Introduction to Python

# Knowledge

* What does a computer do?
  + Perform **calculations**
  + **Remembers** results
* Kinds of calculations
  + Built in language
  + Ones defined by programmer
* Simple calculations aren’t enough to solve problems
* Good algorithm design is needed to accomplish some tasks
* Computers have limits
  + Problems too complex
    - Accurate weather prediction at a local scale
    - Some encryption schemes
  + Fundamentally impossible to compute
    - Predicting whether a piece of code will always halt with an answer for any input (check on this [here](https://en.wikipedia.org/wiki/Halting_problem))
* Types of knowledge
  + Computers know what you tell them
  + Declarative knowledge([wiki](https://en.wikipedia.org/wiki/Descriptive_knowledge))
    - Statement of fact
    - **Example**: There is candy taped to the underside of one chair
    - It declares or states something
  + Imperative knowledge
    - Is a recipe or “how-to”
      * Sequence of simple steps
      * Flow of control that specifies when each step is executed
      * A mean of determining when to stop
    - **Example**: A cooking recipe, instructions manual, etc.
    - AKA: An algorithm

# Machines

* Fixed Program

Machine does only one thing

* Stored Program
  + Machine stores and executes instructions
  + Sequence of instructions stored inside computer
    - Built from predefined primitive instructions
      * Arithmetic and Logic
      * Simple Tests
      * Moving Data
  + Interpreter
    - Executes each instruction in order
      * Uses tests to change flow of control through sequence
      * Stop when done
* Basic Machine Architecture
  + Memory
    - Where data is stored
  + Control Unit([wiki](https://en.wikipedia.org/wiki/Control_unit))
    - Has a program counter (controls what operation comes next)
  + Arithmetic Logic Unit
    - Does primitive operations
  + Input
  + Output
* Basic Primitives
  + You can compute anything using 6 primitives([wiki](https://stackoverflow.com/questions/28148562/what-are-the-six-basic-primitives-in-turing-complete))
  + Modern programming languages have more primitives
    - They abstract methods to create new primitives
  + Anything computable in one language is computable in any other programming language
    - The difference is that in some programming languages it can be easier to solve some kind of problems than others based on the primitives they have.

**Example**: They abstracted more methods to make some problems easier to solve.

# Languages

* A programming language provides a set of primitive operations
* Expressions are complex but legal combinations of primitives in a programming language
* Expressions and computations have values and meaning in a programming language
* Primitive constructs
  + In English we have words
  + In a programming language we have numbers, strings, simple operations, etc.
* Aspects of language
  + Syntax

How phrase is legally constructed

* + Static Semantics

Syntactically valid phrase has meaning

* + Semantics

Meaning associated with a syntactically correct phrase with no static semantic error

Programming languages have one meaning but may not be what programmer intended

* Things go wrong in programming
  + Syntactic errors
    - Common and easily caught
  + Static semantics errors
    - Some languages check for these before running program
    - Can cause unpredictable behavior
  + No semantic error but different meaning than what programmer intended
    - Program crashes, stops running
    - Program runs forever
    - Program gives and answer but different than expected
* Goal of course
  + Learn the syntax and semantics of a programming language
  + Learn how to use those elements to translate “recipes” for solving a problem into a form that the computer can use to do the work for us
  + Learn computational modes of thought to enable us to leverage a suite of methods to solve complex problems

# Types

* A **Program** is a sequence of definitions and commands
  + Definitions are evaluated
  + Commands are executed by the Python interpreter in a shell
* **Commands** (Statements) instruct interpreter to do something
* Can be typed directly in a shell or stored in a **file** that is read into the shell and evaluated

# Variables

# Operators and Branching