CS151 Fall 2012 Lecture 6

Stephanie R Taylor

September 17, 2012

1 Administrative Topics

- Return quiz 1
- I emailed your project grades to you today. Project grades will always be sent on the Mondays after they are due.
- Sign up for the csstu email list.

2 Debugging

Here are three bits of advice I can share with you about debugging your code:

1. Read the Python error message carefully. In particular, that message will give you the line on which the error was encountered. Set up your text editor to display line numbers to make it trivial to find the line with the problem.
2. When there is a syntax error reported on line X, you might want to look at line X-1 to see if there is a problem like a missing colon or a missing close parenthesis.
3. Use **print** statements like crazy. If you are unsure about a calculation you are doing, then print it out before you use it.

3 Booleans

A Boolean value is either True or False. The name Boolean comes from George Boole, who invented Boolean Algebra and Boolean Logic.

they are `True`

and `False`

And Python calls this type "<bool>".

There are many built-in operators that return Boolean values.

4 Comparison (Relational) Operators

The comparison operators are listed below. They work for all types in Python, though we are going to focus on them for numerics.

Relational Operators in Python

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal

The precedence of these operators is lower than that of the mathematical operators. All of these operators have equal precedence to each other and are evaluated from left to right.

5 Boolean Operations

Boolean operations take booleans as operands and result in a boolean value:

Boolean Operators in Python

Operation	Result
x or y	True if either x or y is True, False if they are both False
x and y	True if both x and y are True, False if either is False
not x	True if x is False, False if x is True

"Not" has the highest precedence of the boolean operators, but it still has lower priority than the mathematical operators. "And" has higher priority than "or".

We define the results of AND and OR by enumerating the possible operand values. This is called a truth table.

Truth table for AND

True and True	True
True and False	False
False and True	False
False and False	False

Truth table for OR

True or True	True
True or False	True
False or True	True
False or False	False

6 Writing a Function with a Conditional

Let's put it altogether in a function. Here it is:

```
# weatherF : prints a string describing the weather
#     based on the temperature given as input
# input
#     f : the temperature in degrees fahrenheit
#             (<int> or <float>)
# output:
#     a description of the weather (<str>)
def weatherF(temp):
  if temp > 70:
    print 'hot'
  else:
    print 'not hot'
```

```
#main
weatherF(81)
```

This introduces a new type of statement – a conditional, or "if" statement. We use the keyword if followed by a Boolean or an expression that evaluates to a Boolean. If that expression is true, we evaluated the statements "in" the first nested block (e.g. `weather = 'hot'`). In other words, we take the first "branch". If that expression is not true, we evaluate the code in the "else" block (taking the second "branch").

The if statement can have additional conditions and blocks. We can use the "if ... else if ... else if ... else" construction. We do this by using the "elif" keyword (a contraction for "else if"). For example, we can add additional conditions to test more values of the temperature (see code example below). Only one block is executed and that is the first block associated with a true condition. For example, if the temperature is 65 degrees F, then the weather will be warm.

```
# weatherF : prints a string describing the weather
#    based on the temperature given as input
# input
#    ftemp : the temperature in degrees fahrenheit
#            (<int> or <float>)
# output:
#    a description of the weather (<str>)
def weatherF( ftemp ):
    if ftemp >= 70:
        print 'hot'
    elif ftemp >= 55:
        print 'warm'
    else:
        print 'cold'

#main
weatherF(81)
```

Note that the elif expression is not **elif** temp > 55 **and** temp <= 70. It would be entirely correct if it were. But it isn't necessary to do the second comparison. Why not? Because if-statement expression statements are evaluated in order. If the temp were greater than 70, then the first branch would have been taken. And the elif expression wouldn't be evaluated. So, if we get to the elif then

we already know that the temperature is less than or equal to 70.

6.1 Stepping through the code

We didn't step through the symbol tables in class, but here are notes about it if you are interested.

Let's draw the symbol table for weatherF when it is called with a temperature of 81 degrees F.

I run it, and at the first line of top-level (main) code, the symbol table looks like this:

main

Name	Value
weatherF	→ <function>

When the line weatherF(81) is executed, a new symbol table appears:

main

Name	Value
weatherF	→ <function>

weatherF(81)

weatherF

Name	Value
temp	81<int>

The first line executed in weatherF is **if** temp > 75:. The condition evaluates to True, so the first branch is taken. This means the next line executed is **print** 'hot'. The string 'hot' is printed to the Terminal. Then the function is finished executing and the table goes away.

Finally, the weatherF symbol table is erased, and the assignment statement (in main) is completed:

main

Name	Value
weatherF	→ <function>

~~weatherF(81)~~

Then the main code is done and the table goes away.

