1 Administrative Topics

- Go over the homework.
- We take the quiz.

2 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.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>strictly less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>strictly greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
</tr>
<tr>
<td>==</td>
<td>equal</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
</tr>
</tbody>
</table>

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.
3 Conditional (If) Statements

We can use the comparison operator to execute difference code, depending on the truthiness of the condition. Here is an example in a function that prints how the weather “feels”, depending on the temperature.

```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” 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.

```python
# 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.

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:

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

When the line `weatherF(81)` is executed, a new symbol table appears:
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>
<tr>
<td>weatherF(81)</td>
<td></td>
</tr>
</tbody>
</table>

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

### 4 Advice for Project 2

- **grep** - Bruce has updated the lab instructions regarding grep (he did so Wed night).

- **which buoy** - There had been some confusion earlier about which buoy to grab data from. You should be using the LEA buoy.

- **import** - Import allows you to access functions and variables in another file. You need to use the “dot” to access the functions.

```python
import sys
import math
print sys.argv
print math.sin(0)
```

- **in** - You can test to see if one string is a substring of another using in.
s1 = 'catsup'
print 'cat' in s1
print 'dog' in s1
print 'up' in s1
print s1 in "Scott really doesn't like catsup"