Name _____

CPADS Programming Activity V – Due 11/21

"Let It Snow!"

So far we have covered the basics of creating functions and implementing (fixed) iteration. The next important programming concept is decisions, i.e. *conditional* execution. The most common structure for decisions is the `if-else` construct. In Python the syntax is:

```
if condition:
      true statements
elif condition2:
      true2 statements
      ...
else:
      false statements
```

Just as with functions and iteration, the body of each branch of the decision logic is indicated by *indentation*, otherwise any valid Python code may be used within the branch.

We can also combine iteration and decision structures to create *conditional* iteration (a `while` loop). For conditional iteration, the loop will execute *until* the condition statement becomes false. In Python, the syntax is:

```
while condition:
      statements to execute while condition is true
```

1. User Input

Typically, one important aspect of a program is to obtain input from the user. This can be done graphically via the mouse, or textually via the keyboard. In Python, the command to get user input – ***as a string*** – from the keyboard is

```
var = input('prompt')
```

where *var* is a variable used to store the user input and *prompt* is a literal string that is displayed to the user to explain what type of value they should enter.

To convert the string into a number, Python has the following functions:

`n = int(var)` – (tries) to convert the string *var* to an integer that is stored in `n`
`n = float(var)` – (tries) to convert the string *var* to a float (i.e. decimal) that is stored in *n*

Name _____

- Open PyCharm and create a new project named **Activity5**. Open the Python console: **Tools -> Run Python Console …**. In the console, enter the following bold Python commands in the console window. Note the output:

```
>>> var = input('Enter the number 123.4:')
Enter the number 123.4: 123.4

_____

>>> int(var)

_____

>>> float(var)

_____
```

Note that if you try to convert a string of incompatible type, Python returns an error message called an *exception*. As with many other programming languages, Python provides a mechanism for the programmer to decide how to handle exceptions when they occur (via a `try/except` structure) but we will not deal with that in this course.

2. Snowflake

An interesting geometric figure known as a *Koch* curve can be drawn using a *recursive* function, i.e. a function that calls itself. One important consideration that must be observed when dealing with recursive functions is to ensure that they eventually reach a *termination* point, otherwise they will recurse indefinitely and eventually crash your program. To determine where to terminate, a recursive function always has some type of decision logic within the function. The recursive algorithm for drawing a Koch curve of length *L* is show below. Note the different drawing done in each branch of the conditional statement.

```
Koch(L):
      if L > 2:
            Draw a Koch of length L/3
            Turn left 60 degrees
            Draw a Koch of length L/3
            Turn right 120 degrees
            Draw a Koch of length L/3
            Turn left 60 degrees
            Draw a Koch of length L/3
      else:
            Draw a line of length L
```

Name _____

Create a new Python file named **snowflake.py** and input the following skeleton code:

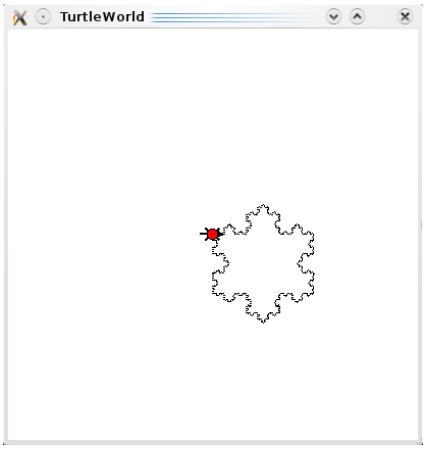
```
# Load TurtleWorld functions
from TurtleWorld import *

# TODO: Koch function
def Koch(t, length):
      pd(t)

# TODO: Input function

# Main program function
def main():
      # Create TurtleWorld objects
      world = TurtleWorld()
      turtle = Turtle()
      turtle.delay = 0.001

      # Snowflake size
      size = 100

      # Draw graphics
      Koch(turtle,size)

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

# Call main function
main()
```

- Write a function named **Koch()** that takes two parameters – **t** for the drawing turtle and **length** for the length of the curve. Save the program as **snowflake.py.** Show the instructor your output.

