Objectives

- Introduction to
  - Problem solving
  - Programming languages

Typical Class Period Organization

1. Pearls of wisdom from Professor Sprenkle
2. Review in pairs
   - Consult your notes, slides from recent classes (see course web site)
3. Review as a class
4. New stuff!
   - Some think-pair-share work

Course Logistics: In-person Preferred

- Virtual for staggered start, quarantining, immunity concerns, sickness, …
- After class, ***sanitize*** your spot
  - Alcohol wipes in the back

Course Logistics: Handouts

- On Canvas under Files → handouts
- A few copies in back of the room
  - Copies: not great solution
- Notes
  - Slide number won't always line up with slides
  - Won't always get to all
  - Don't look ahead

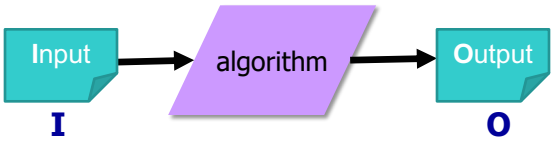
Lab Review

- What are the names of our student assistants and tech support person?
- What OS do the lab computers run?
- What is ssh?
- Why do we need an X server?
- What is computer science?
- What is this course about?
- What is an algorithm?

Review: 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
  - Large programs, solving many problems together → application

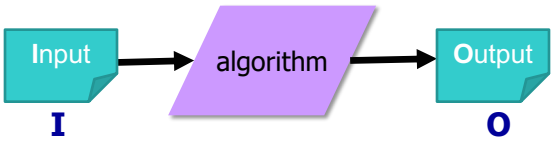
Algorithms: Input and Output



- 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:
- More examples
  - averaging numbers, recipes

Input	Output
5	50
.32	
x	

Algorithms: Input and Output



- 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:
- More examples
  - averaging numbers, recipes

Input	Output
5	50
.32	3.2
x	10x

How Do You Draw *X*?

- You were given a "secret object" in the email with your computer science account password
- Write down the instructions for someone to draw that object
  - Describe the steps that they would take
  - You cannot reveal what they are drawing
  - The person only has a drawing utensil (pen/pencil) and paper

Win, Lose, or Draw!

- Roles
  - **Instructor**: Read your instructions to your partner who will do what you say
    - You cannot look at what they're drawing
  - **Drawer**: Do *exactly* what instructor says and *only* what they say
- As far as you can get in 3 minutes
  - Then, switch roles!

Drawing Instructions are Algorithms!

- What was good about your algorithm?
- What would you change about your algorithm if you were to write it again?
- What did you learn about algorithms in general?

Discussion of Drawing Directions

- 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 Drawing Directions

- Steps need to be done in a particular order
- 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

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

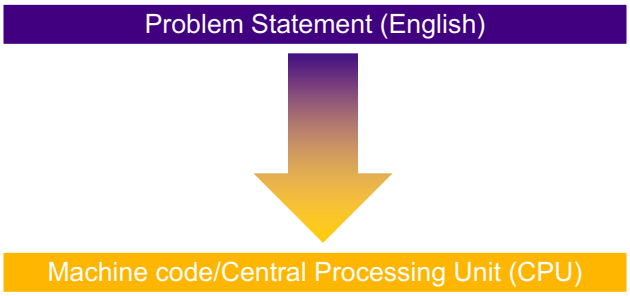
- **Computational Problem:**
  A problem that can be solved by logic
- To solve the problem:
  - Create a **model** of the problem
  - Design an **algorithm** for solving the problem using the model
  - Write a **program** that *implements* the algorithm

Why Do We Need Programming Languages?

- Computers can't understand English
  - Too ambiguous

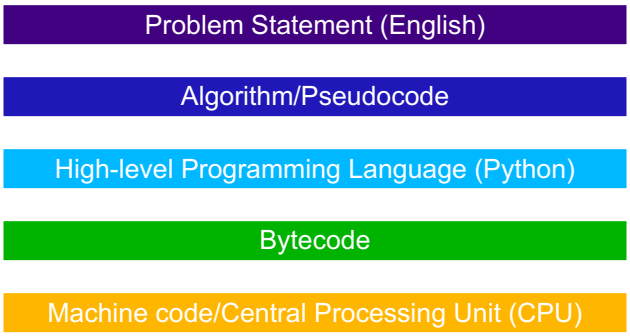
Live Jazz!

