Name:		Course Code:	

Workbook

Contents

Course Introduction

Welcome to grade 11 computer science/programming (ICS3U/3C). You will be learning how to program a computer using the C# programming language.

Mark Breakdown

Category	Percent
Term Work C# Workbook, Coding Challenges, Written Tests	60%
Senior Project Write a proposal document (if you didn't already write it in grade 10) and start development on a project. It can be a game, mobile app, web application, robotics application, etc.	20%
Exam	20%

Email

The majority of communication between you and I will be via your school email account. Make sure you check your school email account at the start and at the end of each period.

GitHub

You will be using an industry-standard code storage and versioning system called GitHub in this course. All your work will be pushed (uploaded) to GitHub daily, so I have constant access to your work. When you want me to mark something, you simply must make sure your work is pushed to GitHub and that you send me a notification email.

I have created a private repository (storage area) on GitHub for you. To access it, you need to have a GitHub account. So, head over to https://github.com/ and create an account. Then email me your GitHub username so I can add you as a collaborator on the repository.

NOTE: You can access GitHub from school and from home.

Follow along with https://youtu.be/hMymT4qJB2c to setup and use GitHub.

Course Software

All the software in this course is free to download and use. This means you can set up your home computer just like your one at school.

Daily Workflow

Follow this workflow whenever you work on this course (at school or at home):

1. Log into your computer.
2. Pull your repository from GitHub.
3. Check email.
4. Work on this workbook, do coding challenges, work on final project, etc.
5. Commit and push your repository to GitHub.
6. Send me any required notification emails.
7. Log off your computer.

Mastery System

The best way to learn to program is to keep working away at it until you've mastered the basics. That's why I require you to keep redoing your assignments in this course until they are perfect. I mark everything on a 4-point scale:

Level	Description
1	You are just beginning to understand a concept.
2	Your understanding is developing.
3	You are proficient, but still make some mistakes.
4	You have mastered a concept and very rarely make mistakes.

All lessons in this workbook must be completed to LEVEL 4 before you can go onto the course Coding Challenges. Any given Coding Challenge must be mastered to LEVEL 4 before you can write the Coding Test associated with that Coding Challenge.

C# Workbook

You will learn the basics of the C# programming language by completing the lessons in this workbook. When you have completed a lesson in this workbook, ensure it is pushed to GitHub and then send me a notification email.

You must complete all lessons in this workbook to LEVEL 4 <u>before</u> going on to the course Coding Challenges.

Coding Challenges

There are several Coding Challenges in the course. This is where you really learn to solve problems and program! Each challenge is followed by a written test which tests the concepts learned during the challenge.

You must achieve LEVEL 4 on a Coding Challenge before you can attempt the test.

Coding Tests

After completing a Coding Challenge to LEVEL 4, you are eligible to write the Coding Test associated with the challenge. The test will cover the basic principles learned in the Coding Challenge.

You are <u>encouraged</u> (but not required) to write the Coding Tests over and over until you reach LEVEL 4.

Senior Project

You will be working on a senior project in this course, which will span grades 11 and 12. It can be a game, mobile app, web application, desktop application, robotics application, or anything you can think of. This project will be worked on in grades 11 and 12. If you did not submit a project proposal in grade 10 last year, then you will start with that.

Final Exam

The final exam is similar to the Coding Tests and written on paper.

What is a programmer?

Watch https://youtu.be/aoptJnQNipo and then answer the following questions.

1. What is a *computer programmer*?

2. What is a *computer program*?

3. What is *computer science*?

4. Why is a programmer like a mathematician?

5. Why is a programmer like an engineer?

6. Why is a programmer like a scientist?

7. List and describe *three* important aspects of problem solving.

✓ Set up your development environment

You will be using Visual Studio to write your programs. You can download and install Visual Studio Community free from https://visualstudio.microsoft.com/downloads/

You only need to install the ".NET desktop environment" component for this course, but feel free to install other components and explore!
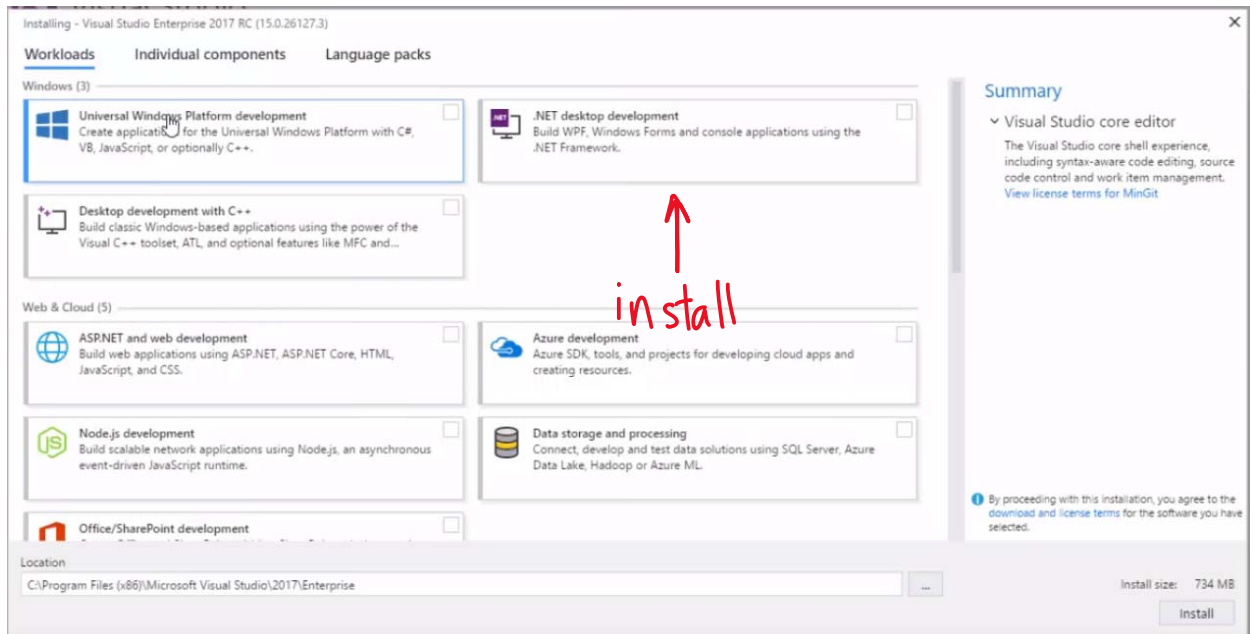
More About Programs

