for the length of *each segment*. In the function definition, use the formulas above to compute the number of segments needed and the angle to turn between each segment. Then add a **loop** (similar to the **`draw_polygon()`** function from part 1) to construct the arc. NOTE: You should only be using the parameters and any locally created variables in this function!

- Test your function using the values 50 for the radius, 60 for the angle, and 4 for the segment length. Show your output to the instructor.

- Test your function again using 360 for the angle. You should see a COMPLETE circle.

3. "Smell the Flowers!"

We will now use the **`draw_arc()`** function (since you did show it to your instructor?) to make "spiro-graph" type pictures. Using the skeleton code below:

```
# Load TurtleWorld functions
from TurtleWorld import *
import math

# TODO: Arc function
def draw_arc(t,r,theta,seg_len):
      pd(t)

# TODO: Petal function

# TODO: Flower function

# Main program function
def main():
      # Create TurtleWorld objects
      world = TurtleWorld()
      turtle = Turtle()
      turtle.delay = 0.01

      # Flower variables
      num_petals = int(input('Enter the number of petals '))

      # Draw graphics
      # draw_petal(turtle, 100, 90, 4)
      # draw_flower(turtle, num_petals)

      # Press enter to exit
      key = input('Press enter to exit')
      world.destroy()

# Call main program
main()
```

Name _____

- Create a new Python file named **`flower.py`** and enter the above skeleton code as a starting point. Now, copy your **`draw_arc()`** function from part 2 into the new file (**note** this is code reuse!)

- Write a function called **`draw_petal()`** (place it after the **`draw_arc()`** function) that takes four parameters – **`t`** for the drawing turtle, **`r`** for the arc radius, **`theta`** for the arc angle, and **`seg_len`** for the length of *each segment*. The function should draw **a single petal** at the current turtle position. The turtle should end up at *exactly the same position and orientation* as it started. **TEST** your **`draw_petal()`** function by uncommenting the call in **`main()`**. Try changing the third argument (arc angle) to 45 -- it should now draw a thinner petal than before.

- Write a function called **`draw_flower()`** (place it after the **`draw_petal()`** function) that takes two parameters – a turtle **`t`**, and the number of petals **`petals`** and constructs a figure similar to below. Note: the width of the petals should scale with the *number* of petals such that the flower remains approximately the same size. Consider using the **`draw_petal()`** function in a **loop** (think about how many petals you need to draw and the angular adjustment between each one) to construct the figure.

- When you're ready to submit your program:
  - Print out and STAPLE a copy of your **`flower.py`** file to this activity.
  - Submit your source file through Marmoset (**https://cs.ycp.edu/marmoset/**).
    - Enter your login information which you should have received in an e-mail (you probably should change your password to match your YCP account)
    - Select **CS100: Computer Science Practice and Design Studio**
    - Select the **submit** link under **web submission** for **program03**
    - Click **Choose File…** , navigate to your program directory and select your **`flower.py`** file (do not worry about the instructions for jar and zip files).
- Click **Submit project!**

HINTS:

- **Think** about how to draw a single petal. In particular, if the turtle begins facing right it will end up at a *corner* of the regular polygon (refer to the first activity in this lab) with the same number of sides as the flower has petals.

- Then consider how far the turtle has to turn to draw the *same* arc to get back to the center.

- Finally think about how far the turtle must then turn to draw the next petal, and repeat the process via a loop.

- Adjust the arc length, i.e. size of the flower, based on the number of petals. Note that the larger the angle of the arc, the larger the arc (since we are using a fixed segment length). Consider how to adjust the *radius* of the arc to compensate.

CS100 Fall 2014

Name _____