1 Administrative Topics

- Showcase some Proj 2 wiki pages.

2 Boolean Operations

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

<table>
<thead>
<tr>
<th>Boolean Operators in Python</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \text{ or } y$</td>
<td>True if either $x$ or $y$ is True, False if they are both False</td>
</tr>
<tr>
<td>$x \text{ and } y$</td>
<td>True if both $x$ and $y$ are True, False if either is False</td>
</tr>
<tr>
<td>$\text{not } x$</td>
<td>True if $x$ is False, False if $x$ is True</td>
</tr>
</tbody>
</table>

“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

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>True and True</td>
<td>True</td>
</tr>
<tr>
<td>True and False</td>
<td>False</td>
</tr>
<tr>
<td>False and True</td>
<td>False</td>
</tr>
<tr>
<td>False and False</td>
<td>False</td>
</tr>
</tbody>
</table>

## Truth table for OR

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>True or True</td>
<td>True</td>
</tr>
<tr>
<td>True or False</td>
<td>True</td>
</tr>
<tr>
<td>False or True</td>
<td>True</td>
</tr>
<tr>
<td>False or False</td>
<td>False</td>
</tr>
</tbody>
</table>

### 3 Writing a Function with a Conditional

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

```python
# 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”
The key word (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'

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.

3.1 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:
When the line `weatherF(81)` is executed, a new symbol table appears:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
</tbody>
</table>

Weather table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>temp</td>
<td>81&lt;int&gt;</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
</tbody>
</table>

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

## 4 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.

```python
# Stephanie Taylor
# Lecture 6 code
# weather-related functions

# Print out a subjective description of how hot it is, given the temperature (in F)
def weatherF(ftemp):
    if ftemp >= 70:
        print 'hot'
    elif ftemp >= 55:
        print 'warm'
    else:
        print 'cold'

# Convert the temperature from Fahrenheit to Celsius
def f2c(ftemp):
    ctemp = (ftemp - 32) * 5.0/9
    return ctemp

# Convert the temperature from Celsius to Fahrenheit
def c2f(ctemp):
    ftemp = ctemp * 9.0/5 + 32
```
```python
return ftemp

# main code
# Does a temp of 30 degrees Celsius mean it is hot out?
ctemp = 30
ftemp = c2f( ctemp )
weatherF( ftemp )
```

### 4.1 Stepping through the code

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>f2c</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>c2f</td>
<td>→ &lt;function&gt;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>weatherF</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>f2c</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>c2f</td>
<td>→ &lt;function&gt;</td>
</tr>
<tr>
<td>ctemp</td>
<td>30 &lt;int&gt;</td>
</tr>
</tbody>
</table>

As Python begins to execute line 28, it first evaluates the code for the input to c2f
Then \( c2f \) begins to execute. It has one parameter \( ctemp \), which is placed in the table with the value that was given as input.

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

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 \textbf{return}:
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:

Then the function’s symbol table disappears

And Python can complete the assignment statement from line 28.
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
  weatherF
    Name           Value
    ftemp   86.0 <flt>
  weatherF(ftemp 86.0)
```

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

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