CS151 Fall 2012 Lecture 4

Stephanie R Taylor

September 12, 2012

1 Administrative Topics

- Look at example write-ups from Project 1.
- I will assign homework tonight. If you email me your answers by Thursday night at 10:30, I will respond with comments. The homework is designed to help you study for the quiz.

2 Organizing Code

A Python program written for this class should be organized as follows:

1. Header comments (name of file, name of author)
2. Any import statements
3. Any function definitions (each of which should have a comment indicating what it does)
4. A comment indicating the "main" section of code is about to appear
5. The main code that you want this program to run

For example, a simple Python program that uses the turtle to draw a few squares is this:

```
# squares.py
# Stephanie Taylor
import turtle

# Draw a square with side of the given length
def drawSquare(length):
  turtle.color((0,0.5,0))
  turtle.forward(length)
  turtle.left(90)
  turtle.forward(length)
  turtle.left(90)
  turtle.forward(length)
  turtle.left(90)
  turtle.forward(length)
  turtle.left(90)

# main
```

```
turtle.reset()
turtle.tracer(False)
len = input('what length? ')
drawSquare(len)
turtle.right(30)
drawSquare(100)
turtle.right(30)
drawSquare(100)
turtle.update()
raw_input("press enter when ready ")
```

3 Understanding what happens when Python executes code

The next example is meant to deepen our understanding of symbol tables and the process of execution. The code calculates and prints the amount you should tip on two different restaurant bills:

```
1  # Stephanie Taylor
2  # September 12, 2012
3  # Lecture 4 example to illustrate symbol tables
4  # and functions.
5
6  # Print a message telling us the amount to tip for the given bill
7  def printTip( bill ):
8      rate = 0.18
9      print 'You should tip $' + str(bill*rate) + ' when the bill is $' + str(bill)
10
11 # main code
12 bill1 = 20;
13 printTip( bill1 )
14 bill2 = 120.50
15 printTip( bill2 )
```

3.1 Stepping through code

Let's step through the code first, then think about memory. The basic approach is this: Python reads the file from the beginning to the end, executing the code as it goes along.

- Python reads the definition of printTip and stores it for use later. Now Python "knows" what the function printTip is.
- Python executes line 12, which is an assignment statement. Now Python "knows" what the amount of the first bill is.
- Python executes line 13, which means it needs to execute lines 8 and 9. A message is printed to the screen as a result of the code in line 9. It is

  You should tip $3.6 when the bill **is** $20

- Python executes line 14, which is an assignment statement. Now Python "knows" what the amount of the second bill is.
- Python executes line 15, which means it needs to execute lines 8 and 9. A message is printed to the screen as a result of the code in line 9. It is

  You should tip $21.69 when the bill **is** $120.5

- There is no more code. So Python stops executing.

3.2 How to view memory when functions are being executed

Now, let's look at what happens to memory as this code is executed. To do that, I am going to draw symbol tables. We talked on Monday about symbol tables. We learned that there is a main symbol table when you are running the interpreter. We also learned that each module has a symbol table. Today, we are going to add to that: Each function has a symbol table while it is executing.

Here is how I will draw this. We will be executing "main" code and code within a function. I am going to write the line of code that is currently being executed underneath the appropriate symbol table.

Let's step through the code again.

- Python reads the definition of printTip and stores it for use later. Now Python "knows" what the function printTip is. So let's assume we are at line 11, which has no code, so I won't write anything underneath the table.

main

Name	Value
printTip	→ <fcn>

- Python moves to line 12, which I will write underneath the table (think of this as a "before" picture)

main

Name	Value
printTip	→ <fcn>

bill1 = 20

Python executes line 12, which is an assignment statement. Now Python "knows" what the amount of the first bill is. (think of this as an "after" picture – I am striking out the line of code to show that it has completed executing)

main

Name	Value
printTip	→ <fcn>
bill1	20 <int>

~~bill1 = 20~~

- Python executes line 13. This will be a multi-step process:
  - Python is about to call the printTip function. The first thing it does is look evaluate the code between the parentheses. It is going to pass (or send, or input) a VALUE to printTip. (think of this as a "during" picture - I am crossing out the variable name and replacing it with its value to indicate that Python looked up the variable's value in the table).

