Objectives

- Continuing fundamentals of programming in Python
- Software development practices
  - Testing
  - Debugging
  - Iteration
- Numeric Operations

Review

- What are some data types and what do they represent?

  - CS Issues:
    - Technology literacy is important
    - Big questions:
      - What technical skills should be taught?
      - Are the teachers skilled to teach these classes?
      - How to assess the skills?
    - Broader issues too (costs!)

Recap of Programming Fundamentals

- Most important data types (for us, for now):
  - `int`, `float`, `str`, `bool`
  - Use these types to represent various information
- Variables have identifiers, (implicit) types
  - Should have “good” names
  - Names: start with lowercase letter; can have numbers, underscores
- Assignments
  - `x = y` means “x gets value y” or “x is assigned value of y”
  - Only LHS of statement changes

Review: Assignment statements

- Assignment statements are NOT math equations!
  - `count = count + 1`
- These are commands!
  - `x = 2`
  - `y = x`
  - `x = x + 3`
  - What’s the value of `y`?

Review: What are the values?

- After executing the following statements, what are the values of each variable?
  - `a = 5`
  - `y = a + -1 * a`
  - `z = a + y / 2`
  - `a = a + 3`
  - `y = (7+x)*z`
  - `x = z*2`

  Runtime error: x doesn’t have a value yet
  --- not initialized, can’t use on RHS

Printing Output

- `print` is a special command
  - Displays the result of expression(s) to the terminal
  - `print “Hello, class”`
    - "string literal"
    - Print automatically adds a ‘\n’ (carriage return) after it’s printed
- `print “Your answer is”, 4*4`
  - Displays same as:
    - `print “Your answer is”, print 4*4`
    - Syntax: commas
    - Semantics: print multiple “things” in one line
Getting Input From User

- **input** and **raw_input** are functions
  - Function: A command to do something
  - Prompts user for input, gets the user's input
  - **input**: to read in numbers
  - **raw_input**: to read in strings/text
- **Syntax:**
  - `input(<string_prompt>)`
  - `raw_input(<string_prompt>)`

Getting Input From User

- Typically used in assignments
- **Examples:**
  - `width = input("Enter the width: ")`
    - Prompt displayed to user
    - **width** is assigned the number the user enters
      - Use **input** because expect a number from user
  - `name = raw_input("What is your name? ")`
    - **name** is assigned the string the user enters
      - Use **raw_input** because expect a string from user

Documenting Your Code

- Use English to describe what your program is doing in **comments**
  - Everything after a `#` is a comment
    - Color-coded in IDLE, jEdit
    - Python does not execute comments
  - Does not affect the correctness of your program
  - Improves program's readability
    - Easier for someone else to read and update your code

```python
# Demonstrate numeric and string input
# by Sara Sprenkle for CS111 on 9/12/07
#
color = raw_input("What is your favorite color? ")
print "Cool!  My favorite color is _light_ blue!"

scale = input("On a scale of 1 to 10, how much do you like Matt Damon? ")
print "Cool!  I like him ", scale*1.8, " much!"
```

When to Use Comments

- Document the author, high-level description of the program at the top of the program
- Provide an outline of an algorithm
  - Separates the steps of the algorithm
- Describe difficult-to-understand code

Identify the Pieces of a Program

```python
# input_demo.py
color = raw_input("What is your favorite color? ")
print "Cool!  My favorite color is _light_ blue!"

scale = input("On a scale of 1 to 10, how much do you like Matt Damon? ")
print "Cool!  I like him ", scale*1.8, " much!"
```
# Demonstrate numeric and string input
# by Sara Sprenkle for CS111 on 9/12/07
#
color = raw_input( "What is your favorite color? " )
print "Cool! My favorite color is " + color + "!

scale = int(input("On a scale of 1 to 10, how much do you like Matt Damon? "))
print "Cool! I like him", scale * 1.8, "much!"

Identify the comments, variables, functions, expressions, assignments, literals

---

## Identify the Pieces of a Program

### Putting it all together

- Find the area of a rectangle (which has a width and height)
  - What is the algorithm for this problem?

### Errors/ Bugs

- Sometimes the program doesn’t work
- Types of programming errors:
  - Syntax error
    - Interpreter shows where the problem is
  - Logic/semantic error
    - answer = 2+3
    - answer should be 2*3
  - Exceptions/Runtime errors
    - answer = 2/0
    - Undefined variable name
- Expose errors when Testing

### Testing Process

- Need good test cases to help determine if program is correct
  - Tester plays devil’s advocate
  - Want to expose all bugs!
  - Find before customer/professor!

---

## Testing Process

- Test case: input used to test the program, **expected output** given that input
- Verify if **output** is what you expected
Practice: Test Cases
• Test cases for finding the area of a rectangle
  ➢ Test both integers
  ➢ Test with at least one float for width, height
  ➢ Test numbers less than or equal to 0
    • Shouldn’t compute area for those

Debugging
• Edit the program, re-execute/test until everything works
• The error is often called a “bug”
• Diagnosing and fixing it is called debugging

Good Development Practices
• Design the algorithm
  ➢ Break into pieces
• Implement and Test each piece separately
  ➢ Identify the best pieces to make progress
  ➢ Iterate over each step to improve it
• Write comments FIRST for each step
  ➢ Elaborate on what you’re doing in comments when necessary

Review: Arithmetic Operations
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>Left</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Left</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Left</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>Left</td>
</tr>
<tr>
<td>%</td>
<td>Remainder (“mod”)</td>
<td>Left</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation (power)</td>
<td>Right</td>
</tr>
</tbody>
</table>

NOT Math Class
• Need to write out all operations explicitly
  ➢ In math class, a \((b+1)\) meant \(a \cdot (b+1)\)

Precedence rules: P E DM% AS

Associativity matters when you have the same operation multiple times
Math Practice

- How should we verify our answers?

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 + 3 * 2</td>
<td>11</td>
</tr>
<tr>
<td>2 * 3 ** 2</td>
<td>18</td>
</tr>
<tr>
<td>-3 ** 2</td>
<td>-1</td>
</tr>
<tr>
<td>2 ** 3 ** 3</td>
<td>1024</td>
</tr>
</tbody>
</table>

Two Types of Division

- Float Division: Result is a `float`
  - `3.0 / 6.0` --> 0.5
  - `6.0 / 3.0` --> 2.0
  - **At least** one of numerator and denominator must have a decimal, i.e., have type `float`
- Integer Division: Result is an `int`
  - `3 / 6` --> 0
  - `6 / 3` --> 2
  - `x / y`, if both x and y are `ints`
  - If both numerator and denominator are `ints`, result is `int`

Division Practice (NOT Math class)

- What is the result? What is the type of the variable?

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x = 6 / 4</code></td>
<td>1.5</td>
<td>float</td>
</tr>
<tr>
<td><code>y = 4 / 6 * 5.0</code></td>
<td>3.3333</td>
<td>float</td>
</tr>
<tr>
<td><code>a = 6 / 12.0</code></td>
<td>0.5</td>
<td>float</td>
</tr>
<tr>
<td><code>b = 6.0 / 12</code></td>
<td>0.5</td>
<td>float</td>
</tr>
<tr>
<td><code>z = .3</code></td>
<td>0.3</td>
<td>float</td>
</tr>
<tr>
<td><code>z = x / y</code></td>
<td>Not always obvious</td>
<td></td>
</tr>
<tr>
<td><code>z = x / 3</code></td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

This Week

- **Tuesday: Lab 1**
  - Due Friday
- For Friday, read Four Puzzles from Cyberspace