

## Objectives

- Review
- Lab 1
  - Linux practice
  - Programming practice
    - Print statements
    - Numeric operations, assignments
    - Input statements

## Lab 0 Feedback

- Overall, did well
  - Lost points because didn't check work
    - E.g., broken Web page links, not including required text
  - Generally, lab grades should be high
- Interesting article links!
  - Consider reviewing for extra credit
- Sakai extra credit Easter egg
  - Great fun facts!

## Review

- How do we display output?
- What are the data types available in Python?
- How should we name variables?
  - Describe what good identifiers look like
- How do we assign values to variables?

## 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: Assignment statements

- Assignment statements are NOT math equations!

```
count = count + 1
```

- These are commands!

```
x = 2
```

```
y = x
```

```
x = x + 3
```

What are the values of x, y?

## Review: Numeric Arithmetic Operations

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

Remember PEMDAS

## Review

- What is our development process?
  - What is the two-part verification process we need to do after we implement a program?

## Review: Development Process

1. Sketch algorithm to solve problem
  - Write steps in comments
2. Fill in details in Python
3. Come up with good test cases
  - Input and expected output
4. Repeat until know the code works and is “good”
  - Test code
  - Debug
  - Refine until “good”
    - For now: good variable names, good/pretty output

## Review

- How do we get input from the user?
  - How is getting numeric input different from getting text input?

## 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?

## 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 Chadwick Boseman? "))
print("Cool! I like him", rating*1.8, "much!")
```

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

## 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 Chadwick Boseman? "))
print("Cool! I like him", rating*1.8, "much!")
                        expression
```

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

## Improving average2.py

- With what we just learned, how could we improve average2.py?
- Example of suggested approach to development
  - Input is going to become fairly routine.
  - Wait on input until you have figured out the rest of the program/problem.

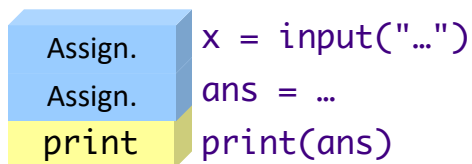
Examples from each class period are on schedule page.

## Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution

## 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

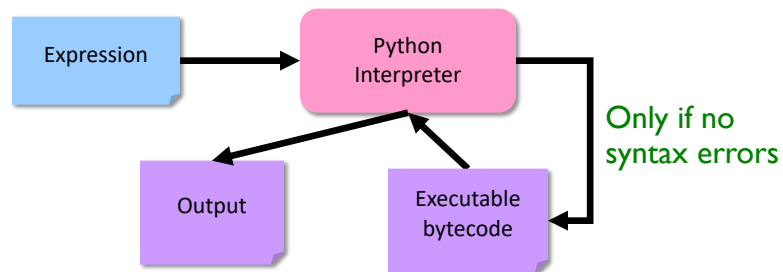


```
x = input("...")
ans = ...
print(ans)
```



## Python Interpreter

1. Validates Python programming language expression(s)
  - Enforces Python syntax rules
  - Reports syntax errors ← Have a lot of these early on!
2. Executes expression(s)



Jan 15, 2019

Sprenkle - CSCI111

17

## Two Modes to Execute Python Code

- **Interactive**
  - Try out Python expressions
- **Batch:** execute *scripts* (i.e., files containing Python code)
  - What we'll write usually

Jan 15, 2019

Sprenkle - CSCI111

18

## Python Interpreter: Interactive Mode

Run by typing `python3` in terminal

```
sprengle@fred:~/algorithms_next_frontier$ cd
sprengle@fred:~$ cd cs111/
sprengle@fred:~/cs111$ python3
Python 3.5.3 (default, May 11 2017, 09:10:41)
[GCC 6.3.1 20161221 (Red Hat 6.3.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 3
3
>>> 4+5
9
>>> 1-7
-6
>>> "word"
'word'
>>> word
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'word' is not defined
>>> print 4 + 5
File "<stdin>", line 1
print 4 + 5
      ^
SyntaxError: Missing parentheses in call to 'print'
>>> print(4+5)
9
>>> if
```

Python displays the result

Type in the expression

**Error Message:**  
Thinks word must be a variable and it is not defined

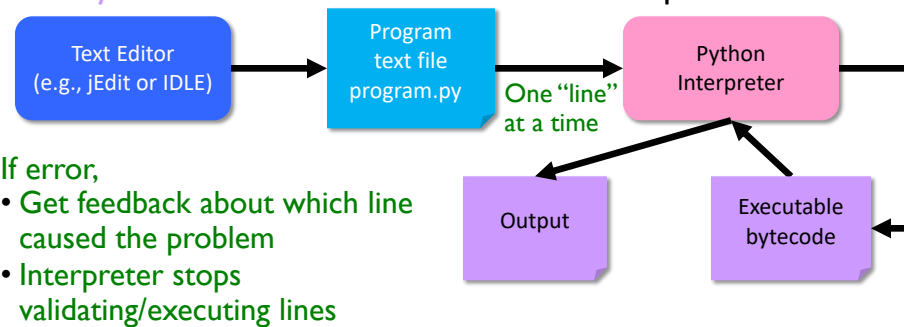
`print`: Special function to display output

`runHelpClient &`

Jan 15, 2019      Sprengle - CSCI111      19

## Batch Mode: Execute Scripts

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



## Example Python Script

Text file named: `hello.py`

```
# A first program
# by Sara Sprenkle, 01/15/2019

print("Hello, world!")
```

Print statement

- What does this program do?
  - Validate your guess by executing the program
    - Go into  
/csdept/courses/cs111/handouts/lab1  
directory
    - **python3 hello.py**

## Example Python Script

```
# A first program
# by Sara Sprenkle, 01/15/2019
print("Hello, world!")
```

} Documentation  
-- good style

- Only **Hello, world!** is printed out
- Python ignores everything after the “#”
  - Known as “**comments**” or, collectively, as **documentation**

Your program should *always* start with a high-level description of what the program does, your name, and the date the program was written

## IDLE Development Environment

- Runs on top of Python interpreter
- Command: **idle3 &**
  - **&** Runs command in “background” so you can continue to use the terminal

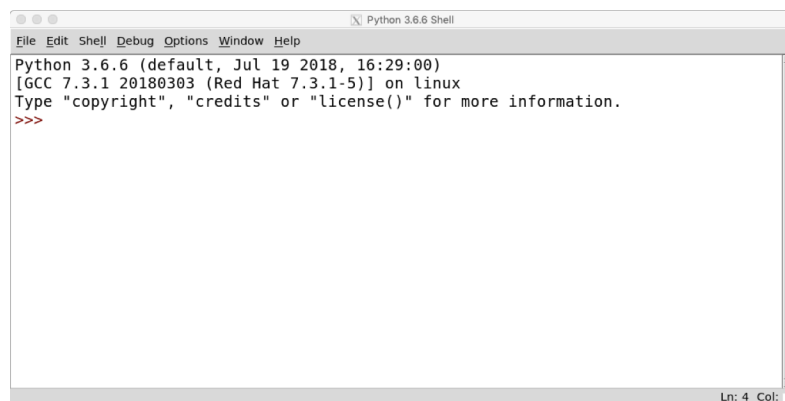


Since our programming language is named after Monty Python, what is the development environment named after?

- 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



## Your Turn in Interactive Mode...

- Run `idle3`
- Enter the following expressions and see what Python displays:
  - `3`
  - `4 * -2`
  - `-1+5`
  - `2 +`
  - `print("Hello!")`
- Alternatively, can use `python3`
  - If you used `python3`, to quit the interpreter, use `Control-D`

Jan 15, 2019

Sprenkle - CSCI111

25

## IDLE

- In IDLE, 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

Jan 15, 2019

Sprenkle - CSCI111

26

## Recap: Executing Python

- Interactive Mode
  - Try out expressions
  - `python3`
- Batch Mode
  - Execute Python scripts
  - `python3 <pythonscript>`
- **IDLE** combines these two modes into one integrated development environment
  - `idle3 &`

Jan 15, 2019

Sprenkle - CSCI111

27

## Lab 0 Feedback

- If there were any issues with your web page, go back and fix them first.
  - We can help!
  - Goal: Make sure you're set up for the semester, when we create more web pages
    - Otherwise, you won't remember how to fix them

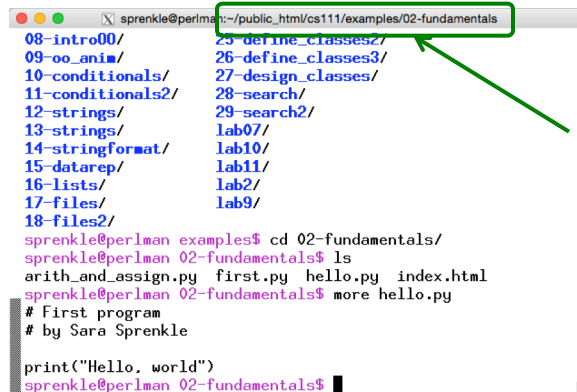
Jan 15, 2019

Sprenkle - CSCI111

28

## Lab 1: Linux Practice

- Review your notes, handouts from last lab
- Setting up directories
  - Make the directory, copy files
- Note: terminal tells you which directory you're in

A terminal window screenshot showing a user navigating through a directory structure. The user starts in a directory with many subdirectories, including '02-fundamentals/'. They use the 'cd' command to enter '02-fundamentals/' and then 'ls' to list the contents, which include 'arith\_and\_assign.py', 'first.py', 'hello.py', and 'index.html'. Finally, they use 'more hello.py' to view the contents of the file. The terminal output shows a simple 'Hello, world' program.

```
sprengle@perlman:~/public_html/cs111/examples/02-fundamentals/
08-intro00/ 25-define_classes2/
09-oo_anim/ 26-define_classes3/
10-conditionals/ 27-design_classes/
11-conditionals2/ 28-search/
12-strings/ 29-search2/
13-strings/ lab07/
14-stringformat/ lab10/
15-datarep/ lab11/
16-lists/ lab2/
17-files/ lab9/
18-files2/

sprengle@perlman examples$ cd 02-fundamentals/
sprengle@perlman 02-fundamentals$ ls
arith_and_assign.py first.py hello.py index.html
sprengle@perlman 02-fundamentals$ more hello.py
# First program
# by Sara Sprengle

print("Hello, world")
sprengle@perlman 02-fundamentals$
```

Jan 15, 2019

29

## 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
  - Pledge the Honor Code on printed sheets

Jan 15, 2019

Sprengle - CSCI111

30

## 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
- You will run some programs **multiple times** to demonstrate that the program works with different values of variables.
- 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 results in poor outcomes

## Lab 1 Submission

- Electronic as well as printed
  - I can execute your program, help find mistakes
  - Copy your lab directory into your turnin directory
- Instructions are in the 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

Reintroduce lab assistants

## On to the Lab!

- When you get to practice.py, add a print statement in practice.py that says “I read the slides!” for 2 points extra credit.

## 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

## 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
  - 7 pts for blog entry
    - Common issue – missing answers to one of questions
  - 3 pts for participation in class
- Example programs posted for each day on course web site

## What Does This Program Do?

- How can we make it easier to understand?

`program_before.py`  
`program_after.py`

## 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/
```



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