

CMSC201

Computer Science I for Majors

Lecture 03 – Operators

Last Class We Covered

- Variables
 - Rules for naming
 - Different types
 - How to use them
- Printing output to the screen
- Getting input from the user
- Written programs vs Python interpreter

Any Questions from Last Time?

Today's Objectives

- To learn Python's operators
 - Arithmetic operators
 - Including mod and integer division
 - Assignment operators
 - Comparison operators
 - Boolean operators
- To understand the order of operations

Pop Quiz!

- Which of the following examples are correct?
 1. `500 = numStudents`
 2. `numStudents = 500`
 3. `numCookies * cookiePrice = total`
 4. `mpg = miles_driven / gallons_used`
 5. `"Hello World!" = message`
 6. `_CMSC201_doge_ = "Very learning"`
 7. `60 * hours = days * 24 * 60`

Pop Quiz!

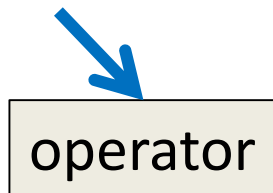
- Which of the following examples are correct?
 - x** 1. `500 = numStudents`
 - ✓** 2. `numStudents = 500`
 - x** 3. `numCookies * cookiePrice = total`
 - ✓** 4. `mpg = miles_driven / gallons_used`
 - x** 5. `"Hello World!" = message`
 - ✓** 6. `_CMSC201_doge_ = "Very learning"`
 - x** 7. `60 * hours = days * 24 * 60`

Python's Operators

Python Basic Operators

- ***Operators*** are the constructs which can manipulate and evaluate our data
- Consider the expression:

num = 4 + 5



Types of Operators in Python

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Membership Operators
- Bitwise Operators
- Identity Operators

focus of
today's lecture

Operators – Addition & Subtraction

- “Lowest” priority in the order of operations
 - Can only change this with parentheses
- Function as they normally do
- Examples:
 1. `cash = cash - bills`
 2. `(5 + 7) / 2`
 3. `(((2 + 4) * 5) / (9 - 6))`

Operators – Multiplication & Division

- Higher priority in the order of operations than addition and subtraction
- Function as they normally do
- Examples:
 1. `tax = subtotal * 0.06`
 2. `area = PI * (radius * radius)`
 3. `totalDays = hours / 24`

Operators – Integer Division

- Reminder: integers (or ints) are **whole numbers**
 - What do you think integer division is?
- Remember division in grade school?
- Integer division is
 - Division done without decimals
 - And the remainder is discarded

$$\begin{array}{r} \boxed{025} \text{ r } 3 \\ 5 \overline{) 128} \\ \underline{-0} \\ 12 \\ \underline{-10} \\ 28 \\ \underline{-25} \\ 3 \end{array}$$

Examples: Integer Division

- Integer division uses double slashes (//)

- Examples:

1. $7 / 5 = 1.4$

2. $7 // 5 = 1$

3. $2 / 8 = 0.25$

4. $2 // 8 = 0$

5. $4 // 17 // 5 = 0$



evaluate from left to right

Operators – Mod

- Also called “modulo” or “modulus”
- Example: $17 \% 5 = 2$
 - What do you think mod does?
- Remember division in grade school?
- Modulo gives you the remainder
 - The “opposite” of integer division

$$\begin{array}{r} 025 \text{ | } 3 \\ 5 \overline{) 128} \\ \underline{-0} \\ 12 \\ \underline{-10} \\ 28 \\ \underline{-25} \\ 3 \end{array}$$

Examples: Mod

- Mod uses the percent sign (%)

- Examples:

$$1. \quad 7 \quad \% \quad 5 \quad = \quad 2$$

$$2. \quad 5 \quad \% \quad 9 \quad = \quad 5$$

$$3. \quad 16 \quad \% \quad 6 \quad = \quad 4$$

$$4. \quad 23 \quad \% \quad 4 \quad = \quad 3$$

$$5. \quad 48692451673 \quad \% \quad 2 \quad = \quad 1$$

Modulo Answers

- Result of a modulo operation will always be:
 - Positive
 - No less than 0
 - No more than the divisor minus 1


- Examples:

1. $8 \% 3 = 2$

2. $21 \% 3 = 0$

3. $13 \% 3 = 1$

no more than the
divisor minus 1



no less than zero



Operators – Exponentiation

- “Exponentiation” is just another word for raising one number to the power of another
- Examples:
 1. `binary8 = 2 ** 8`
 2. `squareArea = length ** 2`
 3. `cubeVolume = length ** 3`
 4. `squareRoot = num ** 0.5`

Arithmetic Operators in Python

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Integer division
%	Modulo (remainder)
**	Exponentiation

Order of Operations (Arithmetic)

- Expressions are evaluated from left to right

Operator(s)	Priority
**	highest
* / // %	
+ -	lowest

- What can change this ordering?
 - Parentheses!

Floating Point Errors

Division: Floats and Integers

- Floats (decimals) and integers (whole numbers) behave in two different ways in Python
 - And in many other programming languages
- Biggest difference is how division works
 - Python 3 automatically performs decimal division
 - Have to explicitly call integer division
 - Floats also automatically perform decimal division

Division Examples

- What do the following expressions evaluate to?

