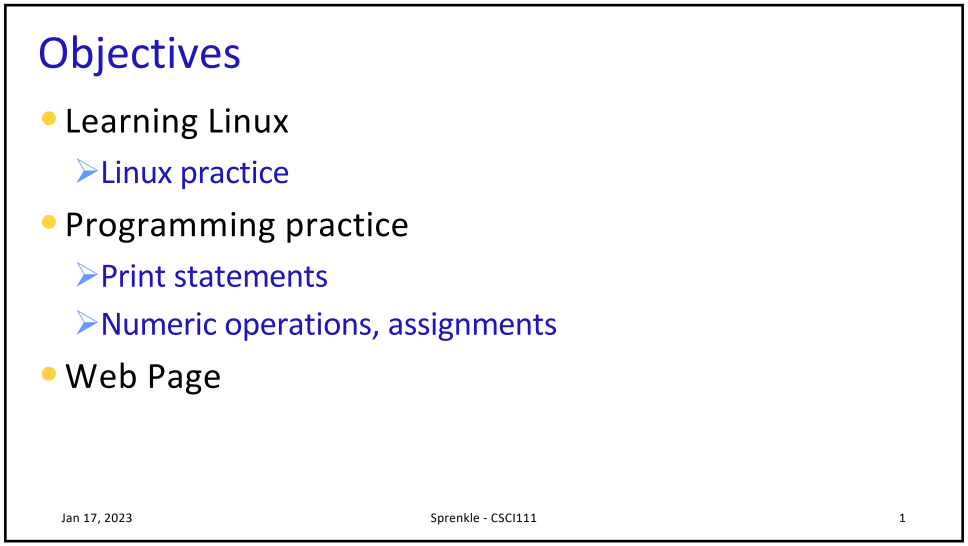
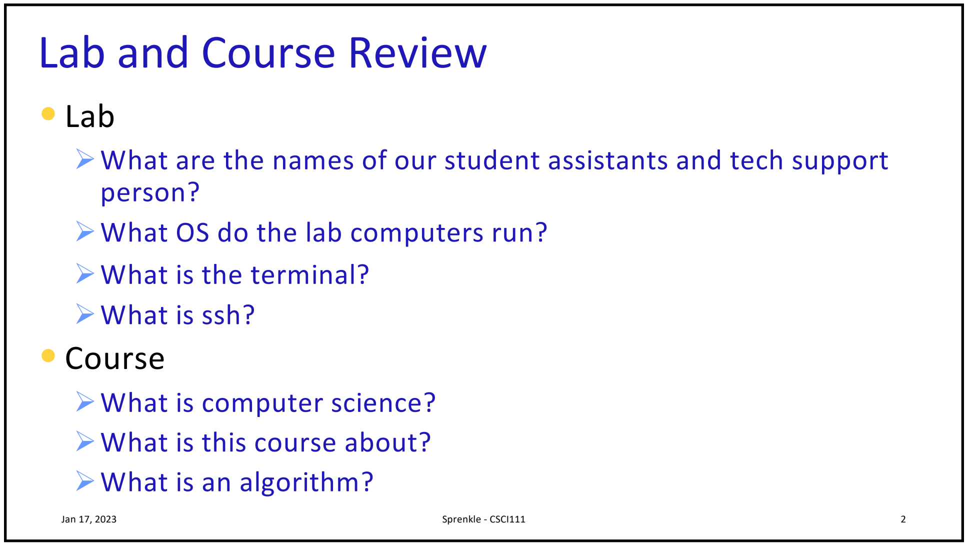


Objectives

- Learning Linux
  - Linux practice
- Programming practice
  - Print statements
  - Numeric operations, assignments
- Web Page



Lab and Course Review

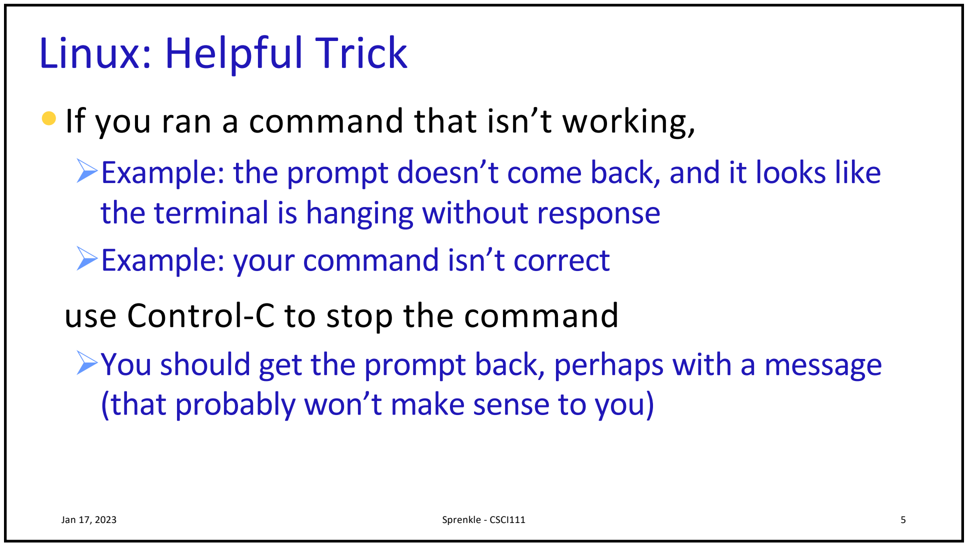
- Lab
  - What are the names of our student assistants and tech support person?
  - What OS do the lab computers run?
  - What is the terminal?
  - What is ssh?
- Course
  - What is computer science?
  - What is this course about?
  - What is an algorithm?

Lab System Review

- Everything you do in lab on these machines (if you save it), you can access remotely (on lab machines)
- Everything you do remotely on lab machines (if you save it), you can see on the lab machines in person

Lab 0 Feedback

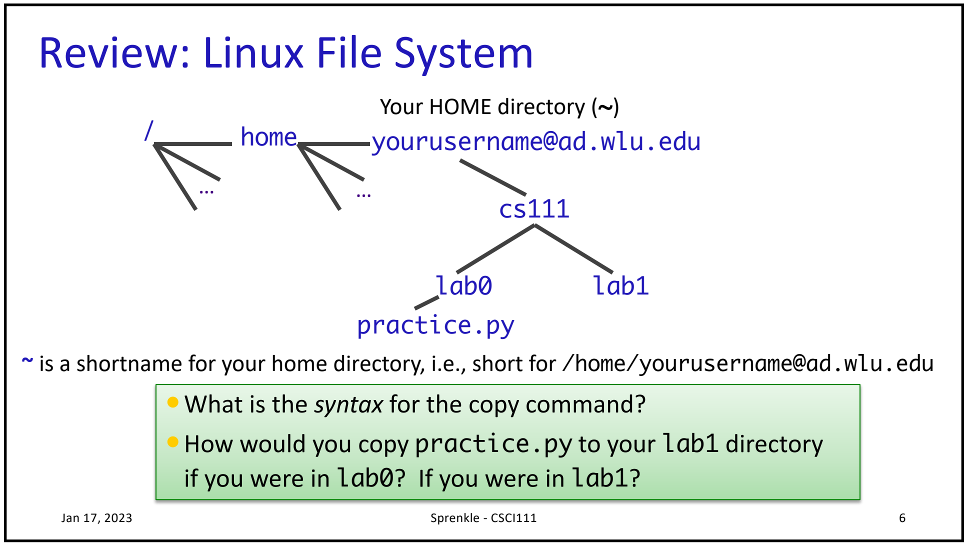
- Overall, did well
  - ➢Generally, lab grades should be high
- Canvas extra credit Easter egg
  - ➢Great fun facts!



