Announcements

- Try to get help from me and tutors

- Reading assignment for this week:
  - Chapters 5 and 6 of Downey

- When you see a page on the web, be sure to reload it to see the latest version of the page!

- Break around 10:15am
Conditionals: Making Choices
Boolean expressions

- **Boolean expression**: evaluates to **True** or **False**
  
  $$5 == 5 \quad \rightarrow \quad \text{True}$$
  
  $$5 > 6 \quad \rightarrow \quad \text{False}$$
  
  \[\text{type}(True) \rightarrow \text{<type `bool'>}\]
  
  \[\text{type}(5 \leq 5) \rightarrow \text{<type `bool'>}\]

- **Relational operators**:
  
  \[x \quad == \quad y\]
  
  \[x \quad != \quad y\]
  
  \[x \quad > \quad y\]
  
  \[x \quad < \quad y\]
  
  \[x \quad >= \quad y\]
  
  \[x \quad <= \quad y\]

- **Boolean operators**:
  
  \[x \quad \text{and} \quad y\]
  
  \[x \quad \text{or} \quad y\]
  
  \[\text{not} \quad x\]
Logical/Boolean operators

- Truth tables for **and**, **or**, **not**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>or</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>not</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

- Examples

  \[ x > y \]
  \[(x < y) \text{ and } (x \% 2 == 0) \]
  \[(x \neq y) \text{ or } (x \geq 3) \text{ and } (x == y) \]

  Try evaluating these with \( x = 3 \) and \( y = 7 \)

Note: Any nonzero number or nonempty string is interpreted as **True**! None is treated as **False** though.
Conditional execution

- **Conditional statement** lets you do one thing or another depending on a condition.

- **Syntax:**
  ```python
  if <condition>:
      <block>
  else:
      <block>
  # else part is optional.
  ```

- **Examples**

  ```python
  x = 3
  if x < 0:
      print 'x is negative'
  if x < 0:
      pass
  ```

  ```python
  x = 14
  if x%2 == 0:
      print str(x) + ' is even'
  else:
      print 'x is odd'
  ```
Chained conditionals

- Multiple branches are useful in some situations

```python
def day(d):
    if d == 0:
        return 'Sunday'
    elif d == 1:
        return 'Monday'
    elif d == 2:
        return 'Tuesday'
    elif d == 3:
        return 'Wednesday'
    elif d == 4:
        return 'Thursday'
    elif d == 5:
        return 'Friday'
    elif d == 6:
        return 'Saturday'
    else:
        return 'Illegal'
```

```python
ph = 4
if ph > 7:
    print 'base'
elif ph == 7:
    print 'neutral'
else:
    print 'acid'
```
Nested conditionals

- Conditionals may be *nested* in any depth

```python
def foo(x, y):
    if x % 2 == 0:
        if y % 2 == 0:
            print 'both even'
        else:
            print 'x even, y odd'
    else:
        if y % 2 == 0:
            print 'x odd, y even'
        else:
            print 'both odd'
```
Exercise

- Write a function called `heart_risk` that has two parameters: `age` and `bmi` and returns an appropriate value (‘Low’, ‘Medium’, or ‘High’) according to the following table:

<table>
<thead>
<tr>
<th>BMI</th>
<th>&lt;22.0</th>
<th>&gt;=22.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>&lt;45</td>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td>&gt;=45</td>
<td>Medium</td>
<td>High</td>
</tr>
</tbody>
</table>
Short-circuit evaluation (optional)

- When Python evaluates an expression containing `and` or `or`, it does so from left to right.
- As soon as it knows enough to stop evaluating, it stops even if some operands have not been looked at yet.
- We call it **short-circuit evaluation**.
- Examples:

  ```python
  >>> (2 < 4) or (2 / 0)
  True
  >>> (2 > 4) and (2 / 0)
  False
  >>> (2 < 4) or (2 / 0) and (2 > 4)
  True
  >>> (5 < 4) or ((2 / 0) and (2 > 4))
  ZeroDivisionError
  ```
Comparing strings

