1 Administrative Topics

• Time for Quiz 2

2 Lists

We have talked about very simple data types (strings, ints, floats, booleans) and very complex ones (modules and functions). Today, we introduce lists, which are somewhere in the middle. Lists in Python are lists of values. For example, here is a list:

\[0, 1, 2, 3\]

It contains the numbers 0, 1, 2, and 3.

In fact this is the return value from a call to the built-in function `range(4)`.

```python
my_list = range(4)
print(my_list)
```

outputs

\[0, 1, 2, 3\]

We can create our own lists:

```python
taylor_ages = [2, 4.5, 36, 36]
```
One of the major uses of lists is to tell for loops what values to give their loop control variables.

3 For Loops

The syntax of a for loop is:

for [item] in [list] :
    [block of code “in the loop”]

[item] is the loop control variable. It is just a variable. It can have any name. [list] is a list of values.

Python executes a for loop by setting the loop control variable to the first item in the list, then executing the block of code. Then, Python sets the loop control variable to the second item in the list, then executes the block of code. This process is repeated until there are no more elements in the list.

How does execution of the for loop affect the memory? First, Python determines what the list is and puts it into memory in a hidden place. We will leave this out of the symbol table. Then python adds the loop control variable to the symbol table (or just updates it if it is already there). It sets its value to the first item in [list]. Then, Python executes the block of code in the loop. This may affect the symbol table. The process is repeated for each item in the list. Once the loop has finished execution, the loop control variable remains in the table with the last value from the list.

Let’s return to our example of the Taylor family’s ages. We can loop through that list and print each age individually.

Notice that the loop control variable does not need to be named i. In fact, there are times when that isn’t an appropriate name. Here, for instance, it should be age because an item in a list of ages is an age. In general I use i for a counter loop control variable (e.g. one that is 0, 1, 2, 3, etc.). I use x and y for positions. I try to choose variable names that are appropriate to the context.

We can nest for loops. For example
for i in range(3):
    for j in range(2):
        print i, j

and the output is
0 0
0 1
1 0
1 1
2 0
2 1

The body of the outer loop executes 3 times because there are 3 items in the list returned by range(3). That means the inner loop is executed 3 times. The body of the inner loop (i.e. the print statement) executes twice each time the inner loop is executed. Therefore, the body of the inner loop (i.e. the print statement) is executed 6 times. Basically, the inner-most code executes once for each combination of inner and outer values. And its does so in the above order.

4 Advice for Project 3

- I made sure my scene-within-a-scene worked, and then worked on my larger scene without drawing the full smaller scene. I added a parameter to the smaller scene (drawn by colby1) called drawBackgroundOnly. If drawBackgroundOnly is True, then the function returns before drawing all the complicated foreground stuff.

- It is possible to give parameters default values. Below is a function addANum that adds two numbers. The second number is optional – if the called doesn’t supply a value for the second parameter, then the default value will be used:

  def addANum( base , num = 1 ):
      return base + num

  print addANum(4)
  print addANum(4,1)
  print addANum(4,2)
and the output is

5
5
6

• Here is the first example of code that sets a boolean value for controlling whether or not it is daytime. It assumes that “python task2.py 1” or ‘python task2.py” on the command line indicates daytime. Anything else indicates nighttime:

```python
day_part = 1
if len(sys.argv) > 1:
    day_part = int(sys.argv[1])
daytime = day_part == 1
```

• Here is the second example. It assumes that “python task2.py day” or ‘python task2.py” on the command line indicates daytime. Anything else indicates nighttime:

```python
daytime = True
if len(sys.argv) > 1:
    daytime = sys.argv[1] == 'day'
```

5 Draw a Night/Day Scene (More advice for the notes, but not given in class)

Let’s draw a scene with ground, sky, and a house. We want daytime and nighttime versions of the scene. But it will be the same scene. So let’s use the same code, but use if-statements to draw objects different colors based on the time of day.

My block shape function takes the color and whether or not it is filled as input.

```python
# If the turtle is oriented to the right, then x,y
# is the bottom left corner of the block.
def block(x, y, width, height, color, filled):
    goto(x, y)
    turtle.color(color)
    turtle.fill(filled)
```
for i in range(2):
    turtle.forward(width)
    turtle.left(90)
    turtle.forward(height)
    turtle.left(90)

turtle.fill(False)
turtle.update()

So I can draw the sky and ground as filled, colored blocks. Let us use variable
daytime that will be True for a daytime scene and False for a nighttime scene.
We will use different colors, depending on the time of day:

import turtle
import shapes

# main code
turtle.tracer(False)
daytime = True

# sky
if daytime:
    shapes.block(-300, -150, 600, 450, 'DeepSkyBlue1', True)
else:
    shapes.block(-300, -150, 600, 450, 'blue4', True)

