CMSC201
Computer Science I for Majors

Lecture 05 – Comparison Operators and Boolean (Logical) Operators

Prof. Katherine Gibson

Prof. Jeremy Dixon

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 1, 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

- From now on, use `main()` in your code:

declaring our `main()` function

```
def main():
    class = int(input("What class is this? "))
    print(class, "is awesome!")

main()
```

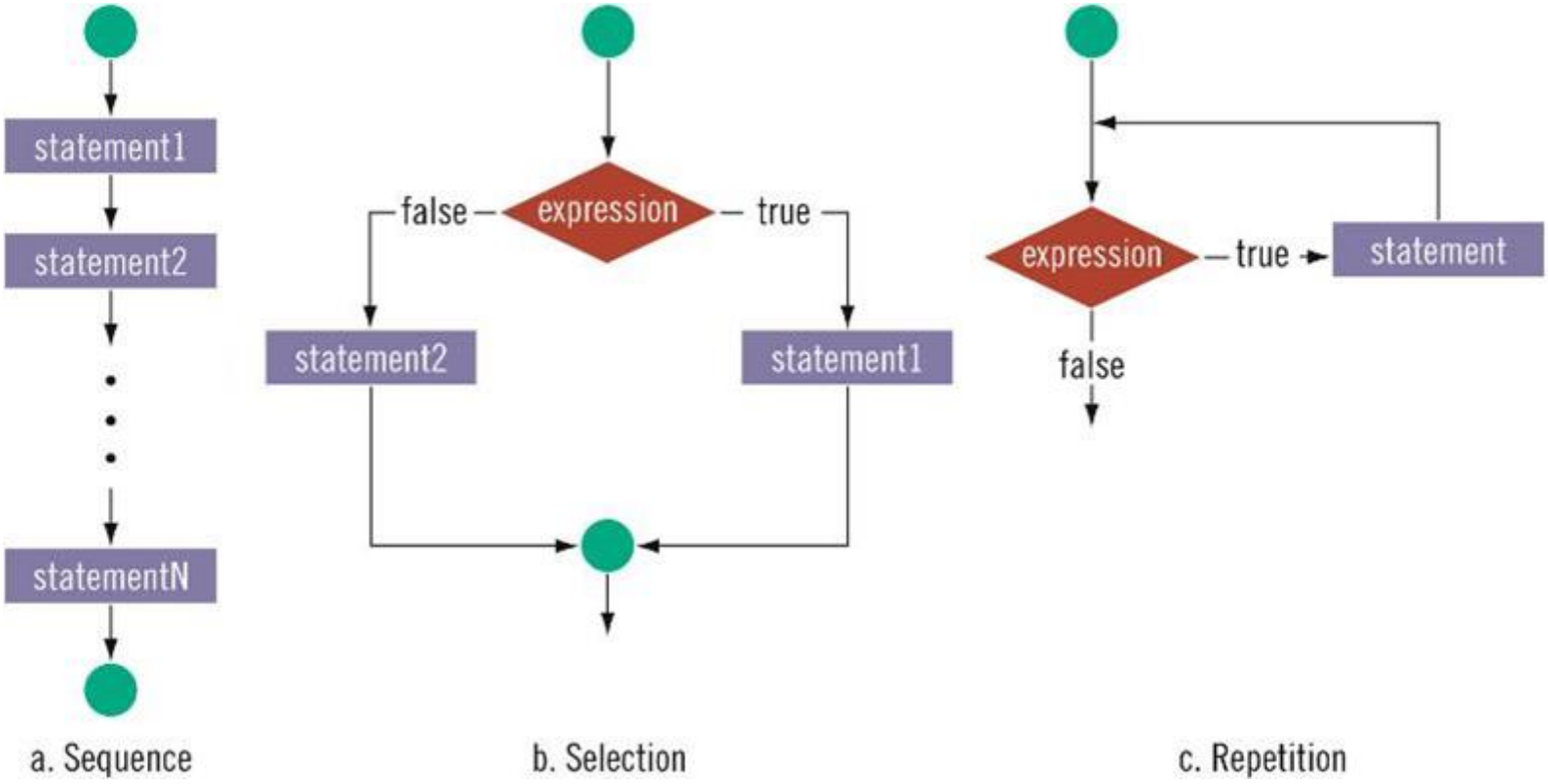
calling our `main()` function

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



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 **>**, **>=**, **<**, **<=**, **==**, **!=**

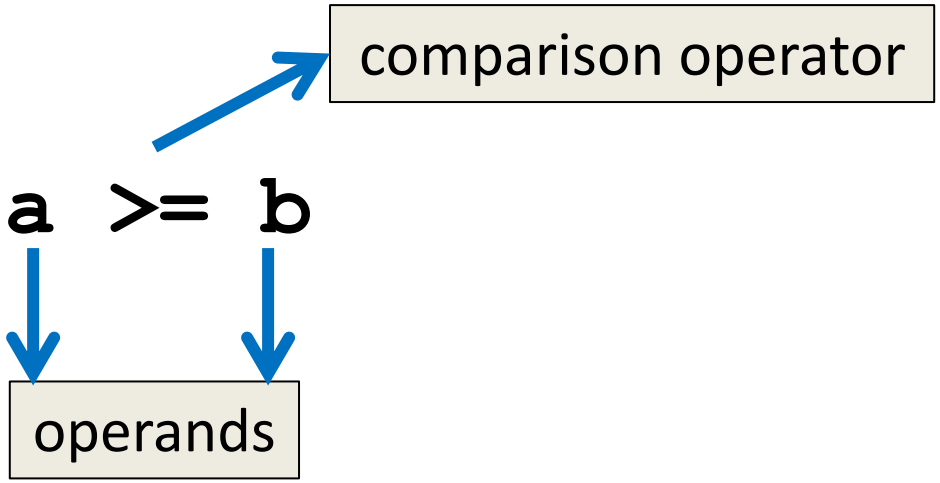
Vocabulary

- 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



Comparison Examples

- What is the following comparison asking?

  `a >= b`

  – Is **`a`** greater than or equal to **`b`**?

  `a == b`

  – Is **`a`** equivalent to **`b`**?

List of Operators

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal

Comparison Examples (Continued)

- What do these evaluate to if `a = 10` and `b = 20`?

  `a >= 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` equivalent to `b`?
  - Is `10` equivalent 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!

- This type of mistake does trigger an error in Python, but you may still make it on paper!

Equals vs Equivalence

- What does `a = b` do?
  - Sets `a` equal to `b`
  - Replaces `a`'s value with the value of `b`

- What does `a == b` do?
  - Checks if `a` is equivalent to `b`

Comparison Operator Examples

Comparison Operators and Simple Data Types

- Examples:

`8 < 15`	evaluates to	**`True`**
`6 != 6`	evaluates to	**`False`**
`2.5 > 5.8`	evaluates to	**`False`**
`5.9 <= 7.5`	evaluates to	**`True`**

"Value" of Boolean Variables

- When we discuss Boolean outputs, we think

  `True` and **`False`**

- We can also think of it in terms of

  `1` and `0`

- **`True  = 1`**
- **`False = 0`**

"Value" of Boolean Variables

- Other data types can also be seen as "**`True`**" or "**`False`**" in Python

- Anything empty or zero is **`False`**
  - `""` (empty string), `0`, `0.0`
- Everything else is **`True`**
  - `81.3`, `77`, `-5`, `"zero"`, `0.01`
  - Even `"0"` evaluates to **`True`**

Comparison Operation Examples

```
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

bool1 = int(a == a)
bool2 = a == a >= 10
bool3 = (a == a) + (b == b) + (c == c)

print(bool1, bool2, bool3)
```

Prints:

```
1 True 3
```

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

```
bool1 = a and b
```

Value of `a`	Value of `b`	Value of `bool1`

Logical Operators – `and`

- Let's evaluate this expression

```
bool1 = a and b
```

Value of `a`	Value of `b`	Value of `bool1`
`True`	`True`	
`True`	`False`	
`False`	`True`	
`False`	`False`	

Logical Operators – `and`

- Let's evaluate this expression

```
bool1 = a and b
```

Value of `a`	Value of `b`	Value of `bool1`
`True`	`True`	`True`
`True`	`False`	`False`
`False`	`True`	`False`
`False`	`False`	`False`

- For `a and b` to be `True`, both `a` and `b` must be true

Logical Operators – `and`

- Two ways to write **`and`** expressions
  1. Explicitly use the keyword:

     ```
     3 > 2 and 2 > 1
     ```

  2. String them together, like in math:

     ```
     x > y > z
     ```

     – Evaluates to **`x > y and y > z`**

Examples of **and**

```
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)
```

Prints:

```
True True True
```

More Examples of `and`

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

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

print(bool1, bool2, bool3)
```

Prints:

```
False False True
```

Logical Operators – `or`

- Let's evaluate this expression

```
bool2 = a or b
```

Value of `a`	Value of `b`	Value of `bool2`

Logical Operators – `or`

- Let's evaluate this expression

```
bool2 = a or b
```

Value of `a`	Value of `b`	Value of `bool2`
`True`	`True`	
`True`	`False`	
`False`	`True`	
`False`	`False`	

Logical Operators – `or`

- Let's evaluate this expression

```
bool2 = a or b
```

Value of `a`	Value of `b`	Value of `bool2`
`True`	`True`	`True`
`True`	`False`	`True`
`False`	`True`	`True`
`False`	`False`	`False`

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

Examples of `or`

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

ex1 = a > b or c < b
ex2 = a + b <= c + 1 or b > c
ex3 = a == c or b + 10 <= a or c/3 == a

print (ex1, ex2, ex3)
```

Prints:

```
False True True
```

Usage Example

- Here's an easy way to remember how the **`and`** and **`or`** logical operators work

- In order to pass the class, you must have:

```
(grade >= 70) and (cheating == False)
```

- For the grade to count for CMSC majors:

```
ltrGrade == "A" or ltrGrade == "B"
```

Logical Operators – `not`

- Let's evaluate this expression

```
bool3 = not a
```

Value of `a`	Value of `bool3`
`True`	`False`
`False`	`True`

- `not a` calculates the Boolean value of `a` and returns the opposite of that

Complex Expressions

- We can put multiple operators together!

```
bool4 = a and (b or c)
```

- What does Python do first?
  - Computes `(b or c)`
  - Computes `a and` the result

Complex Expression Example

```
bool4 = a and (b or c)
```

Value of a	Value of b	Value of c	Value of bool4
True	True	True	
True	True	False	
True	False	True	
True	False	False	
False	True	True	
False	True	False	
False	False	True	
False	False	False	

Complex Expression Example

```
bool4 = a and (b or c)
```

Value of a	Value of b	Value of c	Value of bool4
True	True	True	True
True	True	False	True
True	False	True	True
True	False	False	False
False	True	True	False
False	True	False	False
False	False	True	False
False	False	False	False

Truth Table Layout

- Truth tables follow a pattern for their values

Value 1	Value 2	Value 3	Answer
True	True	True	
True	True	False	
True	False	True	
True	False	False	
False	True	True	
False	True	False	
False	False	True	
False	False	False	

“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:

```
bool1 = a and (b 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:

```
bool1 = a or (b 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 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
```

Decision Making

- So, why do we care about comparison operators and logical operators so much?
- We can use them to *control* how our program works and what code it runs
  - We'll discuss this next time

Announcements

- Your Lab 2 is meeting this week!
  - Make sure you attend your correct section
- Homework 2 is out
  - Due by Monday (Feb 15th) at 8:59:59 PM
- Homework 2 is on Blackboard
  - Complete Academic Integrity Quiz to see HW2

Practice Problems

- Evaluate these expressions – do them yourself before testing them in Python!

```
False and False
1 == 1 or 2 == 1
True and 1 == 1
False and 0 == 0
True or 1 == 1
"test" == "testing"
"test" == 1
not (True and False)
not (1 == 1 and 0 >= 1)
not (10 == 1 or 1000 == 1000)
not (1 <= 10 or 3 == 4)
1 >= 0 and 2 == 1
not ("testing" == "testing" and "Zed" == "Cool Guy")
1 == 1 and (not ("testing" == 1 or 1 == 0))
```


From: http://learnpythonthehardway.org/book/ex28.html

Practice Problems

- Create and fill out truth tables for the following Boolean expressions
  - Try it with and without using short circuiting!

```
a or b or c
not a and not b
a or (b and not c)
a and (b or c) and not d
```