Watch https://youtu.be/YKp08xD_v3E and then answer the following questions.

1. What is a program?

2. List and describe the *five* basic instructions supported by the majority of programming languages.

Debugging

Watch https://youtu.be/lPsfMiPu1ww and then answer the following questions.

1. When programming, what is a **bug**?

2. What is the term for the process of removing errors from software?

3. List the *three* types of errors and describe each one.

SoloLearn – Basic Concepts – What is C#?

C# is a general purpose programming language used in a variety of applications, including: desktop applications, web applications, mobile applications, games, and robotics. It runs on a huge library of useful classes called the .NET Framework. A class is a piece of computer code which performs a useful function.

For this lesson, head over to https://www.sololearn.com/, create an account, and complete the **What is C#?** lesson. When you have completed the lesson, answer the following questions.

1. What is C#?

2. What is the .NET Framework?

3. What is the relationship between C# and the .NET Framework?

4. List *five* things you can create using C# and the .NET Framework.

5. The .NET Framework is composed of what two components?

6. What is the CLR? What does it do?

7. What is the class library? Why is it useful?

SoloLearn – Basic Concepts – Variables

Complete the **Variables** lesson at SoloLearn and then answer the following questions.

1. What is a **variable**?

2. What is another term for a **variable name**?

3. What are the rules of naming variables in C#?

4. The best variables names are "descriptive". What does this mean?

5. What is a **data type**?

6. What does it mean to "declare a variable"?

7. What does it mean to "initialize a variable"?

8. List and describe the *six* built-in data types: *int, float, double, char, bool,* and *strin*g.

9. Write code to declare and initialize variables for the following values:
   * The age of a person.
   * The price of a pizza.
   * The first letter of your last name.
   * Whether or not it is currently raining outside.
   * A user's password.

SoloLearn – Basic Concepts – Your First C# Program

Complete the *Your First C# Program* lesson at SoloLearn and then answer the following questions.

1. What is a ***console application***?

2. Why must every C# application contain a Main method? A method is simply a block of code which performs some task and can be executed by another block of code. You will learn more about them in future lessons.

SoloLearn – Basic Concepts – Printing Text

Remember, the .NET Framework contains a class library, which contains thousands of useful classes we can use. You can think of a class as a mini-program which we can incorporate into our own programs to perform tasks.

The Console class allows us to print text to the console window for the user to read. If we want to use the Console class, we have to specify the path to the class in the .NET Framework at the top of a code file with something called a "using statement".

```
1 ▾ using System;
2
3   namespace Example
4 ▾ {
5       class Program
6 ▾     {
7           static void Main(string[] args)
8 ▾         {
9               Console.WriteLine("Hello World!");
10          }
11      }
12  }
```

Line	
1	We want to use the Console class to output text to the user, so we include the path to the Console class within the .NET Framework. The Console class is contained in the System namespace. A namespace is basically a folder in the .NET Framework which contains classs.
3	We also declare our own namespace to hold our code. It's like making a folder on your computer to store a bunch of related files in.
4, 12	Code blocks are defined using { } parentheses. These define the body of our namespace.
5	Our Example namespace can contain one or more classes. In this case, we have a single class in our namespace called "Program". This class contains the code for our simple program.
6, 11	These parentheses define the body of our Program class. Any code between these brackets is contained within the Program class.
7	Classes are composed of **properties** and **methods**. A property is some data associated with the class and a method is something that the class does. In this case, we have a method called "Main" which is automatically run when our program is launched. This "Main" method is called the "entry point" to our application.
8, 10	These parentheses define the body of the Main method. All code contained between then parentheses is considered part of the Main method.
9	Here we use the WriteLine method of the Console class to write a message to the console window for the user to see. Note the command is terminate with a semicolon.

I know it seems like a whole lot of work to print some text! C# is a powerful language and with that power comes a bit of complication.

Complete the **Printing Text** lesson at SoloLearn and then answer the following questions.

1. What is the difference between the Write and WriteLine methods of the Console class?

2. What is the purpose of the ( ) parentheses after the WriteLine method?

3. Write a program which declares and initializes a variable and then prints the value of the variable to the console window.

4. Debug the following program.

```
namespace PrintingText
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 2
            int y = 3;
            int sum = x + y;

            Console.WriteLine("{0} + {2} = {3}", x, y, sum);
        }
    }
}
```

Sololearn – Basic Concepts – Getting User Input

You can use the ReadLine method of the Console class to get user input, as in the following example.

```
1   using System;
2
3   namespace Example
4 ▾ {
5       class Program
6 ▾     {
7           static void Main(string[] args)
8 ▾         {
9               string yourName;
10              Console.WriteLine("What is your name?");
11              yourName = Console.ReadLine();
12
13              Console.WriteLine("Hello {0}", yourName);
14          }
15      }
16  }
```

Line	
1	The Console class is located in the System namespace of the .NET Framework. Therefore, we must specify the path to the required namespace.
3	Our example project is contained in its own namespace.
5	Our program is its own class contained in our own namespace.
7	The Main method is the entry point to our program.
9	Declare a *string* variable named *yourName*. This variable will hold the person's name when they enter it.
10	Write a message to the Console so the user knows to enter their name.
11	Use the ReadLine method of the Console to get the user's name. Note that we still have to include the ( ) parentheses, even though we are not specifying any arguments. After the user enters their name, the ReadLine method returns their name to the yourName variable, where it is stored.
13	Now we can greet the user by name using the WriteLine method.

