How to teach programming:

1. Define computer terms!
   1. Hardware
      1. Physical pieces of the computer
         1. External
            1. Monitor
            2. Keyboard
            3. Mouse
            4. Tower
            5. Printer
         2. Internal
            1. CPU ( processor )
            2. Motherboard
            3. Main Memory ( RAM )
            4. Secondary Storage ( Harddrive )
   2. Software
      1. Non physical pieces
         1. Applications
            1. iTunes
            2. Internet Explorer
            3. Pac-Man
         2. Systems Software ( Operating systems)
            1. Mac OS
            2. Windows XP, 7, 8
   3. I/O
      1. Input / Output
         1. Input
            1. Keyboard
            2. Mouse
            3. Scanner
         2. Output
            1. Monitor
            2. Printer
2. Programming
   1. High-Level
      1. Human written
      2. Source code – ( text authored by a person )
   2. Low-Level
      1. Machine language
      2. Hardware specific
      3. Not easy to read
   3. Compiled (Java, C++, Objective-C)
      1. Write code
      2. Compile ( convert from high-level -> low-level )
      3. Execute the low level program on the hardware
   4. Interpreted (Python, Perl, Ruby)
      1. Write code
      2. Run each line of code through an interpreter
      3. Each line runs on the hardware, one at a time
3. Designing a Program
   1. Design
      1. Flowcharts
         1. Functionality, business side of things
         2. UML terms
            1. Oval- terminator

Start/end

* + - * 1. Rectangle- process

Basic functions

* + - * 1. Parallelogram – data

i/o

* + - * 1. Diamond – decision

Decision made, path chosen

* + 1. Pseudocode
       1. Fake code, human terms
  1. Write code
     1. Functions
        1. A block of code to perform a specific task
     2. Variables
        1. Live in source code
        2. Have values and names
        3. Declared
           1. Created
        4. Initialized
           1. Given an initial value
     3. Algorithm
        1. Specific set of steps to accomplish a specific task
  2. Syntax Errors
     1. Typed errors
  3. Test Code
     1. Unit testing
  4. Logic Errors
     1. Functionality described incorrectly

1. Functions- group or collection of programming statements designed to perform a specific task
   1. Example: print(“hello”)
      1. Print- name of the function
      2. Always ()
      3. “hello” – argument
      4. parameter – within the function
   2. Two steps
      1. Define function
      2. Call function
   3. Two types of functions
      1. Pre-built
         1. Input, print, etc
      2. Customized
         1. User built
   4. Add example:
      1. Add(num1, num2)
         1. Sum = num1 + num2
         2. Print(sum)
   5. Scope
      1. How long does something live in your program
   6. Modularize our code
      1. Segment out specific pieces of functionality into their own named functions
   7. Main function
      1. Heart of your program
2. Decision Logic
   1. Permits you to build flow in your program with Boolean questions
   2. If / else if / else
   3. Decision Structures
      1. Question: True or False?
      2. How to implement a decision in source code
   4. Boolean logic
      1. Data type: true or false?
   5. Quiz!
      1. Quiz on if statement
      2. Function call
      3. Variable assignment
3. Repetition Structures
   1. Goal- repeat a block of code until a condition is no longer true
   2. Flag- a variable to manage your loops
   3. While loop
      1. Condition controlled loop – boolean test
      2. While the condition is true, the same code will repeat
         1. Each repeat is an iteration
      3. Iterates 0 – pseudo infinite times
   4. For loop
      1. Count controlled loop
      2. Iterates a specific number of times
      3. Target variable, counting variable
      4. Accumulator variable, running total
4. Modules
   1. Examples: math, random
   2. Black box, we don’t care how it works we just want to use the function
   3. Dot notation: how to call a function, module\_name. function\_name()
5. Files and exceptions
   1. If the code cannot continue during runtime, we throw an exception
   2. File
      1. Open, process, close the file, usually process with a loop! How novel :3
      2. Text
         1. Encoded with characters, readable
      3. Binary
         1. 1’s and 0’s, unreadable
      4. Sequential access, direct access.
      5. File objects – go into OOP ahhhhh
      6. ‘r’ –read, ‘w’ –write, - ‘a’ – append
         1. overwriting with w
         2. data preserved with a
   3. Oh shit objects already?!,
      1. Human
         1. Hair
         2. Weight
         3. Height
         4. Name
      2. Dave is an instance of the human object
      3. Andy is another instance of the human object
      4. Attributes (variables)
      5. Methods ( functions
      6. Instantiate- create an instance of the object
      7. Dot notation, string objects
6. Sequences, Lists, Tuples
   1. Sequences – collection of data represented by a name
   2. List – mutable sequence ( can be changed )
      1. Do not have to be the same data type
      2. Allowed to have a list of lists ( nesting lists)
      3. Range of indices values is not the size of your list
      4. Explain 2-d lists and how they are referenced
      5. IndexException, ValueException
      6. Copying a list- addresses and things
      7. Contains elements
   3. Tuples = immutable sequence ( cannot be changed )
7. Strings
   1. Case sensitive
   2. Spaces take up index values
   3. isDigit, isUpper, isLower….
   4. Similar to a list
8. Dictionaries
   1. Mutable
   2. Key value pairs. D = { 3626338:”Andy Beers”} – one element
   3. Edit the values NOT the keys
9. Objects so far
   1. Lists
   2. Tuples
   3. Files
   4. Strings
10. Sets
    1. Unordered
    2. All items must be unique
    3. Can be of different data types
    4. Eliminate data redundancy
    5. Operations
       1. Union – coverage (or)
       2. Intersection – overlap (and)
       3. Difference – exclusive
11. Serializing Objects
    1. Pickling
12. OOP
    1. Creating classes
    2. Instances of objects
13. Questions
    1. How do I access a local variable outside of its scope?
       1. Her answer: no can’t, ramble about scope
       2. My answer: no can’t, but you can return the value
    2. Are parentheses the way a program knows it’s a function?
       1. Her answer: yes
       2. My answer: no, def is the keyword that defines a function () is syntax
    3. Where do built-in functions live?
       1. Her answer: Modules that are included in the files, basic functions are in the interpreter
       2. My answer: included
    4. Arguments in math expressions
14. Chapter 1
    1. Hardware
    2. I/O
    3. Software
    4. Types of programming languages
15. Chapter 2
    1. Built-in function calls
    2. Variables
    3. Data types
16. Chapter 3
    1. Functions you create
    2. Define -> call
    3. Debugging is easier, code reuse
17. Chapter 4
    1. Decision structures [if]
18. Chapter 5
    1. Repetition structures [while,for]
19. Chapter 6
    1. Value returning functions
    2. modules
20. Chapter 7
    1. File I/O
    2. Exception handling
21. Chapter 8 – 10
    1. Sequences
       1. Lists
       2. Tuples
       3. Strings
       4. Dictionaries
       5. Sets
    2. Data storage
22. Chapter 11
    1. OOP