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).

my_list = range(4)
print my_list

outputs

[0, 1, 2, 3]

We can create our own lists:

taylor_ages = [3, 7.0/12, 34, 35]
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

```python
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 loop. 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.

```python
for age in taylor_ages:
    print age
```

And the output is

3
0.583333333333
34
35
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

```python
outer_vals = [4, 5, 9]
inner_vals = [3, 2]

for oval in outer_vals:
    for ival in inner_vals:
        print oval, ival
```

and the output is

4 3
4 2
5 3
5 2
9 3
9 2

The body of the outer loop executes 3 times because there are 3 items in outer_vals. 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 A Fun Example

Here is concise code to draw a black and white checker board:

```python
import turtle

# black block with lower left at (x,y)
def block(x,y):
    turtle.up()
    turtle.goto(x,y)
    turtle.down()
    turtle.fill(True)
    for i in range(4):
        turtle.forward(30)
        turtle.left(90)
    turtle.fill(False)

for i in range(8):
    for j in range(8):
        if (i+j) % 2 == 0:
            block(-300+i*30,-300+j*30)
```

i is used to index columns and j is used to index rows. The remainder/mod operator (%) is used to determine whether or not the column/row sum is even. If it is, then a black block is drawn.