**Introduction to Computer Science (Lecture 1)**

* Types of knowledge:
  + Imperative: the description of how to do something
  + Declarative: statements of fact or assertions of truth
* Earliest computers: fixed programmed computer
  + Has a piece of circuit designed to do a specific computation (i.e. a calculator)
* Stored Program Computer: allows you to provide to the computer a sequence of instructions describing the process you want it to execute
  + Has a sequence of instructions in it
    - Once it gets through one, it goes to the next one
  + Has a memory. Connected to:
    - Control unit
    - ALU
      * Input / output
    - Program counter
      * Basis of computation
      * Points to some place in memory. Typically to the first instruction of the sequence
* Program is a recipe. A sequence of instructions
* Within the language:
  + Syntax: what are the legal expressions in the language
  + Static Semantics: Which programs are meaningful; which expressions make sense.
  + Semantics: what does the program mean; what’s going to happen when you run the program
* Values (Primitive data): numbers (123), and strings (‘abc’)
  + Numbers: Integers, floats
  + Can do various operations with numbers
* Variables: stores values
  + myName = ‘John’

**Lecture 2**

* Combine in expressions: operands and operators
* Interpreter evaluates and prints
* Inside a code (a script) there is no print unless you specify the code to do so
* Type conversion: if you want to convert the type a primitive, you have to specify it to do so
  + Type checking
* Operator precedence holds (PEMDAS)
* Ability to create variables is done with an assignment statement (x = 5)
  + Links the variable x to the value 5.
* Variable types inherits it from its value. The type value is dynamic
  + So don’t change types arbitrarily!! (can cause bugs)
* Statements: legal commands that the programming language can interpret
* Comment the code to help the reader understand what’s going on in the code
* Use meaningful variable names (again, to make the code readable)
* Straight line program: a program we can execute the sequence of instructions one by one
* Branching program: something that can change the order of instructions based on some test
  + The test is usually a value of a variable
  + Conditional statements
  + If / else statements
* Iterations or loops
  + While loops (continues executing while the statement holds)
    - Can lead to infinite loops
  + Check to see if the condition is true. If it is true, then execute the code

**Lecture 3**

* Data:
  + Numbers
  + Strings
  + Booleans
* Operations:
  + + / -
  + \*
  + and/ or
* Commands:
  + Assignment
  + Input/ output
  + Conditionals
  + Loop mechanisms
    - While
* Iterative programs:
  + Choose a variable that is going to “count”
    - What is the thing that is going to change when you execute the loop
  + Initialize the variable outside of the loop
  + Set up the right end test
    - How do you know when you’re done with the loop?
      * Deals with the variable
  + Construct the block of code
    - The same set of instructions that is going to be executed
    - Change the variable somewhere in there
  + Decide what to do when you’re done
* Assure yourself the loop will always terminate and if it does, make sure the answer you get makes sense
  + Defensive programming: make sure you’re going through all paths of the code, that you’re returning out useful information within the code, and make sure for all possible inputs you’re not breaking the code
    - Always test to make sure you catch any and all mistakes
* Exhaustive enumeration: try all “reasonable” values until you find the solution
* For loop:

For some variable in some collection:

*Execute this block of code*

* + As long as the collection is finite, the loop will end
* Tuple: ordered sequence of elements
  + Immutable (cannot be changed)
  + X = (1, 2, 3, 4)
  + Selection: X[0] gives you 1
* Strings also support selection and slicing
  + String are an ordered collection of characters
  + Can use a loop to get through the individual characters

**Programming 101**

* Binary data:
  + Works with base 2
  + Bits: either 0 or 1
    - How data is stored in memory
  + Bytes: 8 bites
* How is 145 represented in binary data? (use an unsigned 8 bit system)
  + 10010001
* Bit unsinged system- you can only have positive numbers
  + Range is 0 to 255
* Signed bit system- allows you to do positive or negative numbers
  + Range is -127 to 127
    - Take off the 128 bit and use that to choose whether the number is going to be positive or negative
      * 0 is positive, 1 is negative
      * 10000001 = -1
  + Cannot be used in the system because you can have -0: 10000000
* GPU- Graphics processing unit
  + Processes the bits and produces a pixel on the screen
* SPU- Sound processing unit
  + Processes the bits and sends a signal to the speakers and outputs a sound wave
* CPU- Core processing Unit
  + Deals with logic
    - Where you can perform complex calculations
* Firmware- physically program or software that exists as positive or negative charges (0s or 1s)
  + BIOS- Basic Input Output System
    - Receive input and provide output
* Types of memory
  + Volatile- Temporary memory
    - RAM (random access memory)
    - When a user simply writes a file
  + Nonvolatile- Persistent memory
    - Hard drive
    - When a user wants to save a file
* Types of connections:
  + Wired connection
    - LAN (local area network) cables (also called an Ethernet cables) can connect your computers together to transfer data
      * Creates a local network
      * Can also connect your computer to a router and similarly transfers data
  + Wireless connection
* Types of networks
  + Local Network: Intranet
  + Global Network: Internet
* IP Address- used for computers to connect over a global network
  + Unique for each device
* Routers have the public IP address and in turn provides the different devices connect to the router with a private IP address
  + Router routs the different package to and from your devices
* MAC (media access control) address- unique number for each device
  + Better for identification since IP addresses can change
  + 48 bits
* All packets are written in HTTP
* Operating system controls how all miniature programs communicate with one another to make the system work
* API (Application Programming Interface): allows for different programs to communicate with one another
  + External or internal APIs
* Console- text interface
  + Allows you to communicate with the computer (gets information from the computer)
  + Every operating system has a console window (the terminal or command prompt)
* Complier- takes the programming language (human readable language) and turns it into machine language—something the computer can understand
  + Different for every language because the syntax can vary within languages
  + Doesn’t stop when it encounters a problem with the code. It just keeps going.
* Trans-complier (transpiler)- takes the human readable language and makes it into another human readable language
  + Takes HAXE code and can be put into the transpiler to be changed into another programming language such as C#
  + Converts the HAXE syntax to the other language’s syntax
    - You then put it through the corresponding language’s complier to use the code
  + Only have to write one program
* Interpreter- interprets the code line by line. Stops when it encounters a problem with the code
  + Can be slow if there are many problems with the code
  + JIT-Complier combines the Interpreter and the Complier to make the process quicker
    - Helpful for debugging
* Types of languages
  + Programming languages and scripting languages
    - Made for programmers to define and run sets of instructions and provide logic
    - Programming languages are pre-compiled languages
      * Distributed in machine language rather than the source code
      * Can sometimes make it harder a certain program to work on a different operating system
      * Java, C, C#, C++
    - Scripting languages- source code is distributed
      * Interpreted on each computer and then complied
      * JavaScript, PHP, SQL
  + Markup languages
    - NOT programming languages
    - Presentational languages (HTML, CSS)
    - Defines the structure of the page
* Client vs server side languages
  + Client language- a language where it is complied on your computer
    - JavaScript
  + Server language- a language where it is complied on your server
    - PHP, Ruby
    - You see the output of the code, not the code itself
* Higher level vs lower level language
  + Higher level- Programming languages; languages which we can understand
    - Don’t worry too much about the hardware
    - Send it to the complier for the operating system
  + Lower level- closer to machine level instructions (machine code or assembly code)
    - Specific for that piece of hardware
* Primitive data (noun, adjectives)
  + String: inside a single or double quote. Can contain letters or numbers
  + Integers: whole numbers
  + Float: a number with a decimal point
  + Boolean: true or false
* Function- allows us to perform actions in the program (verbs)
  + Contains a list of instructions within the body
  + Can have a return statement
    - Once it’s hit, it stops executing the rest of the code
    - Normally at the end of the function
  + Called by its name
  + Can have arguments passed in to make it more adaptable
* Object- a collection of nouns, adjectives, and verbs
* Variables and constants- different types of containers in programming languages (the box)
  + Can be addressed by their name
  + Has data stored in it (the content in the box)
  + Variables- vary in value; can be changed
  + Constants- cannot be changed
* Strict vs loose programming languages:
  + Strict languages- allows you to lock the data type
    - Only allows for a variable to contain a certain data type
      * Ex. Cannot switch from integer to Boolean
  + Loose languages- allow you to change the data type
    - Allows for a variable to change data types
      * Ex. Can switch the data stored in a variable from integer to Boolean
* Objects encapsulate nouns, adjectives, and verbs
  + Can be owned or possessed
  + Has properties which are variables (adjectives of the object)
  + Has methods which are functions (verbs of the object)
  + Can encapsulate other objects as well
    - Object hierarchy
* Arrays- objects that register similar types of data
  + Values are elements
  + Iterable for processing the data
* Operators invoke functions within the complier
  + +, -, \*, /, %
* Paradigm- a typical example or pattern of something; a pattern or model
  + You can mix and match paradigm in order to best suit your application
* Assembly Paradigm- line upon line; precept upon precept
  + Executed line by line
* Procedural paradigm- set of subroutines
  + Break it down into smaller chunks of code that can be run at a later date
* Functional paradigm- functions only
* OOP (object oriented programming)- all about manipulating and accessing objects
* Classes and constructor functions build new objects
  + Template to build objects
  + Allows you to reuse the code to create an object with the same properties
  + Can then modify the object