Boolean Expressions and If

- Flow of Control / Conditional Statements
- The if Statement
- Logical Operators
- The else Clause
- Block statements
- Nested if statements
- Reading for this class:
  - Dawson, Chapter 3
  - http://introcs.cs.princeton.edu/python/13flow
Flow of Control

• Default order of statement execution is linear: one after another in sequence

• But, sometimes we need to decide which statements to execute and/or how many times

• These decisions are based on boolean expressions (or “conditions”) that evaluate to True or False

• The resulting order of statement execution, according to these decisions, is called the flow of control
Conditions/Boolean Expressions

• A condition is often expressed as a boolean expression (which returns a boolean result).
• Boolean expressions, like arithmetic ones, use operators, such as the following equality and relational operators:
  
  ```
  ==   equal to
  !=   not equal to
  <    less than
  >    greater than
  <=   less than or equal to
  >=   greater than or equal to
  ```

• **Note:** `==` and `=` are **not** the same!
Boolean Expressions

5 < 7  
7 >= 5  
x == 98

len(password) >= MIN_LENGTH

ins_prem * months != benefits - deductible

(volume - (1 / ph_value)) * 2 <= 1 / q_factor

a-- * (b / ((c - d) % e)) ==

(b * (c / a) + ((3 % q) + 7))

offer < minimum_bid

grade+1 >= a_grade

t_weight < weight
Logical Operators

• The following *logical operators* can also be used in boolean expressions:

  - `not` Logical NOT
  - `and` Logical AND
  - `or` Logical OR

• They operate on *boolean operands* and produce *boolean results*
  
  - Logical NOT is a *unary* operator => *one operand*
  - AND and OR are *binary* operators => *two operands*
Logical NOT

- The logical NOT operation is also called logical negation or logical complement.
- If some boolean condition \( a \) is True, then \( \text{not } a \) is False.
- If \( a \) is False, then \( \text{not } a \) is True.
- Logical operations can be shown with a truth table.

<table>
<thead>
<tr>
<th>a</th>
<th>not a</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
Logical AND and Logical OR

• The *logical AND* expression
  \[ a \text{ and } b \]
  • is True if **both** \( a \) and \( b \) are True, and False otherwise

• The *logical OR* expression
  \[ a \text{ or } b \]
  • is True if **at least** one of \( a \) or \( b \) is True, and False otherwise
### Logical Operators

- A truth table shows all possible True-False combinations of the terms.
- Since **and** and **or** each have two operands, there are four possible combinations of conditions \(a\) and \(b\).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>a and b</th>
<th>a or b</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Short-Circuited Operators

• The processing of logical AND and logical OR is “short-circuited”

• If the left operand is sufficient to determine the result, the right operand is not evaluated. Example:

```
if count != 0 and total/count > MAX:
    print ("Testing...")
```

• Why would you do this?

• This coding technique must be used carefully
More Boolean Expressions

- **NOTE:** You should look at these primarily as examples of how boolean expressions can be combined into more complex ones.

\[
\begin{align*}
5 & < 7 \quad \text{or} \quad \text{offer} & < & \text{min\_bid} \\
7 & \geq 5 \quad \text{and} \quad x & = & 98 \\
\text{not done} & \quad \text{and} \quad x & = & 47 \\
\text{not} \quad (5 & < 7 \quad \text{or} \quad \text{offer} & < & \text{MIN}) \quad \text{or} \quad (7 & \geq 5 \quad \text{and} \quad x & = & 98) \\
\text{not} \quad (\text{grade} & \geq & \text{a\_grade}) \quad \text{and} \quad \text{not} \quad (t\_weight & < & \text{weight}) \\
\text{not} \quad (\text{len}(\text{password}) & \geq & \text{MIN}) \quad \text{or} \quad \text{my\_boolean}
\end{align*}
\]
Conditional Statements

• A *conditional statement* decides which program statement will be executed next

• We use conditional statements to make basic decisions as the program runs.

• Recall the quadratic formula example:
  o Check if $a = 0$, if $b = 0$, etc.

• In Python, we have a number of variations of the conditional statement:
  – *if statement*
  – *if-else statement*
  – *if-elif-else statement*
The if Statement

• The *if statement* has the following syntax:

```python
if condition:
    statement
    statement
    statement
```

The *condition* must be a boolean expression. It must evaluate to either True or False.

