Objectives

- Review Lab
- Introduction to
  - Problem solving
  - Programming languages
  - Writing Python programs

Review: Lab

- Learned some UNIX commands
- Created a Web page
- Started writing Python programs

- Lessons learned:
  - Problems are fixable (often just typos!)
  - Find a good solution

Review: UNIX

- UNIX is a bad parent
  - Doesn't tell you when you've done something right
  - Only tells you when you've done something wrong

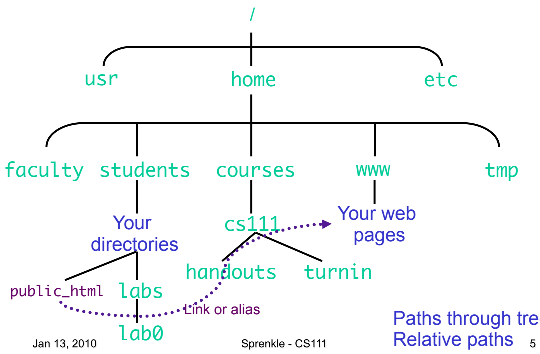
```
sprenkle@spartacus Desktop$ mv lab00.ppt.pdf lab00.pdf
sprenkle@spartacus Desktop$
```

RIGHT! Because didn't get an error message!

Review: Linux

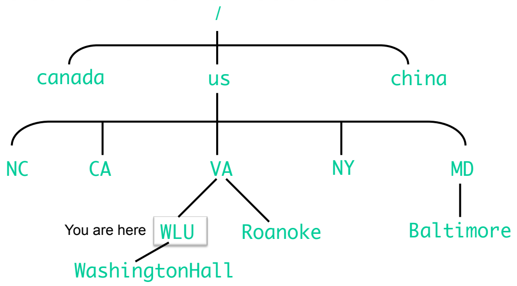
- How do you …
  - Learn more about a Linux command?
  - List the files in a directory?
  - Change your current directory?
  - Make a directory?
  - Find out the current directory?
- What is the shortcut for …
  - The current directory?
  - The parent directory?
  - Your home directory?

Review: Linux File Structure



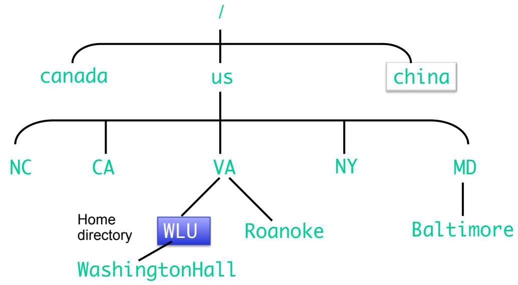
Paths through tree
Relative paths

Relative Paths vs Absolute Paths



- Given that you're at WLU, how would you get to Washington Hall? To Roanoke? To Baltimore?

Relative Paths vs Absolute Paths



- Given that you're in China, how would you go to Canada? WLU? Washington Hall?

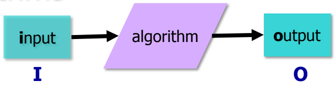
Computational Problem Solving 101

- Computational Problem
  - A problem that can be solved by logic
- To solve the problem:
  - Create a **model** of the problem
  - Design an **algorithm** for solving the problem using the model
  - Write a **program** that *implements* the algorithm

Computational Problem Solving 101

- Algorithm: a well-defined recipe for solving a problem
  - Has a finite number of steps
  - Completes in a finite amount of time
- Program
  - An algorithm written in a **programming language**
  - Also called code
- Application
  - Large programs, solving many problems

More on Algorithms



- Algorithms often have a defined **input** and **output**
- Correct algorithms give the intended output for a set of input
- Example: Multiply by 10
  - I/O for a correct algorithm:

Input	Output
5	50
.32	3.2
x	10x

- More examples: averaging numbers, recipes

Making a Peanut Butter & Jelly Sandwich

- How do you make a peanut butter and jelly sandwich?
- Write down the steps so that someone else can follow your instructions
  - Make no assumptions about the person's knowledge of PB&J sandwiches
  - The person has the following materials:
    - Loaf of bread, Jar of PB, Jar of Jelly
    - 2 Knives, paper plates, napkins

Discussion of PB&J

- The computer: a blessing and a curse
  - Recognize and meet the challenge!
- Be unambiguous, descriptive
  - Must be clear for the computer to understand
  - "Do what I **meant**! Not what I said!"
    - Motivates programming languages
- Creating/Implementing an algorithm
  - Break down pieces
  - Try it out
  - Revise

Discussion of PB&J

- Be prepared for special cases
  - Any other special cases we didn't discuss?
- Aren't necessarily spares in real life
  - Need to write correct algorithms!
- Reusing similar techniques
  - Do the same thing with a little twist
- Looping
  - For repeating the same action

Parts of an Algorithm

- Input, Output
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

An overview for the semester!

Other Lessons To Remember

- A cowboy's wisdom: Good judgment comes from experience
  - How can you get experience?
  - Bad judgment works every time
- Program errors can have **bad** effects
  - Prevent the bad effects--especially before you turn in your assignment!

Computational Problem Solving 101

- Computational Problem
  - A problem that can be solved by logic
- To solve the problem:
  - Create a **model** of the problem
  - Design an **algorithm** for solving the problem using the model
  - ➔ Write a **program** that *implements* the algorithm

Why Do We Need Programming Languages?

