

Objectives

- Review
- Lab 1
 - Linux practice
 - Programming practice
 - Numeric operations
 - Getting input from the user

Advice from Previous Students

- **Push yourself** on labs. They are a great **opportunity** to try out the **concepts** you're learning.
- Get help from the **student assistants** during lab.
- Focus on **key concepts** and don't fret the details. Make sure you **understand** what's going on **before moving on**.
- Everything in this class **builds** on everything before it.
- Don't settle on a program just because it runs. You can probably still **make it better**.
- Think about how the course material **applies** to your **interests**. Try to see how the work we're doing is similar to how CS is used in the "real world"

Review

- What are the two different types of division?
- What is the two-part verification process we need to do after we implement a program?

Review: Arithmetic Operations


Symbol	Meaning	Associativity
+	Addition	Left
-	Subtraction	Left
*	Multiplication	Left
/	Division	Left
%	Remainder ("mod")	Left
**	Exponentiation (power)	Right

Precedence rules: P E - DM% AS

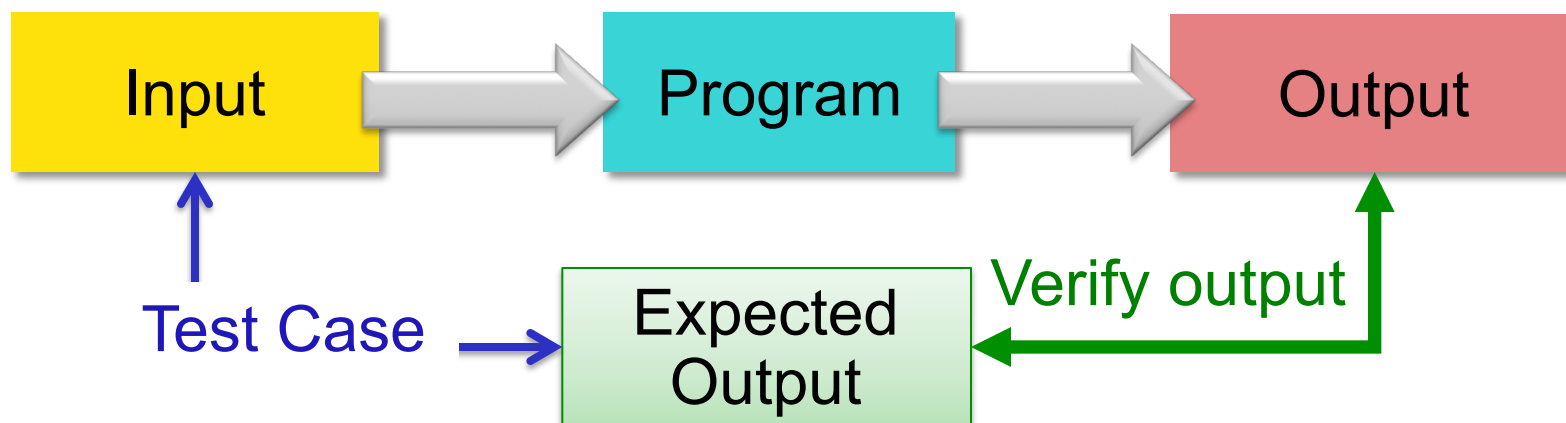
↑
negation

Associativity matters when
you have the same
operation multiple times

Review: Two Types of Division

- Float Division: Result is a **float**
 - **At least** one of numerator and denominator must have a decimal, i.e., have type **float**
 - $3.0/6.0 \rightarrow 0.5$
 - $6.0/3.0 \rightarrow 2.0$
- Integer Division: Result is an **int**
 - x/y , if both x and y are **ints**  Not always obvious
 - If both numerator and denominator are **ints**, result is **int**
 - $3/6 \rightarrow 0$
 - $6/3 \rightarrow 2$

