# CMSC201
# Computer Science I for Majors

## Lecture 04 – Expressions

Prof. Katherine Gibson

Prof. Jeremy Dixon

Based on slides by Shawn Lupoli and Max Morawski at UMBC

# Last Class We Covered

- Variables
  - Rules for naming
  - Different types
  - How to use them
- Printing output to the screen
- Getting input from the user
  - Mad Libs

# Any Questions from Last Time?

# Today’s Objectives

- 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

# Expressions

- Expressions are code that produces or calculates new data and data values
- Allow us to program interesting things
- Always on the **right hand side** of the assignment operator

# 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`

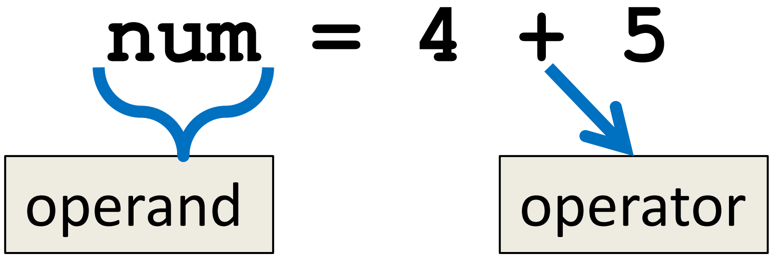
✓ 6. `_CMSC201_doge_ = "Very learning"`

✗ 7. `60 * hours = days * 24 * 60`

# Python’s Operators

# Python Basic Operators

- *Operators* are the constructs which can manipulate the value of *operands*
- Consider the expression:



```
num = 4 + 5
```

operand

operator

- Here, **`num`** is the operand and **+** is the operator

# Types of Operators in Python

- Arithmetic Operators

focus of today's lecture

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

# Operators in Python

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

# 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:

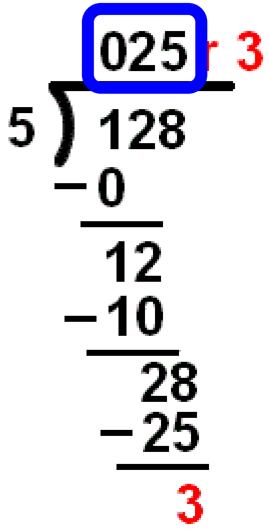
1. `tax = subtotal * 0.06`
2. `area = PI * (radius * radius)`
3. `totalDays = hours / 24`

# 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



# Examples: Integer Division

- Integer division uses double slashes (//)

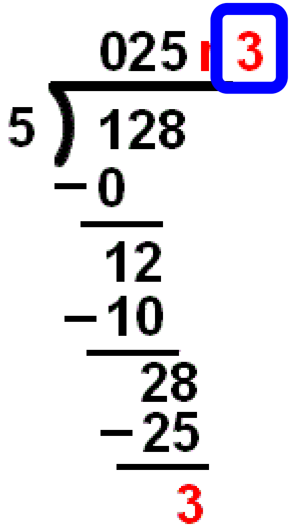
- Examples:

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 – Modulo

- Also called “modulo,” “modulus,” or “mod”
- 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

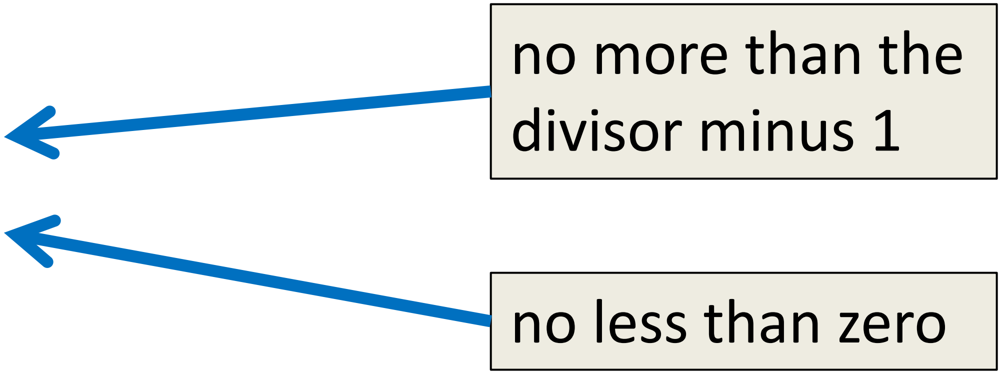
- Mod uses the percent sign (`%`)

- Examples:
  1. `7  % 5      = 2`
  2. `5  % 9      = 5`
  3. `17 % 6      = 5`
  4. `22 % 4      = 2`
  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)`

# Order of Operations

- Expressions are evaluated in what direction?