Linux: Helpful Trick

- If you ran a command that isn't working,
  - Example: the prompt doesn't come back, and it looks like the terminal is hanging without response
  - Example: your command isn't correct

  use Control-C to stop the command
  - You should get the prompt back, perhaps with a message (that probably won't make sense to you)



Review: Linux File System

Your HOME directory (~)

`/` … `home` … `yourusername@ad.wlu.edu`

`cs111`

`lab0` `lab1`

`practice.py`

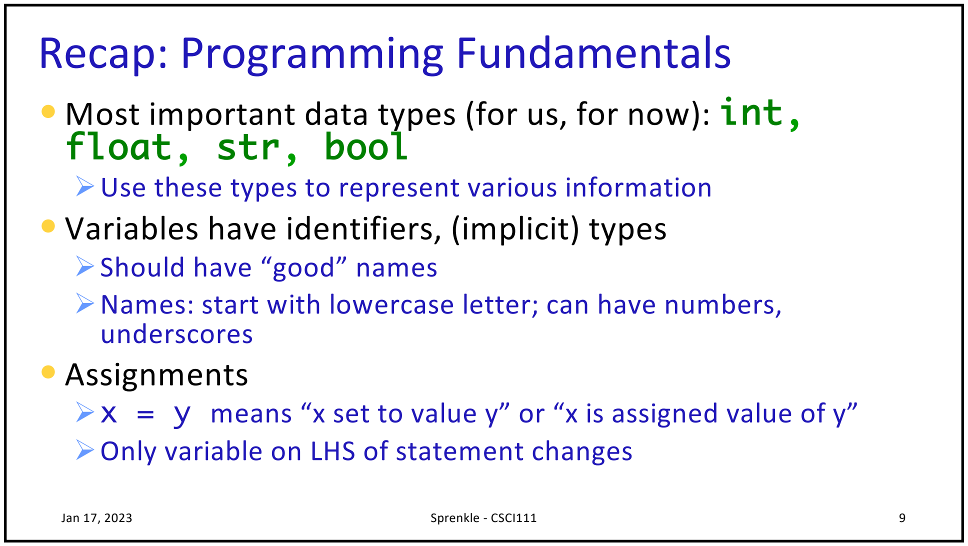
~ is a shortname for your home directory, i.e., short for `/home/yourusername@ad.wlu.edu`

- What is the *syntax* for the copy command?
- How would you copy `practice.py` to your `lab1` directory if you were in `lab0`? If you were in `lab1`?

PYTHON PROGRAMMING

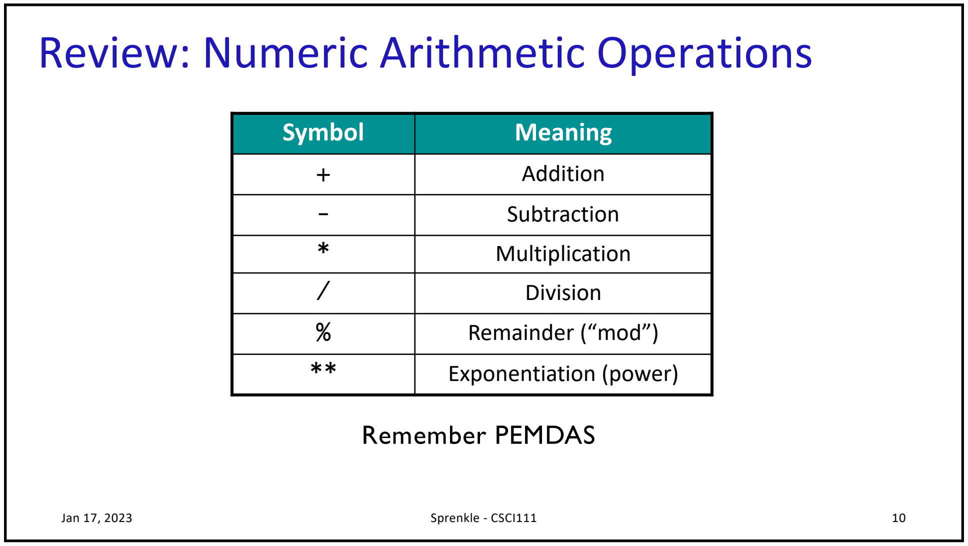
Review

- What are the two ways to run the Python interpreter?
- Give three examples of data types
- How do we display output from a program?
- How do we assign values to variables?
- What arithmetic operators are available?
    - What rules do they follow?



Recap: 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 set to value y" or "x is assigned value of y"
  - ➢ Only variable on LHS of statement changes



Review: Numeric Arithmetic Operations

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

Remember PEMDAS

Review: Arithmetic & Assignment

- You can use the assignment operator (=) and arithmetic operators to do calculations
  1. Calculate right hand side
  2. Assign value to variable
- Remember your order of operations! (PEMDAS)
- Examples:

```
x = 4+3*10
y = 3/2.0
z = x+y
```

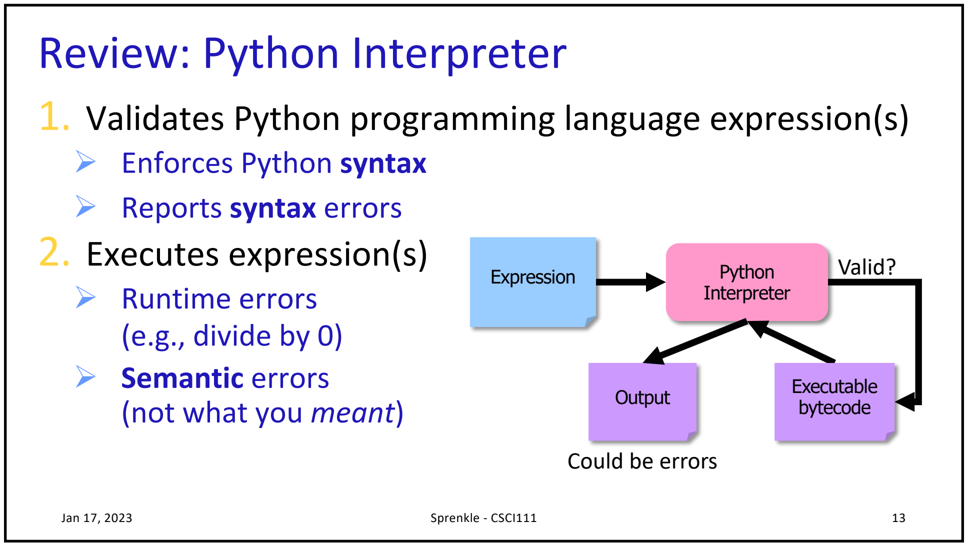
The right-hand sides are ***expressions***, just like in math.

