ENGR 102 – Final Exam Learning Objectives

The second exam is cumulative.

Be prepared to answer questions regarding *definitions* or *behavior*, write *test cases*, *list steps* in a computation, *outline a design* and *write short programs* on paper. Expect to combine several covered topics into problem solutions. **Always follow good programming practices.**

Given a piece of Python code, be prepared to answer questions such as:

- what will the code output,
- what values are stored in memory, and of what data type
- what type a variable is,
- where there is an error or bug

In reviewing for the second exam, be familiar with all the topics we have covered. You should be able to:

Introduction to Programming

- Define the terms IDE, Compiler, and Interpreter
- Use the print statement in Python
- Define the purpose of, and utilize in a program, the import function in Python (e.g., `from math import *`)
- Use simple mathematical operations in Python (`+`, `-`, `*`, `/`, `//`, `%`, `**`, `sqrt`, `cos`, `sin`, `tan`, `log`, `log10`, `exp`)
- Use comments in a Python program

Sequential Steps, Variables and Assignments

- Utilize variables in Python programming to store data
- Apply Python naming rules and give descriptive names to variables
- Identify necessary programming variables from a given problem statement
- Assign or reassign values to variables
- Use the special assignment operators (`+=`, `-=`, `*=`, `/=`)
- Write a sequence of steps to successfully complete a complex task
- Follow a sequence of steps to understand the complex task that was performed

Input, Output and Data Types

- Identify the integer, floating-point number, Boolean and string data types in Python
- Know common usages of these data types in programming
- Create, read and use each data type within a Python program
- Use the escape character (`\`) in Python to define special characters in strings
- Convert between data types in Python, and know when an error will occur
- Find the value of a calculated expression during data type conversions
- Format output in good form using the print command (including use of %s, %f, %i string format)
- Concatenate strings using the + operator
- Modify the print item-separator and line-end commands (`sep = ""`, `end = ""`)
- Use the input command to read information from the user
- Provide a useful prompt to the user with the print command
- Use input information to perform calculations within Python
- Use the newline command (`\n`) in Python where appropriate

Conditionals and Boolean Expressions

- Identify a linear and a branching process from an ordering of steps or a flow chart
- Create and solve Boolean expressions using relational operators, following correct order of operations
- Create Boolean expressions in Python
- Use `if`, `elif` and `else` statements to correctly branch a Python program
- Identify correct instances to use `if-only`, `if-else`, and `if-elif-else` forms of branching in a program.
- Use nested conditional statements

Creating and Testing Programs

- Utilize comments in your program to section code, define and clarify variables or computations, refer to sources and improve readability
- Define incremental coding, pyramid- and arch-style software construction, and the terms typical, edge and corner as they relate to program testing
- List major steps of solving a problem
- Structure a program by using comments from the problem-solving step list
- Write test cases prior to writing programming code that will incrementally verify your program
- Write tests that demonstrate your program calculates correct results for a wide variety of cases, including unlikely scenarios
- Create and test code incrementally

Loops and Iteration

- Identify a repetition process from an ordering of steps or a flow chart
- Create a while loop in Python, and correctly form the conditional statement
- Recognize an infinite loop, and how to correct one
- Create a for loop in Python, and correctly define a range for iteration
- Identify values of variables and iterators for each iteration through a loop
- Use nested looping statements

Lists of Data and List Operations

- Create and identify lists in Python
- Assign or reference a single element within a list
- Find the length of a list using the len() command
- Loop through the elements of a list where the iterator is the index value through a range
- Loop through the elements of a list where the iterator is the value of each list element
- Create and index lists of lists
- Use the list operations of delete, append, insert, and concatenation
- Slice a list to print, delete, or replace portions of a list or create a new list
- Use the shortcut methods of slicing to refer to the start or end of a list (e.g., `[:b]`, `[a:]`, `[:]`)
- Slice a string to print or define a new string
- Recognize limitations of slicing when used on string data types and when errors will occur
- Create and identify dictionaries in Python
- Define and identify key-value pairs of a Python dictionary
- Assign or reference a single element within a dictionary
- Use a `for` loop with a dictionary in Python
- Use the `in` command with a dictionary or list in Python

Top-Down Design of Programs

- Create appropriately detailed top-down hierarchies when given a primary (complex) goal
- Explain and give examples of advantages and disadvantages of top-down design
- Define and create trees, roots, nodes, parent, children and leaves in this context
- Utilize a top-down design approach as part of creating a Python program

File Input and Output

- Use 'open' and '.close' commands and 'with open()' commands within Python to read and write external files
- Define and utilize file open designators (e.g., `'r'`, `'w'`, `'a'`)
- Write to files using write command
- Read from files using `read`, `readline`, `readlines` commands
- Use for or while loops to read complete or required contents of a file
- Process strings using `split` and `join` commands
- *csv.reader* and *csv.writer*

Using Functions, Parameters, Return Values and Modules

- Define the terms function call and return
- Explain and give examples of the benefits of functions
- Import modules to a program following good programming practice (i.e., don't use `import *`)
- Call functions from modules correctly (e.g., `math.cos()`)
- Use a function with or without parameters
- Use a function with or without return variable
- Create a tuple and use indexing with tuples
- Assign a value from a tuple or slice of a tuple
- Utilize any function if provided with a function definition or description
- Use matplotlib to create a plot with axis labels, a title, a line and/or marker, and a legend
- Create plots with several basic colors, solid and dashed lines, and a couple of basic marker types
- Use numpy to create an array and perform simple operations on arrays
- Use numpy with linspace in conjunction with matplotlib to create a plot

Writing Functions, Scope

- Define a function without parameters, with parameters, and/or including default parameters
- Define a function with or without return variable
- Define a function utilizing another function
- Recognize when errors may or may not appear when defining or calling functions
- Define and explain the scope of a variable and where it will be recognized in a program
- Define local and global variables
- Account for how the mutable data within a list behaves with functions calls

Functions and Use in Top-Down / Bottom-Up Design

- Create docstrings for functions
- Explain advantages and disadvantages of using a top-down design while utilizing functions
- Utilize functions within a top-down design framework, documenting steps of the process
- Define and give examples of bottom-up design
- Explain and give examples of advantages and disadvantages of bottom-up design
- Utilize a bottom-up design approach as part of creating a Python program
- Compare top-down and bottom-up design

Systematic Debugging

- Recognize general types of errors
- Read or write try-except statements for error handling
- Identify and follow code utilizing `TypeError`, `OSError`, and `ZeroDivisionError` exceptions
- Describe a proper debugging process
- Describe the use of an interactive debugger tool in a programming IDE