1. LANGUAGES
   a. To specify a recipe, we need a language.
   b. Formal versus natural languages.
      i. Natural languages - English, French - were not designed by people. They evolved naturally.
      ii. Formal languages are designed for people for specific applications. Mathematicians used formal
          languages to specify relationships between numbers. Chemists use a formal language to represent the
          chemical structure of molecules.
      iii. Programming languages are formal languages designed to express computations. They are meant to
          communicate instructions to a machine. (Note that programming languages come before computers.)
   c. The description of a programming language is split into two components:
      i. Syntax (form) - the possible combinations of symbols that form a correct program. (Context-free
         grammars; so the interpreters work like finite-state machines, cf. Q250).
         1. Formal languages have strict rules about syntax. 3+3=6 is syntactically correct. 3+=3%6 is not.
         2. There are two types of errors that occur in syntax.
            a. Illegal tokens.
            b. Invalid structures.
      ii. Semantics (meaning) -
   d. There are thousands of different programming languages.
      i. Markup languages (e.g., XML, HTML) are not generally considered programming languages.
      ii. There is a whole taxonomy of programming languages or paradigms. (We will get back to this
          taxonomy later on in the course.)

2. DEBUGGING (part 1)
   b. Errors in the static semantics. Meaningful expressions. “My desk is Susan.”
   c. Errors in the full semantics. Expressing what you are meaning to communicate.
   d. (We will return to strategies for debugging later on in the course).

3. PYTHON
   a. Why Python?
      i. Platform independent.
      ii. Easy and clean syntax.
      iii. Highly readable.
      iv. Rapid development time.
      v. Scalable.
      vi. Object-oriented.
      vii. Dynamic type.
      viii. Automated memory management.
      ix. Extendable and embeddable.
      x. Widely used and large community.
      xi. Free and open source.
   b. Make sure you can program from your own computer. You can download Python here.
   c. There are two main different versions, Python 2 and 3. We’ll be using Python 2 (version 2.7.3).
   d. Different ways of running Python programs.
      i. Interactive.
         1. It evaluates and prints.
         2. You can use the interactive interpreter in IDLE or in a Terminal.
         3. Three useful keyboard shortcuts: control-c, control-p, and quit().
      ii. Script.
         1. No automatic printing.
         2. Scripts can be run in IDLE or in a Terminal.
   e. Saving and running a script.
      i. print “hello, world” example
   f. Assignment hand-in clarification.
4. VALUES
   a. Types
      i. Numbers (Integers, Floating points)
      ii. Strings (strings can include numbers)
      iii. Booleans
   b. Function type.
      i. type(2), type(‘abc’), type(‘True’)
   c. Those are the basic types. But as we go along, we'll build up this taxonomy.
   d. No matter how complex the data structure. All of them, at their basis, is going to be some combination of those primitive data structures.
   e. With each type, there's a different set of operators that can deal with it.

5. OPERATORS
   a. Arithmetic: +, -, *, /, **
   b. Show an expression. E.g., 5*7
   c. Different operators for different types.
   d. But sometimes operators can be overloaded (e.g., ‘ab’+’ab’, 3*’ab’).

6. SIMPLE EXAMPLES
   a. 3*5, 3**5
   b. 3/5 = 0 ?? It's doing integer division.
   c. 3.0 / 5 = 0.599999..
   d. Similar things for strings
   e. concatenate strings

7. ERRORS (revisit)
   a. Syntax error. E.g., print 52a (invalid token). E.g., 52 print (invalid structure).
   b. Static semantic error. E.g., 3+'ab'

8. TYPE CONVERSIONS
   a. str(3) + ‘ab’
   b. int(‘3’) + 3

9. OPERATOR PRECEDENCE
   a. All other things being equal, exponentiation, before multiplication and division, before addition and subtraction.
   b. You don’t need to know this by heart. When in doubt, use parenthesis.

10. VARIABLES
    a. We would like to store values. In order to operate on them later on. Cook metaphor.
    b. mystring = 'eric'
    c. Start with letter or underscore (not numbers). No spaces anywhere.
    d. Some 28 keywords are blocked.
    e. Good practice: meaningful names.
    f. a = 1, b = a, what is b?
    g. a = 2, now what is b?
    h. The type of a variable is inherited from its value.

11. INTERACTIVE MODE AND SCRIPT MODE
    a. minutes = 59.0
    b. minutes/60   (print minutes/60)

12. SEQUENTIAL PROGRAMS
    i. E.g., m = 3 #(slope), b = 6 #(intercept), x = 1, y = m*x + b

13. COMMENTS
    a. Useful
b. Not too obvious.
c. Something that will allow someone else to see your code and know what it does.

14. IMPORTING LIBRARIES (or Modules)
   a. import math
   b. Variables: math.pi
   c. Functions: math.sqrt(2), log, exp, sin
   d. You can find out what functions are implemented in the documentation (or google: math python).

15. CREATE YOUR OWN FUNCTIONS
   a. Use: def functionname():
   b. Calculate the y value of a function, given x.
   c. How to compile (F5).
   d. How to add arguments.
   e. How to return results for later use.