ME 5194: HW 2

Adrian Bakhtar

CHAPTER 3

Exercise 3.1

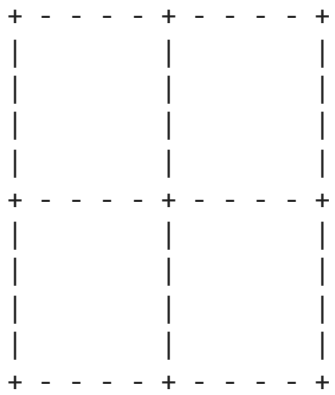
Write a function named right_justify that takes a string named s as a parameter and prints the string with enough leading spaces so that the last letter of the string is in column 70 of the display.

In [1]:
```
def right_justify(string, column_num):
    arg_length = len(string)
    diff_to_column = column_num-arg_length
    result = ' '*diff_to_column+string
    print(result)
    print('The last letter of the inputted string is at column {}.'.format(len(result)

right_justify('monty', 70)
```

```
                                                                     monty
The last letter of the inputted string is at column 70.
```

Exercise 3.3

1. Write a function that draws a grid like the following:



Hint: to print more than one value on a line, you can print a comma-separated sequence of values: print('+', '-')

By default, print advances to the next line, but you can override that behavior and put a space at the end, like this:
print('+', end=' ')
print('-')

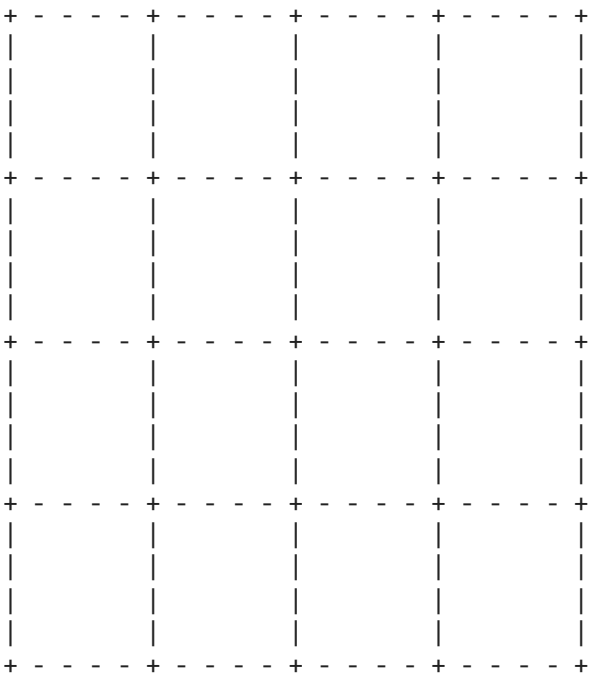
The output of these statements is '+ -' on the same line. The output from the next print statement would begin on the next line.

In [3]:
```
def square_grid(num_rows, num_columns, square_size):
    """
        Generates a square grid based off the number of rows, columns, and size of the
        that are entered by the user.

    Args:
        num_rows: int
            Desired number of rows
        num_columns: int
            Desired number of columns
        square_size: int
            Desired size of the square

    """
    num_squares = num_rows*num_columns
    num_pluses = num_squares +1
    num_dashes = num_vert_bar = square_size

    for j in range(num_rows):
        print(('+ ' + '- '*num_dashes)*num_columns + '+')

        for i in range(square_size):
            print(('| ' + '  '*num_vert_bar)*num_columns + '|')

    print(('+ ' + '- '*num_dashes)*num_columns + '+')

square_grid(2, 2, 4)
```

```
+ - - - - + - - - - +
|         |         |
|         |         |
|         |         |
|         |         |
+ - - - - + - - - - +
|         |         |
|         |         |
|         |         |
|         |         |
+ - - - - + - - - - +
```

1. Write a function that draws a similar grid with four rows and four columns.

In [4]:
```
square_grid(4,4,4)
```



CHAPTER 4: TURTLE ACTIVITY

I was having some trouble running this code. I kept getting a traceback error once I tried to move the turtle. After splitting the code into 3 cells, it works more consistently, but I still get these traceback errors. I would have to restart and clear output from the kernal before running it again. I think it is something to do with running this in a Jupyter Notebook.

In [ ]:
```
import turtle

t = turtle.Turtle()
print(t)
```

In [ ]:
```
for i in range(4):
    t.fd(100)
    t.lt(90)
```

In [ ]:
```
turtle.mainloop()
```

Section 4.3 - Exercise 4.1

Write a function called square that takes a parameter named t, which is a turtle. It should use the turtle to draw a square. Write a function call that passes bob as an argument to square, and then run the program again.

