Programming Foundations in Python
Adapted From: CMSC 201 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

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`

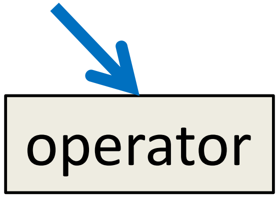
Python’s Operators

Python Basic Operators

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

- Consider the expression:

```
num = 4 + 5
```



operator

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
- Function as they normally do
- Examples:
  1. `cash = cash - bills`
  2. `(5 + 7) / 2`
  3. `( ((2 + 4) * 5) / (9 - 6) )`

Operators – Multiplication & Division

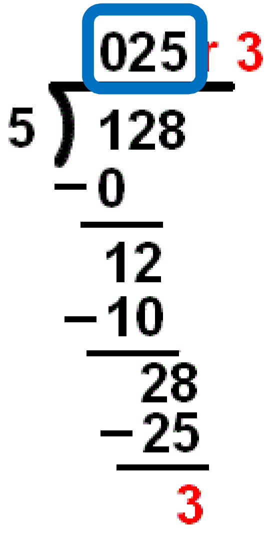
- Higher priority in the order of operations than addition and subtraction
- Function as they normally do

- 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



Examples: Integer Division

- Integer division uses double slashes (//)

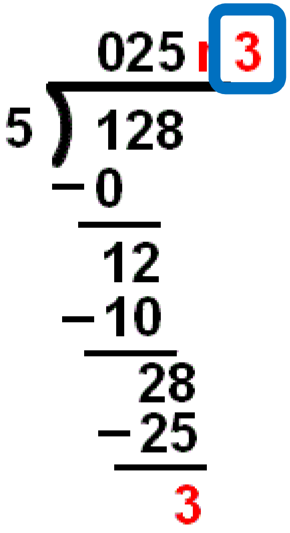
- Examples:

  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



Examples: Mod

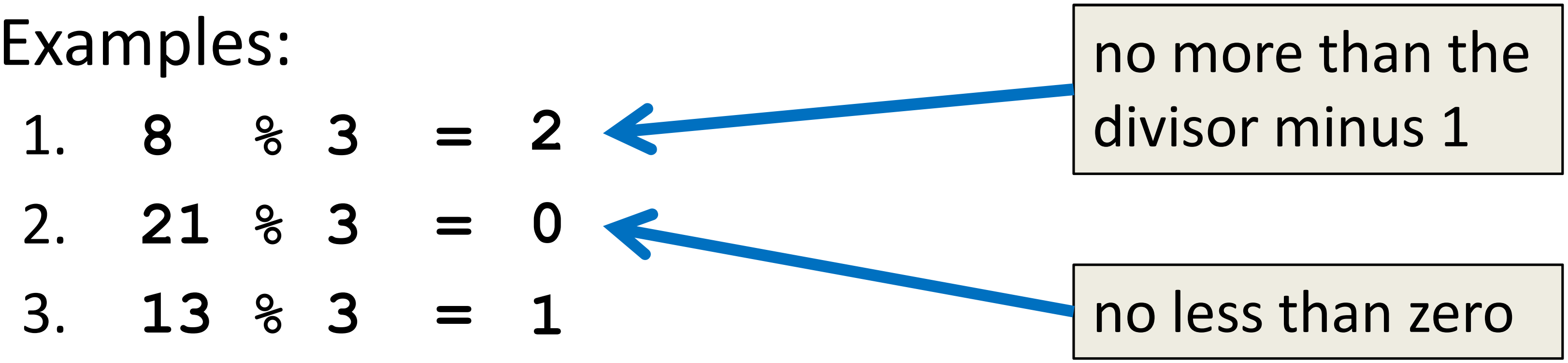
- Mod uses the percent sign (`%`)

- Examples:
  1. `7  % 5     = 2`
  2. `5  % 9     = 5`
  3. `16 % 6     = 4`
  4. `23 % 4     = 3`
  5. `48692451673 % 2 = 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 their division works
  - Python 3 automatically performs decimal division
    - For both integers and floats
  - Have to explicitly call integer division

Division Examples

- What do the following expressions evaluate to?

