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. $500 = \text{numStudents}$
2. $\text{numStudents} = 500$
3. $\text{numCookies} \times \text{cookiePrice} = \text{total}$
4. $\text{mpg} = \frac{\text{miles\_driven}}{\text{gallons\_used}}$
5. "Hello World!" = message
6. `_CMSC201_doge_` = "Very learning"
7. $60 \times \text{hours} = \text{days} \times 24 \times 60$
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:

\[ \text{num} = 4 + 5 \]
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
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. \( \text{tax} = \text{subtotal} \times 0.06 \)
  2. \( \text{area} = \pi \times (\text{radius} \times \text{radius}) \)
  3. \( \text{totalDays} = \text{hours} / 24 \)
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. \(7 \div 5 = 1.4\)
2. \(7 \div 5 = 1\)
3. \(2 \div 8 = 0.25\)
4. \(2 \div 8 = 0\)
5. \(4 \div 17 \div 5 = 0\)

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. $17 \% 6 = 5$
  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)
## Operators in Python

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>//</td>
<td>Integer division</td>
</tr>
<tr>
<td>%</td>
<td>Modulo (remainder)</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
</tbody>
</table>
Order of Operations

• Expressions are evaluated in what direction?

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>highest</td>
</tr>
<tr>
<td>* / // %</td>
<td></td>
</tr>
<tr>
<td>+ −</td>
<td>lowest</td>
</tr>
</tbody>
</table>

• 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:

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

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

– To use the interpreter, enable Python 3, then type “python” into the command line
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. \( 4 \div 3 = 1 \)
3. \( 4 \div 3.0 = 1.0 \)
4. \( 8 / 3 = 2.6666666666666667 \)
5. \( 8 / 2 = 4.0 \)
6. \( 5 / 7 = 0.7142857142857143 \)
7. \( 5 \div 7 = 0 \)
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 value from one type to another using something called **casting**

• Example:

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

The type you want to cast to, then the variable whose value you want to cast

This code means: 
“show what e is as an integer”
Casting to a Type: Assignment

• Casting alone doesn’t change the variable’s type

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

- cast courseNum’s value to an integer
- 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

```python
>>> 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
  – This follows CMSC 201 Coding Standards
Using Constants

• Calculating the total for a shopping order

```python
MD_TAX = 0.06

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

- easy to update if tax rate changes
- we know exactly what this number is
“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?

• 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 any “magic” values

• For example, a blackjack program that uses the strings “H” for hit, and “S” for stay

```python
if userChoice == "H":  ✗

if userChoice == HIT: ✓
```

– Which of these options is easier to understand?
– Which is easier to update if it’s 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”
Announcements

• Your Lab 2 is happening this week!
  – Attend your assigned section

• Homework 2 will be out Wednesday night
  – Due by Wednesday (Sep 21st) at 8:59:59 PM

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