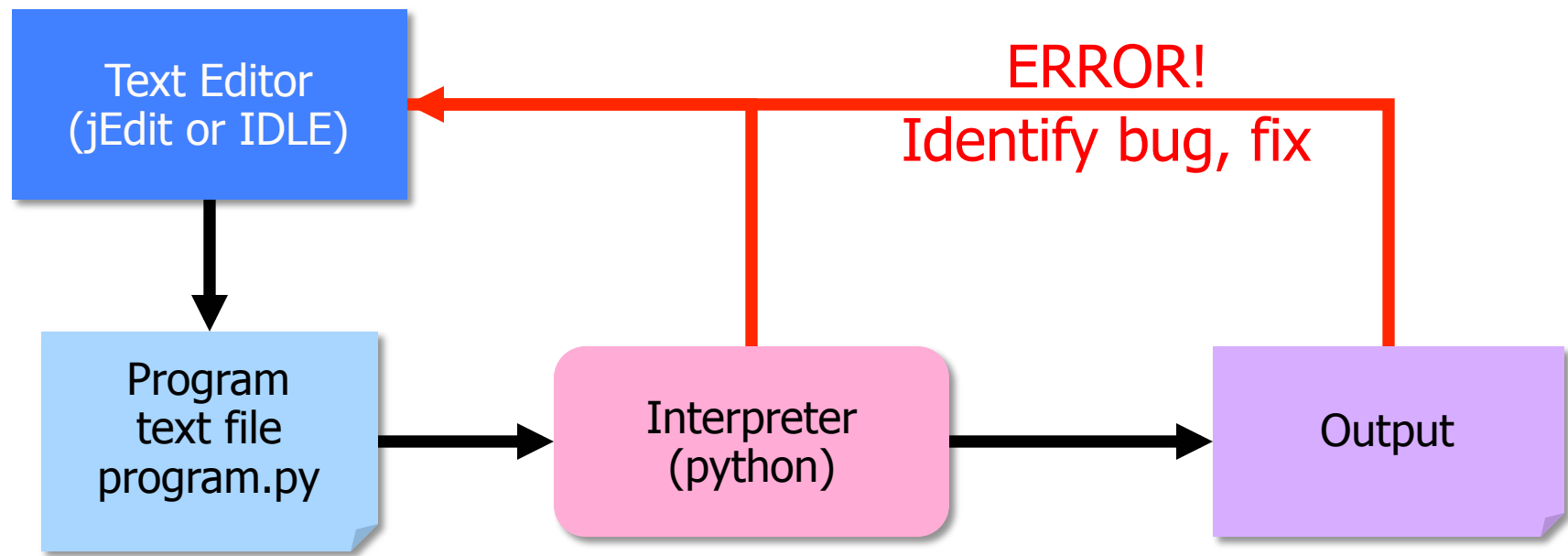
Review: Testing Process



- Test case: **input** used to test the program, **expected output** given that input
- Verify if **output** is what you expected
- Need good test cases
 - Good that you know the “problematic” test cases, even if we don’t know how to address them yet

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

General Announcements

- CS Issues Grading/Expectations
 - 6 pts for blog entry
 - Common issue – missing answers to one of questions
 - 4 pts for participation in class
- Example programs posted for each day on course web site

Lab 0 Feedback

- Overall, did well
 - Often lost points because missed some directions
 - E.g., broken Web page links, documentation in programs, output of programs
 - Generally, lab grades should be high
- Interesting article links
 - Consider reviewing for extra credit
- Missed Sakai extra credit

Lab 0 Feedback


- `ls -l` option
 - Demonstrate how different from `-1` option
- Need electronic as well as printed submission
 - I can execute your program, help find mistakes
 - Copy your lab directory into your turnin directory
 - How do you copy a directory?

Linux Command Conventions

- `<arg>` means fill in the appropriate thing
- `[arg]` means optional argument

- Example: Move or Rename a file
 - `mv <sourcefile> <destination>`
 - If `<destination>` is a *directory*, keeps the original source file's name

