Objectives

- More arithmetic operations
- Getting user input
- Updated software development practices
  - Testing with user input

Review

1. What is our development process?
   - For programming, in general
   - For lab work
2. What are the two division operators?
3. How should you "read" the following expression? What does it mean?
   - `rem = num1 % num2`
   - Complete worksheet started last time

Formalizing Process of Developing Computational Solutions

1. Create a sketch of how to solve the problem (the algorithm)
2. Fill in the details in Pytho

Not necessarily complete program at first

3. Execute the program
4. If output doesn't match your expectation
   - Debug the program (Where is the problem? How do I fix it?)

Our development process will evolve over time

Development Process for Lab

- Develop in **IDLE**

1. Create a new file
2. Develop the program (following previous slide)
3. Close the shell
4. Run the program again
5. Save output from program

Review: Lab Expectations

- Comments in programs
  - High-level comments, author
  - Notes for your algorithms, implementation
- Nice, readable, clearly labeled understandable output
  - User running your program needs to **understand** what the program is saying

Other Lab Notes

- Trying to set you up for success now
  - Develop good development habits
  - You know the expectations and how you should develop as programs get larger, more complex
- I won't check your labs before every submission
- Learning how to solve problems
  - Every week: new problems, new techniques to solve problems
- I am explicit in directions/reminders early
  - Then stop reminding because you should know the process later
- Labs are due on Friday; review before the next lab

Modulo Practice

1. `7 % 2`
2. `3 % 6`
3. `6 % 2`
4. `7 % 14`
5. `14 % 7`
6. `6 % 0`

Brainstorm

- What useful thing does % 10 do?
  - 3 % 10 =
  - 51 % 10 =
  - 40 % 10 =
  - 678 % 10 =
  - 12543 % 10 =
- What useful thing does // 10 do (integer division)?
  - 3 // 10 =
  - 51 // 10 =
  - 40 // 10=
  - 678 // 10 =
  - 12543 // 10 =
- What useful thing does % 2 do?

Trick: Type Conversion

- You can convert a variable's type
  - Use the type's ***constructor***

Conversion Function/Constructor	Example	Value Returned
`int(<number or string>)`	`int(3.77)` `int("33")`	3 33
`float(<number or string>)`	`float(22)`	22.0
`str(<any value>)`	`str(99)`	"99"

Trick: Arithmetic Shorthands

- Called **extended assignment operators**
- Increment Operator
  - `x = x + 1` can be written as `x += 1`
- Decrement Operator
  - `x = x - 1` can be written as `x -= 1`
- Shorthands are similar for `*, /, //` :
  - `amount *= 1.055`
  - `x //= 2`

Parts of an Algorithm

- **Input**, Output
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

Interactive Programs

2.8 in Text Book

- Meaningful programs often need input from users

- Demo: `input_demo.py`

Getting Input From User

- `input` is a ***function***
  - **Function**: A command to do something
    - A "subroutine"
- Syntax:
  - `input(<string_prompt>)`
- Semantics:
  - Display the prompt `<string_prompt>` in the terminal

Getting Input From User

- Typically used in assignments
- Examples:
  - `name=input("What is your name? ")`

    Prompt displayed to user
    - **`name`** is assigned the string the user enters
  - `width=eval(input("Enter the width: "))`
    - What the user enters is *evaluated* (as a number) and assigned to `width`
    - Use **`eval`** function because expect a number from user
    - Alternatively, could use **`int`** or **`float`** (conversion functions) instead of **`eval`**

Getting Input from User

```
color = input("What is your favorite color? ")
```

Semantics: Sets the variable `color` to the user's input

Terminal:

Grabs every character up to the user presses "enter"

```
> python3 input_demo.py
What is your favorite color? blue
Cool!  My favorite color is _light_ blue !
```

`input_demo.py`

Reverse Engineering

Terminal:

```
> python3 input_demo.py
What is your favorite color? blue
Cool!  My favorite color is _light_ blue !
```

- Think about what was displayed
- What code was written to make that happen?
  - Typically, we hear "display", we think "print statement"
  - But, that's not what was used here because we were displaying a *prompt*

`input_demo.py`

What Happens If …?

Program:

```
str_age = input("Enter your age: ")
age = int(str_age)
```

Executing:

```
Enter your age: twelve
```

User enters a *string* but you were expecting an *integer!*

What Happens If …?

Program:

```
str_age = input("Enter your age: ")
age = int(str_age)
```

Executing:

```
Enter your age: twelve
Traceback (most recent call last):
  File "/Users/sprenkles/Library/CloudStorage/Box-Box/
CSCI111/prep/int_input.py", line 5, in <module>
    age = int(str_age)
ValueError: invalid literal for int() with base 10: 'twelve'
```

Restricting User's Inputs

```
>>> x = 7
>>> yourVal = input("My val is: ")
My val is: x
>>> print(yourVal)
x
```

Restricting User's Inputs

```
>>> x = 7
>>> yourVal = input("My val is: ")
My val is: x
>>> print(yourVal)
x
>>> yourVal = eval(input("My val is: "))
My val is: x
>>> print(yourVal)
7
>>> yourVal = int(input("My val is: "))
My val is: x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'x'
```

What happened here?

Restricting User's Inputs

```
>>> x = 7
>>> yourVal = input("My val is: ")
My val is: x
>>> print(yourVal)
x
>>> yourVal = eval(input("My val is: "))
My val is: x
>>> print(yourVal)
7
>>> yourVal = int(input("My val is: "))
My val is: x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'x'
```

What happened here?

Summary of Input

- Use the `input` function to get input from the user
- Typical use: save the result of the `input` function in a variable
- The `input` function returns a string
  - If you want a number, wrap the input in the `int`, `float`, or `eval` function

Identify the Parts of a Program

```
# Demonstrate numeric and string input
# by Sara Sprenkle for CS111
#

color = input("What is your favorite color? " )
print("Cool!  My favorite color is _light_", color, "!")

rating = eval( input("On a scale of 1 to 10, how much do
you like Zendaya? ") )
print("Cool!  I like her", rating*1.8, "much!")
```

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

input_demo.py

Identify the Parts of a Program

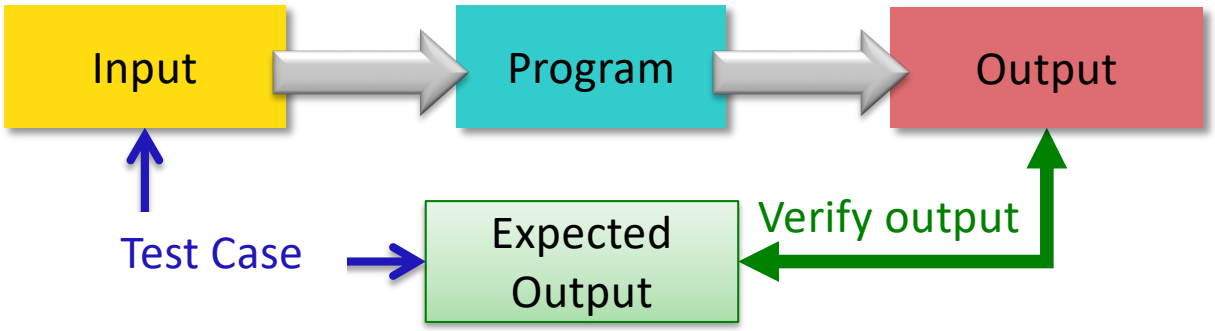
```
# Demonstrate numeric and string input
# by Sara Sprenkle for CS111
#

color = input("What is your favorite color? " )
print("Cool!  My favorite color is _light_", color, "!")

rating = eval( input("On a scale of 1 to 10, how much do
you like Zendaya? ") )
print("Cool!  I like her", rating*1.8, "much!")
```

expression

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

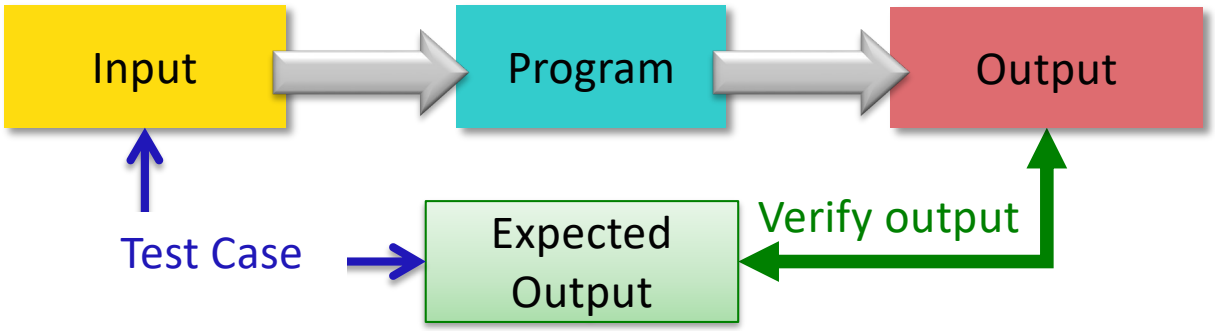
REFINING OUR DEVELOPMENT PROCESS

Testing Process



- Test case:
  - Input used to test the program
  - Expected output given that input
- Verify if output is what you expected

Testing Process

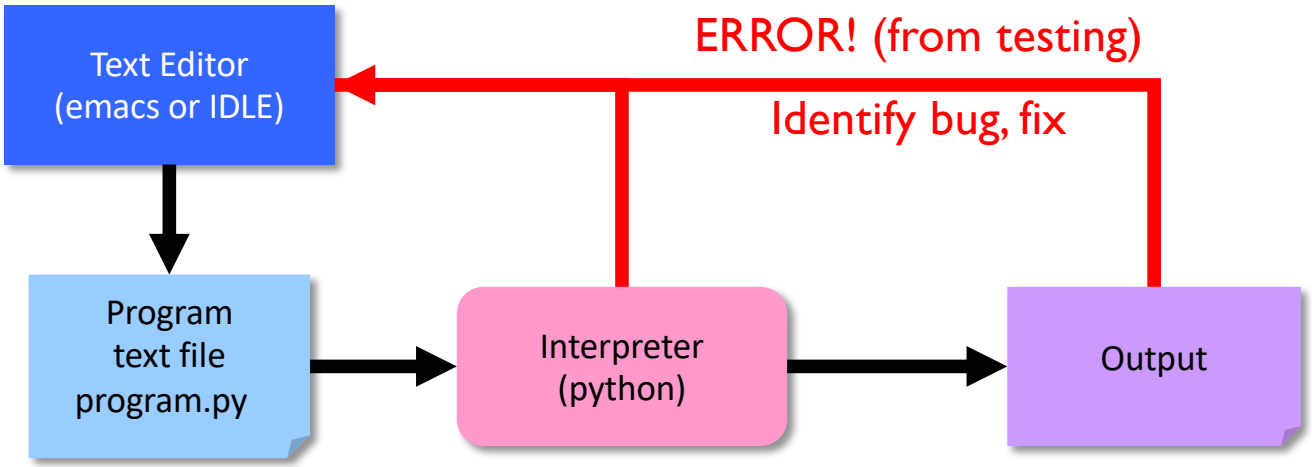


- Test case:
  - Input used to test the program
  - Expected output given that input
- Verify if output is what you expected
- Goal: create *good* test cases that will reveal if there is a problem in your code

If output is not what you expect, we say that the program failed the test case.

Review: Debugging

- After executing program and output did not match what you expected
- Identify the problems in your code
  - Edit the program to fix the problem
  - Re-execute/test until all test cases pass
- The error is called a "bug" or a "fault"
- Diagnosing and fixing error is called ***debugging***



Practice: A Computational Algorithm

- Problem: Find the average of two numbers
- Process:
  1. Consider good test cases for the problem
     - Start thinking about expectations: “When user enters these inputs, this should be displayed.”
  2. Create a sketch of how to solve the problem (the algorithm)
  3. Fill in the details in Python

Practice: Development Process

- Problem: Find the average of two numbers
- Test Cases

Input		
num1	num2	**Expected Output**

Good Test Cases for Finding the Average

- Test both integers
- Test with at least one float
- Test numbers less than or equal to 0

Practice: Develop Algorithm

- Problem: Find the average of two numbers

Looking Ahead

- Pre Lab due Tuesday before lab
- Broader Issue: Algorithm Bias