7 Writing Our Own Functions that return values

We have written functions to draw scenes with the turtle or that print information, but we haven't needed to print or store any results from those functions. When we call the mathematical functions, there is always a value returned to us. So, how do we write functions (like math functions) that return results? The answer is that we use a "return" statement.

As a review, here is what we know about writing functions: Functions have input, instructions, and output

- We begin by writing the "header" comments. Name the function, briefly describe what it does, then give detailed information about what it expects as input (including the types), then list what it returns.
- Define the function. using the keyword "def", the name of the function, and the parameters it takes as input
- Perform the task of the function (e.g. the "do the math")
- Insert a return statement to return the value to the caller.

Below is the code for two functions that convert temperature values – one from Celsius to Fahrenheit, the other that does the reverse. I have also uploaded the code file to the web page.

```
1  # Stephanie Taylor
2  # Lecture 6 code
3  # weather-related functions
4
5  # Print out a subjective description of how
6  # hot it is, given the temperature (in F)
7  def weatherF( ftemp ):
8      if ftemp >= 70:
9          print 'hot'
10     elif ftemp >= 55:
```

```
11          print 'warm'
12      else:
13          print 'cold'
14
15  # Convert the temperature from Fahrenheit to Celsius
16  def f2c( ftemp ):
17      ctemp = (ftemp-32)*5.0/9
18      return ctemp
19
20  # Convert the temperature from Celsius to Fahrenheit
21  def c2f( ctemp ):
22      ftemp = ctemp * 9.0/5 + 32
23      return ftemp
24
25  # main code
26  # Does a temp of 30 degrees Celsius mean it is hot out?
27  ctemp = 30
28  ftemp = c2f( ctemp )
29  weatherF( ftemp )
```

7.1 Stepping through the code

Python reads in the function definitions, so at line 25, the main table is

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>

Then line 27 is executed and ctemp is added to the main symbol table.

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>

As Python begins to execute line 28, it first evaluates the code for the input to c2f

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>

ftemp = c2f(~~ctemp~~ 30)

Then c2f begins to execute. It has one parameter ctemp, which is placed in the table with the value that was given as input.

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>

ftemp = c2f(~~ctemp~~ 30)

c2f

Name	Value
ctemp	30 <int>

Then Python evaluates the assignment statement in the function. It begins by evaluating the mathematical expression on the left-hand side.

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>

ftemp = c2f(~~ctemp~~ 30)

c2f

Name	Value
ctemp	30 <int>

ftemp = ~~ctemp * 9.0/5 + 32~~ 86.0

Then the table is updated and Python moves to the line 23 to execute the return statement. It begins by evaluating the code after the command **return**:

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>

ftemp = c2f(~~ctemp~~ 30)

c2f

Name	Value
ctemp	30 <int>
ftemp	86.0 <flt>

return ~~ftemp~~86.0

To indicate that the value is sent back to the caller, I cross out the function call in the main code and replace it with the value that is being returned:

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>

ftemp = ~~c2f(30)~~86.0

c2f

Name	Value
ctemp	30 <int>
ftemp	86.0 <flt>

return ~~ftemp~~86.0

Then the function's symbol table disappears

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>

ftemp = ~~c2f(30)~~86.0

And Python can complete the assignment statement from line 28.

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>
ftemp	86.0 <flt>

And Python moves to line 29, and evaluates the input to the weatherF function:

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>
ftemp	86.0 <flt>

weatherF(~~ftemp~~86.0)

And weatherF begins execution with its parameter's value in its symbol table.

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>
ftemp	86.0 <flt>

weatherF(~~ftemp~~86.0)

weatherF

Name	Value
ftemp	86.0 <flt>

Python moves to the code in line 8 and evaluates the conditional expression

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>
ftemp	86.0 <flt>

weatherF(~~ftemp~~86.0)

weatherF

Name	Value
ftemp	86.0 <flt>

if ~~ftemp ¿= 70~~True:

Because the condition is True, Python takes the "if-branch" and next executes the code in line 9. This code prints the word 'hot' to the Terminal. There is no code after the if-elif-else statement. So Python is now done executing the function. There is no return statement. In the past, we have just crossed out the call to a function without a return statement. But today, I am going to replace it with the value that Python automatically returns. That value is None.

main

Name	Value
weatherF	→ <function>
f2c	→ <function>
c2f	→ <function>
ctemp	30 <int>
ftemp	86.0 <flt>

~~weatherF(86.0)~~ None

And then Python is done. The main symbol table disappears. And there is nonethingness.