Graded Exercise 1

CSC140 Foundations of Computer Science

21 February 2020

In this exercise you will define 4 functions and use them to draw a picture:

- weighted_avg_nums()
- weighted_avg_points()
- weighted_avg_of_avgs()
- weighted_avg_of_avgs_of_avgs()

The definition of each successive function will build upon definitions of previous functions.

You will also practice using tuples. A tuple is a compound datatype. In this exercise, you will use tuples to represent points in the plane.

Write a Python program. Within that program do the following:

1. Define a function weighted_avg_nums(a, b, t) that computes the weighted average of two numbers a and b with a weight t, where $0.0 \leq t \leq 1.0$.

$$weighted\_avg\_nums = (1 - t) \cdot a + t \cdot b$$

   This function returns to its caller a number.

   Hint:

   Here's how to define and call a function:

```
def arithmetic_mean( a, b ):
    result = (a + b)/2
    return result

if __name__ == "__main__":
    m = 2
    n = 4
    average = arithmetic_mean( m, n )
    print( f"average = {average:2d}" )
```

Examples:

- weighted_avg_nums(12, 20, 0.25) returns 14
- weighted_avg_nums(12, 20, 0.50) returns 16
- weighted_avg_nums(12, 20, 0.75) returns 18

2. Define a function weighted_avg_points(p0, p1, t) that computes the weighted average of two points $p0$ and $p1$ with a weight t, where $0.0 \leq t \leq 1.0$.

   Each point is a tuple:

$$p_0 = (x_0, y_0)$$
$$p_1 = (x_1, y_1)$$

   Each coordinate of the weighted average of two points is the weighted average of the corresponding coordinates of the two points:

$$\begin{aligned} p_x &= \mathit{weighted\_avg\_nums}(x_0, x_1, t) \\ &= (1-t) \cdot x_0 + t \cdot x_1 \\ p_y &= \mathit{weighted\_avg\_nums}(y_0, y_1, t) \\ &= (1-t) \cdot y_0 + t \cdot y_1 \\ \mathit{weighted\_avg\_points}(p_0, p_1, t) &= (p_x, p_y) \end{aligned}$$

   Hints:

   - Your function should include statements that correspond to the first and third equations shown above. Your function should not include expressions that correspond to those on the second and fourth lines. The arithmetic takes place inside weighted_avg_nums(), the function that weighted_avg_points(), and not in weighted_avg_points() itself.
   - Here is one way to create a tuple and get values out of it:

```
p = (3, 4) # create a tuple and give it a name
x = p[0]   # assign first element of p to x
y = p[1]   # assign second element of p to y
```

- Here is another way to get values out of a tuple:

```
p = (3, 4) # create a tuple and give it a name
x, y = p   # copy 1st element of p into x
           # and copy 2nd element into y
```

- Here is a definition of a function whose parameters are tuples and a call to that function:

```
import math

def euclidean_distance( p0, p1 ):
    x0, y0 = p0
    x1, y1 = p1

    delta_x = x1 - x0
    delta_y = y1 - y0

    dx_sqr = delta_x * delta_x
    dy_sqr = delta_y * delta_y

    result = math.sqrt( dx_sqr + dy_sqr )
    return result

if __name__ == "__main__":
    p0 = (0, 0)
    p1 = (3, 4)

    distance = euclidean_distance( p0, p1 )

    print( f"distance = {distance:8.4f}" )
```

This function returns to its caller a tuple that contains the x and y coordinates of a new point.

For example, the weighted average of the points $(2, 2)$ and $(4, 4)$ with weight $t = 0.5$ is $(3, 3)$.

3. Define a function weighted_avg_of_avgs(p0, p1‘, p2, t) that computes the weighted average of two weighted averages of points.

   The first weighted average is the weighted average of $p0$ and $p1$ with weight t.

   The second weighted average is the weighted average of $p1$ and $p2$ with weight t.

   This function computes its result by calling weighted_avg_points() three times.

- the first call computes the weighted average of $p0$ and $p1$
- the second call computes the weighted average of $p1$ and $p2$
- the third call computes the weighted average of the points returned by the first two calls

This function returns to its caller a tuple that contains the x and y coordinates of a new point.

Hint:

Here is an example of a program that defines two functions. The definition of the second function makes use of the first function.

```
MAXIMUM_SEQUENCE_LENGTH = 1024

def hailstone_number( n ):
    if n % 2 == 0:
        return n // 2
    else:
        return 3 * n + 1

def sequence_length( n ):
    count = 0
    while n != 1 and count < MAXIMUM_SEQUENCE_LENGTH:
        n = hailstone_number( n )
        count += 1

    return count

if __name__ == "__main__":
    seed = int( input( "Enter a positive integer: " ) )
    length = sequence_length( seed )
    print( f"Sequence length = {length:4d}" )
```

4. Define a function weighted_avg_of_avgs_of_avgs(p0, p1, p2, p3, t that computes the weighted average of the weighted averages of weighted averages of points.

   This function will compute its result by calling weighted_avg_of_avgs() twice.

   - the first call will the function with the parameters $(p0, p1, p2, t)$
   - the second call will the function with the parameters $(p1, p2, p3, t)$

   Then this function will call weighted_avg_of_points() to compute the averages of the points returned by the two calls to weighted_avg_of_avgs().

   This function will return to its caller a tuple that contains the x and y coordinates of a new point.

5. Use Turtle graphics to create a window.

6. Create tuples that hold the coordinates of four points in the window and assigns these tuple values to variables $p0, p1, p2, p3$.

7. Call weighted_avg_of_avgs_of_avgs() repeatedly with $p0$, $p1$, $p2$, $p3$ and values of t that begin with $t = 0.0$ and increase in equal increments to $t = 1.0$.

   Hint:

   Here is a program that calls a function repeatedly.

```
import math

STEPS = 12

if __name__ == "__main__":
    for i in range(STEPS):
        t = i / (STEPS - 1)
        sqrt = math.sqrt( t )
        print( f"square root({t:6.4f}) = {sqrt:6.4f}" )
```

8. Plot the points $p0$, $p1$, $p2$, $p3$ in one color.

   Hint:

   Here is a program that plots four points.

```
import turtle

if __name__ == "__main__":
    window = turtle.Screen()
    window.colormode( 255 )
    window.screensize( 512, 512 )

    window.bgcolor( "cornflowerblue" )

    pen= turtle.Turtle()
    pen.hideturtle()

    pen.up()
    pen.goto( 128, 128 )
    pen.dot( 12, (128, 0, 112) )

    pen.goto( -128, 128 )
    pen.dot( 12, (128, 0, 112) )

    pen.goto( -128, -128 )
    pen.dot( 12, (128, 0, 112) )
```

```
pen.goto( 128, -128 )
pen.dot( 12, (128, 0, 112) )

window.exitonclick()
```

9. Plot the computed points in another color.

10. Can you connect the computed points with line segments?