

## Objectives

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

## 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!

## 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

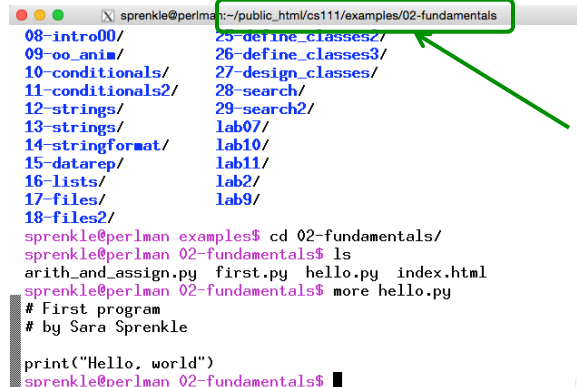
Jan 17, 2018

Sprenkle - CSC111

3

## 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



```
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 Sprenkle

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

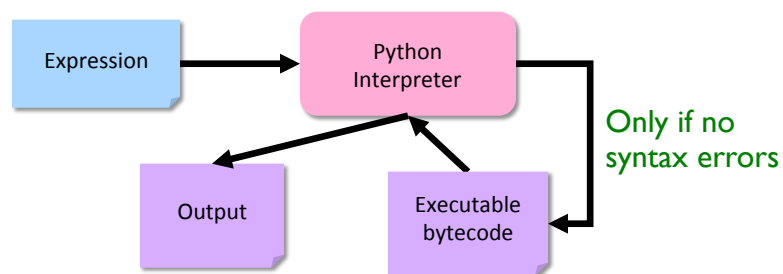
Jan 17, 2018

4

## Terminal Shortcut

## 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)



## Two Modes to Execute Python Code

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

## Python Interpreter: Interactive Mode

Run by typing `python3` in terminal

Python displays the result

Type in the expression

**Error Message:**  
We'll talk more later about why this is an error

`print`: Special function to display output

## IDLE Development Environment

- 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

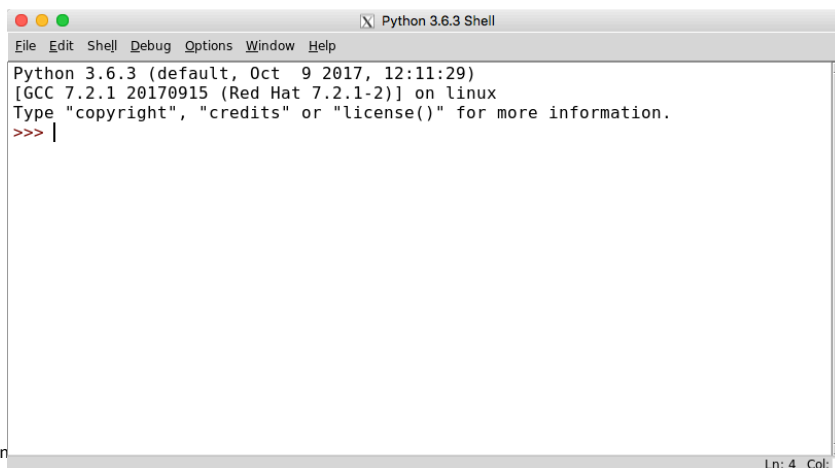
Jan 17, 2018

Sprenkle - CSCI111

9

## IDLE

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

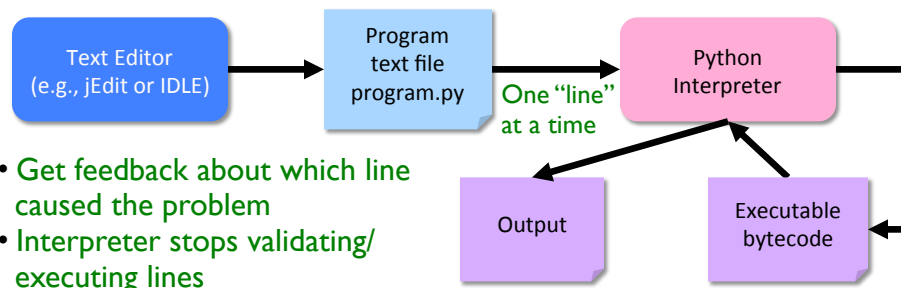


## Your Turn in Interactive Mode...

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

## Batch Mode

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



## Example Python Script

Text file named: `hello.py`

```
# A first program
# by Sara Sprenkle, 01/16/2018

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

Print statement

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

Jan 17, 2018

Sprenkle - CSCI111

13

## Example Python Script

```
# A first program
# by Sara Sprenkle, 01/16/2018
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

Jan 17, 2018

Sprenkle - CSCI111

14

## 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.

## 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 &**



## 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?

## What are the values?

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

```
➤ a = 5
```

```
➤ y = a + -1 * a
```

```
➤ z = a + y / 2
```

```
➤ a = a + 3
```

```
➤ y = (7+x)*z
```

```
➤ x = z*2
```

## What are the values?

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

➤  $a = 5$   
➤  $y = a + -1 * a$   
➤  $z = a + y / 2$   
➤  $a = a + 3$   
➤  $y = (7+x)*z$   
➤  $x = z*2$

### 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!\*

## Numeric Arithmetic Operations

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

Remember PEMDAS

## Programming Building Blocks

- Each type of statement is a building block

- Initialization/Assignment

- So far: Arithmetic

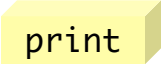
- Print

- We can combine them to create more complex programs

- Solutions to problems



Assign.



print



Assign.

print

Assign.

Assign.

print

## Bringing It All Together: A simple program

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

```
x = 3  
y = 5
```

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

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

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

arith\_and\_assign.py

## Bringing It All Together: A simple program

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

```
x = 3  
y = 5
```

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

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

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

arith\_and\_assign.py

Jan 17, 2018

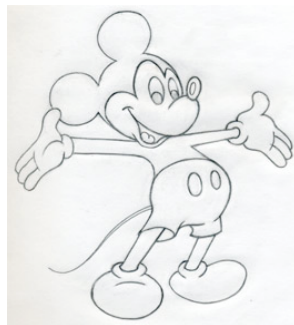
Sprenkle - CSC111

25

## Formalizing Process of Developing Computational Solutions

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

Use comments to describe the steps



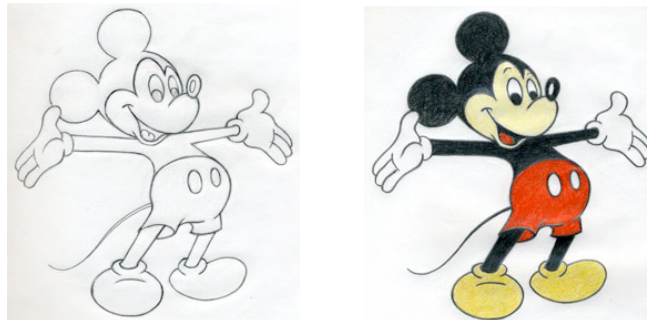
Jan 17, 2018

Sprenkle - CSC111

26

## 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



Jan 17, 2018

Sprenkle - CSC111

27

## Errors

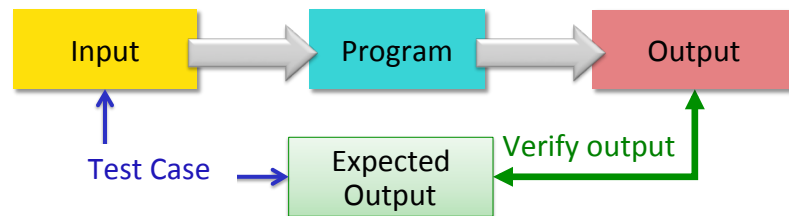
- Sometimes the program doesn't work
- Types of programming errors:
  - Syntax error
    - Interpreter shows where the problem is
  - Logic/semantic error
    - `answer = 2+3`
    - No, answer should be `2*3`
  - Exceptions/Runtime errors
    - `answer = 2/0`
    - Undefined variable name

Jan 17, 2018

Sprenkle - CSC111

28

## Testing Process

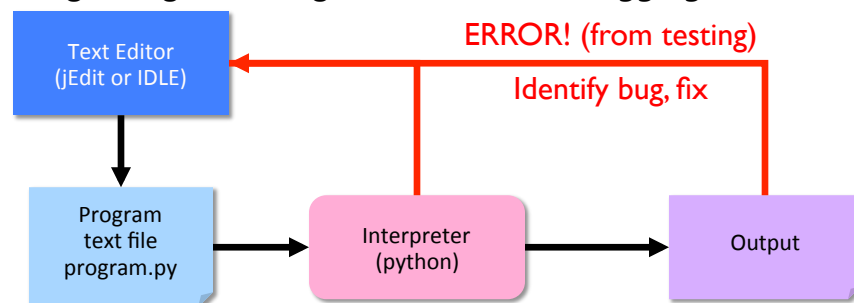


- Test case: **input** used to test the program, **expected output** given that input
- Verify if **output** is what you expected

If output is not what you expect...

## 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 error is called **debugging**



## Practice: A Computational Algorithm

- Find the average of two numbers

## Practice: A Computational Algorithm

- Find the average of two numbers
- Test cases:

Input		Expected Output
num1	num2	



## A Computational Algorithm

- Algorithm for finding the average of two numbers:
  - Hard-code two numbers
    - Later: get the two numbers from user
  - Calculate average
  - Print average
- Test cases for finding the average
  - Test both integers
  - Test with at least one float
  - Test numbers less than or equal to 0

Jan 17, 2018

Sprenkle - CSCI111

average2.py

33

## 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

average2.py

Jan 17, 2018

Sprenkle - CSCI111

34

## 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

## Lab 1 Expectations

- Comments in programs
  - High-level comments, author
  - Notes for your algorithms, implementation
- Nice, readable, understandable output
  - User running your program needs to **understand** what the program is saying
- Honor System
  - Pledge the Honor Code on printed sheets

## 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

- 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 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

Reintroduce lab assistants

## 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.

## Honors System: Rules of Thumb

- Discussion of problems/programs - OK
  - Clarification questions
  - Algorithm discussion (on paper, board)
- 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