- Humans can't easily write machine code



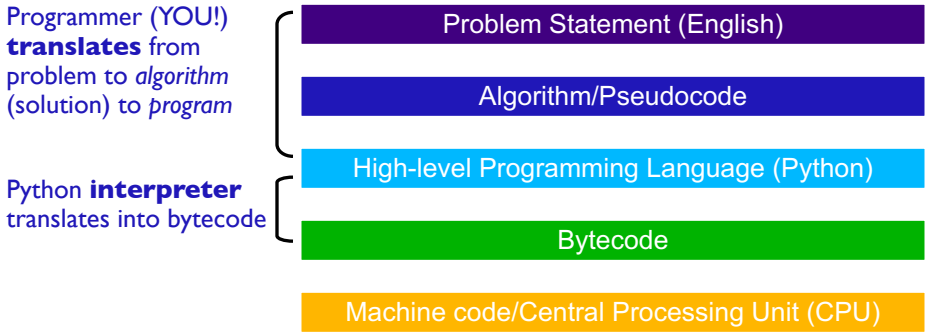
Why Do We Need Programming Languages?

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



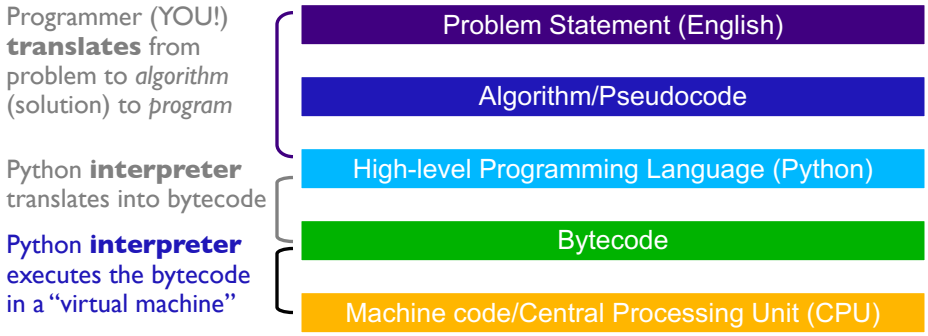
Why Do We Need Programming Languages?

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



Why Do We Need Programming Languages?

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

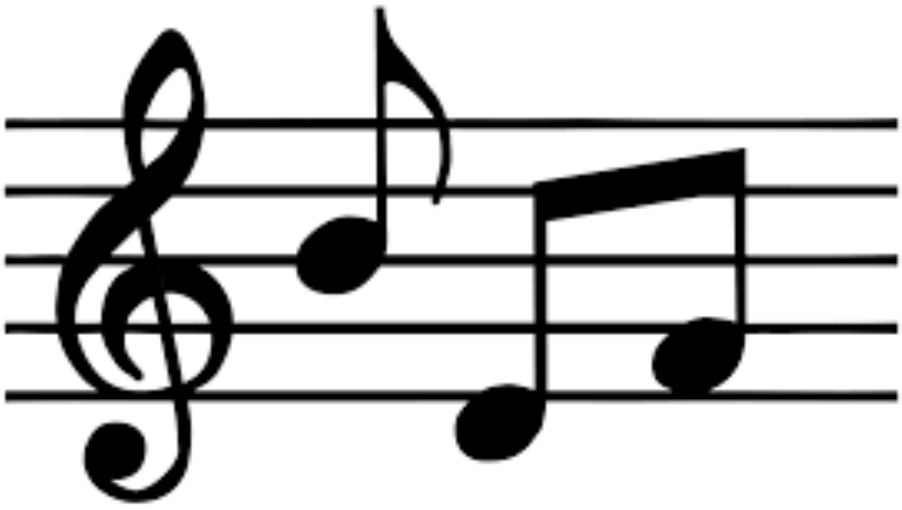


Programming Languages

- 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
  - `cp src dest` means copy the file named src to dest
- Programming languages are unambiguous

Another Syntax and Semantics Example

What is the syntax? What is the semantics?



Python Is …

- A **programming language**
  - 3rd most popular programming language

    http://www.tiobe.com/tiobe-index/

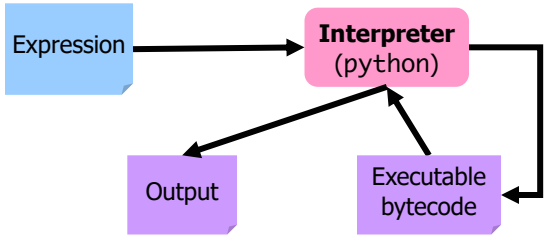
    January Headline: Python is TIOBE's Programming Language of 2020!
- An **interpreter** (which is a *program*) that understands and executes Python code

Python

- A common *interpreted* programming language
  - Runs on many operating systems
- First released by Guido van Rossum in 1991
- Named after *Monty Python's Flying Circus*
- Minimalist syntax, emphasizes readability
- Flexible, fast, useful language
- Used by scientists, engineers, systems programmers