main

Name	Value
printTip	→ <fcn>
bill1	20 <int>

printTip(~~bill1~~ 20)

– Now Python begins executing printTip. To do that, it creates a symbol table for it, and makes entries in the symbol table for each of the parameters.

main

Name	Value
printTip	→ <fcn>
bill1	20 <int>

printTip(~~bill1~~ 20)

printTip

Name	Value
bill	20<int>

– We move to line 8 and execute it, which results in a new row in the printTip symbol table.

main

Name	Value
printTip	→ <fcn>
bill1	20 <int>

printTip(~~bill1~~ 20)

printTip

Name	Value
bill	20<int>
rate	0.18<flt>

~~rate = 0.18~~

– We move to line 9 and execute it. This line requires Python to look up value in the table, but does not make any changes to the tables, so I am not doing to redraw the tables. The output is

You should tip $3.6 when the bill **is** $20

– Python is now done executing the function so its table goes away. And I will strike-out the code below the main function to show that the line has completed executing.

main

Name	Value
printTip	→ <fcn>
bill1	20 <int>

~~printTip(20)~~

- Python executes line 14, which is an assignment statement. Now Python "knows" what the amount of the second bill is.

main

Name	Value
printTip	→ <fcn>
bill1	20 <int>
bill2	120.5 <flt>

~~bill2 = 120.50~~

- Python executes line 15. This will be a multi-step process:

– Now Python begins executing printTip. To do that, it creates a symbol table for it, and makes entries in the symbol table for each of the parameters.

main

Name	Value
printTip	→ <fcn>
bill1	20 <int>
bill2	120.5 <flt>

printTip(~~bill2~~ 120.5)

printTip

Name	Value
bill	120.5<flt>

– We move to line 8 and execute it, which results in a new row in the printTip symbol table.

main

Name	Value
printTip	→ <fcn>
bill1	20 <int>
bill2	120.5 <flt>

printTip(~~bill1~~ 20)

printTip

Name	Value
bill	120.5<flt>
rate	0.18<flt>

~~rate = 0.18~~

- We move to line 9 and execute it. This line requires Python to look up value in the table, but does not make any changes to the tables, so I am not doing to redraw the tables. The output is

  You should tip $21.69 when the bill **is** $120.5

- Python is now done executing the function so its table goes away. And I will strike-out the code below the main function to show that the line has completed executing.

main

Name	Value
printTip	→ <fcn>
bill1	20 <int>
bill2	120.5 <flt>

~~printTip(120.5)~~

- There is no more code. So Python stops executing.

4 Math Operators

There are quite a few mathematical operators built in to Python. Table 1 contains a list of binary and unary operators

Table 1: Unary and binary numeric operators in Python

Operation	Result
`x + y`	sum of x and y
`x - y`	difference of x and y
`x * y`	product of x and y
`x / y`	quotient of x and y
`x % y`	remainder of x/y
`-x`	x negated
`+x`	x unchanged
`x ** y`	x to the power y

Here are some interesting points regarding math in Python:

- The same precedence rules apply to the operations as in math. For operators with the same precedence, Python executes them from left to right.
- For floats, +,-,*,/ do what you expect.
- For integers +.-.* are also straightforward

- What about / and % for integers? / results in the quotient and % results in the remainder, e.g. 5/3 is 1 and 5%3/ is 2.
- We can group together operations using parentheses the same way we can with mathematical expressions.

The rules of precedence for the binary operators are:

- ** (highest)
- *, /,
- +, -

A word of advice, when making scale factor for your turtle drawings, never use 1/5. Because it is zero! There are several ways to make it become 0.2 (or, rather 0.200000000001 because the computer can't represent 0.2 precisely). You can multiply by 1.0, or you can use the float function. Suppose you have two variables, A and B, and you want to scale your distances by A/B using float division. If A and B are floats, then you are all set. But there are no guarantees that A and B are floats. However, you can force them to perform float operations multiple ways, e.g.

```
scale = 1.0 * A / B
scale = float(A) / B
scale = A / float(B)
```