In [ ]:
```
import turtle

bob = turtle.Turtle()
print(bob)
```

In [ ]:
```
turtle_square(bob)

def turtle_square(turtle_obj):
    for i in range(4):
        turtle_obj.fd(100)
        turtle_obj.lt(90)
    turtle_obj.mainloop()
```

Section 4.3 - Exercise 4.2

Add another parameter, named length, to square. Modify the body so length of the sides is length, and then modify the function call to provide a second argument. Run the program again. Test your program with a range of values for length.

In [ ]:
```
import turtle

bob = turtle.Turtle()
print(bob)
```

In [ ]:
```
def turtle_square(turtle_obj, length):
    counter = 0
    for i in range(20):
        turtle_obj.fd(length+counter)
        turtle_obj.lt(90)
        counter+=25

    turtle_obj.mainloop()

turtle_square(bob, 100)
```

Section 4.12 - Exercise 4.1

Download the code in this chapter from http: // thinkpython2. com/ code/ polygon.py.

In [16]:
```
"""This module contains a code example related to

Think Python, 2nd Edition
by Allen Downey
http://thinkpython2.com

Copyright 2015 Allen Downey

License: http://creativecommons.org/licenses/by/4.0/
"""

from __future__ import print_function, division

import math
import turtle


def square(t, length):
    """Draws a square with sides of the given length.
```

```
    Returns the Turtle to the starting position and location.
    """
    for i in range(4):
        t.fd(length)
        t.lt(90)


def polyline(t, n, length, angle):
    """Draws n line segments.

    t: Turtle object
    n: number of line segments
    length: length of each segment
    angle: degrees between segments
    """
    for i in range(n):
        t.fd(length)
        t.lt(angle)


def polygon(t, n, length):
    """Draws a polygon with n sides.

    t: Turtle
    n: number of sides
    length: length of each side.
    """
    angle = 360.0/n
    polyline(t, n, length, angle)

def arc(t, r, angle):
    """Draws an arc with the given radius and angle.

    t: Turtle
    r: radius
    angle: angle subtended by the arc, in degrees
    """
    arc_length = 2 * math.pi * r * abs(angle) / 360
    #print(arc_length)

    n = int(arc_length / 4) + 3
    #print(n)

    step_length = arc_length / n
    #print(step_length)

    step_angle = float(angle) / n
    #print(step_angle)

    # making a slight left turn before starting reduces
    # the error caused by the linear approximation of the arc
    t.lt(step_angle/2)
    polyline(t, n, step_length, step_angle)
    t.rt(step_angle/2)

def arc_og(t, r, angle):
    arc_length = 2 * math.pi * r * angle / 360
    n = int(arc_length / 3) + 1
    step_length = arc_length / n
```

```
    step_angle = float(angle) / n
    polyline(t, n, step_length, step_angle)

def circle(t, r):
    """Draws a circle with the given radius.

    t: Turtle
    r: radius
    """

    arc(t, r, 360)
    #arc_og(t, r, 360)
```

