Week 5

while loops; logic; random numbers; tuples

Special thanks to Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.
Except where otherwise noted, this work is licensed under:
http://creativecommons.org/licenses/by-nc-sa/3.0
while test:
  statements

>>> n = 91
>>> factor = 2  # find first factor of n

>>> while n % factor != 0:
...     factor += 1
... 
>>> factor
7
while test:
    statements
else:
    statements

- Executes the else part if the loop never enters
- There is also a similar for / else statement

```python
>>> n = 91
>>> while n % 2 == 1:
...     n += 1
...     print n, "was even; no loop."
91 was even; no loop.
```
• Python's logic type, equivalent to boolean in Java
  - True and False start with capital letters

```python
>>> 5 < 10
True

>>> b = 5 < 10
>>> b
True

>>> if b:
...     print "The bool value is true"
...
The bool value is true

>>> b = not b
>>> b
False
```
# Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equals</td>
<td><code>1 + 1 == 2</code></td>
<td>True</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>does not equal</td>
<td><code>3.2 != 2.5</code></td>
<td>True</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
<td><code>10 &lt; 5</code></td>
<td>False</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
<td><code>10 &gt; 5</code></td>
<td>True</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
<td><code>126 &lt;= 100</code></td>
<td>False</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
<td><code>5.0 &gt;= 5.0</code></td>
<td>True</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>and</code></td>
<td><code>2 == 3 and -1 &lt; 5</code></td>
<td>False</td>
</tr>
<tr>
<td><code>or</code></td>
<td><code>2 == 3 or -1 &lt; 5</code></td>
<td>True</td>
</tr>
<tr>
<td><code>not</code></td>
<td><code>not -1 &lt; 5</code></td>
<td>False</td>
</tr>
</tbody>
</table>
from random import *

randint(min, max)
– returns a random integer in range [min, max] inclusive

choice(sequence)
– returns a randomly chosen value from the given sequence
  • the sequence can be a range, a string, ...

>>> from random import *
>>> randint(1, 5)
  2
>>> randint(1, 5)
  5
>>> choice(range(4, 20, 2))
  16
>>> choice("hello")
  'e'
Exercise

- Rewrite the *Dice* program from Java to Python:

\[
\begin{align*}
2 + 4 &= 6 \\
3 + 5 &= 8 \\
5 + 6 &= 11 \\
1 + 1 &= 2 \\
4 + 3 &= 7
\end{align*}
\]

You won after 5 tries!
Tuple

tuple_name = (value, value, ..., value)
– A way of "packing" multiple values into one variable

```python
>>> x = 3  
>>> y = -5  
>>> p = (x, y, 42)  
>>> p  
(3, -5, 42)
```

name, name, ..., name = tuple_name
– "unpacking" a tuple's contents into multiple variables

```python
>>> a, b, c = p  
>>> a  
3  
>>> b  
-5  
>>> c  
42
```
Using Tuples

- Useful for storing multi-dimensional data (e.g. (x, y) points)

```python
>>> p = (42, 79)
```

- Useful for returning more than one value

```python
>>> from random import * 
>>> def roll2():
...     die1 = randint(1, 6)
...     die2 = randint(1, 6)
...     return (die1, die2)
... 
>>> d1, d2 = roll2()
>>> d1
6
>>> d2
4
```
def name((name, name, ..., name), ...):
    statements

    - Declares tuple as a parameter by naming each of its pieces

    >>> def slope((x1, y1), (x2, y2)):
        ...     return (y2 - y1) / (x2 - x1)
        ...
        >>> p1 = (2, 5)
        >>> p2 = (4, 11)
        >>> slope(p1, p2)
        3
def name(parameters):
    statements
    return (name, name, ..., name)

>>> from random import *
>>> def roll2():
    ...    die1 = randint(1, 6)
    ...    die2 = randint(1, 6)
    ...    return (die1, die2)
    ...
    ...
    d1, d2 = roll2()
>>> d1
6
>>> d2
4
Exercise

• Write a program that performs a graphical "random walk".
  – Create a DrawingPanel of size 150x150.
  – Draw a circle of radius 75.
  – Start a "walker" pixel at (75, 75), the circle's center.
  – Every 10 ms, randomly move the walker by 1 pixel in either the x or y direction and redraw the walker.
  – Once the walker reaches the edge of the circle, stop walking.

• Key questions:
  – How do we randomly move by 1 pixel?
  – How do we know when we have reached the edge of the circle?