1. 4 / 3 = 1.3333333333333333
2. 4 // 3 = 1
3. 8 / 3 = $2.666666666666\boxed{6667}$
4. 8 / 2 = 4.0
5. 5 / 7 = $0.714285714285\boxed{7143}$
6. 5 // 7 = 0

Rounding Errors

- Sometimes we need to approximate the representation of numbers
  - 0.66666666666666666666666667…
  - 3.14159265358979323846264338328…
- We know that this can lead to incorrect answers when doing calculations later
  - Something similar happens when numbers are stored in a computer's memory

Float Arithmetic Examples

- What do the following expressions evaluate to?

```
1.  8 /   3         = 2.6666666666666667
2.  5 /   7         = 0.7142857142857143
3.  1.99 + 0.12 = 2.11
4.  0.99 + 0.12 = 1.1099999999999999
5.  1.13 * 1.19 = 1.3446999999999998
```

What's going on here???

Because computers store numbers differently, they sometimes run into different sets of rounding errors

Handling Floating Point Errors

- How to fix floating point errors?
  - You can't!

    ¯_(ツ)_/¯
  - They're present in every single programming language that uses the float data type

- Just be aware that the problem exists
  - Don't rely on having exact numerical representations when using floats in Python

Assignment Operators

Basic Assignment

- All assignment operators
  - Contain a single equal 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

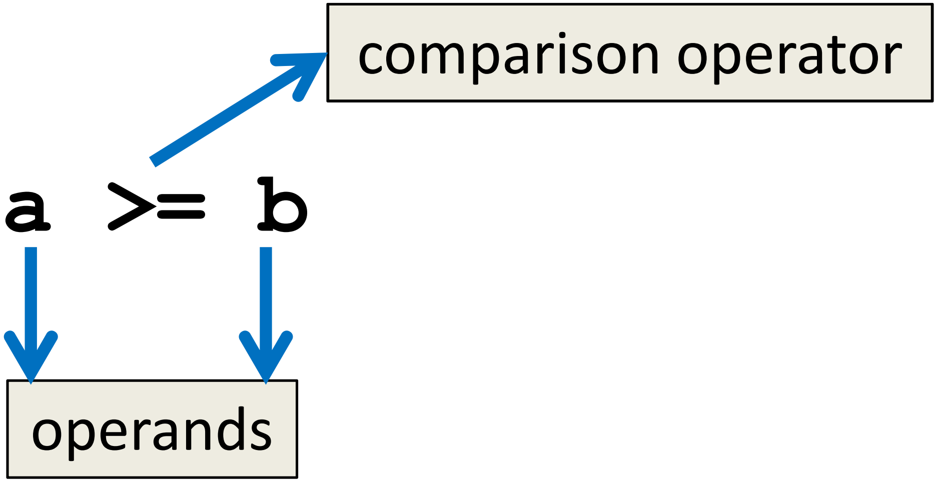
Overview

- 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 operator

`a >= b`

operands

Comparison Examples

- What are the following comparisons asking?

  `a >= 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 <= b`

  - Is **`a`** less than or equal to **`b`**?
  - Is **`10`** less than or equal to **`20`**?
  - **`True`**

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 `a = b` do?
  - Assigns **a** the value stored in **b**
  - Changes **a**'s value to the value of **b**

- What does `a == b` do?
  - Checks if **a** is equivalent to **b**
  - Does <u>not</u> change the value of **a** or **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 use

  `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

Logical Operators

- Sometimes also called Boolean operators
- There are three logical operators:
  - `and`
  - `or`
  - `not`
- They let us build 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

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)
```

output:

```
True True 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`	`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 a	Value of `bool3`
`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 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)`
  - 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 False)

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

Order of Operations (All)

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