Complete the **Getting User Input** lesson at SoloLearn and then answer the following questions.

1. Which Console class method allows you to get user input?

2. The Console.ReadLine method returns a string. Which other class can be used to convert this string to another data type?

3. Write a program which prompts the user for two integers and then prints the sum of the integers to the console.

SoloLearn – Basic Concepts – Comments

It's a good idea to comment your code. This helps you remember what you were thinking when you wrote it and other programmers can use your comments to understand your code!

Complete the **Comments** lesson at SoloLearn and then answer the following questions,

1. What is a ***comment***?

2. What are the *two* methods of commenting your code?

3. Why is it a good idea to comment your code?

SoloLearn – Basic Concepts – The var Keyword

Complete the ***The var Keyword*** lesson at SoloLearn so you can continue on to the next lesson. However, I don't want you using the *var* keyword in this course. Instead, you will be explicitly specifying the data type of your variables when you declare them.

SoloLearn – Basic Concepts – Constants

Complete the **Constants** lesson at SoloLearn and then answer the following questions.

1. What is the difference between a constant and a variable?

2. What is the **const** modifier used for?

3. Why might a programmer prefer to use a constant in their code instead of a variable?

SoloLearn – Basic Concepts – Arithmetic Operators

Complete the **Arithmetic Operations** lesson at SoloLearn and then answer the following questions.

1. What is an **operator**?

2. List the *five* arithmetic operators supported by C#.

3. What is the result of 10 / 3, assuming both operands are integers?

4. What is the result of 10 / 3.0, assuming both operands are floats?

5. What happens if you attempt to divide by zero?

6. What is the modulus operator (%) used for?

7. What does the term *operator precedence* mean?

8. Determine the output of the following program.

```
1    using System;
2    namespace OperatorPrecedence
3    {
4        class Program
5        {
6            static void Main(string[] args)
7            {
8                Console.WriteLine((10 / 3 * 2 + 10) % 2);
9                Console.WriteLine(10 - 3 * 2 / (7 + 1) % 5);
10               Console.WriteLine(8 % 5 * 3.0 / (15 - 3) + 1);
11               Console.WriteLine(3 * 3 + (15 / 5) % 2 - 2);
12           }
13       }
14   }
```

9. Write a program which prompts the user for two floats and then determines the result of dividing the first by the second.

SoloLearn – Basic Concepts – Assignment & Increment Operators

Complete the **Assignment & Increment Operators** lesson at SoloLearn and then answer the following questions.

1. What is the purpose of the **assignment operator (=)**?

2. Write the equivalent shorthand syntax in each case.

```
x = x + 2
x = x - 2
x = x * 2
x = x / 2
x = x % 2
```

3. Rewrite the following code using the shortest possible shorthand.

```
x = x + 1
x = x - 1
```

4. What is the difference between the prefix increment operator (++x) and the postfix increment operator (x++)?

5. Predict the output of the following code.

```
1      using System;
2
3      namespace Temp
4      {
5          class Program
6          {
7              static void Main(string[] args)
8              {
9                  int x = 10;
10                 Console.WriteLine(x++);
11                 Console.WriteLine(++x);
12             }
13         }
14     }
```

SoloLearn – Conditionals and Loops – The if-else Statement

Complete *The if-else Statement* lesson at SoloLearn and then answer the following questions.

1. When are *if-else statements* used?

2. What the purpose of the *else if* statement?

3. Write the start of a basic text-based adventure game. The user enters a direction (N, S, E, or W) and the game tells them what they are looking at. For example, if the user enters 'N' the game might say "You see a tall mountain shrouded in ominous mist." If the user does not enter a valid direction, the game should inform them of their mistake.

SoloLearn – Conditionals and Loops – The switch Statement

Complete **The switch Statement** at SoloLearn and then answer the following questions.

1. What is the **switch** statement used for?

2. Refactor your text-based adventure from the previous lesson to use a **switch statement** instead of the **if structure**.

SoloLearn – Conditionals and Loops – The while Loop

Complete *The while Loop* lesson at SoloLearn and then answer the following questions.

1. Why are loops used in programming?

2. Predict the output of the following program.

```
1      using System;
2
3      namespace Temp
4      {
5          class Program
6          {
7              static void Main(string[] args)
8              {
9                  int x = 20;
10
11                 while (x > 0)
12                 {
13                     if (x % 2 == 0)
14                     {
15                         Console.WriteLine(x);
16                     }
17                     x--;
18                 }
19             }
20         }
21     }
```

3. Predict the output of the following program.

```csharp
1    using System;
2
3    namespace Temp
4    {
5        class Program
6        {
7            static void Main(string[] args)
8            {
9                int x = 20;
10               int y = 0;
11
12               while (x-- > 0)
13               {
14                   while (++y < 5)
15                   {
16                       Console.WriteLine(x / y);
17                   }
18               }
19           }
20       }
21   }
```

4. Write a program which calculates the sum of all numbers from 0 to a specified positive integer. The program should prompt the user for the integer. Here is a sample session...

```
C:\WINDOWS\system32\cmd.exe

Enter a positive integer:
5
The sum from 1 to 5 is 15.
Press any key to continue . . .
```

SoloLearn – Conditionals and Loops – The for Loop

Complete **The for Loop** lesson at SoloLearn and then answer the following questions.

