CIS 122

Logical Conditioning
Homework Note

● Last week, your code *did* something when you ran it
  ○ Printed out an info sheet
  ○ Printed out some skittle counts

● This week's homework is more passive
  ○ Less printing
  ○ More defining

● It's ok if nothing happens when you run your code
  ○ Check your definitions in the shell
  ○ Test your functions in the shell
Functions so far

- Take (zero or more) values as input
- Perform a set of operations
  - Assignments
  - Other function calls
- Return some value as output
Functions so far

- Currently, functions always follow the same steps
- Great if we want to treat every input the same way
  - `addOne` - Given a number, return its successor
  - Temperature Conversion
- But what if we want different things in different situations?
  - `abs` - Given a number, return its absolute value
  - `middle` - Given a string, return the middle character(s)
Conditional Logic

- We'd like to allow our programs to branch
  
  ```python
  if <something is true>:
      <do one thing>
  
  else:
      <do something else>
  ```

- But what is truth?
  - We need a new object type
Booleans

- A very simple object type

- Most types have infinitely many values
  - Booleans only have two
  - True / False

- Where have we seen them before?
We produce booleans when we compare objects:

- $a > b$ - greater than
- $a < b$ - less than
- $a \geq b$ - greater than or equal to
- $a \leq b$ - less than or equal to
- $a == b$ - equal to
- $a != b$ - not equal to
Comparisons

- Note, the equality operator is `==`
  - `=` was already taken for assignment
  - When you compare values, make sure to use `==`
  - Strange things will happen otherwise

```python
>>> a = 5
Assigns the value 5 to the variable a

>>> a == 5
Returns True if a holds the value 5, False otherwise
```
Comparisons

● Any two objects can be compared to return a boolean
  ○ $1 > 2$
  ○ $3.5 \leq 8.0$
  ○ `'a' == 'b'`
  ○ `True != False`

● We can even compare multiple objects simultaneously
  ○ $1 < x < 5$

● Which is greater, True or False?
Conditional Logic

- What can we do with booleans?
  - Branch!

- The `if` keyword runs code only if some condition is true
  - Always followed by a boolean condition

```python
if x == 0:
    print "x is zero"
```

- Note the colon
  - About to define a block of code
  - Indented text
if x == 0:
    print "x is zero"
Conditional Logic

- The `else` keyword runs code if a condition is false
  - Always paired with an `if`
  - Not followed by a condition

```python
if x == 0:
    print "x is zero"
else:
    print "x is not zero"
```
```python
if x == 0:
    print "x is zero"
else:
    print "x is not zero"
```
Conditional Logic

What if we want to choose between multiple conditions?
  - We could nest if statements...

```python
if x == 0:
    print "x is zero"
else:
    if x == 1:
        print "x is one"
    else:
        if x == 2:
            print "x is two"
        else:
            print "beats me"
```
Conditional Logic

- Python provides a shortcut for nesting if statements
  - The `elif` keyword acts as a combined `else` and `if`

```python
if x == 0:
    print "x is zero"
elif x == 1:
    print "x is one"
elif x == 2:
    print "x is two"
else:
    print "beats me"
```
if x == 0:
    print "x is zero"
elif x == 1:
    print "x is one"
elif x == 2:
    print "x is two"
else:
    print "beats me"
Conditional Logic Applied

- Let's put what we've learned to use
  - Finish the function

```python
def abs(x):
  """Return the absolute value of x"""
```
Conditional Logic Applied

- Let's put what we've learned to use
  - Finish the function

```python
def abs(x):
    """Return the absolute value of x""
    if x < 0:
        return -x
    else:
        return x
```
Conditional Logic Applied

- Let's put what we've learned to use
  - How about this one?

```python
def middle(string):
    """Return the middle character(s) from string"""

- How would we solve this problem?