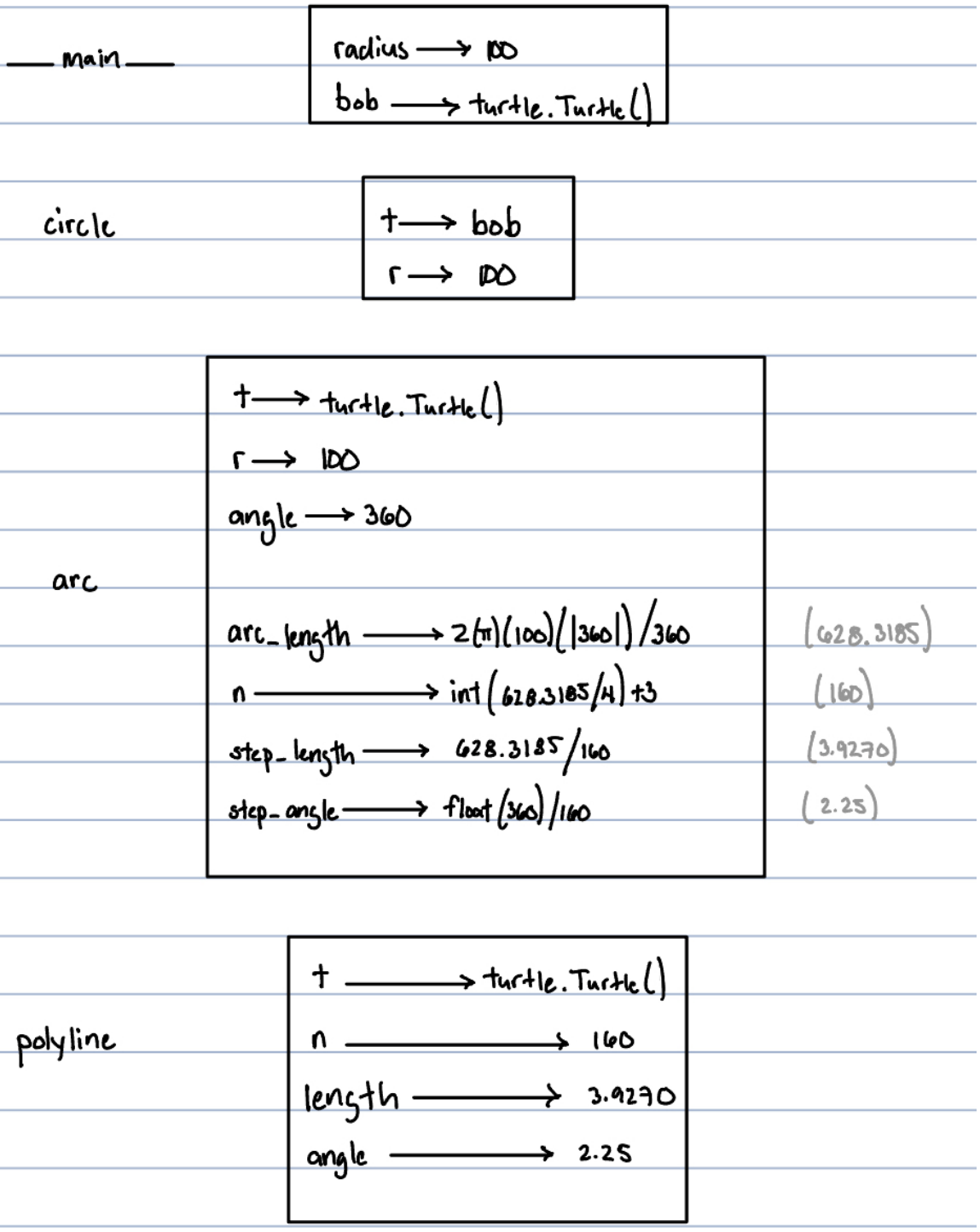
In [19]:
```
# the following condition checks whether we are
# running as a script, in which case run the test code,
# or being imported, in which case don't.

if __name__ == '__main__':
    bob = turtle.Turtle()

    # draw a circle centered on the origin
    radius = 200
    bob.pu()
    bob.fd(radius)
    bob.lt(90)
    bob.pd()
    circle(bob, radius)

    # wait for the user to close the window
    turtle.mainloop()
```

1. Draw a stack diagram that shows the state of the program while executing circle(bob,radius). You can do the arithmetic by hand or add print statements to the code.

In [1]:
```
from IPython.display import Image
Image("/home/abakhtar/Downloads/4_3_exercise_4_1.jpeg")
```

Out[1]:

Exercise 4.1:



1. The version of arc in Section 4.7 is not very accurate because the linear approximation of the circle is always outside the true circle. As a result, the Turtle ends up a few pixels away from the correct destination. My solution shows a way to reduce the effect of this error. Read the code and see if it makes sense to you. If you draw a diagram, you might see how it works.

