1 If Statements And The Boolean Type

1.1 if statements

In everyday life, we often make decisions based on the current conditions. For example,

If the weather is warm,
I wear shorts.
Otherwise,
I wear pants.

The same conditional process appears when writing code. To handle these situations we use an if statement. For example,

<table>
<thead>
<tr>
<th>English</th>
<th>Python</th>
</tr>
</thead>
</table>
| Check The Temperature: | def check_temp(temperature):
| If the temperature > 0 then it is above the freezing point | if temperature > 0:
| return "above the freezing point." |
| Otherwise, if the temperature = 0 then it is at the freezing point | elif temperature == 0:
| return "at the freezing point." |
| Otherwise it is below the freezing point | else:
| return "below the freezing point." |

Exercise: Write a docstring for check_temp.

Form for writing an if statement:

```
if condition_1:
  statements_1
elif condition_2:
  statements_2
elif ...
  ...
else:
  statements_N
```

The execution:

- evaluate condition_1
- if the result is True, execute statements_1 and skip the rest of the if statement
- otherwise, evaluate condition_2
- if the result is True, execute statements_2 and skip the rest of the if statement
- ...
- otherwise, execute statements_N

We can also nest if statements. Fill in the Python lines below that are missing:
Check The Temperature:  
If the temperature > 0 then  
  If the temperature > 100 then  
    it is above the boiling point  
  Otherwise, if the temperature > 37 then  
    it is above body temperature  
  Otherwise,  
    it is above the freezing point  
Otherwise, if the temperature = 0 then  
  it is at the freezing point  
Otherwise  
  it is below the freezing point

What is the type of the conditions in the if statements above?

1.2 The Boolean Type

A True/False value belongs to a type called a boolean type, named after George Boole, a philosopher and logician of the early 19th century. In Python we denote the boolean type by bool. We have already seen boolean expressions such as:

```python
temperature > 0 and temperature == 0
```

There are many different operators in Python that evaluate to True or False (note the capital letters).

Suppose we begin by assigning 7 to variables x and y. For each of the following conditions that use comparison operators write what they evaluate to:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 &lt; 4</td>
<td>True</td>
</tr>
<tr>
<td>3 &gt; 8</td>
<td>False</td>
</tr>
<tr>
<td>8 &gt; 3</td>
<td>True</td>
</tr>
<tr>
<td>3.5 &gt;= 3.5</td>
<td>True</td>
</tr>
<tr>
<td>7 == 7</td>
<td>True (Why not just one equals sign?)</td>
</tr>
<tr>
<td>x == 7</td>
<td>True</td>
</tr>
<tr>
<td>y == 7.0</td>
<td>True (Why not just one equals sign?)</td>
</tr>
<tr>
<td>x == y</td>
<td>True</td>
</tr>
<tr>
<td>3 != 4</td>
<td>True</td>
</tr>
</tbody>
</table>

Suppose we assign to variables snowing and sunny as follows:

snowing = False
sunny = True

For each of the following conditions that use logical operators, write down what they evaluate to. Most common logical operators are not, or and and.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>not snowing</td>
<td>False (&quot;not&quot; is a unary operator: 1 operand)</td>
</tr>
<tr>
<td>not sunny</td>
<td>False (&quot;not&quot; is a unary operator: 1 operand)</td>
</tr>
<tr>
<td>sunny and snowing</td>
<td>True (&quot;and&quot; and &quot;or&quot; are binary operators: 2 operands)</td>
</tr>
<tr>
<td>sunny or snowing</td>
<td>True (&quot;and&quot; and &quot;or&quot; are binary operators: 2 operands)</td>
</tr>
</tbody>
</table>
Let \( a, b, c, d \) be variables assigned the following values:

\[
\begin{align*}
  a &= \text{True} \\
  b &= \text{True} \\
  c &= \text{False} \\
  d &= \text{False}
\end{align*}
\]

Determine the value of each of the following:

\[
\begin{align*}
  \neg (a \land c) \\
  \neg a \lor \neg c \\
  (a \lor d) \land \neg (c \lor b) \\
  \neg (\neg a \lor \neg c)
\end{align*}
\]

### 1.3 Short circuiting and and or

Python is “lazy” when it evaluates boolean expressions. If both \( x \) and \( y \) are boolean expressions, then the expression

\[
x \land y
\]

is evaluated as follows:

- evaluate expression \( x \)
- if it evaluates to \text{False}, the resulting value of the \( \land \) expression is \text{False}
- otherwise, evaluate the expression \( y \); the resulting value of the \( \land \) expression is the value of the expression \( y \)

If the value of \( x \) is \text{False}, Python doesn’t bother to evaluate \( y \).

**Q.** What value of \( x \) would force Python to evaluate \( y \) in the expression \( x \lor y \)?

**A.**

Short circuiting allows us to simplify our code. For example,

Suppose you write a program that reads in some temperature data. It has just read all the data recorded for January 13, 2010. Variable \texttt{total} has the total of those values, and \texttt{n} has the number of values that were read. If the average temperature reading that day was above freezing, you want to print a message saying so.

\[
\text{if total} / \text{n} > 0:
  \text{print("Average was above freezing.")}
\]

**Q.** What problem may occur?

**A.**
Q. How can we fix this?

A.

1.4 Boolean Good Practice

- Use parentheses to indicate precedence – not only is this easier for someone else to read, it also ensures that errors are not made!

- Use the simplest expression possible. Some examples:

  Avoid double negatives like not not a; a is much clearer.
  Avoid stylistic errors:

  ```python
  if (temperature < 0) == True:
      print "Brrr!"
  ```

  Instead write:

  ```python
  if temperature < 0:
      print "Brrr!"
  ```

Q. Can you simplify this function?

```python
def is_odd(x):
    if x % 2 == 1:
        return True
    else:
        return False
```