**Overview**

The purpose of this activity is to teach basic coding practices using tasks that build on one another. The exercises are chosen to help students better understand core programming aspects of mechatronics using programming tasks that are integrated and due with each homework assignment. The tasks are designed to be serial in nature, allowing students to self-pace. Further, they are interdependent such that, for example, task 3 would have to be completed before the student could move on to task 4. This inter-task dependency is created by having each task result produce, upon completion, a password that unlocks the subsequent task. Each homework assignment has approximately five tasks to complete, and these tasks are designed to relate to the content being taught in the homework module.

For example, Module 1 covers basic programming concepts such as using functions, for and while loops, or if statements.

As the semester progresses, the tasks may get more complicated, incorporating multiple instances of these concepts and applying the coding sections to the course content. Below is presented a summary of the tasks for each module and the learning objectives associated with each.

A MATLAB script with possible solutions to each task is also provided. The first few tasks follow the building up structure where each task builds on the last, but the remaining tasks are relatively free from context and seemed like tasks that could be interesting or challenging. Tasks that build up to or build on concepts developed in these later tasks would be a good place to start when coming up with additional tasks for the course. Later tasks are designed to rely on a student ID number to seed the MATLAB random number generator, generating unique but repeatable passwords for each student as they progress through the lessons.

**Task 1**

Learning Objectives: Students demonstrate the use of giving inputs to functions

This task simply requires a student to provide input to a function. By inputting an ID number they have created at the beginning of the semester, they will generate a unique password which can be used to access the next task.

**Task 2**

Objectives: Students demonstrate that they can write simple for loops

This task requires a student to input a specific number between 1 and 9 to receive a password and thereby access the next task. While a student could try to input all the numbers between 1 and 9 themselves manually, the repeated queries demonstrate the utility of code to handle repeated events, and how to implement this repetition with for-loops.

**Task 3**

Objectives: Write simple for loops

Similar to Task 2, this task requires a student to input a number to a function, this time between 1 and 99,999. A student will not be able to guess this password, so a for loop will have to be written to achieve this goal.

**Task 4**

Objectives: Use modular division

This task requires students to use modular division to determine whether the elements in an array of numbers are even or odd. The password will be a string of binary numbers corresponding to each element in the array, where odd numbers are represented as 1 and even numbers are represented as 0. It may be simpler to have the students simply count the number of odd numbers in the array for an earlier task.

**Task 5**

Objectives: Use modular division, convert binary to decimal, write for loops, write while loops

This task requires a student to convert an 8-bit binary number to decimal. This task can be done programmatically or by hand, but students should be encouraged to do it programmatically. Modular division can be used to extract digits from the number. The decimal number will be the password to the next task.

**Task 6**

Objectives: Use modular division, convert binary to decimal, write for loops, write while loops

This task requires a student to convert a 16-bit binary number to decimal. This too could probably be done by hand, but it is much harder and requires much larger numbers. Larger binary numbers could be used if necessary, but the structure of the number will need to be done differently than it currently is in this task. The decimal number will be the password to access the next task.

**Task 7**

Objectives: Write while loops, understand binary search, use if statements

This task is similar to Task 3, where a number between 1 and 99,999 must be input into the function, but now instead of the function simply saying right or wrong, the function also specifies whether the input number was higher or lower. While not currently implemented, it could be arranged such that the correct number must be found in a certain amount of time or the password will reset, incentivizing students to use a faster method of searching than linear search, like binary search. The password to the next task would be output when the correct input number is found.

**Task 8**

Objectives: Convert binary to hexadecimal, use modular division, write for loops, write while loops, use switch statements

This task requires the student to convert a 16-bit binary number to hexadecimal. This task uses the same function as Task 5, which generates a 16-bit binary number. Again, a longer number could be used, but not as the code is currently set up. Students might find it convenient to use a switch statement to handle the letters which are part of hexadecimal code. The password for the next task will be the hexadecimal number, using lowercase letters.

**Task 9**

Objectives: Understand the polling timing method

This task requires students to start a timer and then run the function between 1 and 2 seconds later, and returns a password if this is accomplished. Later tasks which build on this one could be centered around polling loops, running the function every second for a minute, for example, with each attempt giving a piece of the password needed to progress to the next lesson.

**Task 10**

Objectives: Understand parity in serial communication, write for loops

This task generates an array of 8-bit communications with a parity bit at the end, using even parity. Students are required to determine whether a bit may have flipped in the process of transmitting the data, or whether the communication agrees with what they expect. The password is a string of ones and zeros, where 1 means the corresponding communication was transmitted correctly and 0 means it was not.

**Task 11**

Objectives: Understand ASCII code communication, write for loops, use switch statements

This task generates an array which represents a “secret message,” a list of decimal representations of capital letters in the ASCII code. This task is rather tedious, because a possible solution involves writing a switch statement for 26 letters in the ASCII code, but it could be fun to see it work. The password to move on to the next task is a randomly generated string of letters attached at the end of the message.