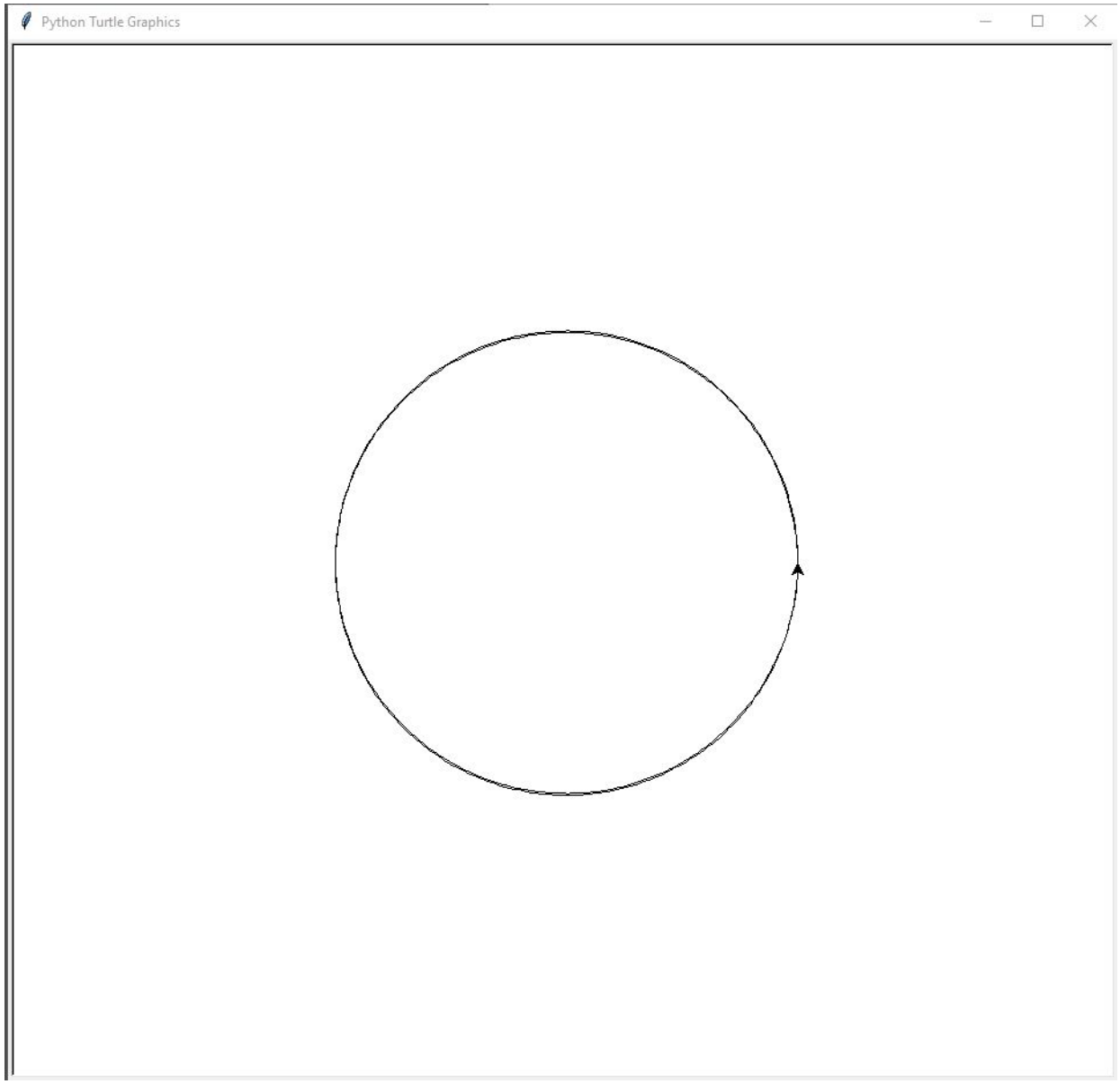
After running both the original (arc_og) and corrected (arc) functions on top of each other, it can be seen that there is an outer circle that is just slightly larger than the inner circle. This outer circle is from the inacurracy of the linear approximation of the uncorrected arc function. The inner circle is from the corrected arc function, proving that arc_og's linear approximation is just outside the true circle.

The way the arc function corrects this is by making a very fine left turn before calling the polyline function to draw the line segments, then by making a right turn by the same amount.

The output of the two circles can be seen in the next cell.

In [18]:
```
from IPython.display import Image

Image("//coeit.osu.edu/home/b/bakhtar.1/Downloads/exercise_4_1_thinkpython_HW2.jpg")
```

Out[18]:



Section 4.12 - Exercise 4.2

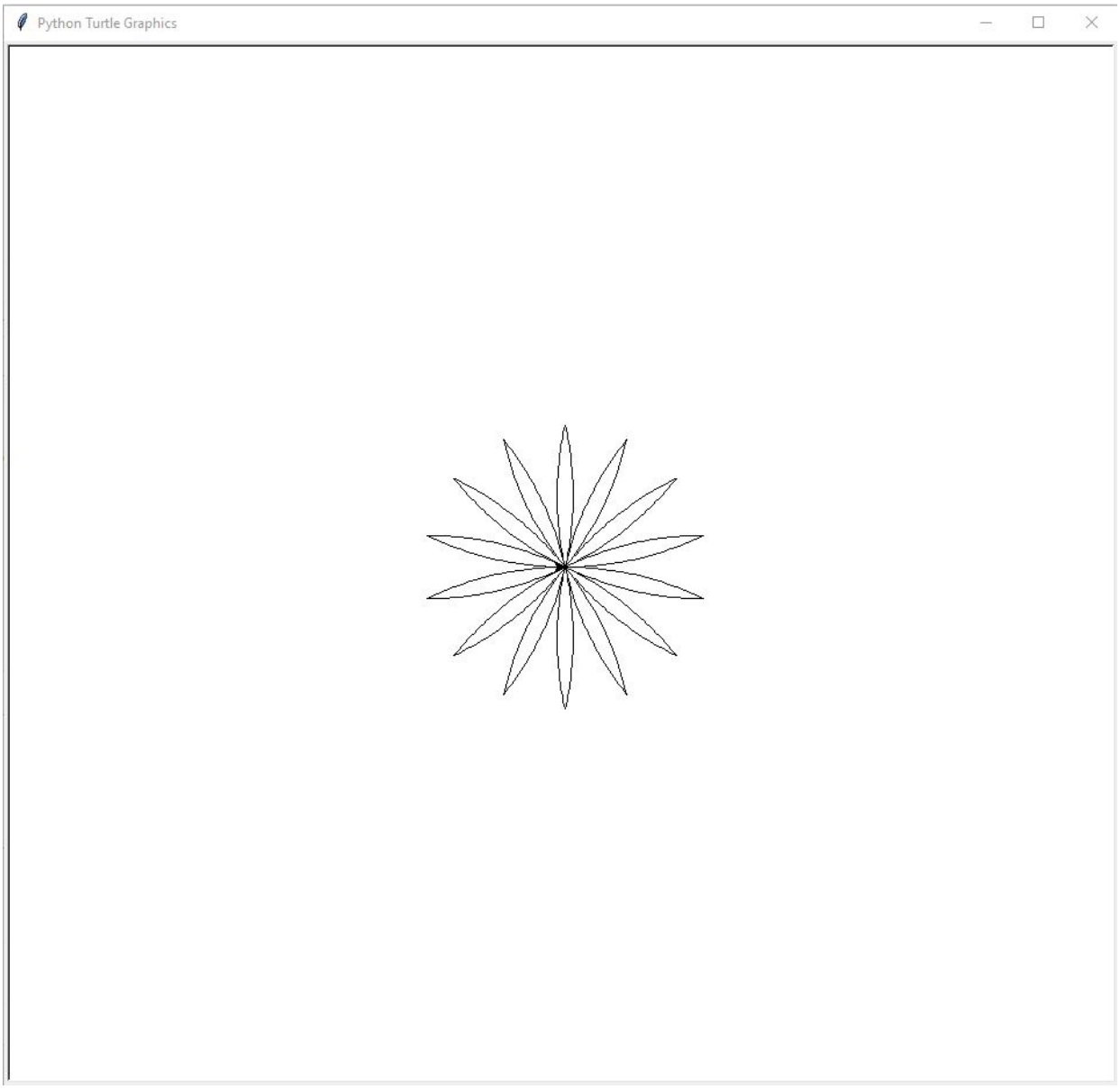
Write an appropriately general set of functions that can draw flowers as in Figure 4.1.

In [11]:
```
#Run the cell containing all of the functions before running this one.
from turtle import Turtle

t = Turtle()
print(t)


def symmetrical_flower(t_obj, num_petals, radius):
    angle = 360.0/num_petals

    for i in range(num_petals):
        for i in range(2):
            arc(t, radius, angle)
            t.lt(180-angle)
        t.lt(360.0/num_petals)


symmetrical_flower(t, 14, 275)
```

```
<turtle.Turtle object at 0x000001834DD232E0>
```

In [17]:
```
Image("//coeit.osu.edu/home/b/bakhtar.1/Downloads/exercise_4_2_thinkpython_HW2.jpg")
```

Out[17]:



CHAPTER 5

Recursion Exercise

Exercise 5.1

The time module provides a function, also named time, that returns the current Greenwich Mean Time in “the epoch”, which is an arbitrary time used as a reference point. On UNIX systems, the epoch is 1 January 1970.

Write a script that reads the current time and converts it to a time of day in hours, minutes, and seconds, plus the number of days since the epoch.

