Introduction to Computing is the entry course for CS majors but is also taken by a wide variety of non-CS majors. No course pre-requisites are required to take this course, so it must cater to students with differing technical backgrounds and experience. Establishing a firm base of programming knowledge is intrinsic to this course with an emphasis on teaching students how to design, code, and test well at an introductory level. Core introductory computer science concepts are also introduced. The current programming language taught is Python 3, but the choice of language is secondary and may change with department approval. As such, this document is written generally and applicable to most programming languages. The course includes 3 hours of lecture and 2 hours of “hands-on” lab per week.

Course Description

Introduction to software development through algorithmic problem solving and procedural abstraction. Programming in the small. Fundamental control structures, data modeling, and file processing. Significant emphasis on program design and style.

General Course Goals/Outcomes

1. Students can analyze a programming problem and create a programming solution using the basics of programming: actions (input, processing, output), control structures (selection, repetition), basic data structures (strings, arrays/lists, hash tables/dictionaries), modularization (student-written procedures/functions), and data persistence (file input and output).

Specific Outcomes/Assessments

1. Produce code for a problem requiring basic action elements (input, processing, output).

2. Produce code for a problem requiring basic control structures (selection, repetition).

3. Produce code for a problem requiring the use of nested basic control structures (selection, repetition).

4. Produce code for a problem requiring the use of basic data structures (strings, arrays/lists, hash tables/dictionaries).

5. Produce code that can read, process, and write data to/from persistent storage.

6. Produce a design document that “breaks up” a larger programming problem into a series of procedures/functions.

7. Produce code for a problem using procedures/functions.
8. Trace code using basic action elements, control structures, data structures, files, and/or procedures/functions and describe what it does in English.

9. Debug and correct code using basic techniques (understanding of runtime errors, code tracing, print statements, varying inputs).

Specific Course Goals/Content

Programming concepts:

• Data & actions
  - Declaring variables of appropriate type (Integer, Float, Boolean, String) to accomplish a task
  - Using input (e.g. from the console) to accomplish a task
  - Using output (e.g. to the console) to accomplish a task
  - Processing input and data to accomplish a given task using primitive operations, properties, and functions for
    - Mathematical operations such as addition, subtraction, multiplication, division, integer division, modulus, exponential evaluation, etc.
    - Boolean operations <, >, ==, >=, <=, and, or, not, etc.
    - Casting from one variable type to another.

• Control structures
  - Sequence—interpreting and producing code for relatively simple tasks (to learn basic actions)
    - As part of IPO (input, process, output) “problems”
    - As part of other control structures, e.g., initialization before loops and if sequences; processing/reporting after loops and if sequences, etc.
  - Selection—interpreting and producing code for tasks involving conditional execution of code
    - Various combinations of if’s: single if, sequence of independent if’s, sequence of connected if’s (if ... elseif ..., or generalized case logic)
    - Various conditionals: numeric, string, Boolean flags, and mixed
    - Nested selection statements
  - Repetition—interpreting and producing code for tasks involving repeated execution of code
    - Counting loops
    - Indefinite loops (using various conditionals: numeric, string, Boolean flags, and mixed)
    - Nested loops
  - Modularization—producing code modules and using them appropriately
    - Procedures
    - Functions
    - With & without parameters

• Use of built-in data structures, such as
  - Strings
    - Empty string, constructor, concatenation, indexing, manipulation, copying
- Searching strings (e.g. use of loops)
- Comparing strings (e.g. use of selection and Boolean statements)
- Possibly converting single characters to ASCII encoding and back

  o Arrays/lists
    - Empty array, constructor, indexing, manipulation, copying
    - Searching arrays (e.g. use of loops)
    - Comparing arrays and elements in arrays (e.g. use of selection and Boolean statements)

  o Hash tables/dictionaries
    - Empty hash table, constructor, manipulation (adding and removing elements), searching

• File programming
  o Designing and using files as input for given problems
  o Designing and using files as output for given problems

• Programming Design
  o “Break up” large programming problem into manageable chunks before starting to program
    - Use a top-down design tree or other design document tool to chart needed procedures/functions to be created
    - Write pseudo-code for procedures/functions to be created

Teaching Suggestions

• Use search and sorting algorithms as a way to exercise the use of loops and arrays.