CMSC201
Computer Science I for Majors
Lecture 05 – Comparison Operators and Boolean (Logical) Operators

Prof. Katherine Gibson

Based on slides by Shawn Lupoli and Max Morawski at UMBC
Last Class We Covered

• Expressions
• Python’s operators
  – Including mod and integer division
• The order of operations
• Different variables types
  – How to cast to a type
• Constants (and why using them is important)
Any Questions from Last Time?
Today’s Objectives

• To learn a bit about `main()`
• To learn more of Python’s operators
  – Comparison operators
  – Logical operators
• To practice using these new operators
• To become more familiar with using Boolean variables
Quick Note about `main()`
main()

• In Lab 2, we introduced the code
def main():
    —as the first line of code in our file

• main() is an example of a function

• We can use functions to organize our code
Functions

• We’ll cover functions in more detail later

• For now, think of them as something similar to a variable
  – Variables hold data
  – Functions hold code
Calling `main()`

• With variables, we use the variable name to access the data they store

• We must do the same with functions like `main()`, using the function name to execute the code they store
Using `main()` for Your Code

• For our purposes, use `main()` with your code from now on:

```python
def main():
    class = int(input("What class is this? ")
    print(class, "is awesome!"))
main()```
Review:
Control Structures & Operators
Control Structures

• What are the three control structures?
  – Sequential
  – Decision Making
    • Also known as “Selection”
  – Looping
    • Also known as “Repetition”

• We can also call a function
Control Structures: Flowcharts

- a. Sequence
- b. Selection
- c. Repetition
Types of Operators in Python

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Focus of today’s lecture
Comparison Operators
Vocabulary

• Comparison operators
• Relational operators
• Equality operators
  – Are all the same thing

• Include things like $>$, $\geq$, $<$, $\leq$, $==$,$!=$
Vocabulary

• Logical operators
• Boolean operators
  – Are the same thing

• Include **and**, **or**, and **not**
Comparison Operators

• Always return a Boolean result
  – True or False
  – Indicates whether a relationship holds between their operands

\[
a \geq b
\]
Comparison Examples

• What is the following comparison asking?

  \[ a \geq b \]
  
  – Is \( a \) greater than or equal to \( b \)?

  \[ a == b \]
  
  – Is \( a \) equal to \( b \)?
## List of Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>If the values of two operands are equal, then the condition becomes true.</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>If values of two operands are not equal, then condition becomes true.</td>
</tr>
<tr>
<td><code>&lt;&gt;</code></td>
<td>If values of two operands are not equal, then condition becomes true.</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>If the value of left operand is greater than the value of right operand, then condition becomes true.</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>If the value of left operand is less than the value of right operand, then condition becomes true.</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>If the value of left operand is less than or equal to the value of right operand, then condition becomes true.</td>
</tr>
</tbody>
</table>

*Note: `<>` is outdated. Use `!=` for “not equal to.”*
List of Operators (Continued)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>strictly less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>strictly greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal</td>
</tr>
<tr>
<td><code>==</code></td>
<td>equal</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal</td>
</tr>
<tr>
<td><code>is</code></td>
<td>object identity</td>
</tr>
<tr>
<td><code>is not</code></td>
<td>negated object identity</td>
</tr>
</tbody>
</table>

https://docs.python.org/3.3/library/stdtypes.html
Comparison Examples (Continued)

• What do these evaluate to if \( a = 10 \) and \( b = 20 \)?

\[ a \geq b \]

– Is \( a \) greater than or equal to \( b \)?
– Is \( 10 \) greater than or equal to \( 20 \)?
– FALSE
Comparison Examples (Continued)

• What do these evaluate to if $a = 10$ and $b = 20$?

  $a == b$
  
  – Is $a$ equal to $b$?
  
  – Is 10 equal to 20?
  
  – FALSE
Comparison vs Assignment

• A common mistake is to use the assignment operator (\(=\)) in place of the relational (\(==\)) – This is a very common mistake to make!

What does \(a=b\) do? Sets \(a\) equal to \(b\).

What does \(a==b\) do? Asks does \(a\) equal \(b\)?

This type of mistake will usually not trigger an error!
Comparison Operator Examples
Comparison Operators and Simple Data Types

- Examples:
  - $8 < 15$ evaluates to $True$
  - $6 \neq 6$ evaluates to $False$
  - $2.5 > 5.8$ evaluates to $False$
  - $5.9 \leq 7.5$ evaluates to $True$
“Value” of Boolean Variables

• When we discuss Boolean outputs, we think
  – True and False
• but we can also think of it in terms of
  – 1 and 0

• True = 1
• False = 0
Comparison Operation Examples