- Extend your program to add a function called **get_input()** that has one parameter – **prompt** that will contain the string to display to the user in the Python console. The function **get_input()** should *return* the **integer value** of the user input (assume the user will enter a string representing an integer). In the **main()** function, replace the assignment statement for **size** to set its value using a function call to **get_input()**. Only draw the Koch curve if the length entered is greater than zero (a programming technique known as *data validation*). **Hint:** The data validation should be done in **main()** not **get_input().**

Name _____

- Extend your program by adding code to the **`main()`** function to draw a *snowflake* using 3 Koch curves, see the sample output below. Hint: Consider each Koch curve as the side of a triangle and use a *loop*.



- Test your program using both valid values, e.g. 100, and invalid values, e.g. -2.

3. Blizzard

- Create a new Python file named **`blizzard.py`** and copy the code from your **`snowflake.py`** into your new **`blizzard.py`** file.

- In your **`blizzard.py`** file, move the snowflake drawing code from the main program into a function called **`draw_snowflake()`** that has two parameters – **`t`** for the turtle to use for drawing and **`size`** for the size of the snowflake to draw.

- Call the **`get_input()`** function a second time in the main program to obtain the number of snowflakes the user would like to draw. Note: You will want supply a different prompt string as the *argument* in this call with appropriate text.

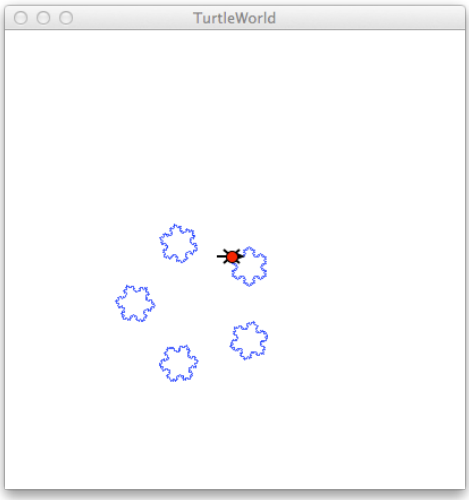
- Write a function called **`draw_blizzard()`** that has three parameters – **`t`** for the turtle to use for drawing, **`size`** for the size of the snowflakes to draw, and **`n`** for the number of snowflakes to draw. The function should draw snowflakes at what would be the corners of a polygon with *n* sides.

- Add a function call in the **`main()`** funtion to **`draw_blizzard()`** (with appropriate arguments) that continually draws the snowflakes **AS LONG AS** the user continues to enter a positive number of snowflakes. If the user enters a negative number of snowflakes, your program should terminate. If the user enters a negative number for the snowflake size your program should re-prompt the user for a valid input. You should clear the TurtleWorld screen between each drawing using the command **`world.clear()`**.

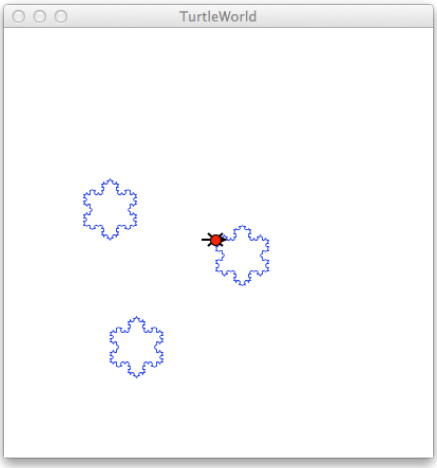
Name _____

- A sample output run is shown below (note this is a ***single*** execution of the program).

```
Please enter the number of snowflakes: 5
Please enter the snowflake size: 30
```




```
Please enter the number of snowflakes: 3
Please enter the snowflake size: 50
```




```
Please enter the number of snowflakes: -2
Thank you for playing in the snow.
```

- When done, print out and attach a copy of your **`blizzard.py`** program to this activity. Additionally, submit your source file through Marmoset as **program04** (https://cs.ycp.edu/marmoset).