In [18]:
```
import time as t
```

```
def days_since_epoch():
    time_sec = t.time() #in seconds since "the epoch"

    #Takes current time and divides it by number of seconds in a day
    #Stores the quotient as total whole days, modulus as the number of seconds left
    days, remaining_seconds1 = divmod(time_sec, 60*60*24)

    #Takes remaining seconds and divides it by number of seconds in an hour
    #Stores the quotient as total whole hours, modulus as the number of seconds left
    hours, remaining_seconds2 = divmod(remaining_seconds1, 60*60)

    #Takes remaining seconds and divides it by number of seconds in a minute
    #Stores the quotient as total whole minutes, modulus as the remaining seconds left
    hours, remaining_seconds2 = divmod(remaining_seconds1, 60*60)
    minutes, seconds = divmod(remaining_seconds2, 60)

    print('It has been {} days, {} hours, {} minutes, and {} seconds since the epoch.'


days_since_epoch()
```

```
It has been 19383 days, 0 hours, 49 minutes, and 9 seconds since the epoch.
```

Exercise 5.6

The Koch curve is a fractal that looks something like Figure 5.2. To draw a Koch curve with length x, all you have to do is

1. Draw a Koch curve with length x/3.
2. Turn left 60 degrees.
3. Draw a Koch curve with length x/3.
4. Turn right 120 degrees.
5. Draw a Koch curve with length x/3.
6. Turn left 60 degrees.
7. Draw a Koch curve with length x/3.

In [1]:
```
import turtle
from IPython.display import Image


t = turtle.Turtle()
print(t)
```

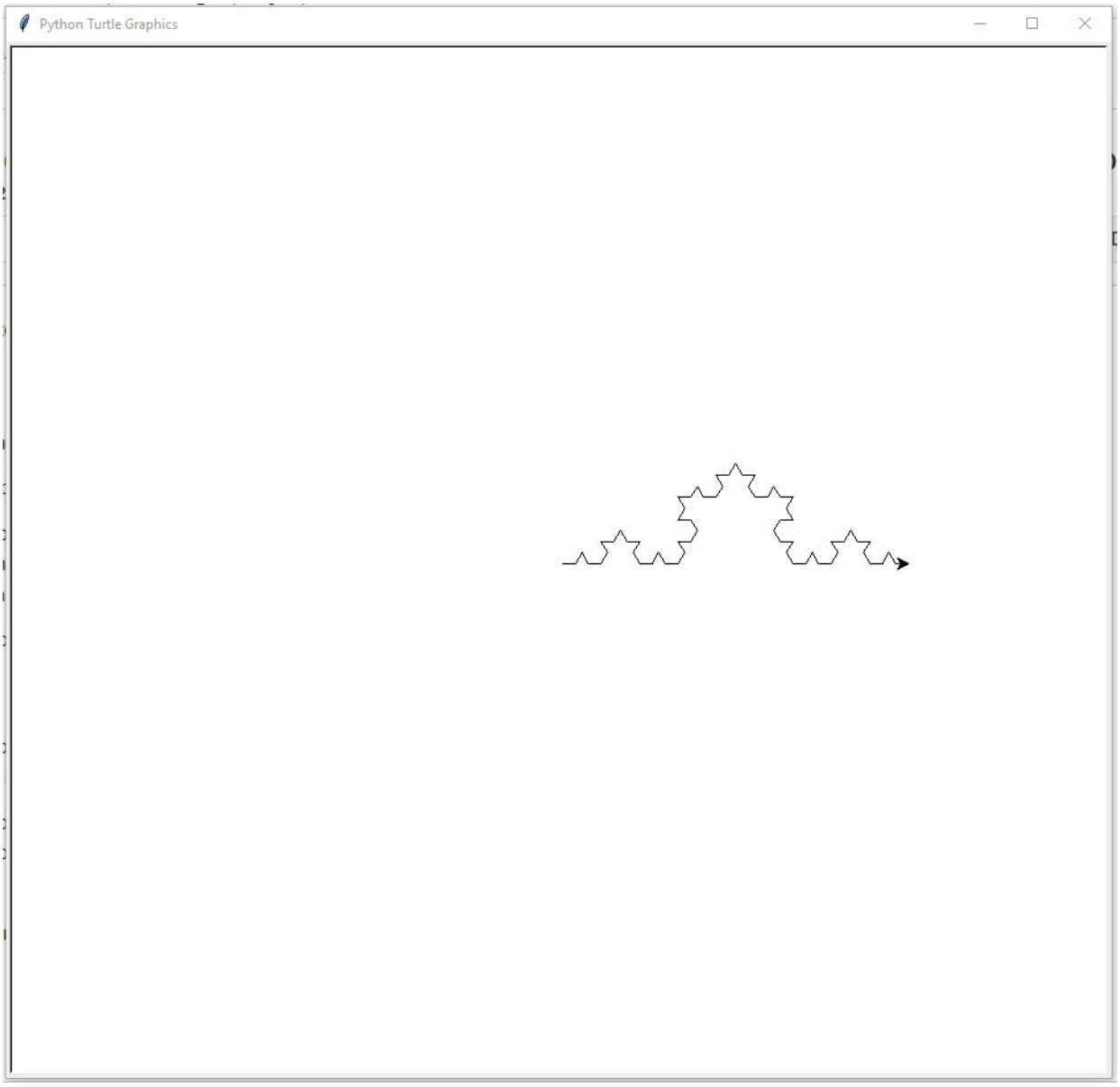
```
<turtle.Turtle object at 0x000001ABD2F84F40>
```