Assignment statements

- Assignment statements are NOT math equations!
  - ➢Valid expression: `count = count + 1`
- These are commands!

```
x = 2
y = x
x = x + 3
```

After these 3 statements execute, what are the values of x, y?



Review: Python Interpreter

1. Validates Python programming language expression(s)
   - Enforces Python **syntax**
   - Reports **syntax** errors
2. Executes expression(s)
   - Runtime errors (e.g., divide by 0)
   - **Semantic** errors (not what you *meant*)

Expression

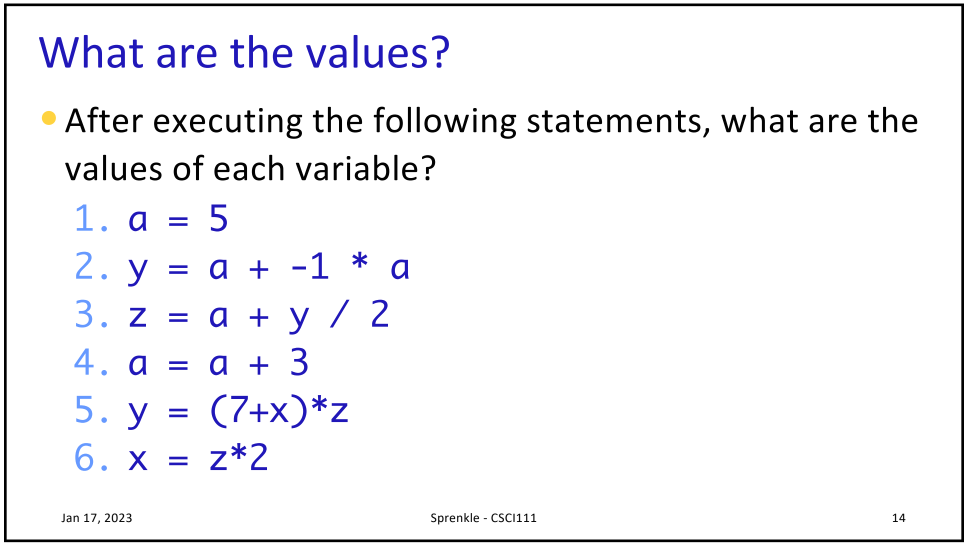
Python Interpreter

Valid?

Output

Executable bytecode

Could be errors



What are the values?

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

What are the values?

- After executing the following statements, what are the values of each variable?

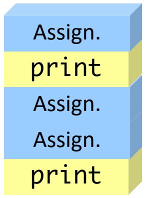
```
1. a = 5
2. y = a + -1 * a
3. z = a + y / 2
4. a = a + 3
5. y = (7+x)*z
6. x = z*2
```

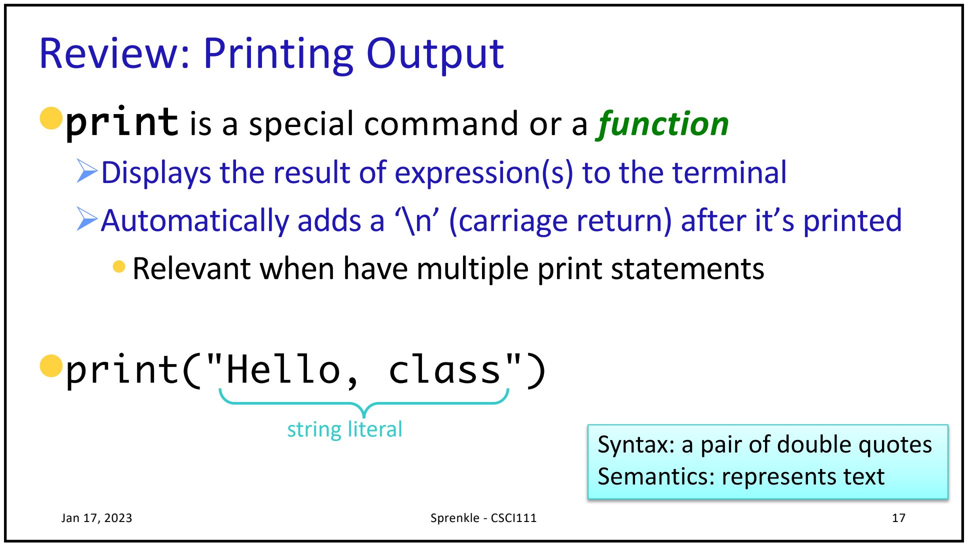
Runtime error: x doesn't have a value yet!

- We say "x was not initialized"
- Can't use a variable on RHS until seen on LHS!*

Programming Building Blocks

- Each type of statement is a building block
  - Initialization/Assignment Assign.
    - So far: Arithmetic
  - Print print
- We can combine them to create more complex programs
  - Solutions to problems





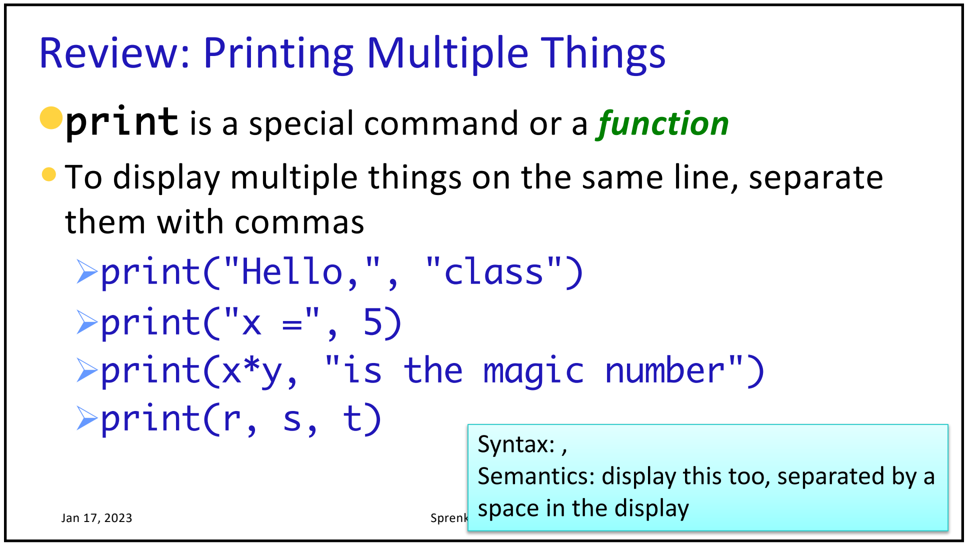
Review: Printing Output

- `print` is a special command or a ***function***
  - Displays the result of expression(s) to the terminal
  - Automatically adds a ‘\n’ (carriage return) after it’s printed
    - Relevant when have multiple print statements