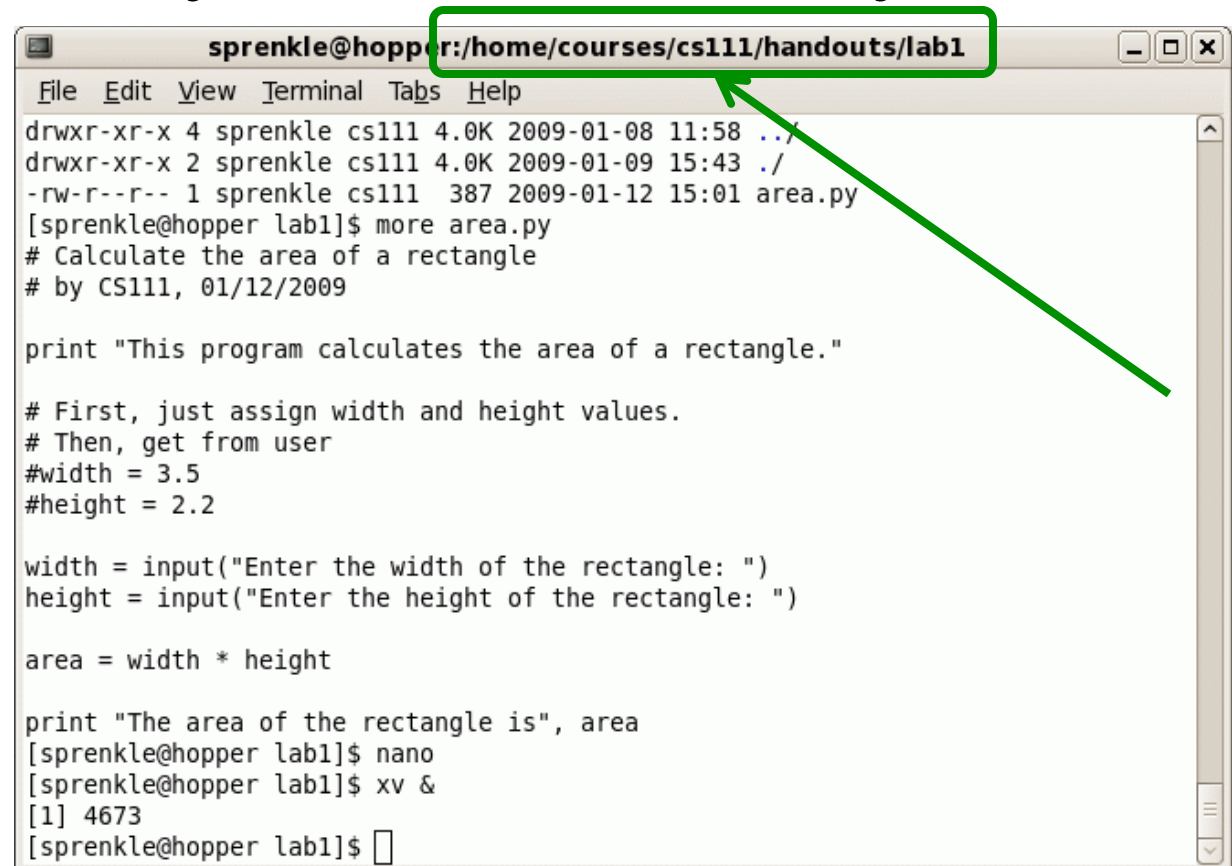
```
mv ~/labs/file.py labs/lab1/
```



- File `file.py` will be in `labs/lab1` directory

Lab 1: Linux Practice

- Setting up directories
- Renaming/moving files
- Note: terminal tells you which directory you're in



A terminal window titled 'sprenkle@hopper' with the current directory path '/home/courses/cs111/handouts/lab1' highlighted in a green box. The terminal shows the output of the 'ls' command, listing files 'cs111', 'area.py', and their permissions, sizes, and timestamps. Below this, the command 'more area.py' is executed, displaying the contents of the 'area.py' file. The script is a Python program that calculates the area of a rectangle. It includes comments, a print statement, variable assignments for width and height, and a final print statement. The user has also run 'nano' and 'xv &' commands, with the output '[1] 4673' visible.

```
sprenkle@hopper:/home/courses/cs111/handouts/lab1
File Edit View Terminal Tabs Help
drwxr-xr-x 4 sprenkle cs111 4.0K 2009-01-08 11:58 ../
drwxr-xr-x 2 sprenkle cs111 4.0K 2009-01-09 15:43 ./
-rw-r--r-- 1 sprenkle cs111 387 2009-01-12 15:01 area.py
[sprenkle@hopper lab1]$ more area.py
# Calculate the area of a rectangle
# by CS111, 01/12/2009

print "This program calculates the area of a rectangle."

# First, just assign width and height values.
# Then, get from user
#width = 3.5
#height = 2.2

width = input("Enter the width of the rectangle: ")
height = input("Enter the height of the rectangle: ")

area = width * height

print "The area of the rectangle is", area
[sprenkle@hopper lab1]$ nano
[sprenkle@hopper lab1]$ xv &
[1] 4673
[sprenkle@hopper lab1]$
```

Lab 1: Programming Practice

- Name them **lab1.n.py**, where n is the problem you're working on
- After completed, demonstrate that your program works
 1. Close IDLE/Python shell, rerun program
 - Get rid of the output from when you were developing/ debugging ("scratch work")
 2. Execute using **good** test cases
 - More than one test case if dealing with user input
 - Don't need to exhaustively test
 3. Save output for each program in file named **lab1.n.out** where n is the problem you're working on

Lab 1 Notes

- Expect comments in programs
 - High-level comments, author
 - Notes for your algorithms, implementation
- Expect testing on programs
 - What are good test cases for your programs?
 - Show the output from those test cases
 - But don't go overboard, testing every possible number!
- Honor System
 - Pledge the Honor Code on printed sheets