Python Interpreter

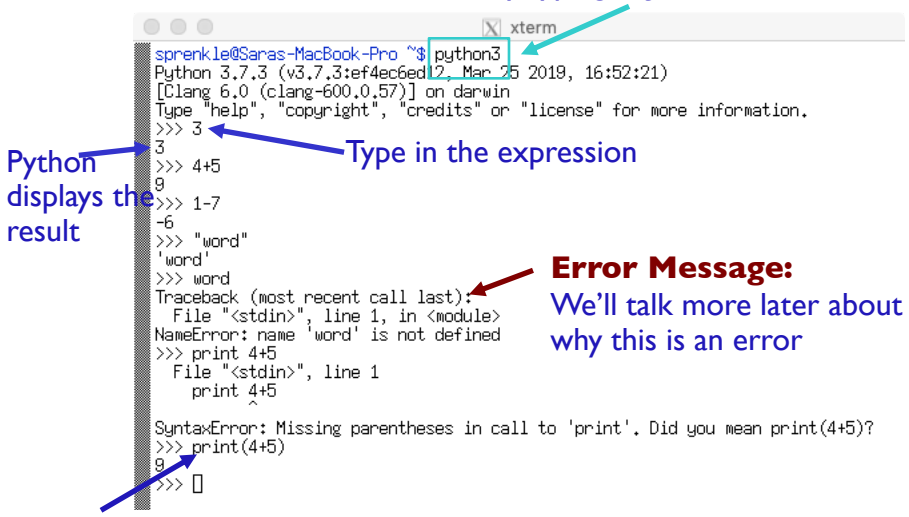
1. Validates Python programming language expression(s)
   - Enforces Python **syntax**
   - Reports **syntax** errors
2. Executes expression(s)
   - Runtime errors (e.g., divide by 0)
   - **Semantic** errors (not what you *meant*)



Two Modes to Execute Python Code

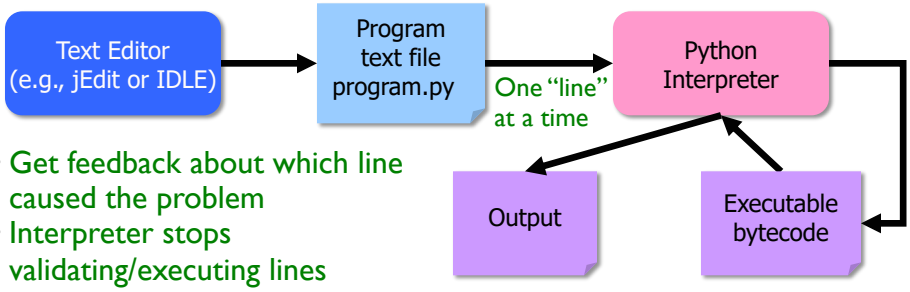
- **Interactive**: using the interpreter
  - Try out Python expressions
- **Batch**: execute *scripts* (i.e., files containing Python code)
  - What we'll usually write

Interactive Mode



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



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

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 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 set of double quotes
Semantics: represents text

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

Primitive Data Types

- Primitive data types represent data
- Python provides some basic or ***primitive*** *data types*
- Broadly, the categories of primitive types are
  - Numeric
  - Boolean
  - Strings

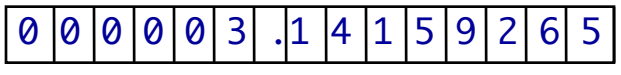
Numeric Primitive Types

Python Data Type	Description	Examples
`int`	Plain integers (32-bit precision)	-214, -2, 0, 2, 100
`float`	Real numbers	.001, -1.234, 1000.1, 0.00, 2.45
`complex`	Imaginary numbers (have real and imaginary part)	1j * 1J → (-1+0j)

How big (or small or precise) can we get?

- Computer cannot represent all values
- Problem: Computer has a **finite** capacity
  - The computer only has so much memory that it can devote to one value.
  - Eventually, reach a cutoff
    - Limits size of value
    - Limits precision of value

PI has more decimals, but we're out of space!



Example: in Python interpreter, .1 + .1 + .1 yields 0.30000000000000004.
* In reality, computers represent data in binary.

Strings: `str`

- Indicated by double quotes " " or single quotes ' '
- Treat what is in the " " or ' ' literally
  - Known as **string literals**
- Examples:
  - "Hello, world!"
  - 'c'
  - "That is Buddy's dog."

Single quote must be inside double quotes*
* Exception later

Booleans: `bool`

- 2 values
  - `True`
  - `False`
- Much more on these later…

What is the value's type?

Value	Type
52	
-0.01	
4+6j	
"3.7"	
4047583648	
True	
'false'	

Looking Ahead

- Lab 0 due Friday
- Survey (on Canvas) due Friday
- Broader Issue write up on Canvas due Friday at 11 a.m.