- `print("Hello, class")`

string literal

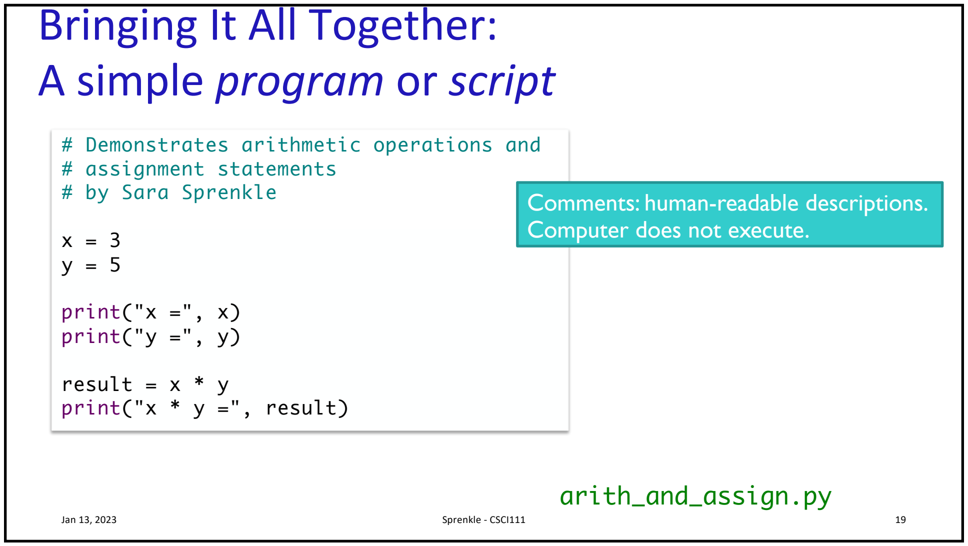
Syntax: a pair of double quotes
Semantics: represents text



Review: Printing Multiple Things

- `print` is a special command or a ***function***
- To display multiple things on the same line, separate them with commas
  - `print("Hello,", "class")`
  - `print("x =", 5)`
  - `print(x*y, "is the magic number")`
  - `print(r, s, t)`

Syntax: ,
Semantics: display this too, separated by a space in the display

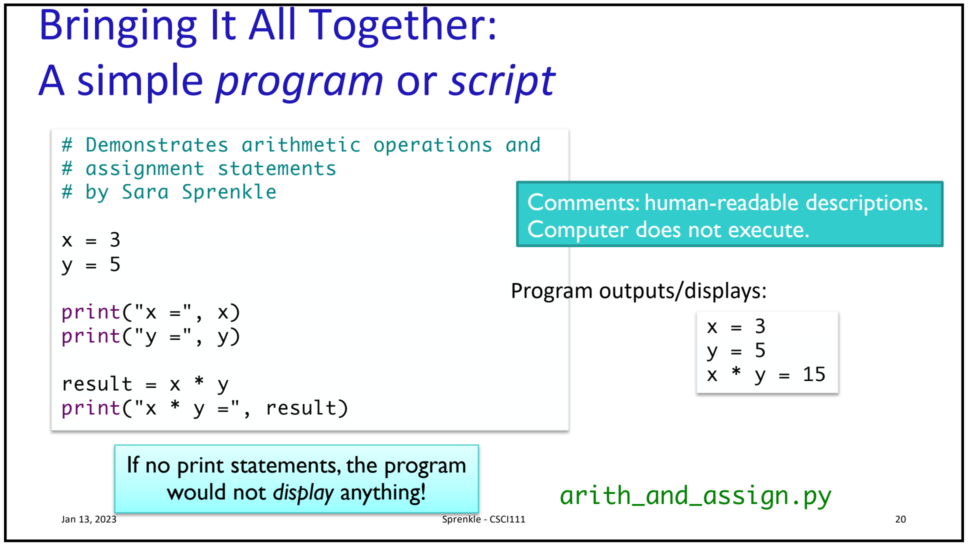


Bringing It All Together: A simple *program* or *script*

```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle

x = 3
y = 5

print("x =", x)
print("y =", y)

result = x * y
print("x * y =", result)
```

Comments: human-readable descriptions. Computer does not execute.

arith_and_assign.py



Bringing It All Together: A simple *program* or *script*

```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle

x = 3
y = 5

print("x =", x)
print("y =", y)

result = x * y
print("x * y =", result)
```

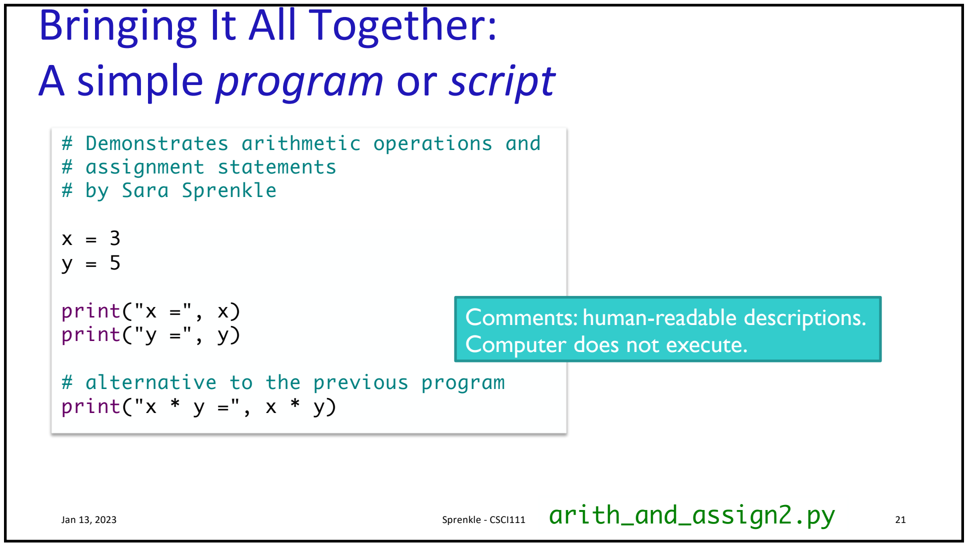
Comments: human-readable descriptions. Computer does not execute.