In [2]:
```
def koch_curve(t_obj, length, n):
    if n == 0:
        t_obj.fd(length)
        return
    else:
        koch_curve(t_obj, length/3, n-1)
        t_obj.lt(60)

        koch_curve(t_obj, length/3, n-1)
        t_obj.rt(120)
```

```
        koch_curve(t_obj, length/3, n-1)
        t_obj.lt(60)

        koch_curve(t_obj, length/3, n-1)

koch_curve(t,300, 3)


## Tried to directly save turtle output

#ts = turtle.getscreen()
#ts.getcanvas().postscript(file = "turtle_koch2.eps")
#!convert -density 150 -rotate 90 turtle_koch2.eps turtle_koch2.png

#Image(filename = 'turtle_koch2.png')
```

In [3]:
```
from IPython.display import Image
Image("//coeit.osu.edu/home/b/bakhtar.1/Downloads/exercise_5_6_thinkpython_HW2.jpg")
```

Out[3]:



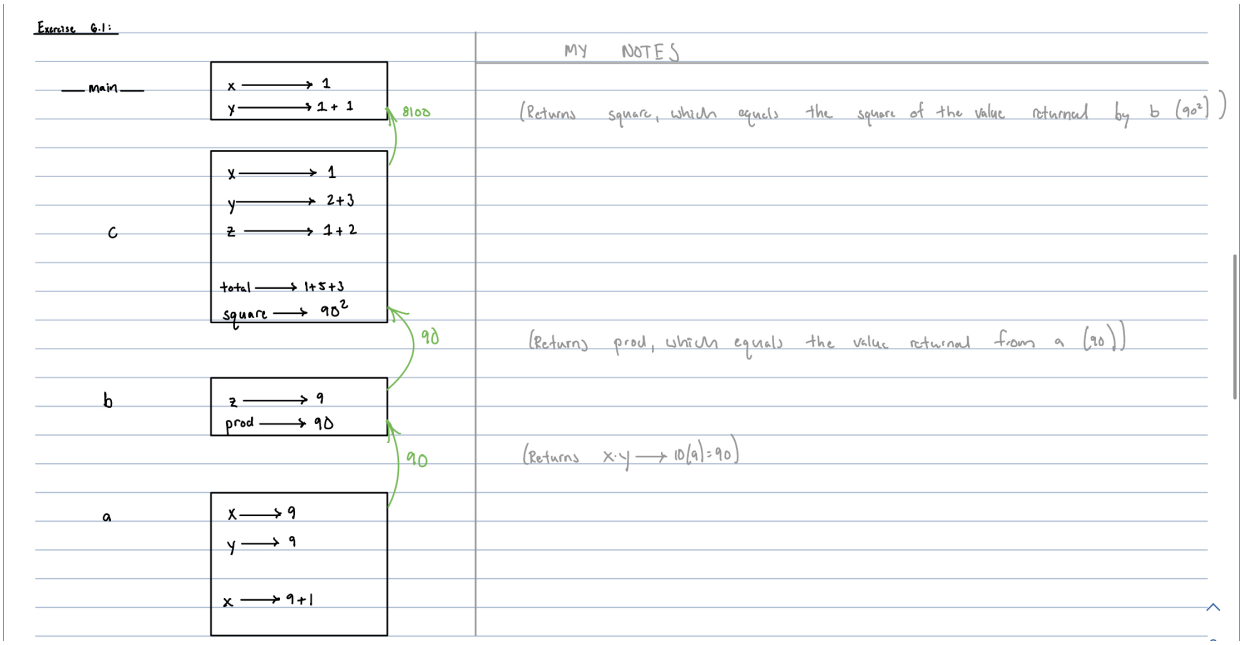
CHAPTER 6

Exercise 6.1

Draw a stack diagram for the following program. What does the program print?