1. When might a programmer prefer a **for loop** over **while loop**?

2. Predict the output of the following program.

```
1    using System;
2
3    namespace Temp
4    {
5        class Program
6        {
7            static void Main(string[] args)
8            {
9                for (int i = 0; i < 3; i++)
10               {
11                   for (int j = 3; j > 0; j--)
12                   {
13                       Console.WriteLine(i + j);
14                   }
15               }
16           }
17       }
18   }
```

3. Alter your summing program from the last lesson to use a **_for loop_** instead of the **_while loop_**.

SoloLearn – Conditionals and Loops – The do-while Loop

Complete **The do-while Loop** lesson at SoloLearn and then answer the following questions.

1. How is the ***do-while loop*** different than the ***while loop***?

2. Predict the output of the following program.

```
1    using System;
2
3    namespace Temp
4    {
5        class Program
6        {
7            static void Main(string[] args)
8            {
9                int a = 10;
10               int b = 0;
11
12               do
13               {
14                   Console.WriteLine(a / ++b);
15                   a--;
16               } while (a > b++);
17           }
18       }
19   }
```

3. Write a program which calculates the factorial of a given number. The program should loop until the user exits. Here is a sample session.

```
C:\Windows\system32\cmd.exe
This program will calculate the factorial of a number.

Enter a positive integer or enter 'quit' to exit:
3
The factorial of 3 if 6.

Enter a positive integer or enter 'quit' to exit:
4
The factorial of 4 if 24.

Enter a positive integer or enter 'quit' to exit:
5
The factorial of 5 if 120.

Enter a positive integer or enter 'quit' to exit:
quit

Press any key to continue . . . _
```

SoloLearn – Conditionals and Loops – break and continue

Complete the **break and continue** lesson at SoloLearn and then answer the following questions.

1. What does the **break** command do?

2. What does the **continue** command do?

3. Predict the output of the following program.

```csharp
using System;

namespace Temp
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 10;
            do
            {
                for (int i = 0; i < x; i++)
                {
                    if (i % 2 == 0)
                    {
                        continue;
                    }
                    else if (i * x > 60)
                    {
                        break;
                    }
                    Console.Write("{0} ", i);

                }
            } while (x-- > 0);

            Console.WriteLine();
        }
    }
}
```

Output:

```
1 3 5 1 3 5 1 3 5 7 1 3 5 1 3 5 1 3 1 3 1 1
```

SoloLearn – Conditionals and Loops – Logical Operators

Complete the *Logical Operators* lesson at SoloLearn and then answer the following questions.

1. Predict the output of the following program.

```
1   using System;
2
3   namespace Temp
4   {
5       class Program
6       {
7           static void Main(string[] args)
8           {
9               bool a = true;
10              bool b = true;
11
12              Console.WriteLine(a && b);
13              Console.WriteLine(a || b);
14              Console.WriteLine(a && !b);
15              Console.WriteLine(!a || !b);
16              Console.WriteLine(a && b || !b);
17          }
18      }
19  }
```

2. Predict the output of the following program.

```csharp
using System;

namespace Temp
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    if (i < j || i % 2 == 1)
                    {
                        Console.WriteLine("Snap!");
                    }
                    else if (i > j && j % 2 == 1)
                    {
                        Console.WriteLine("Crackle!");
                    }
                    else if (i == j || j / i == 1)
                    {
                        Console.WriteLine("Pop!");
                    }
                }
            }
        }
    }
}
```

SoloLearn – Conditionals and Loops – The Conditional Operator

Complete *The Conditional Operator* lesson at SoloLearn and then answer the following questions.

1. When is the *conditional operator* used?

2. Predict the output of the following program.

```
1    using System;
2
3    namespace Temp
4    {
5        class Program
6        {
7            static void Main(string[] args)
8            {
9                int sum = 0;
10               int x = 1;
11               int y = 1;
12               while (x < 10)
13               {
14                   sum += y / x > 1 ? x++ : y;
15                   x += 1;
16                   y += 3;
17               }
18               Console.WriteLine(sum);
19           }
20       }
21   }
```

SoloLearn – Methods – Introduction to Methods

Complete the **Introduction to Methods** lesson at SoloLearn and then answer the following questions.

1. What is a **method**?

2. List *four* advantages of using methods.

3. Which method is required in every C# program? Why?

4. What are the *three* parts of every method declaration?

5. What is the return type of a method which doesn't return a value?

6. What is a method *parameter*? Are they optional or required? Explain.

7. Write a method which prompts the user for input and then returns the input as a string. The message signature is:

```
static string input(string message)
```

SoloLearn – Methods – Method Parameters

Complete the **Method Parameters** lesson at SoloLearn and then answer the following questions.

1. Predict the output of the following program. Explain the output.

```
1      using System;
2
3      namespace Temp
4      {
5          class Program
6          {
7              static void Main(string[] args)
8              {
9                  int x = 2;
10                 calc(x);
11                 Console.WriteLine(x);
12             }
13
14             static void calc(int x)
15             {
16                 x *= x;
17             }
18         }
19     }
```

2. Predict the output of the following program.

```
1      using System;
2
3      namespace Temp
4      {
5          class Program
6          {
7              static void Main(string[] args)
8              {
9                  int x = 2;
10                 x = calc(calc(x));
11                 Console.WriteLine(x);
12             }
13
14             static int calc(int x)
15             {
16                 return x * x;
17             }
18         }
19     }
```

SoloLearn – Methods – Multiple Parameters

Complete the **Multiple Parameters** lesson at SoloLearn and then answer the following questions.

