1 Administrative Topics

2 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.
3 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.

4 Boolean Operations

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x or y</td>
<td>True if either x or y is True, False if they are both False</td>
</tr>
<tr>
<td>x and y</td>
<td>True if both x and y are True, False if either is False</td>
</tr>
<tr>
<td>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.
True table for AND

| True and True | True |
| True and False | False |
| False and True | False |
| False and False | False |

True table for OR

| True or True | True |
| True or False | True |
| False or True | True |
| False or False | False |

5 Writing a Function with a Conditional

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

```python
# weatherF : returns 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 > 80:
        ret = 'hot'
    else:
        ret = 'not hot'
    return ret
```

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”).

Notice also that we can use + to concatenate strings.
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 if 72 degrees F, then the weather 
will be warm.

```python
# weatherF : returns 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 > 80:
        ret = 'hot'
    elif temp > 70:
        ret = 'warm'
    else:
        ret = 'not hot'
    return ret
```

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

I write the code for weatherF in a module named lec7 and then import it 
into Python. The main table, then looks like this:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>lec7</td>
<td>&lt;module&gt;</td>
</tr>
</tbody>
</table>
```

Then, I type `w = lec7.weatherF(81)` at the command line. The new symbol 
table for weatherF is created, and the parameter temp is given the value 81:
The first line executed in weatherF is `if temp > 80:`. The condition evaluates to `True`, so the first branch is taken. This means the next line executed is `ret = 'hot'`. This adds a row to the weatherF symbol table:

```
weatherF
Name   Value
temp   81<int>
ret    'hot'<str>
```

The next line to be executed is `return ret`, which returns `'hot'` to the caller:

```
weatherF
Name   Value
temp   81<int>
ret    'hot'<str>
```

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

```
main
Name   Value
lec7   <module>
w = lec7.weatherF(81)
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```

```
main
Name   Value
lec7   <module>
w    'hot' <str>
```