In [6]:
```
Image("//coeit.osu.edu/home/b/bakhtar.1/Downloads/exercise_6_1_thinkpython_HW2.jpeg")
```

Out[6]:



The program first prints the value of of the input argument z and the value of prod from function b, which are equal to 9 and 90 respectively. The program will then print out 8100, which is the value of square that is returned from function c.

In [7]:
```
def b(z):
    prod = a(z, z)
    print(z, prod)
    return prod

def a(x, y):
    x = x + 1
    return x * y

def c(x, y, z):
    total = x + y + z
    square = b(total)**2
    return square

x = 1
y = x + 1
print(c(x, y+3, x+y))
```

```
9 90
8100
```

Exercise 6.2

The Ackermann function, A(m, n), is defined:

```
A(m, n) =
    {
        n + 1 if m = 0
        A(m − 1, 1) if m > 0 and n = 0
        A(m − 1, A(m, n − 1)) if m > 0 and n > 0
    }
```

See http: // en. wikipedia. org/ wiki/ Ackermann_ function .

1. Write a function named ack that evaluates the Ackermann function. Use your function to evaluate ack(3, 4), which should be 125.

In [10]:
```
def ack_checks(m,n):
    """
        Checks to see if the arguments passed into the ack() function are both positiv

    Args:
        m: int
            Positive integer
        n: int
           Positive integer
   Returns:
       bool: True if ack() arguments were positive integers, False if not

    """
    if isinstance(m, int) & isinstance(n, int):
        if (m>=0) & (n>=0): return True
        else:
            print('Make sure that positive numbers were entered!\n')
            #print('m: {}\t type = {}'.format(m, type(m)))
            #print('n: {}\t type = {}\n\n'.format(n, type(n)))
            return False
    else:
        print('Make sure that integer numbers were entered!\n')
        #print('m: {}\t type = {}'.format(m, type(m)))
        #print('n: {}\t type = {}\n\n'.format(n, type(n)))
        return False



def ack(m, n):
    """
        Represents the Ackermann Function, which is used to demonstrate the concept of
        function that is not primitive recursive

    Args:
        m: int
            Positive integer
        n: int
           Positive integer
   Returns:
       int: result of the Ackermann function calculations

    """
    if ack_checks(m, n):
        if m == 0: return n+1
```

```
        if n == 0: return ack((m-1), 1)

        return ack(m-1, ack(m, n-1))
```

In [11]:
```
ack(3, 4.0)
ack(-3, 4)
ack(3,4)
```

```
Make sure that integer numbers were entered!

Make sure that positive numbers were entered!

125
```

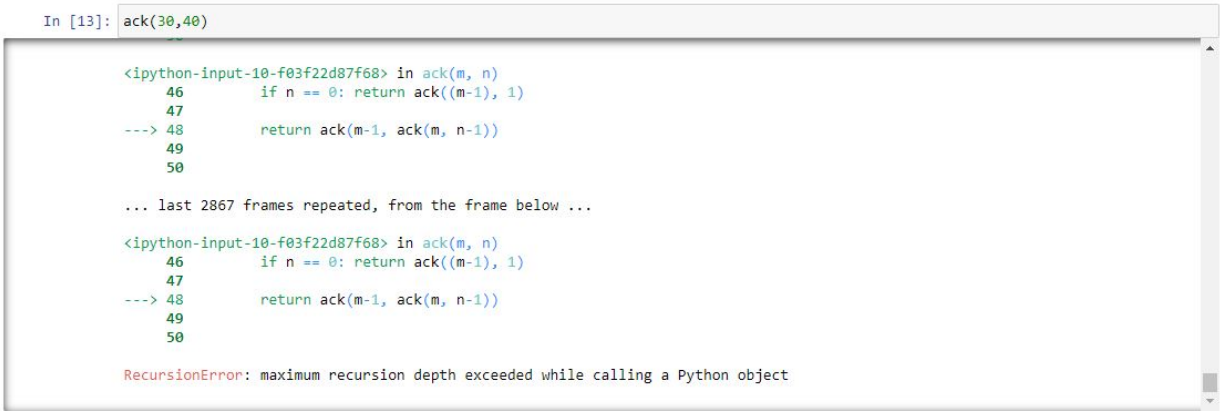
Out[11]:

1. What happens for larger values of m and n?

   For larger values of m and n, a RecursionError is raised, stating that maximum recursion depth for calling a Python object has been exceeded. The ack() function call terminates without providing the ouptut. An image of the error that is produced can be seen below.

In [10]:
```
Image("//coeit.osu.edu/home/b/bakhtar.1/Downloads/exercise_6_2_thinkpython_HW2.jpg")
```

Out[10]:



In [ ]: