CMSC201
Computer Science I for Majors

Lecture 05 –Comparison Operators and Boolean (Logical) Operators

Prof. Jeremy Dixon

Last Class We Covered

- To learn more about expressions
- To learn Python's operators
  – Including mod and integer division
- To understand the order of operations
- To learn more about types
  – How to cast to a type
- To understand the use of constants

Today’s Objectives

- To introduce the usage of modules and main()
- To review of control structures
- To discuss vocabulary considerations
- To introduce Python’s relational operators
- To introduce Python’s logical operators
- To reinforce the order of operations

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:

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

main()
```

declaring our **`main()`** function

calling our **`main()`** function

Review of Control Structures

- A computer can proceed:
  - In sequence
  - Selectively (branch): making a choice
  - Repetitively (iteratively): looping
  - By calling a function
- Two most common control structures:
  - Selection
  - Repetition

Review Control Structures (cont'd.)

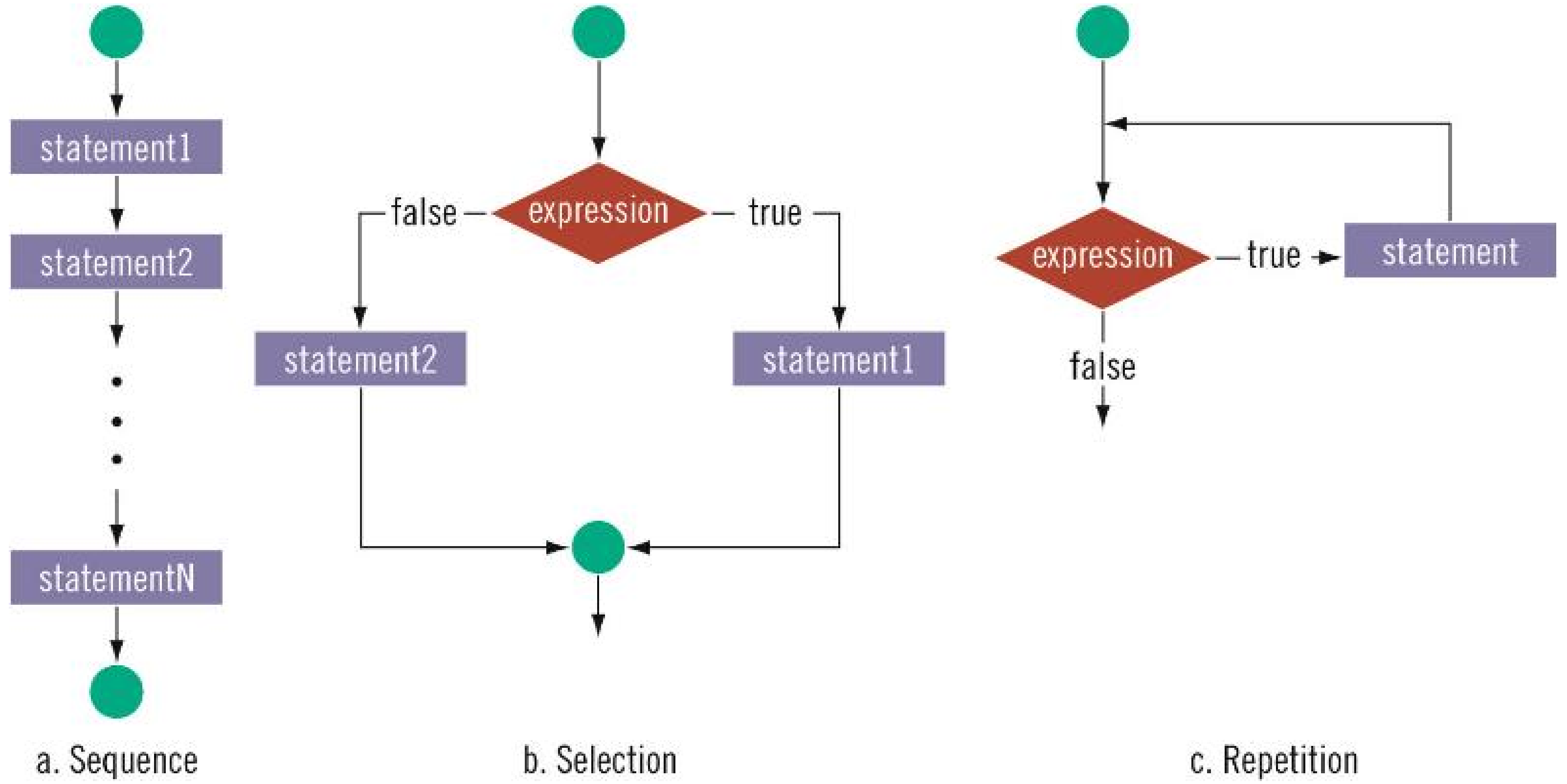

FIGURE 4-1 Flow of execution

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

Vocabulary Considerations

- *Comparison* operators, *relational* operators, and *equality* operators are all the same thing
  - Include >, >=,<,<=,==,!=
- *Logical* operators and *Boolean* operators are the same things
  - Include and, or, and not

Comparison Operators

- **Comparison operations** always return a Boolean (True or False) result that indicates whether some relationship holds between their operands.
  - Asks the question, “what is the relationship between these two things”

a>=b

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

a==b

Is **a** equal to **b**?

Comparison Operators(cont'd)

Operator	Description
==	If the values of two operands are equal, then the condition becomes true.
!=	If values of two operands are not equal, then condition becomes true.
<>	If values of two operands are not equal, then condition becomes true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.

<> Is outdated
Use != for "not equal to"

Comparison Operators (cont'd)

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Comparison Operators (cont'd)

- As previously mentioned, relational operators always return a Boolean response (true or false)

a=10
b=20

a>=b

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

Is **10** greater than or equal to **20**?

false

a==b

Is **a** equal to **b**?

Is **10** equal to **20**?

false

Common Pitfall with Comparison Operators

- We commonly use the assignment operator (=) in place of the relational operator (==)

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 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`

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

Other Comparison Considerations

- 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 Operators 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, False, 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`
`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>1` **`and`** `2>1`

  2. String them together, like in math:

     `x > y > z`

     evaluated: `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`

bool1 = a or b

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

Not

bool1 = not a

Value of a	Value of bool1
True	False
False	True

Not a returns the opposite boolean from a

Complex Expressions

- We can put multiple operators together!

  `bool1 = a and (b or c)`

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

Complex Expressions

We can combine these operators however we like!

bool1 = a and (b or c)

Value of a	Value of b	Value of c	Value of bool1
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

"Short Circuit" Evaluation

Short Circuit Evaluation

- “and” statements short circuit when the first expression evaluates to `False`
- “or” statements short circuit when the first 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 false won't bother with the rest of the expression.

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

Practice 2

- 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!
  - Due by this Thursday (Sept 17th) at 8:59:59 PM
- Homework 2 is out
  - Due by Tuesday (Sept 15th) at 8:59:59 PM
- Both of these assignments are on Blackboard
  - Weekly Agendas are also on Blackboard