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

- What can change this ordering?
  - Parentheses!

# Types in Python

# Variable Types

- There are many different kinds of variables!
  - Numbers
    - Whole numbers (Integers)
    - Decimals (Floats)
  - Booleans (**`True`** and **`False`**)
  - Strings (collections of characters)

# Finding a Variable’s Type

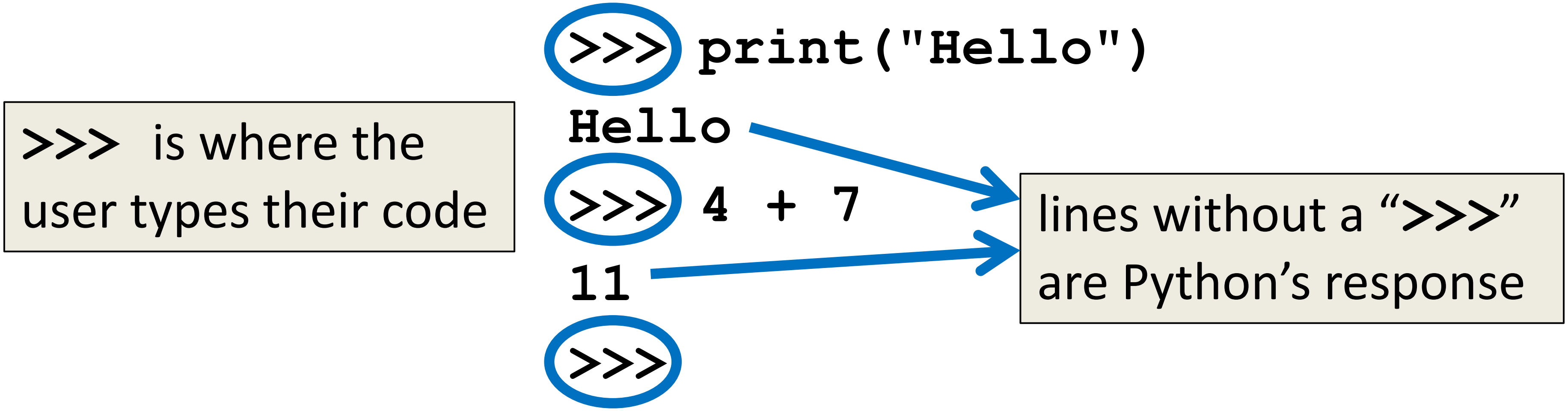
- To find what type a variable is, use **`type()`**

- Example:

```
>>> a = 3.0                 >>> b = "moo"
>>> type(a)                 >>> type(b)
<class 'float'>             <class 'str'>
```

# Quick Note: Python Interpreter

- Sometimes in class and the slides, you’ll see use of Python’s “interactive” interpreter
  - Evaluates each line of code as it’s typed in



```
>>> print("Hello")
Hello
>>> 4 + 7
11
>>>
```

>>> is where the user types their code

lines without a “>>>” are Python’s response

# Division: Floats and Integers

- Floats (decimals) and integers (whole numbers) behave very differently in Python
  - And in many other programming languages
- Biggest difference is with 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. `4 // 3.0 = 1.0`
4. `8 /  3   = 2.6666666666666667`
5. `8 /  2   = 4.0`
6. `5 /  7   = 0.7142857142857143`
7. `5 // 7   = 0`

# Floating Point Errors

- In base 10, some numbers are approximated:
  - 0.66666666666666666666666667…
  - 3.14159265358979323846264338328…
- The same is true for base 2
  - 0.00011001100110011001100… (0.1 in base 10)
- This leads to rounding errors with floats
  - **General rule**: Don’t compare floats for equality after you’ve done division on them!

# Casting to a Type

- We can change a variable from one type to another using casting

- Example:

```
>>> e = 2.718
>>> int(e)
2
>>> str(e)
'2.718'
```

type you want to cast to, then the variable to cast
*"change e to an integer"*

# Casting to a Type: Assignment

- Casting alone doesn’t change a variable’s type

```
>>> courseNum = "201"
>>> int(courseNum)
201
>>> type(courseNum)
<class 'str'>
```

cast courseNum as an int

type is still a string (!?)

- To make an actual change, you need to “save” it with the assignment operator

# Casting to a Type: Assignment

- Use the assignment operator (=) to actually change the variable’s type

```
>>> courseNum = "201"
>>> type(courseNum)
<class 'str'>
>>> courseNum = int(courseNum)
>>> type(courseNum)
<class 'int'>
```

this is what actually causes the variable’s type to change

# Constants

# What are Constants?

- Constants are values that are **not** generated by the user or by the code
  - But are used a great deal in the program

- Constants should be ALL CAPS with a “_” (underscore) to separate the words
  - Coding standards

# Using Constants

- Calculating the total for a shopping order

```
MD_TAX    = 0.06
```

easy to update if tax rate changes

```
subtotal = input("Enter subtotal:")
tax    = subtotal * MD_TAX
total = tax + subtotal
print("Your total is:", total)
```

we know exactly what this number is for

# “Magic” Numbers

- “Magic” numbers are numbers used directly in the code – should be replaced with constants

- Examples:
  - Mathematical numbers (pi, e, etc.)
  - Program properties (window size, min and max)
  - Important values (tax rate, maximum number of students, credits required to graduate, etc.)

# "Magic" Numbers Example

- You're looking at the code for a virtual casino
  - You see the number 21
  - What does it mean?

```
if (value < 21)
```
✘

- Blackjack? Drinking age? VIP room numbers?

```
if (customerAge < DRINKING_AGE)
```
✓

- Constants make it easy to update values – why?
  - Don't have to figure out which "21"s to change

# “Magic” Everything

- Can also have “magic” characters or strings
  - Use constants to prevent <u>any</u> “magic” values
- For example, a blackjack program that uses the chars “**H**” for hit, and “**S**” for stay

```
if (userChoice == "H"):
```

✗

```
if (userChoice == HIT):
```

✓

  - Which of these options is easier to understand?
  - Which is easier to update if needed?

# Are Constants Really Constant?

- In some languages (like C, C++, and Java), you can create variables that CANNOT be changed
- This is not possible with Python variables
  - Part of why coding standards are so important
  - If you see code that changes the value of a variable called **`MAX_ENROLL`**, you know that's a constant, and *shouldn't* be changed

# Quick Note: Version of Python

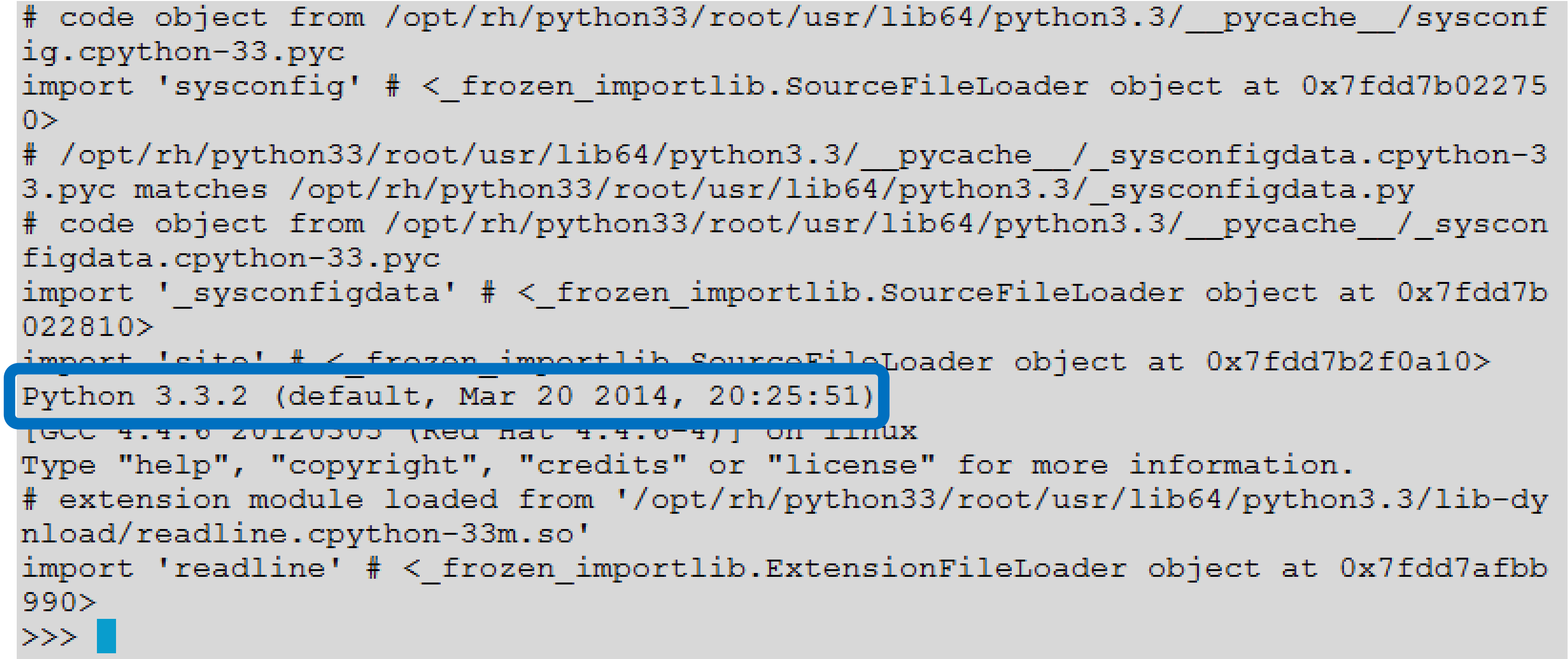
- Before you run any Python code, you need to tell GL you want to use Python 3 instead:

```
scl enable python33 bash
```

- You can double-check which version is running with the command **`python –v`**
  - It will print out a bunch of text, but near the bottom you should see "**`Python 3.3.2`**"

# Version of Python

- After typing “**`python -v`**”



```
# code object from /opt/rh/python33/root/usr/lib64/python3.3/__pycache__/sysconf
ig.cpython-33.pyc
import 'sysconfig' # <_frozen_importlib.SourceFileLoader object at 0x7fdd7b02275
0>
# /opt/rh/python33/root/usr/lib64/python3.3/__pycache__/_sysconfigdata.cpython-3
3.pyc matches /opt/rh/python33/root/usr/lib64/python3.3/_sysconfigdata.py
# code object from /opt/rh/python33/root/usr/lib64/python3.3/__pycache__/_syscon
figdata.cpython-33.pyc
import '_sysconfigdata' # <_frozen_importlib.SourceFileLoader object at 0x7fdd7b
022810>
import 'site' # <_frozen_importlib.SourceFileLoader object at 0x7fdd7b2f0a10>
Python 3.3.2 (default, Mar 20 2014, 20:25:51)
[GCC 4.4.6 20120305 (Red Hat 4.4.6-4)] on linux
Type "help", "copyright", "credits" or "license" for more information.
# extension module loaded from '/opt/rh/python33/root/usr/lib64/python3.3/lib-dy
nload/readline.cpython-33m.so'
import 'readline' # <_frozen_importlib.ExtensionFileLoader object at 0x7fdd7afbb
990>
>>> 
```

# Announcements

- Your Lab 1 is happening this week!
  – First graded lab; attend your assigned section

- Homework 2 will be out Monday night
  – Due by Monday (Feb 15th) at 8:59:59 PM

- Both of these assignments are on Blackboard
  – Complete Academic Integrity Quiz to see HW2

# Practice Problems

- Write a program that gets a price from the user, and uses arithmetic operators to calculate the dollars and pennies (*e.g.*, 7.55 = $7, 55¢)
  - Update the program to check if the value is negative, and print out an error message if it is
- Explain why you would use constants in a program. Give an illustrative example.
- Write a program that calculates the volume of a cylinder. (Try to write it using exponentiation!)