1. Write a method accepts two integer parameters and then returns the product of the multiplication of the two integers. Here is the method signature:

```
static int multiply(int a, int b)
```

2. Write a method which accepts two integers and determines if the first integer is greater than the second integer. Here is the method signature:

```
static bool isGreater(int a, int b)
```

SoloLearn – Methods – Optional and Named Arguments

Complete the *Optional and Named Arguments* lesson at SoloLearn and then answer the following questions.

1. Why might a programmer want to use optional parameters in a method?

2. Why might a programmer want to use named arguments instead of regular arguments?

3. Write a method which outputs a line of specified length made up of a specified character. Your method should use optional arguments. Also include a method call from the Main method which uses named arguments. Here is the method signature:

```
static void line(char c, int n)
```

SoloLearn – Methods – Passing Arguments

Complete the **Passing Arguments** lesson at SoloLearn and then answer the following questions.

1. List and describe the *three* ways of passing arguments.

2. Predict the output of the following program. Explain the output.

```
1    using System;
2
3    namespace Temp
4    {
5        class Program
6        {
7            static void Main(string[] args)
8            {
9                int x = 1;
10               init(ref x);
11               Console.WriteLine(x);
12           }
13
14           static void init(ref int x)
15           {
16               x = 999;
17           }
18       }
19   }
```

3. Predict the output of the following program.

```
1      using System;
2
3      namespace Temp
4      {
5          class Program
6          {
7              static void Main(string[] args)
8              {
9                  int result = 0;
10                 sum(1, 2, out result);
11                 Console.WriteLine(result);
12             }
13
14             static void sum(int a, int b, out int sum)
15             {
16                 sum = a + b;
17             }
18         }
19     }
```

SoloLearn – Methods – Method Overloading

Complete the *Method Overloading* lesson at SoloLearn and then answer the following questions.

1. What is *method overloading*?

2. Why might a programmer want to use method overloading?

3. Write two method overloads which calculate the modulus of two numbers. Here are the method signatures.

```
static float mod(float x, float y)

static int mod(int x, int y)
```

SoloLearn – Methods – Recursion

Complete the **Recursion** lesson at SoloLearn and then answer the following questions.

1. What is a **recursive method**?

2. Why must every recursive method have an exit condition?

3. Write a recursive algorithm to calculate the first 10 terms of the Fibonacci sequence.

Fibonacci sequence: https://www.mathsisfun.com/numbers/fibonacci-sequence.html

SoloLearn – Classes and Objects – Classes & Objects

Complete the **Classes and Objects** lesson at SoloLearn and then answer the following questions.

1. What is a **class**?

2. What is an **object**?

3. What is the relationship between classes and objects?

4. What is **instantiation**?

SoloLearn – Classes & Objects – Value and Reference Types

Complete the **Value and Reference Types** lesson at SoloLearn and then answer the following questions.

1. Give *four* examples of data types that are **value types**.

2. What is the **stack**?

3. What is the **heap**?

4. When you instantiate an object, where is the object stored? Where is the memory address to the object stored?

SoloLearn – Classes & Objects – Class Example

Complete the **Class Example** lesson at SoloLearn and then answer the following questions.

1. Object-oriented programming (OOP) is programming based on classes and objects. As you can see, C# is an object-oriented programming language. Other examples are Java and C++.

   OOP is often used in game development. Define a class called Enemy with fields *name, health,* and *damage* and methods *move, attack,* and *die.*

   Instantiate your class in the Main method and try calling its methods.

SoloLearn – Classes & Objects – Encapsulation

Complete the *Encapsulation* lesson at SoloLearn and then answer the following questions.

1. What is *encapsulation*?

2. What is an *access modifier*?

3. What does the *private* modifier do?

4. What does the *public* modifier do?

5. Add access modifiers to your Enemy class from the last lesson. Make all fields private and all methods public. Can you access the fields from the Main method? Can you access the methods?

SoloLearn – Classes & Objects – Constructors

Complete the **Constructors** lesson at SoloLearn and then answer the following questions.

1. What is a **class constructor**?

2. Add a constructor to your Enemy class which allows you to specify the Enemy name, health, and damage.

SoloLearn – Classes & Objects – Properties

Complete the **Properties** lesson at SoloLearn and then answer the following questions.

1. What is a **property**?

2. What is an **accessor**?

3. Add public accessors to your Enemy class for the private fields *name, health,* and *damage.* The name of the enemy should be read only.

4. Why is it a good practice to use properties instead of accessing the member fields of a class directly?

SoloLearn – Arrays & Strings – Arrays

Complete the *Arrays* lesson at SoloLearn and then answer the following questions.

1. What is an *array*?

2. Arrays are *zero-indexed*. What does this mean?

3. Predict the output of the following program.

```
1      using System;
2
3      namespace Temp
4      {
5          class Program
6          {
7              static void Main(string[] args)
8              {
9                  int[] ints = { 2, 6, 4, 7, 9, 0, 3, 2, 1, 5, 7 };
10                 int sum = 0;
11                 for (int i = 0; i < ints.Length; i++)
12                 {
13                     if (ints[i] <= 3 || ints[i] % 2 == 0)
14                     {
15                         continue;
16                     }
17                     sum += ints[i];
18                 }
19                 Console.WriteLine(sum);
20             }
21         }
22     }
```

4. Write a method which accepts an array of floats and returns the average of the floats. Here is the
   method signature.

```
static float avg(float[] floats)
```

SoloLearn – Arrays & Strings – Using Arrays in Loops

Complete the **Using Arrays in Loops** lesson at SoloLearn and then answer the following questions.

1. What is the benefit of using a **foreach** loop over a **for loop** when iterating over an array?

2. Refactor your average calculating method from the last lesson to use a foreach loop.

SoloLearn – Arrays & Strings – Multidimensional Arrays

Complete the **Multidimensional Arrays** lesson at SoloLearn and then answer the following questions.

1. What is the maximum number of dimensions an array can have?

2. Predict the output of the following program.

```
1      using System;
2
3      namespace Temp
4      {
5          class Program
6          {
7              static void Main(string[] args)
8              {
9                  int[,] matrix = {
10                     {1, 1, 1 },
11                     {0, 0, 1 },
12                     {0, 0, 1 }};
13
14                 for (int i = 0; i < 3; i++)
15                 {
16                     int sum = 0;
17                     for (int j = 0; j < 3; j++)
18                     {
19                         sum += matrix[i, j];
20                     }
21                     Console.WriteLine(sum);
22                 }
23             }
24         }
25     }
```

3. Write a method which accepts two 2D arrays of the same size and returns the sum of the arrays. Here is the method signature. You might find the GetLength method useful.

```
static int[,] add(int[,] a, int[,] b)
```

Adding Arrays:
https://www.mathsisfun.com/algebra/matrix-introduction.html

GetLength method:
https://stackoverflow.com/questions/2044591/what-is-the-difference-between-array-getlength-and-array-length

SoloLearn – Arrays & Strings – Jagged Arrays

Complete the *Jagged Arrays* lesson at SoloLearn and then answer the following questions.

1. What is a *jagged array*?

2. What is the difference between a jagged array and a matrix (i.e. two dimensional array)?

3. When might a programmer want to use a jagged array instead of a matrix?

SoloLearn – Arrays & Strings – Array Properties and Methods

Complete the **Array Properties and Methods** lesson at SoloLearn and then answer the following questions.

1. When might a programmer use the **Length** property of an array object?

2. When might a programmer use the **Rank** property of an array object?

SoloLearn – Arrays & Strings – Working with Strings

Complete the **Working with Strings** lesson at SoloLearn and then answer the following questions.

1. Describe each or the following string methods: *IndexOf, Insert, Remove, Replace, Substring,* and *Contains.*

2. You can also concatenate strings (i.e. add them together) using the + operator. Write a method which accepts an array of words and then concatenates them into a sentence. Here is the signature.

```
static string concat(string[] words)
```

Concatenating strings: https://www.dotnetperls.com/string-concat

SoloLearn – More on Classes – Destructors

Complete the **Destructors** lesson at SoloLearn and then answer the following questions.

1. What is a *destructor*?

2. When is the destructor automatically invoked?

3. Why might a programmer use a destructor?

4. Add a destructor to your Enemy class.

SoloLearn – More About Classes – Static Members

Complete the **Static Members** lesson at SoloLearn and then answer the following questions.

1. What is a **static member**?

2. What is a **static method**?

3. Is it possible for a static method to access a non-static member? Explain.

4. When are **static constructors** used?

5. Add a static member and a static method to your Enemy class.

SoloLearn – More About Classes – Static Classes

Complete the **Static Classes** lesson at SoloLearn and then complete the following questions.

1. When are **static classes** useful?

2. Describe *three* examples of static classes found in the .NET Framework class library.

3. Write a static class called *Calc* which contains static methods (*Add, Subtract, Multiply, Divide, Modulus*, and *Exponent*) and static constant members (*Pi* and *E).*

SoloLearn – More About Classes – this & readonly

Complete the **this & readonly** lesson at SoloLearn and then complete the following questions.

1. What is the **this** keyword used for?

2. Change your Enemy class constructor so that it uses the *this* keyword. This way your constructor arguments can have the same name as your class members.

3. Describe *two* differences between **readonly** values and **const** values.

SoloLearn – More About Classes – Indexers

Complete the *Indexers* lesson at SoloLearn and then answer the following questions.

1. When might a programmer use an *indexer*?

2. Create an Enemies class which contains an array of Enemy objects. Implement an indexer in the
   class.

SoloLearn – More About Classes – Operator Overloading

Complete the **Operator Overloading** lesson at SoloLearn and then answer the following questions.

1. When might a programmer want to use operator overloading?

SoloLearn – Inheritance and Polymorphism – Inheritance

Complete the **Inheritance** lesson at SoloLearn and then answer the following questions.

1. What is **inheritance**?

2. What is the relationship between a **base class** and a **derived class**?

3. Derive a class from your Enemy class (e.g. Koopa) and give it two new methods and two new properties

SoloLearn – Inheritance and Polymorphism – Protected Members

Complete the **Protected Members** lesson at SoloLearn and then answer the following questions.

1. What is the difference between a private member and a protected member?

2. What is the purpose of the **sealed** keyword?

3. Add a protected member to your Enemy base class.

SoloLearn – Inheritance & Polymorphism – Derived Class Constructor & Destructor

Complete the **Derived Class Constructor & Destructor** lesson at SoloLearn and then answer the following questions.

1. Predict the output of the following code.

```
1    using System;
2
3    namespace Temp
4    {
5        class Program
6        {
7            static void Main(string[] args)
8            {
9                Dog dog = new Dog();
10           }
11       }
12
13       class Animal
14       {
15           public Animal()
16           {
17               Console.WriteLine("Animal created.");
18           }
19       }
20
21       class Mammal : Animal
22       {
23           public Mammal()
24           {
25               Console.WriteLine("Mammal created.");
26           }
27       }
28
29       class Dog : Mammal
30       {
31           public Dog()
32           {
33               Console.WriteLine("Dog created.");
34           }
35       }
36   }
```

SoloLearn – Inheritance & Polymorphism – Polymorphism

Complete the **Polymorphism** lesson at SoloLearn and then answer the following questions.

1. What does the term **polymorphism** mean?

2. What is the purpose of the **virtual** keyword?

3. What is the purpose of the **override** keyword?

4. Make the Attack method in your base Enemy class virtual and then override the attack method in a derived class.

SoloLearn – Inheritance & Polymorphism – Abstract Classes

Complete the **Abstract Classes** lesson at SoloLearn and then answer the following questions.

1. What is an **abstract class**?

2. Why might a programmer want to use an abstract class?

3. Is it possible to create a sealed abstract class? Why or why not?

4. Go back and make your Enemy class an abstract class.

SoloLearn – Inheritance & Polymorphism – Interfaces

Complete the *Interfaces* lesson at SoloLearn and then answer the following questions.

1. What is an *interface*?

2. Why might a programmer prefer to use an interface instead of an abstract class?

SoloLearn – Inheritance & Polymorphism – Nested Classes

Complete the **Nested Classes** lesson at SoloLearn and then answer the following questions.

1. Why might a programmer want to use nested classes?

SoloLearn – Inheritance & Polymorphism – Namespaces

Complete the **Namespaces** lesson at SoloLearn and then answer the following questions.

1. What is a **namespace**?

2. What is the purpose of the **using** keyword?

3. List and describe *three* namespaces found in the .NET Framework class library/

SoloLearn – Structs, Enums, Exceptions, & Files – Structs

Complete the **Structs** lesson at SoloLearn and then answer the following questions.

1. What is a **struct**?

2. Describe *three* differences between classes and structs?

3. When might a programmer use a struct instead of a class?

SoloLearn – Structs, Enums, Exceptions & Files – Enums

Complete the **Enums** lesson at SoloLearn and then answer the following questions.

1. What is an **enumeration**?

2. Create an enum for *five* computer peripheral devices (e.g. Mouse).

SoloLearn – Structs, Enums, Exceptions & Files – Exception Handling

Complete the **Exception Handling** lesson at SoloLearn and then answer the following questions.

1. What is an **exception**?

2. Describe *four* scenarios that would cause an exception.
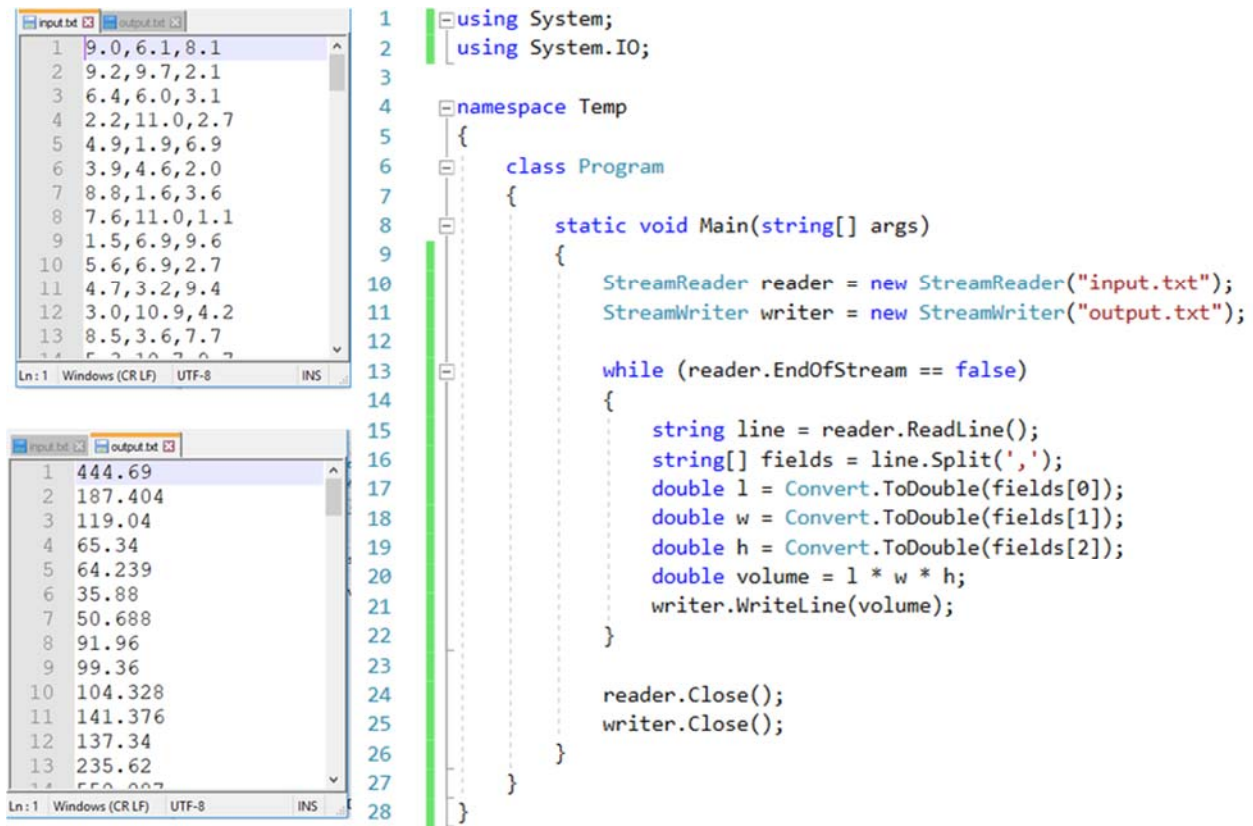

3. What is the **try-catch** statement used for?

4. What is the purpose of the **finally** statement?

5. Alter the following code so it handles a DivisionByZeroException and doesn't crash if the user enters zero for the second number.

```
1    using System;
2
3    namespace Temp
4    {
5        class Program
6        {
7            static void Main(string[] args)
8            {
9                int a = getInt("Enter the first number:");
10               int b = getInt("Enter the second number:");
11               Console.WriteLine("{0} divided by {1} is {2}", a, b, a / b);
12           }
13
14           static int getInt(string prompt)
15           {
16               Console.WriteLine(prompt);
17               return Convert.ToInt32(Console.ReadLine());
18           }
19       }
20   }
```