Program outputs/displays:

```
x = 3
y = 5
x * y = 15
```

If no print statements, the program would not *display* anything!

arith_and_assign.py

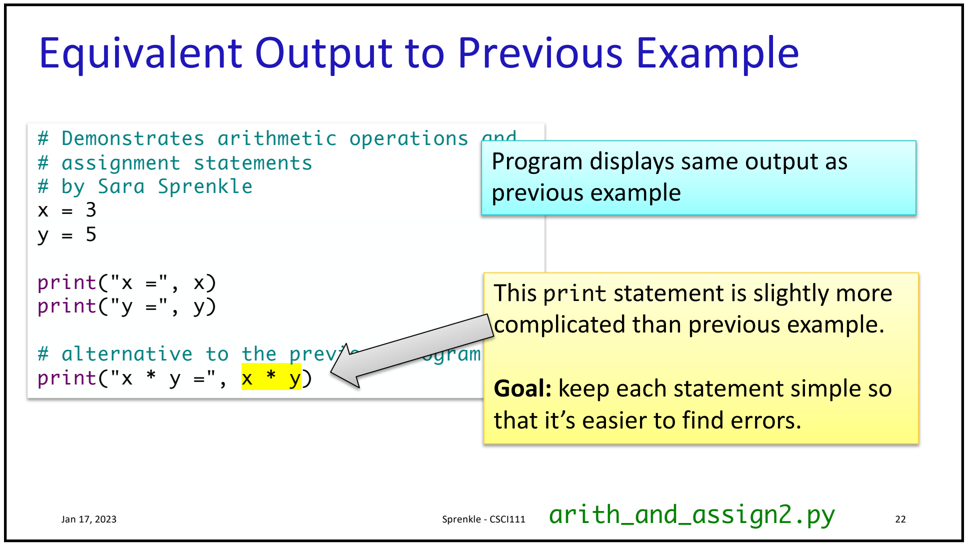


Bringing It All Together:
A simple *program* or *script*

```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle

x = 3
y = 5

print("x =", x)
print("y =", y)

# alternative to the previous program
print("x * y =", x * y)
```

Comments: human-readable descriptions. Computer does not execute.

arith_and_assign2.py



Equivalent Output to Previous Example

```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle
x = 3
y = 5

print("x =", x)
print("y =", y)

# alternative to the previous program
print("x * y =", x * y)
```

Program displays same output as previous example

This `print` statement is slightly more complicated than previous example.

Goal: keep each statement simple so that it's easier to find errors.

arith_and_assign2.py

A Documented Program

```
# Demonstrates arithmetic operations and
# assignment statements
# by Sara Sprenkle

x = 3
y = 5

print("x =", x)
print("y =", y)

result = x * y
print("x * y =", result)
```

Comments: human-readable descriptions.
Computer does not execute.
Can be anywhere in code.

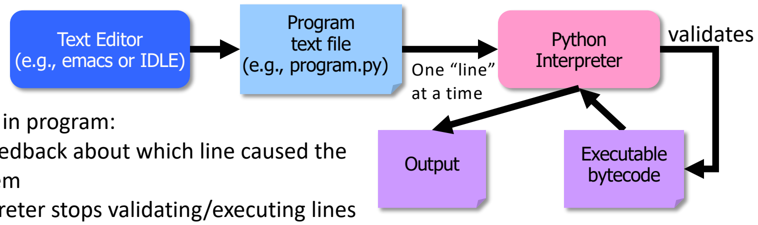
All your submitted programs ***must*** have

1. high-level description of what the program does
2. Your name as author and date you authored it

arith_and_assign.py

Review: Batch Mode

1. Programmer types a program/script into a **text editor**
2. An interpreter turns each expression into bytecode and then executes each expression



If errors in program:

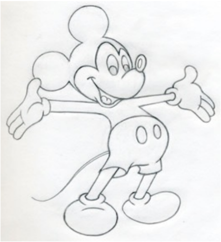
- Get feedback about which line caused the problem
- Interpreter stops validating/executing lines

DEVELOPMENT PROCESS

Formalizing Process of Developing Computational Solutions

1. Create a sketch of how to solve the problem (the algorithm)

Use comments to describe the steps



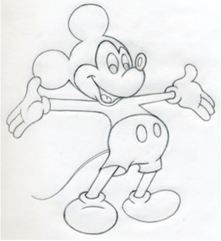
Example sketch for previous Python program:

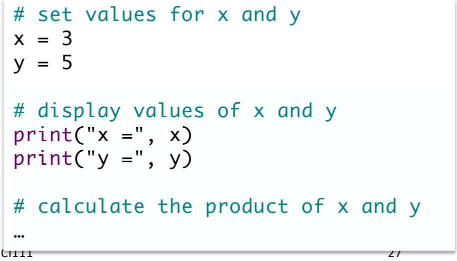
```
# set values for x and y

# display values of x and y

# calculate the product of x and y

# print the results
```

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

Use comments to describe the steps





```
# set values for x and y
x = 3
y = 5

# display values of x and y
print("x =", x)
print("y =", y)

# calculate the product of x and y
…
```

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. Execute the program

May not have everything filled

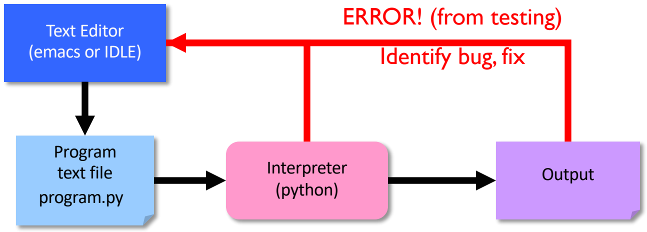
- ➢ Test: does the program’s output match your expectation?

