# IT 230 Coding Activity Submission Template

Submit your work on the coding activities for Modules One, Two, Three, Four, and Six in this document. In addition to this document, you should submit a ZIP file containing all your Visual Studio project files and source code that can be run in Visual Studio on a different computer.

For each coding activity, complete the following steps:

* Download and rename this document to meet the file naming conventions requested in the assignment instructions.
* Fill in the required information below by replacing the bracketed text with the relevant information.
* Submit this document and your ZIP file for grading and feedback. Your ZIP file should follow the same naming conventions.

Document your work in the coding activity by completing each of the following items:

1. Provide a screenshot of the output that resulted from running your program successfully in Visual Studio. See the coding assignment instructions for an example of what should be included in the screenshot. Your screenshot must include the following elements:
   1. Your last name as the first printed text on the screen
   2. Verification that the program is fully functioning and data results are accurate for the given problem

A computer screen shot of a black screen

Description automatically generated

1. Copy and paste the source code text you wrote for this assignment from the \*.cs file into the space below. Only providing the \*.cs files or a screenshot does not meet the requirements for this part of the assignment. Code should be logically organized. It should also follow proper syntax and conventions noted in the Coding Activity Guidelines and Rubric.

using System;

namespace DebugFixMethods

{

class Program

{

static void Main(string[] args)

{

(new Program()).Run();

}

void Run()

{

int choice = 0;

WritePrompt();

choice = ReadChoice();

while (choice == -1)

{

WritePrompt();

choice = ReadChoice();

}

WriteChoice(choice);

}

void WritePrompt()

{

Console.WriteLine("Couch's Copy");

Console.WriteLine("Please select a course for which you want to register by typing the number inside []");

Console.WriteLine("[1] IT 145\n[2] IT 200\n[3] IT 201\n[4] IT 270\n[5] IT 315\n[6] IT 328\n[7] IT 330");

Console.Write("Enter your choice: ");

}

int ReadChoice()

{

string s = Console.ReadLine();

int choice;

if (int.TryParse(s, out choice))

{

if (choice >= 1 && choice <= 7)

{

return choice;

}

else

{

Console.WriteLine("Invalid choice. Please enter a number between 1 and 7.");

return -1;

}

}

else

{

Console.WriteLine("Invalid input. Please enter a number.");

return -1;

}

}

void WriteChoice(int choice)

{

Console.WriteLine("Your choice is {0}", choice);

Console.WriteLine("Press any key to continue . . .");

Console.ReadKey();

}

}

}

1. Show that you understand the task by explaining the design of your program in the space below. Include the process and steps you took to write your code. Explain how you arrived at the solution to the problem and completed the activity.

To debug and fix the DebugFixMethods program, I first reviewed the provided code to identify syntax and logical errors. The primary issues included incorrect method names, missing data type declarations, and inadequate input validation. I corrected the capitalization in the WritePrompt method and ensured the parameter type was specified in the WriteChoice method. I then implemented input validation in the ReadChoice method using int.TryParse to handle non-integer inputs gracefully. To prevent the program from closing immediately after an invalid input, I added a loop in the Run method that repeatedly prompts the user until a valid choice is entered. This ensured the program could handle various incorrect inputs and still function as expected when a correct input was provided.

1. Reflect on your learning experience and what you learned from completing the activity.

Through this activity, I learned the importance of thorough debugging and testing to ensure a program functions as intended. I gained a deeper understanding of input validation and error handling, which are crucial for creating robust applications. This exercise also highlighted the significance of clear and consistent naming conventions and method signatures for code readability and maintainability. Overall, this experience enhanced my debugging skills and reinforced the importance of writing clean, error-free code that anticipates and handles user input errors promptly.