SoloLearn – Structs, Enums, Exceptions & Files – Working with Files

The System.IO namespace contains classes useful for reading and writing files. In this course, you will be using the StreamReader class to read files and the StreamWriter class to write files. Here is a small example program to illustrate their use. This program calculates the area of boxes given the length, width, and height of each box. The input file contains the length, width, and height (separated by commas), with each line representing a box. The output file contains the volumes of each box on separate lines.

Input file (input.txt):

```
1   9.0,6.1,8.1
2   9.2,9.7,2.1
3   6.4,6.0,3.1
4   2.2,11.0,2.7
5   4.9,1.9,6.9
6   3.9,4.6,2.0
7   8.8,1.6,3.6
8   7.6,11.0,1.1
9   1.5,6.9,9.6
10  5.6,6.9,2.7
11  4.7,3.2,9.4
12  3.0,10.9,4.2
13  8.5,3.6,7.7
```

Output file (output.txt):

```
1   444.69
2   187.404
3   119.04
4   65.34
5   64.239
6   35.88
7   50.688
8   91.96
9   99.36
10  104.328
11  141.376
12  137.34
13  235.62
```

```csharp
using System;
using System.IO;

namespace Temp
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamReader reader = new StreamReader("input.txt");
            StreamWriter writer = new StreamWriter("output.txt");

            while (reader.EndOfStream == false)
            {
                string line = reader.ReadLine();
                string[] fields = line.Split(',');
                double l = Convert.ToDouble(fields[0]);
                double w = Convert.ToDouble(fields[1]);
                double h = Convert.ToDouble(fields[2]);
                double volume = l * w * h;
                writer.WriteLine(volume);
            }

            reader.Close();
            writer.Close();
        }
    }
}
```

Line	
1	We need to include the System namespace because we are using the Convert class.
2	We need to include the System.IO namespace because we are using the StreamReader and StreamWriter classes.
4	This project is contained in its own namespace.
6	This program is its own class. Everything in C# must be a class because it is a fully Object-Oriented Programming (OOP) language.
8	The Main method is the entry-point to our program. It is the first method ran by the operating system.

10	We declare and instantiate a StreamReader object which we will use to read the contents of the input.txt file.
11	We declare and instantiate a StreamWriter object which we will use to write the contents of the output.txt file.
13	This while loop will run while the StreamReader is <u>not</u> at the end of the file. Every time we call ReadLine (see line 15) there is an internal pointer which moves line-by-line through the file. When the pointer is at the end of the file, the EndOfStream property is true and the while loop will end.
15	Grab a line from the file (and move the internal line pointer to the next line in the file).
16	Use the Split method [https://www.dotnetperls.com/split] to separate the data into fields.
17	The first field is the length of the box.
18	The second field is the width of the box.
19	The third field is the height of the box.
20	Calculate the volume of the box.
21	Write the volume to the output file.
24	Close the StreamReader. Failing to do so can cause unpredictable results and difficult bugs.
25	Close the StreamWriter. Failing to do so can cause unpredictable results and difficult bugs.

There are many other useful classes in the System.IO namespace. Complete the lesson **Working with Files** at SoloLearn and then answer the following questions.

1. Which .NET namespace contains classes useful for working with files?

2. What does the File.WriteAllText method do?

3. What does the File.ReadAllText method do?

SoloLearn – Generics – Generic Methods

Complete the *Generic Methods* lesson at SoloLearn and then answer the following questions.

1. What is the benefit of using generic methods?

SoloLearn – Generics – Generic Classes

Complete the **Generic Classes** lesson at SoloLearn and then answer the following questions.

1. What is the most common use of generic classes?

2. What is a *stack*? Why is it called a LIFO structure?

SoloLearn – Generics – Collections

Complete the **Collections** lesson at SoloLearn and then answer the following questions.

1. In which namespace are the generic collection classes found?

2. What main benefit does the generic List class have over a basic array?

3. Describe the function of each of the following generic collection classes: *Dictionary, List,* and *Queue*.

Course Coding Assignments

The coding assignments are available in the course Google Drive. Remember, each assignment has a follow-up test, and you can only write the test after you have achieved **LEVEL 4** on the assignment.

Coding Assignments: https://goo.gl/Ag7dyY

Senior Project

Now that you have gained some serious programming skill, it's time to think about what you really want to make. Is it a game? A mobile app? A website? A robotics application? The Senior Project is your chance to work on your first big project as a developer.

If you have not already done so, first you need to write up a Software Design Document (SDD). The SDD is a description of how you are going to solve a problem. It outlines the features of a piece of software and serves as a guide while you develop the software. Basically, it the most useful tool for making sure the right work gets done on a project.

Check the course Google Drive for the SDD templates [https://goo.gl/vuks7E]. If you don't find a SDD that suits your needs, then send me an email and I can make up a custom one for you. There are SDDs for each of the main project types.

Once you have your SDD completed its time to start designing and developing your product! I'm here to help you out so don't be afraid to ask questions!