It worked! ☺ Or, it didn't ☹

- Sometimes the program doesn't work
- Types of programming errors:
  - Syntax error
    - Interpreter shows where the problem is
  - Logic/semantic error
    - answer = 2+3
    - No, answer should be 2*3
  - Exceptions/Runtime errors
    - answer = 2/0
    - Undefined variable name

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***



Text Editor (emacs or IDLE)

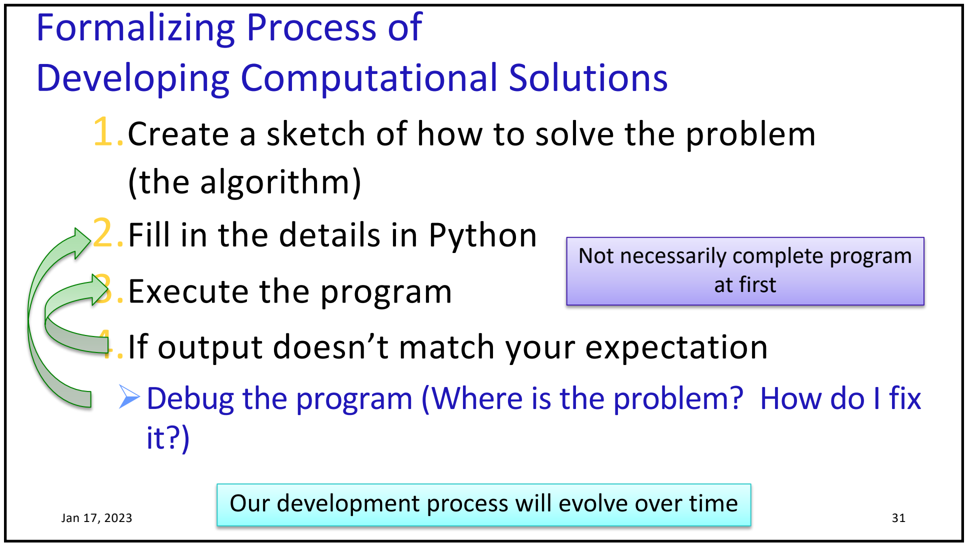
Program text file program.py

Interpreter (python)

Output

ERROR! (from testing)

Identify bug, fix

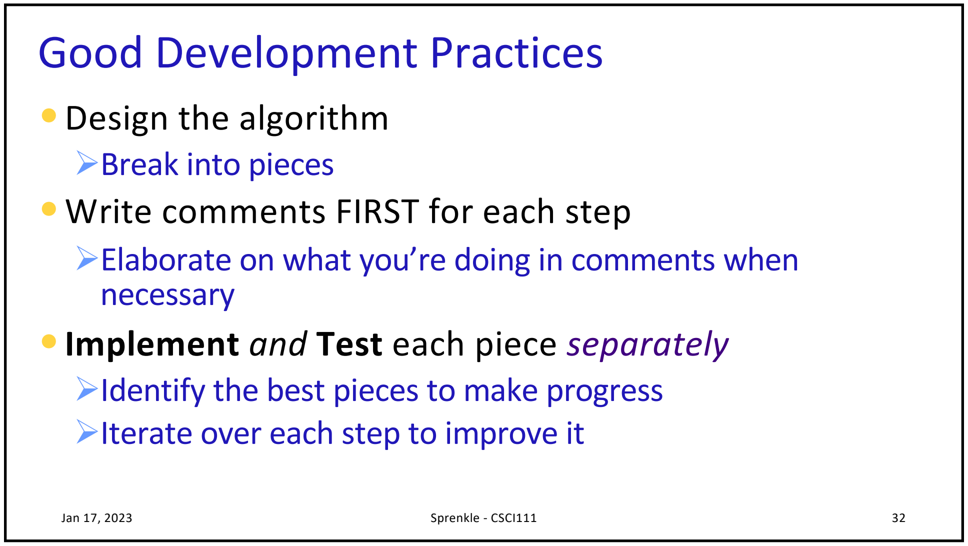


Formalizing Process of Developing Computational Solutions

1. Create a sketch of how to solve the problem (the algorithm)
2. Fill in the details in Python
3. Execute the program
4. If output doesn't match your expectation
   - Debug the program (Where is the problem? How do I fix it?)

Not necessarily complete program at first

Our development process will evolve over time



Good Development Practices

- Design the algorithm
  - Break into pieces
- Write comments FIRST for each step
  - Elaborate on what you're doing in comments when necessary
- **Implement** *and* **Test** each piece *separately*
  - Identify the best pieces to make progress
  - Iterate over each step to improve it

When to Use Comments

- Document the author, high-level description of the program at the top of the program

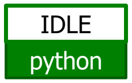
- Provide an outline of an algorithm
  - Separates the steps of the algorithm

- Describe difficult-to-understand code

PYTHON PROGRAMMING IN LINUX

IDLE Development Environment

- Runs on top of Python interpreter
- Command: `idle &`
  - **&** Runs command in "background" so you can continue to use the terminal

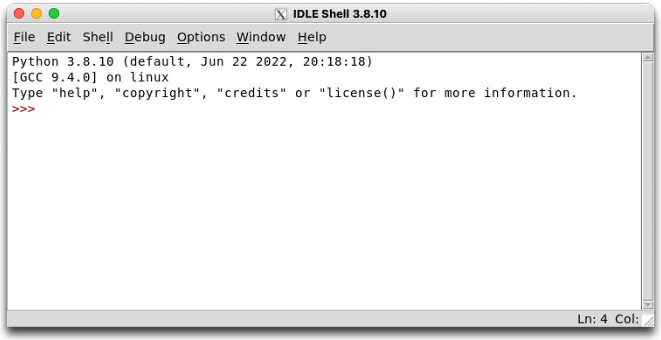


Since our programming language is named after Monty Python, what is the development environment named after?

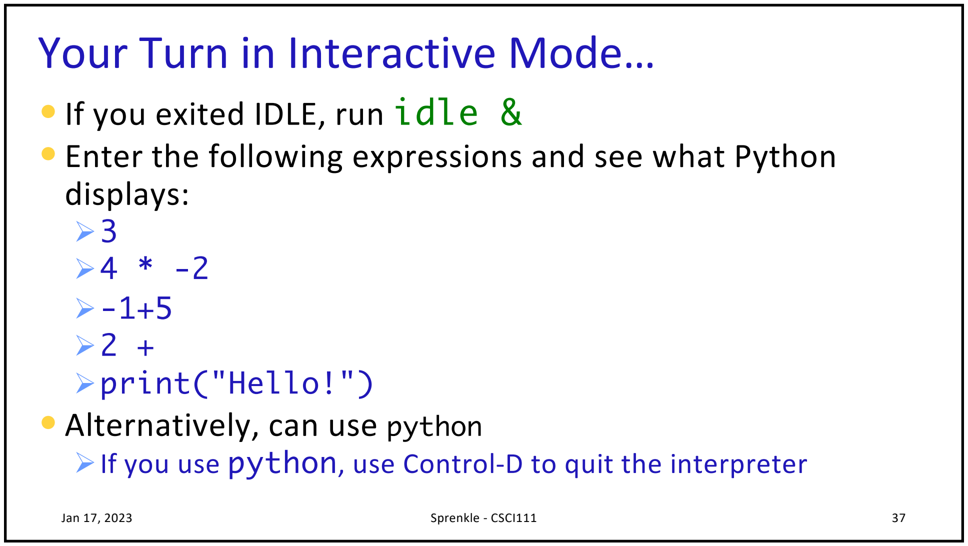
- Can use IDLE to
  - Run Python in **interactive** mode
  - Write and execute scripts in **batch** mode