- Computers can't understand English
  - Too ambiguous

Live Jazz!

- Humans can't easily write machine code

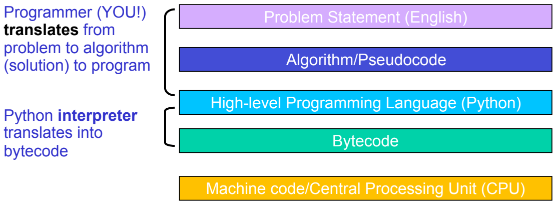
Problem Statement (English)

Machine code/Central Processing Unit (CPU)

000000 00001 00010 00110 00000 100000

Why Do We Need Programming Languages?

- Computers can't understand English
  - Too ambiguous
- Humans can't easily write machine code



Programmer (YOU!) **translates** from problem to algorithm (solution) to program

Python **interpreter** translates into bytecode

Problem Statement (English)

Algorithm/Pseudocode

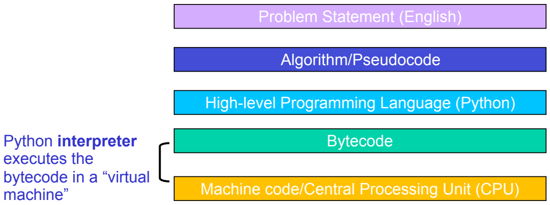
High-level Programming Language (Python)

Bytecode

Machine code/Central Processing Unit (CPU)

Why Do We Need Programming Languages?

- Computers can't understand English
  - Too ambiguous
- Humans can't easily write machine code


Problem Statement (English)
Algorithm/Pseudocode
High-level Programming Language (Python)
Bytecode
Machine code/Central Processing Unit (CPU)

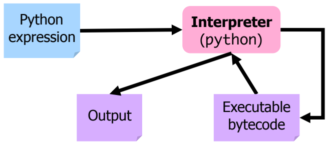
Python **interpreter** executes the bytecode in a "virtual machine"

Programming Languages

- Programming language:
  - Specific rules for what is and isn't allowed
  - Must be exact
  - Computer carries out commands as they are given
- **Syntax**: the symbols given
- **Semantics**: what it means
- Example: III * IV means 3 × 4 which evaluates to 12
- Programming languages are unambiguous

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*)



Parts of an Algorithm

- ➔ Input, **Output**
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

Printing Output

- `print` is a special command
  - Displays the result of expression(s) to the terminal
- `print "Hello, class"`

string literal

`print` automatically adds a '\n' (carriage return) after it's printed

---

Printing Output

- `print` is a special command
  - Displays the result of expression(s) to the terminal
- `print "Hello, class"`

string literal

`print` automatically adds a '\n' (carriage return) after it's printed

- `print "Your answer is", 4*4`
  - Displays same as:
    - `print "Your answer is",`
    - `print 4*4`

Syntax: commas
Semantics: print multiple "things" in one line

Next Time

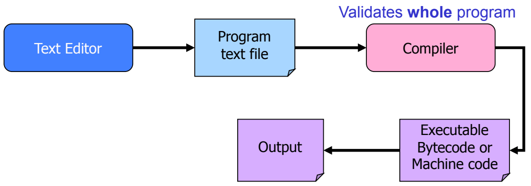
- More programming fundamentals
- Broader Issue: “**New Programs Aim to Lure Young Into Digital Jobs**”
  - Post write up on Sakai, as response to appropriate topic
  - Your write up will include
    - How interesting you found this article on a scale of 0 to 9
    - Summary of the 3 most important points
    - Article’s effect on your understanding of CS
    - Article’s relation to our course specifically (if applicable)
    - Question for class discussion
  - See Course’s CS Issues page for more information

Compiled Languages

- Examples: Java, C++, C, etc.
- Compile whole program into bytecode/ executable format
- Then, execute the bytecode/machine code

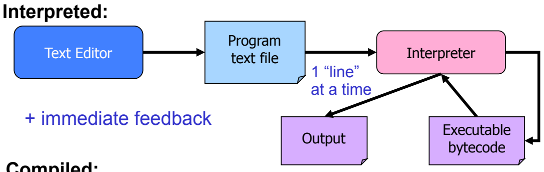
Compiled Language Programming Process

1. Compiler compiles program
   - Validate program, report syntax errors
   - Creates executable (bytecode or machine code)
2. Execute executable


Validates **whole** program
Text Editor
Program text file
Compiler
Executable Bytecode or Machine code
Output

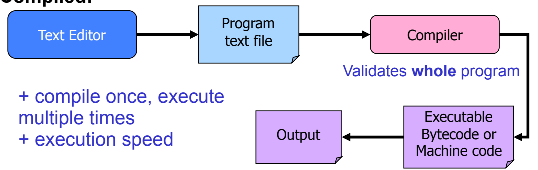
Compiled vs. Interpreted Languages

Interpreted:



+ immediate feedback

Compiled:


Text Editor
Program text file
Compiler
Validates **whole** program
Output
Executable Bytecode or Machine code

+ compile once, execute multiple times
+ execution speed

Python’s Implementation

- Combination of compiled and interpreted
- Interactive mode: interpreted
  - Validate, execute each line
- Python “interpreter” in script mode:
  - Compiles Python script into **bytecode**
  - Runs Python Virtual Machine that interprets the bytecode and executes