# ground
if daytime:
    shapes.block(-300, -300, 600, 150, 'LawnGreen', True)
else:
    shapes.block(-300, -300, 600, 150, 'DarkGreen', True)

turtle.update() # Force turtle to draw everything you told it to draw
raw_input("Press Enter")

We can run this code in its current form to see a day scene (see Figure 1) or
change one line:

daytime = False

and see a night scene (see Figure 2).
Figure 1: Day Background

Figure 2: Night Background
5.1 Redesigning the Code

Since the only difference between night and day is color, then let’s take advantage of that. Instead of having multiple if statements and instead of using nearly identical lines of code for the daytime shapes and the nighttime shapes, let’s introduce variables for the colors and use those. Let’s have one if-statement at the beginning of the code that stores colors for each shape, based on whether it is nighttime or daytime. Then, we use those variables when we actually draw the shapes.

This means we introduce sky_color, ground_color, and window_color in the main section of the code.

```python
# main code
turtle.tracer(False)
daytime = True

if daytime:
    sky_color = 'DeepSkyBlue1'
    ground_color = 'LawnGreen'
else:
    sky_color = 'blue4'
    ground_color = 'DarkGreen'

# sky and ground
shapes.block(-300, -150, 600, 450, sky_color, True)
shapes.block(-300, -300, 600, 150, ground_color, True)

turtle.update() # Force turtle to draw everything you told it to draw
raw_input("Press Enter")
```

The results from calling this code are identical to those in Figures 1 and 2. One advantage of this new organization is that the positions and sizes of the blocks are used only once. So if we decide to change the location of the horizon later, we have less code to update.
5.2 Adding the House

We want to draw a house. If it includes lots of shapes, and we think we might want to draw more than one house, we should place it in a function. And since it is itself a shape, it belongs in shapes.py.

We will allow the user to specify a scale and base color for the house. The code for the house is

```python
# x, y is bottom left hand corner of house
def house(x, y, scale, base_color):
    swidth = 300*scale
    sheight = 350*scale
    block(x, y, swidth, sheight, base_color)
    triangle(x, y+sheight, swidth, 'black')
```

Once we add the line

```python
shapes.house(-100, -200, 0.5, "pink")
```

to the main code, we see the picture in Figure 3 when daytime is True and the picture in Figure 4 when daytime is False.

5.3 Making the scene movable and scalable

What if we want to draw this scene more than once? Say, a day version next to a night version? We should put its code into a function. We will call it scene1. The input to that function should include the position and a scale factor. In our case, we also want to include a parameter indicating whether it is day or night time.

The current scene is written with the “absolute” positions, i.e. we place the ground at (-300,-300), which means its bottom left corner is at the bottom left corner of the window. We need to make these positions relative to a given location. In other words, all of our positions become offsets from the given scene position. We say that the (x,y) position of a scene is the location of its bottom left-hand corner, so we compute our offsets from the bottom left-hand corner (which was (-300,-300))
Figure 3: Day Scene
Figure 4: Night Scene
We must also scale every shape. That means we must scale both the sizes (e.g. edge lengths) and any offsets.

For example

```
shapes.block(-300,-150,600,300,sky_color,True)
```

becomes

```
shapes.block(x, y+150*scale, 600*scale, 450*scale, sky_color, True )
```

In a nutshell the steps are:

1. Create a function definition that includes parameters for the position and scaling.
2. Cut the code from the main program and paste it within the function (don’t forget to indent)
3. Go through the code, adding x and y to the positions and multiplying any offsets or lengths by the scale factor.

When I choose which code to copy from the main program, I choose all code that draws and none of the set-up or final code.

For this example, the result is:

```
# Stephanie Taylor
# lec7.py
import turtle
import shapes

# draw a day or night scene
# x, y is lower lefthand corner of scene
def scene1( x, y, scale, daytime ) :
    # colors depend on the time of day
    if daytime :
        sky_color = 'DeepSkyBlue1'
        ground_color = 'LawnGreen'
    else :
        sky_color = 'blue4'
        ground_color = 'DarkGreen'

    # sky
    shapes.block( x, y+150*scale, 600*scale, 450*scale, sky_color, True )
```
# ground
shapes.block( x, y, 600*scale, 150*scale, ground_color, True )

shapes.house( x+200*scale, y+100*scale, scale*0.5, "pink" )

# main code
turtle.tracer(False)

scene1(-300,-300,1.0, True)  # Draw a daytime scene

turtle.update()  # Force turtle to draw everything you told it to draw

raw_input("Press Enter")

We can now update the main code to draw night and day scenes next to each other:

scene1(-300,-300,0.5, False)
scene1(-150,-300,0.5, True)

and the output is in Fig. 5.

Figure 5: Two scenes at once!