- Characters in strings are represented as integers, e.g., ‘A’ is represented as 65; ‘a’ as 97; and ‘2’ as 50 using ASCII code.
- Using the ASCII encoding Python compares strings as it does with numbers.
- Examples:

```python
>>> 'A' < 'a'
True
>>> 'A' > 'z'
False
>>> 'apple' < 'applf'
True
>>> 'September' in 'September 16, 2014'
True
```
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>ASCII table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td>NUL</td>
<td>(null)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>(start of heading)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX</td>
<td>(start of text)</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX</td>
<td>(end of text)</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT</td>
<td>(end of transmission)</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ</td>
<td>(enquiry)</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK</td>
<td>(acknowledge)</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL</td>
<td>(bell)</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS</td>
<td>(backspace)</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB</td>
<td>(horizontal tab)</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>012</td>
<td>LF</td>
<td>(NL line feed, new line)</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT</td>
<td>(vertical tab)</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF</td>
<td>(NP form feed, new page)</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR</td>
<td>(carriage return)</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO</td>
<td>(shift out)</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI</td>
<td>(shift in)</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>020</td>
<td>DLE</td>
<td>(data link escape)</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>021</td>
<td>DC1</td>
<td>(device control 1)</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>022</td>
<td>DC2</td>
<td>(device control 2)</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>023</td>
<td>DC3</td>
<td>(device control 3)</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>024</td>
<td>DC4</td>
<td>(device control 4)</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>025</td>
<td>NAK</td>
<td>(negative acknowledge)</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>026</td>
<td>SYN</td>
<td>(synchronous idle)</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>027</td>
<td>ETB</td>
<td>(end of trans. block)</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>030</td>
<td>CAN</td>
<td>(cancel)</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>031</td>
<td>EM</td>
<td>(end of medium)</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>032</td>
<td>SUB</td>
<td>(substitute)</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>033</td>
<td>ESC</td>
<td>(escape)</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>034</td>
<td>FS</td>
<td>(file separator)</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>035</td>
<td>GS</td>
<td>(group separator)</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>036</td>
<td>RS</td>
<td>(record separator)</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>037</td>
<td>US</td>
<td>(unit separator)</td>
</tr>
</tbody>
</table>

Source: www.LookupTables.com
Keyboard input

- `raw_input`: to read input from keyboard without evaluating it
- `input`: to read input from keyboard with evaluation

```python
>>> text = raw_input()
How are you?
>>> print text
How are you?
>>> expr = raw_input('Enter an expression: ')
Enter an expression: 4 + 5
>>> print expr
4 + 5
>>> n = input('Enter an expression: ')
Enter an expression: 4 + 5
>>> print n
9
```
Recursive functions (optional)

- A function can call itself – such a function is called *recursive*
- A recursive function typically has three elements:
  - Recursive call
  - Base case ("escape hatch") to end the recursive calls
  - An argument in the recursive call that moves toward the base case

```python
def countdown(n):
    if n <= 0:
        print 'Blastoff!'
    else:
        print n
countdown(n-1)

def factorial(n):
    if n <= 1:
        return 1
    else:
        return n*factorial(n-1)
```
Incremental development

1. Given a problem, **start with a partial solution** that can be improved to reach the full solution.

2. Make sure the partial solution is fully debugged before you add more toward the full solution.

- Example: Suppose you are to write a function named `dollars_to_bills` that converts a number of dollars into the number of twenty dollar bills, the number of ten dollar bills, the number of five dollar bills, and the number of one dollar bills. If you use incremental development, you would try the following:
  - First, write `dollars_to_bills` that only works for $20 bills and fully debug it.
  - When and only when that is done, then add more code to handle $10 bills as well and again fully debug it.
  - Continue this way until you are done.
Boolean functions

- **boolean function**: returns a boolean value as its return value

```python
def isEven(n):
    if n%2 == 0:
        return True
    else:
        return False

def isEven(n):  # does the same but simpler code!
    return n%2 == 0

- Boolean functions are often used in conditional statements

```python
    if isEven(33):
        doSomething()
    else:
        doSomethingElse()
```
Checking types on the fly (optional)

- Some **defensive programming**

```python
def factorial(n):
    if not isinstance(n, int):
        print 'Factorial is defined only for integers.'
        return None
    elif n < 0:
        print 'Factorial is not defined for negative integers.'
        return None
    elif n == 0:
        return 1
    else:
        return n*factorial(n-1)
```

- Be aware of this concept, but don’t try to check for valid input in your programming unless you are specifically asked to do so!
Do these before next class

- Finish reading Chapters 5 and 6
  - Try the examples using your Python installation

- Next time: Chapter 7 Iteration (Repeating things)