```python
a = 10
b = 20
c = 30

bool1 = a == b
bool2 = c < b
bool3 = c != a

print(bool1, bool2, bool3)
```

Prints:

False False True
More Comparison Operation Examples

\[
a = 10 \\
b = 20 \\
c = 30
\]

Prints:

\[
1 \text{ False } 3
\]

\[
\text{bool1} = \text{int}(a==a) \\
\text{bool2} = a==a \geq 10 \\
\text{bool3} = (a==a) + (b==b) + (c==c)
\]

print(bool1, bool2, bool3)
Logical Operators
Logical Operators

• There are three logical operators:
  – and
  – or
  – not

• They allow us to build more complex Boolean expressions
  – By combining simpler Boolean expressions
Logical Operators – **and**

- Let’s evaluate this expression
  
  \[ \text{bool1} = a \text{ and } b \]

<table>
<thead>
<tr>
<th>Value of a</th>
<th>Value of b</th>
<th>Value of bool1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Logical Operators – **and**

- Let’s evaluate this expression
  
  \[
  \text{bool1} = a \text{ and } b
  \]

<table>
<thead>
<tr>
<th>Value of a</th>
<th>Value of b</th>
<th>Value of bool1</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Logical Operators – \texttt{and}

- Let’s evaluate this expression
  \[ \text{bool1} = a \text{ and } b \]

<table>
<thead>
<tr>
<th>Value of ( a )</th>
<th>Value of ( b )</th>
<th>Value of ( \text{bool1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

- For \( a \text{ and } b \) to be \textbf{True}, both \( a \) and \( b \) must be true
Logical Operators – and

• Two ways to write and expressions
  1. Explicitly use the keyword:
     \[ 3 > 2 \text{ and } 2 > 1 \]

  2. String them together, like in math:
     \[ x > y > z \]
     – Evaluates to \[ x > y \text{ and } y > z \]
Examples of \texttt{and}

\begin{verbatim}
a = 10
b = 20
c = 30

ex1 = a < b < c
ex2 = a < b and b < c
ex3 = a+b==c and b-10==a and c/3==a

print (ex1, ex2, ex3)
\end{verbatim}

Prints:

True True True
More Examples of `and`

\[
\begin{align*}
\text{a} &= 10 \\
\text{b} &= 20 \\
\text{c} &= 30
\end{align*}
\]

\[
\begin{align*}
\text{bool1} &= \text{a} > \text{b} > \text{c} \\
\text{bool2} &= \text{a} == \text{b} > \text{c} \\
\text{bool3} &= \text{a} < \text{b} < \text{c}
\end{align*}
\]

Prints:

\[
\text{False False True}
\]
Logical Operators – \texttt{or}

• Let’s evaluate this expression

\[ \text{bool1} = a \text{ or } b \]

<table>
<thead>
<tr>
<th>Value of ( a )</th>
<th>Value of ( b )</th>
<th>Value of ( \text{bool1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Logical Operators – **or**

- Let’s evaluate this expression
  
  $$\text{bool1} = a \text{ or } b$$

<table>
<thead>
<tr>
<th>Value of $a$</th>
<th>Value of $b$</th>
<th>Value of bool1</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Logical Operators – or

- Let’s evaluate this expression
  
  `bool1 = a or b`

<table>
<thead>
<tr>
<th>Value of <code>a</code></th>
<th>Value of <code>b</code></th>
<th>Value of <code>bool1</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

- For `a or b` to be True, either `a` or `b` must be true
Examples of `or`

\[
\begin{align*}
    a &= 10 \\
    b &= 20 \\
    c &= 30 \\

    \text{ex1} &= a > b \text{ or } c < b \\
    \text{ex2} &= a + b \leq c + 1 \text{ or } b > c \\
    \text{ex3} &= a == c \text{ or } b + 10 \leq a \text{ or } c/3 == a \\

    \text{print (ex1, ex2, ex3)}
\end{align*}
\]
Logical Operators – \texttt{not}

- Let’s evaluate this expression
  \[
  \text{bool1} = \text{not a}
  \]

<table>
<thead>
<tr>
<th>Value of a</th>
<th>Value of bool1</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

- \texttt{not a} returns the opposite Boolean value of a
Complex Expressions

• We can put multiple operators together!
  
  ```python
  bool1 = a and (b or c)
  ```

• What does Python do first?
  – Computes `(b or c)`
  – Computes the `and` with `a` and the result
Complex Expression Example

bool1 = a and (b or c)

<table>
<thead>
<tr>
<th>Value of a</th>
<th>Value of b</th>
<th>Value of c</th>
<th>Value of bool1</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
“Short Circuit” Evaluation
Short Circuit Evaluation

- “and” statements short circuit as soon as an expression evaluates to False

- “or” statements short circuit as soon as an expression evaluates to True
Short Circuiting – and

• Notice that in the expression:
  \[ \text{bool1} = a \text{ and } (b \text{ or } c) \]
• If \( a \) is False
• The rest of the expression doesn’t matter

• Python will realize this, and if \( a \) is false won’t bother with the rest of the expression
Short Circuiting – or

• Notice that in the expression:
  \[
  \text{bool1} = a \text{ or } (b \text{ or } c)
  \]

• If \(a\) is True

• The rest of the expression doesn’t matter

• Python will realize this, and if \(a\) is true
  won’t bother with the rest of the expression
More Practice

• Given:
  a = 4
  b = 5
  c = 6
  d = True
  e = False

bool1 = d and (a > b)  
False

bool2 = (not d) or (b != c)  
True

bool3 = (d and (not e)) or (a > b)  
True

bool4 = (a%b==2) and ((not d) or e)  
False
More More More Practice

• Given:
  a = 4
  b = 5
  c = 6
  d = True
  e = False

  bool1 = (d + d) >= 2 and (not e)
          True

  bool2 = (not e) and (6*d == 12/2)
          True

  bool3 = (d or (e)) and (a > b)
          False
Numbers and Booleans

• Python accepts anything that is non-zero as **True**
  – There are some exceptions, but we’ll get into those later

• So technically you can use any integer as a Boolean expression
Decision Making

• So, why do we care about comparison operators and logical operators so much?

Answer: Next Class
Announcements

• Your Lab 3 is meeting normally this week!
  – Make sure you attend your correct section

• Homework 2 is out
  – Due by Tuesday (Sept 15th) at 8:59:59 PM

• Homeworks are on Blackboard
  – Weekly Agendas are also on Blackboard