

Objectives

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

Jan 17, 2012

Sprenkle - CSCI111

1

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".

Jan 17, 2012

Sprenkle - CSCI111

2

Review

- What are the two different types of division?
- How do we get input from the user?
 - Numerical vs textual (string)
- What is our process for solving problems?
- What is the two-part verification process we need to do after we implement a program?

Jan 17, 2012

Sprenkle - CSCI111

3

Review: Formalizing Process of Developing Computational Solutions

1. Create a sketch of how to solve the problem (the algorithm)
2. Fill in the details in Python
3. Test the Python program with *good* test cases
 - a. If errors found, debug program
 - b. Repeat step 3

Jan 16, 2012

Sprenkle - CSCI111

4

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

Jan 17, 2012

Sprenkle - CSCI111

5

Review: Two Division Operators

/ Float Division

- Result is a **float**
- Examples:
 - $6/3 \rightarrow 2.0$
 - $10/3 \rightarrow 3.3333333333333335$
 - $3.0/6.0 \rightarrow 0.5$
 - $10/9 \rightarrow 1.9$

// Integer Division

- Result is an **int**
- Examples:
 - $6//3 \rightarrow 2$
 - $10//3 \rightarrow 3$
 - $3.0//6.0 \rightarrow 0$
 - $10//9 \rightarrow 1$

Jan 17, 2012

Sprenkle - CSCI111

6

Review: Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
 - Template for solution
- Example (Standard Algorithm)
 - Get input from user
 - Do some computation
 - Display output

```

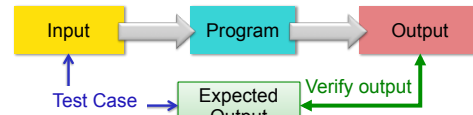
Assign.  x = input("...")
Assign.  ans = ...
print    print(ans)
    
```

Jan 24, 2011

Sprenkle - CSCI111

7

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

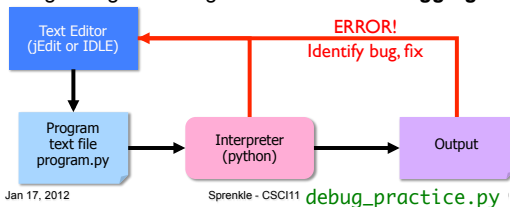
Jan 17, 2012

Sprenkle - CSCI111

8

Review: Debugging

- After identifying errors during *testing*
- 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 errors is called **debugging**



Jan 17, 2012

Sprenkle - CSCI111

debug_practice.py

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

Jan 17, 2012

Sprenkle - CSCI111

10

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

Jan 17, 2012

Sprenkle - CSCI111

11

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

Jan 17, 2012

Sprenkle - CSCI111

12

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 turn in directory
 - How do you copy a directory?

Jan 17, 2012

Sprenkle - CSCI111

13

Linux Command Conventions

- `<arg>` means fill in the appropriate thing
- `[arg]` means optional argument
- Example: Move or Rename a file
 - `mv <sourcefile> <destination>`
`mv ~/labs/file.py newfilename.py`
 - Moves `file.py` to current directory with a new name
 - If `<destination>` is a *directory*, keeps the original source file's name
`mv ~/labs/file.py ~/labs/lab1/` ← *directory*
 - File `file.py` will be in `labs/lab1` directory

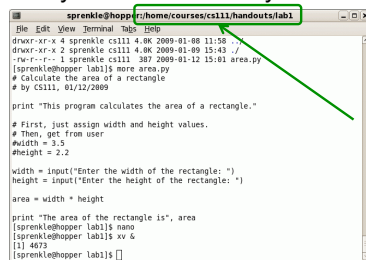
Jan 17, 2012

Sprenkle - CSCI111

14

Lab 1: Linux Practice

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



```
sprenkle@hopper:~/home/courses/csci111/handouts/lab1$  
File Edit View _Terminal_ Help  
dnuxr-xr-x 4 sprenkle csci11 4.0K 2009-01-08 11:58 .  
dnuxr-xr-x 2 sprenkle csci11 4.0K 2009-01-09 15:43 ../  
-rwxr-xr-x 1 sprenkle csci11 387 2009-01-12 15:41 area.py  
[sprenkle@hopper lab1]$ more area.py  
# Calculate the area of a rectangle  
# by CSCI11, 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 6  
[!] 4073  
[sprenkle@hopper lab1]$
```

Jan 17, 2012

IDLE Review

- Run using `idle3 &`

Jan 17, 2012

Sprenkle - CSCI111

16

Lab 1: Programming Practice

- Name program files **lab1.n.py**, where *n* is the problem you're working on
- After completed, demonstrate that your program works
 1. Close IDLE/Python interpreter, 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

Jan 17, 2012

Sprenkle - CSCI111

17

Lab 1 Expectations

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

Jan 17, 2012

Sprenkle - CSCI111

18