If the *condition* is True (i.e., evaluates to True), the *statements* are executed. If it is False, the *statements* are skipped.

See password.py
The if Statement

• An example of an if statement:

```python
if sum > MAX:
    delta = sum - MAX
    print ("The sum is " + str(sum))
```

• First the condition is evaluated -- either the value of `sum` is either **greater** than the value of `MAX`, or **it is not**

• If the condition is True, the assignment statement is executed -- if False, it is not

• The `print` statement, **not** being contingent upon `sum < MAX`, is always executed next
Indentation

- The statement controlled by the `if` statement is **indented** to indicate that relationship
  
  ```python
  if sum > MAX:
      delta = sum - MAX
      print ("The sum is " + str(sum))
  ```

- A consistent indentation style makes a program easier to read and understand

- In Python, unlike many other languages, proper indentation is necessary for the program to be interpreted correctly!

- Moreover, **human readers** care!
Blocks of Statements

• Several statements can be indented in order to create a “block”

```python
if total > MAX:
    print ("Error!!")
    error_count += 1
```

• A block can be used to indicate several statements are subordinate to another

• “if [condition is True]:”
  – one statement => “do this thing”
  – 2 or more => “do this group of things”
The if-else Statement

• An *else clause* can be added to an *if* statement to make an *if-else statement*

```python
if condition:
    statement1
else:
    statement2
```

- *condition is True* => *statement1* is executed
- *condition is False* => *statement2* is executed

• One or the other will be executed, but not both

See granted_or_denied.py
Block Statements

• In an if-else statement, the if portion, or the else portion, or both, could be blocks:

```python
if total > MAX:
    print ("Error!!")
    error_count += 1
else:
    print ("Total: " + str(total))
    current = total * 2
```

• Think of each block as a "game plan" for one situation versus the other
Composing an if(-else) statement

```python
if offer < minimum_bid:
    print ("Offer is too low.")
    print ("Please bid at least", minimum_bid)
    offer = float(input("Your bid: "))
    print ("You bidded "+ str(offer))

else:
    print ("You bidded "+ str(offer))
    print ("Raise your offer to the",
           "current highest? YES or NO")
    answer = input ("Your reply: ")
```
Nested if Statements

• An if statement or an else clause can contain another conditional statement
• The inner if statement is treated as a single statement, but...
• An else clause is matched to the last unmatched if by default, unless...
• **Indentation** is used to specify the if statement to which an else clause belongs
num = 3
if num > 2:
    print ("num > 2")
if num > 4:
    print ("num > 4, too!")
else:
    print ("num <= 2")

-> num > 2
num <= 2
With Correct Indentation

```python
num = 3
if num > 2:
    print("num > 2")
    if num > 4:
        print("num > 4, too!")
else:
    print("num <= 2")

-> num > 2
```

Prints correct result!
The *if-elif-else* Statement

• Sometimes, you may have multiple conditions to consider, in which case you can add *elif clauses* can to your *if* statement:

```python
if condition1:
    statement1
elif condition2:
    statement2
elif condition3:
    statement3
...
else:
    default_statement
```
The if-elif-else Statement

• If condition1 is True, execute that block and continue with the program.
• Otherwise, try condition2, and so forth.
• If no conditions are true, execute the else block.
• The final else clause is optional. Its purpose is to serve as a default, when none of the conditions apply.

See: mood_computer.py
     if_elif_else.py
The Conditional Operator

• Python has a *conditional operator* that uses a boolean condition to evaluate one of two expressions

• Its syntax is:

```
expression1 if condition else expression2
```

• If the *condition* is True, *expression1* is evaluated; if it is False, *expression2* is evaluated

• The value of the entire conditional operator is *the value of the selected expression*

See min_of_three_cond.py
The Conditional Operator

• The conditional operator is similar to an `if-else` statement, except that it is an expression that returns a single value

• For example, these are functionally equivalent:

```python
larger = num1 if num1 > num2 else num2

if num1 > num2:
    larger = num1
else:
    larger = num2
```

• The conditional operator is *ternary* because it requires three operands: a condition and two alternative values
Non-Boolean Values as Conditions

• In Python, you are not limited to using values of True and False as conditions.
• In fact, *any* value can be interpreted as True or False, specifics depending on the type.
• For *numbers*, zero is False while anything else is True.
• For *strings*, the empty string "" is False, while non-empty strings are True.
• More examples to come...

See maitre_d.py