IDLE

- IDLE first opens up a Python ***shell***
  - i.e., the Python interpreter in interactive mode

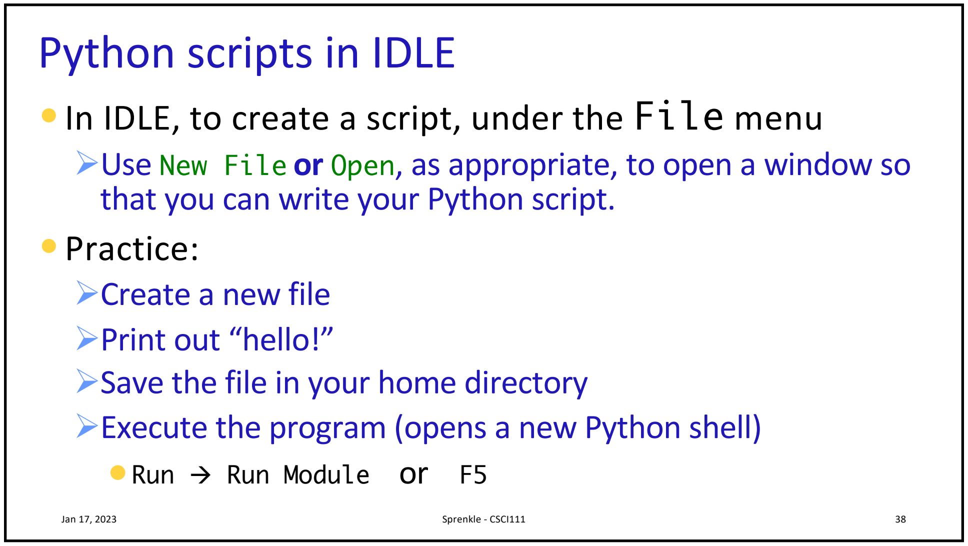


```
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```



Your Turn in Interactive Mode…

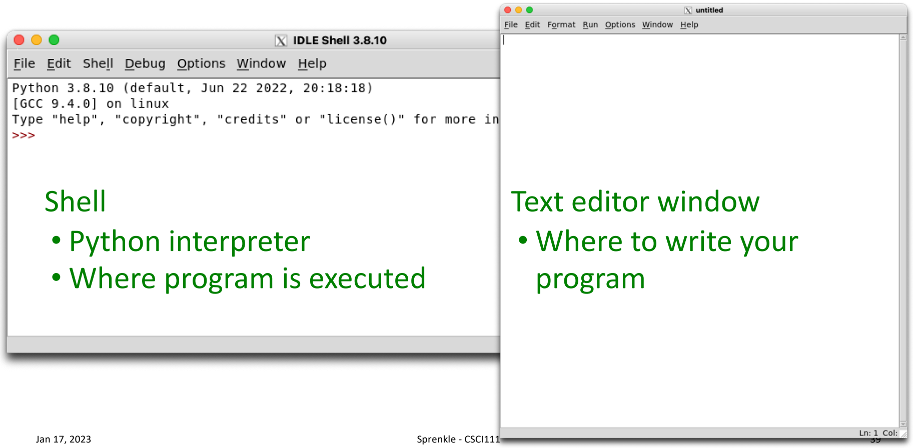
- If you exited IDLE, run `idle &`
- Enter the following expressions and see what Python displays:
  - `3`
  - `4 * -2`
  - `-1+5`
  - `2 +`
  - `print("Hello!")`
- Alternatively, can use `python`
  - If you use `python`, use Control-D to quit the interpreter



Python scripts in IDLE

- In IDLE, to create a script, under the `File` menu
  - Use `New File` **or** `Open`, as appropriate, to open a window so that you can write your Python script.
- Practice:
  - Create a new file
  - Print out “hello!”
  - Save the file in your home directory
  - Execute the program (opens a new Python shell)
    - `Run → Run Module` or `F5`

IDLE: Interactive Development Environment


IDLE Shell 3.8.10
File Edit Shell Debug Options Window Help
untitled
File Edit Format Run Options Window Help
Ln: 1 Col:

```
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license()" for more in
>>>
```

Shell
- Python interpreter
- Where program is executed

Text editor window
- Where to write your program

Recap: Executing Python

- Interactive Mode
  - Try out expressions
  - `python`
- Batch Mode
  - Execute Python scripts
  - `python <pythonscript>`
- **IDLE** combines these two modes into one *integrated development environment* (IDE)
  - `idle &`

Lab 1 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
- Honor System

Lab 1: Programming Practice

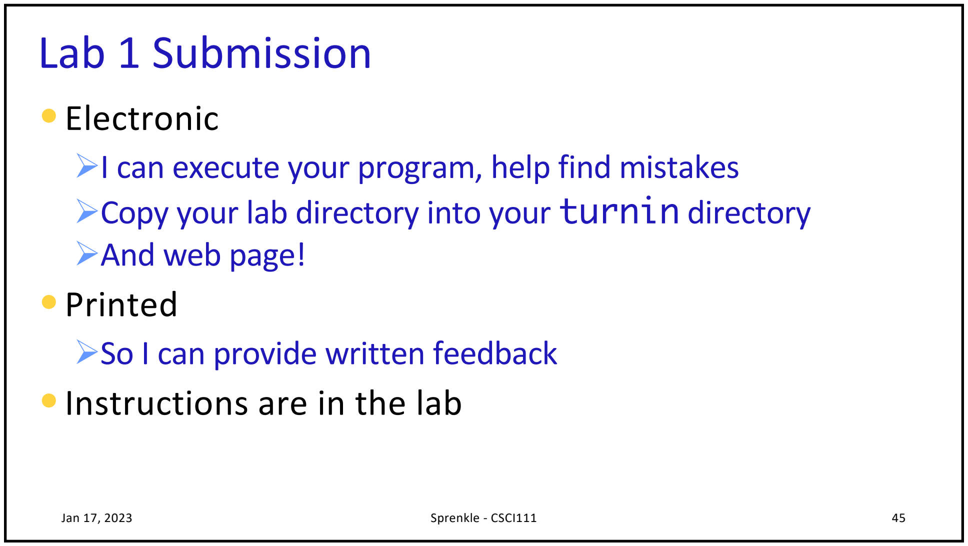
- After the warm up problems…
- Name program files **lab1_n.py**, where *n* is the problem you're working on
- After completed, demonstrate that your program works
  1. Close IDLE/Python interpreter, rerun program
     - Get rid of the output from when you were developing/debugging ("scratch work")
  2. Save output for each program in file named `lab1_n.out` where *n* is the problem you're working on

Lab 1 Expectations: Example Output

- Your program should have clearly labeled output
  - Clear to user what is happening in program
- Resulting output should be saved in a `.out` file

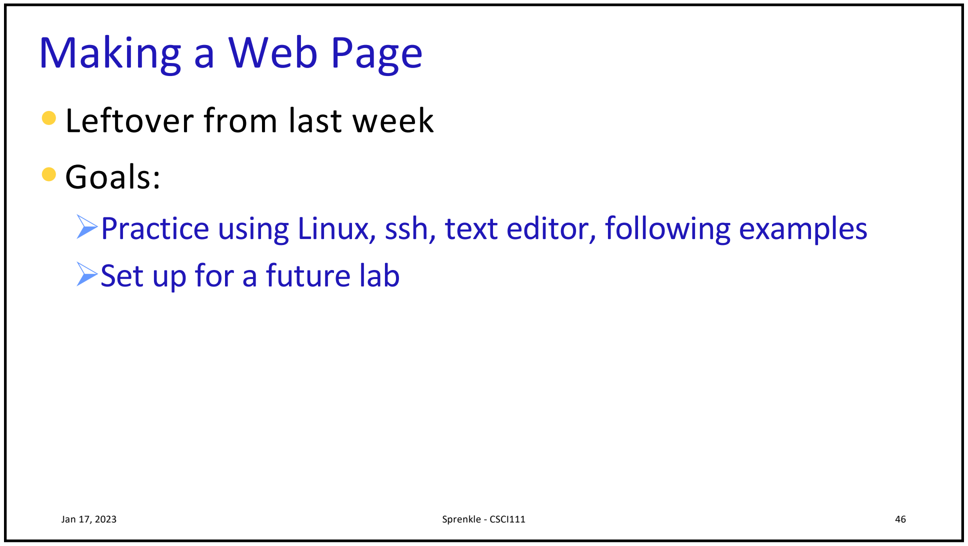
Lab 1 Expectations: Read the Directions

- To ***completion***
- Often the answer to your question is in the next sentence
- Practice patience
  - Rushing → poor outcomes



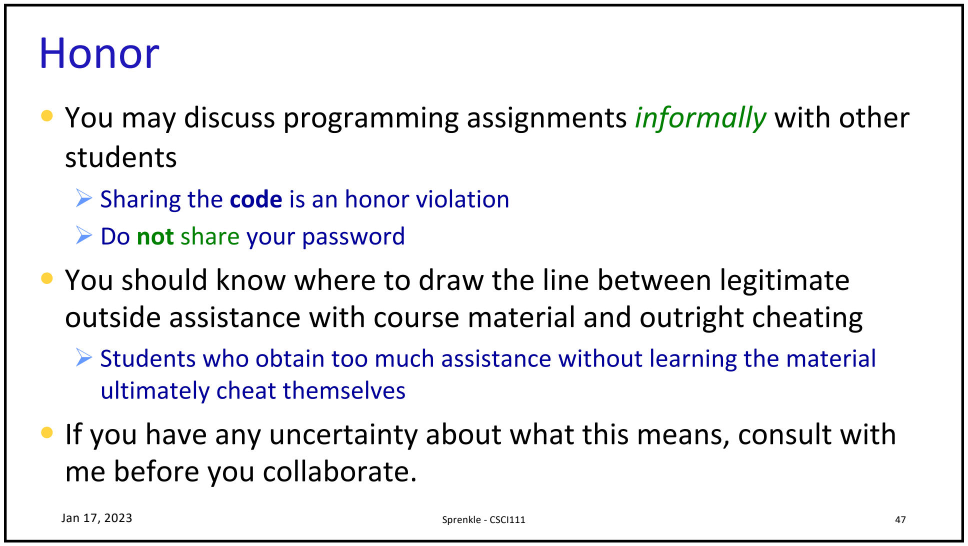
Lab 1 Submission

- Electronic
  - I can execute your program, help find mistakes
  - Copy your lab directory into your `turnin` directory
  - And web page!
- Printed
  - So I can provide written feedback
- Instructions are in the lab



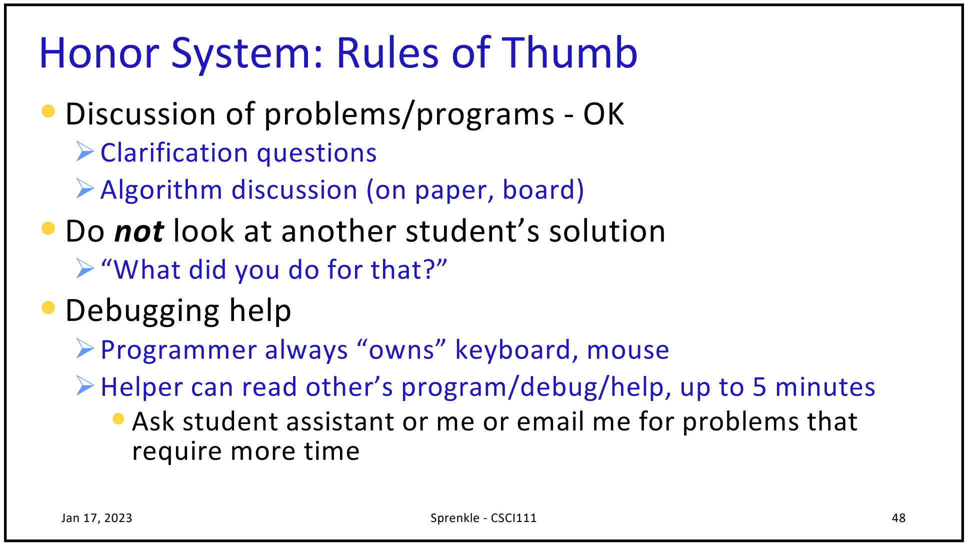
Making a Web Page

- Leftover from last week
- Goals:
  - Practice using Linux, ssh, text editor, following examples
  - Set up for a future lab



Honor

- You may discuss programming assignments *informally* with other students
  - Sharing the **code** is an honor violation
  - Do **not** share your password
- You should know where to draw the line between legitimate outside assistance with course material and outright cheating
  - Students who obtain too much assistance without learning the material ultimately cheat themselves
- If you have any uncertainty about what this means, consult with me before you collaborate.



Honor System: Rules of Thumb

- Discussion of problems/programs - OK
  - Clarification questions
  - Algorithm discussion (on paper, board)
- Do ***not*** look at another student's solution
  - "What did you do for that?"
- Debugging help
  - Programmer always "owns" keyboard, mouse
  - Helper can read other's program/debug/help, up to 5 minutes
    - Ask student assistant or me or email me for problems that require more time

Lab 1 Overview

- Linux practice
- IDLE practice
- Programming practice
- Web page