1. $4 / 3 = 1.3333333333333333$

2. $4 // 3 = 1$

3. $8 / 3 = 2.6666666666666666$ **6667**

4. $8 / 2 = 4.0$

5. $5 / 7 = 0.714285714285$ **7143**

6. $5 // 7 = 0$

Assignment Operators

Basic Assignment

- All assignment operators
 - Contain a single equals sign
 - Must have a variable on the left side
- Examples:
 1. `numDogs = 18`
 2. `totalTax = income * taxBracket`
 3. `numPizzas = (people // 4) + 1`

Combining with Arithmetic

- You can simplify statements like these

```
count    = count + 1
```

```
doubling = doubling * 2
```

– By combining the arithmetic and assignment

```
count    += 1
```

```
doubling *= 2
```

- You can do this with any arithmetic operator

Combined Assignments

- These shortcuts assume that the variable is the first thing after the assignment operator

```
percent = int(input("Enter percent: "))  
# convert the percentage to a decimal  
percent /= 100
```

- The last line is the same as this line

```
percent = percent / 100
```

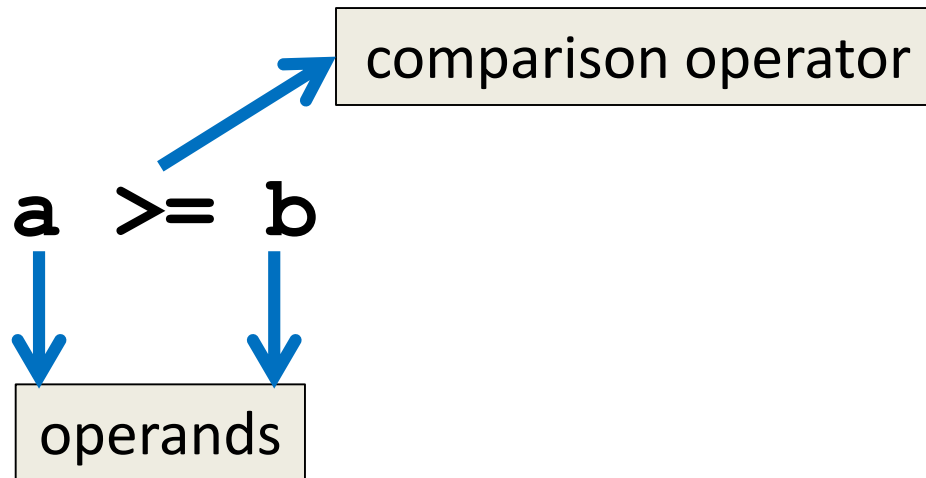
Comparison Operators

Vocabulary

- Comparison operators
- Relational operators
- Equality operators
 - Are all the same thing
- Include things like $>$, $>=$, $<$, $<=$, $==$, $!=$

Comparison Operators

- Always return a Boolean result
 - **True** or **False**
 - Indicates whether a relationship holds between their operands



Comparison Examples

- What are the following comparisons asking?

$a \geq b$

– Is a greater than or equal to b ?

$a == b$

– Is a equivalent to b ?

Comparison Operators in Python

Operator	Meaning
<	Less than (exclusive)
<=	Less than or equal to (inclusive)
>	Greater than (exclusive)
>=	Greater than or equal to (inclusive)
==	Equivalent to
!=	Not equivalent to

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** 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 will trigger an error in Python, but you may still make it on paper!

Equals vs Equivalence

- What does $\mathbf{a} = \mathbf{b}$ do?
 - Assigns \mathbf{a} the value stored in \mathbf{b}
 - Changes \mathbf{a} 's value to the value of \mathbf{b}
- What does $\mathbf{a} == \mathbf{b}$ do?
 - Checks if \mathbf{a} is equivalent to \mathbf{b}
 - Does not change the value of \mathbf{a} or \mathbf{b}

Evaluating to Boolean Values

Comparison Operators and Simple Data Types

- Examples:

`8 < 15` evaluates to **True**

`6 != 6` evaluates to **False**

`2.5 > 5.8` evaluates to **False**

`4.0 == 4` 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"` and `"False"` evaluate to **True**

Logical Operators

Vocabulary

- Logical operators
- Boolean operators
 - Are the same thing
- Include **and**, **or**, and **not**

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	False	
False	True	
False	False	

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

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

Practice with `and`

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

```
output:  
True True True
```

```
ex1 = a < b
```

```
ex2 = a < b and b < c
```

```
ex3 = (a + b == c) and (b - 10 == a) \  
and (c / 3 == a)
```

```
print (ex1, ex2, ex3)
```

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	

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

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

Logical Operators – `not`

- Let's evaluate this expression

`bool3 = not a`

Value of <code>a</code>	Value of <code>bool3</code>
True	
False	

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

Logical Operators – **not**

- Let's evaluate this expression

`bool3 = not a`

Value of <code>a</code>	Value of <code>bool3</code>
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!

```
bool14 = a and (b or c)
```

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

Practice with Comparisons

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

output:

```
False True True False
```

```
bool1 = True and (a > b)  
bool2 = (not True) or (b != c)  
bool3 = (True and (not False)) or (a > b)  
bool4 = (a % b == 2) and ((not True) or \n                        False)
```

```
print (bool1, bool2, bool3, bool4)
```

Order of Operations (All)

Operator(s)	Priority
**	highest
* / // %	
+ -	
< <= >	
>= != ==	
not	
and	
or	lowest

Daily emacs Shortcut

- **CTRL+K**
 - “Kill” from the cursor to the end of the line
 - Cuts the text (saves it to the “kill ring”)
 - Hit twice to get the “enter” at the end too
- **CTRL+Y**
 - “Yank” the killed text back from the dead
 - Pastes the text (from the “kill ring”)
 - Press multiple times to paste the text again

Announcements

- Your discussions start this week!
 - Go to your scheduled location and time

- HW 1 is out on Blackboard now
 - You must first complete the Syllabus and Course Website Quiz to see it
 - Due by Friday (Sept 15th) at 8:59:59 PM