Phase 1

1. Visual Studio 2019 IDE Overview

1.1 Different types of windows (solution exp., properties etc.)

- **Solution Explorer:** *This window is useful for access and manages different files of the project. It is helpful for us to view and organize files of our project in different subfolders.*
- **Code Editor:** *In this window, we can write and edit our code*.
- **Property Windows:** *This window is useful for view and sets different properties of selected objects.*
- **Output Window:** *This window helps to show the output of debugging, compiler and build system.*

1.2 Solution, Project

- **Solution:** *It is simply like a container which use to organize one or more related projects logically. We can organize these projects via Solution Explorer.*
- **Project:** *The project contains all the files like source codes, images, data, configuration files and reference files. Later these files will be built and compile into the website, Web-App or any other Application.*

1.3 Code editor features

These are the main code editor feature that includes in Visual Studio.

- *IntelliSense*
- *Formate Document*
- *Global Undo and Redo*
- *Global Find and Replace*
- *Brace Matching*
- *Comment-Uncomment Selection*
- *Increase-Decrease Line Ident*
- *Syntax coloring and highlighting*
- *Error and warning marks*
- *Auto error correction*
- *Find all reference and function definition*
- *Tabify-Untabify Selected lines*
- *Line numbers and zooming feature*

1.4 Shortcuts

- **Ctrl+Shift+B:** *Build solution*
- **Ctrl+F7:** *Compile*
- **Ctrl+F:** *Find*
- **F3:** *Find next*
- **Ctrl+K, Ctrl+D:** *Format document*
- **Ctrl+K, Ctrl+F:** *Format selection*
- **Ctrl+F12:** *Go to declaration*
- **F12:** *Go to definition*
- **Ctrl+D:** *Go to find combo*
- **Ctrl+K, Ctrl+X:** *Insert snippet*
- **Tab:** *Insert tab*
- **Ctrl+W:** *Select Current Word*
- **Shift+Tab:** *Tab with*
- **Ctrl+K, Ctrl+C:** *Comment Selection*
- **Ctrl+K, Ctrl+U:** *Uncomment Selection*
- **F5:** *Run with Debugging*
- **Ctrl+F5:** *Run Without Debugging*
- **F11:** *Step into*
- **Shift+F11:** *Step out*
- **F10:** *Step Over*
- **Shift+F5:** *Stop Debugging*
- **Alt+F4:** *Exit*
- **Ctrl+N:** *New file*
- **Ctrl+Shift+N:** *New Project*
- **Shift+Alt+N:** *New Website*
- **Ctrl+O:** *Open File*
- **Shift+Alt+O:** *Open Website*
- **Ctrl+Shift+W:** *View in browser*
- **Ctrl+F4:** *Close document window*
- **Ctrl+Tab:** *Navigate Next document window*

2. Project Types

2.1 Windows App, Class Library

- **Windows Application:** *It is a User-build application that directly runs on Windows Operating System. We can create a windows application using Windows Forms (.Net Frameworks) in Visual Studio. Windows Forms is a set of classes that encapsulates the creation of the graphical user interface (GUI) portion of a typical desktop application. A Windows forms application will normally have a collection of controls such as labels, text boxes, list boxes, etc.*
- **Class Library:** *Class Library is a collection of pre-written classes which will be used during developing an Application program. It defines a set of properties and methods which will be called by an application. We can simply use these properties and methods by including these classes in our project.*

2.2 Web Application

- *Unlike Windows applications, Web applications run on Web servers not directly on Operating System. It can be accessed directly by the user through an active network connection.*

- *Web applications are designed and programmed using a client-server modelled structure where servers are hosted on the internet so Mostly Web-App can be accessed by anyone who is connected with the internet.*
- *Web Application is mostly similar to Web Site but has slightly difference from Web-site. Web sites most likely to be referred to as "web applications" are those which have similar functionality to a desktop software application, or a mobile app.*

3. Create First C# Program "Hello World"

3.1 What is namespace?

- *A Namespace is a collection of different classes which belong to the same type of groups or categories.*
- *Namespace gives us the ability to use different classes which may have the same names but different functionality or classes belong to different parts of our projects.*
- *It also helps us to organize classes and controlling their scope. If we didn't have namespaces we'd have to (potentially) change a lot of code any time we added a library, or come up with tedious prefixes to make our function names unique. With namespaces, we can avoid the headache of naming collisions when mixing third-party code with our projects.*
- *In C#, Namespace doesn't have any access modifier because it has public access by default and we cannot change it.*
- *A namespace may also contain other namespaces.*
- *This is syntax of creating namespace.*

```
namespace <namespace_name> {
   // code declarations
}
```

- *We can either include classes of the namespace by using the "using" keyword or we can access all members of the namespace by using the dot(.) operator.*

Example:

```
    using <namespace_name>[.][sub-namespace_name]    // by
including classes

                    or

    <namespace_name>.<member_name>          // by accessing
    directly
```

3.2 What is class?

- *Classes are user-defined data types that have properties and methods.*
- *Classes may have public, private, protected or internal access specifiers which will be used to allow to limit access to other classes.*

Syntax:

```
[access_specifire]  class <class_name> {

}
```

- *We can create an instance of this class which is called an Object. Class is a blueprint of an object.*

Syntax of crating object:-

```
<class_name> <object_name> = new <class_name>()
```

- *Using objects we can directly access property and methods of class where both should be declared as public. That's means objects can only publicly access properties and methods of a class.*

3.3 Variable & Method Declaration

Variable:

- *Variables are used to store information or data in our program. We can also change or overwrite the old value in a variable. Variable act as Identifier for our data. We can give a readable and meaningful name to data because of a variable.*
- There are certain rules for creating variables.
    - *Variable name only contains alphabets, numbers, and underscore and it cannot start with a number.*
    - *There are certain words reserved for keywords so we cannot use those words as variable names.*
    - *A variable name is case sensitive and must be unique.*

Syntax of declaring and initializing Variable in C#:

```
[Access_Spefire] <Datatype> <indetifire> [= value];
```

- *Here you can see We can either just declare a variable called which is called a variable declaration or directly assign its value which is called variable initialization.*
- *We can change the value of the variable later also.*
- *Here **data type** is specified which type of value variable has and what type of data it can only store.*

Method:

- *A Method is a bunch of statements that perform some operation for a specific class/object.*
- *We just have to define a method for the specific type of option in class and later whenever we need to perform that operation we just need to call this method.*

Syntax of creating Method:-

```
[Access_specifire] <Return_type> <method_name> ([parameters , …])
{
   // Lines of code….
}
```

  - *Here **Access-Specifier** represents the access area of Method.*
  - ***Return_Type** represent which data type of value this method will return. If it returns nothing then its data type must be Void.*
  - *Rules for providing methods name is same as Variable name.*
  - *All **parameters** must be defined within parentheses. During calling We can pass data to our method using Parameters. Parameters specify how many numbers of data and which type of data method can accept after calling. The order of this data must be the same as already specified in a method declaration.*
  - *Body of Method must be within **Curly bracket**. The body contains all the codes that will execute after the calling method.*

Syntax of Calling a Method:

```
<Method_name>([Parameters, …]);
```

4. Understanding C# Program

4.1 Program Flow

This is the standard structure of the C# Program.

- *Include Library*
- *Namespace declaration*
- *A class*
- *Class methods*
- *Class attributes*
- *Main method*
- *Statements and Expressions*
- *Comments*

4.2 Understanding Syntax

```
1.  using System;                                       // Include other classes
2.
3.  namespace HelloWorldApplication {                   // Namespace Declaration
4.     class HelloWorld {                               // Class Declaration
5.        private string attribute;                     // Class Attributes
6.        static void Main(string[] args) {             // class  main method
7.           /* my first program in C# */               // comments
8.           Console.WriteLine("Hello World");          // Statements
9.           Console.ReadKey();
10.       }
11.    }
12. }
```

5. Working with code files, projects & solutions

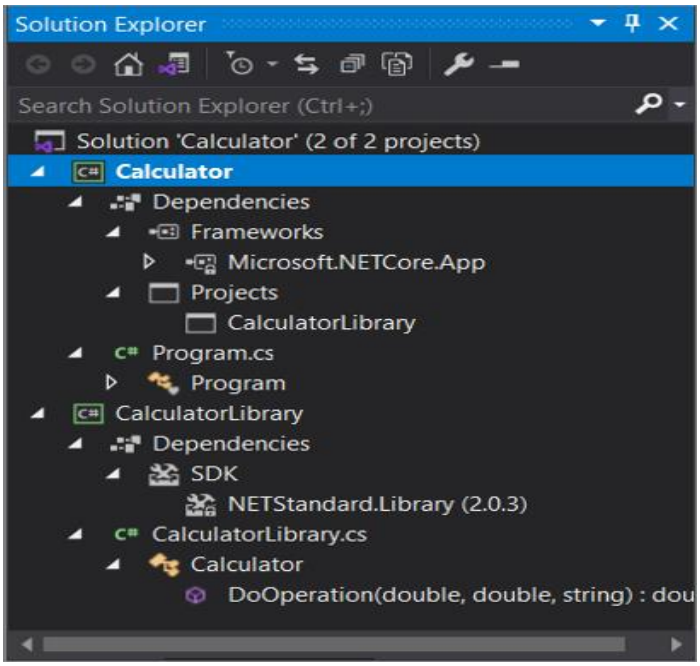
5.1 Understanding the structure of a solution

- *As we discuss before Solution is like a container for one or more related projects, along with build information, project settings and configurations files.*

Visual Studio uses two file types (.sln and .suo) to store settings for solutions:

- **.sln:** This file use for Organizes projects, project items, and solution items in the solution.
- **.suo:** This file Stores user-level settings and customizations, such as breakpoints.

- *After you create a new project, you can use Solution Explorer to view and manage the project and solution and their associated items. The following illustration shows Solution Explorer with a C# solution that contains two projects:*



5.2 Understanding structure of the project (Win App, Web-App, Web-API, class library)

Windows Application:

- *Program.cs file contains the main method for our program so its act as an entry point for our application.*
- *Form1.cs contains all necessary code for different windows of our application*
- *We can see Solution explorer on the right side so we can manage our project.*
- *All necessary references of other projects and packages are contained in ProjectName.csproj file.*

- *Below solution explorer, we can see Property windows that are used for set-unset properties for different objects.*
- *All dependency and reference files are already loaded we can see those files in the solution explorer.*

Web Application:

- *The structure of Web application and Windows application is mostly the same because both follow MVC architecture.*
- *wwwroot folder is a root folder of the website. Static files can be stored in any folder under the webroot folder.*
- *For a Web application, there are addition folder help to manage all files which are related to Models, Views and Controllers.*
- *Globle.json file is that where we can find all configurations of our project.*
- *Appsettings.json file contains all global variables.*

Web API:

- *ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework.*
- *The structure of the Web API project is almost the same as Web Application.*

Class Library:

- *Class Library has sections for Properties, References and .cs file.*
- *In References, we can add as many references as we can and use them for our project.*
- *References are those files with .dll extension. We can add it to our class* library.

5.3 Familiar with different type of file extensions

- **aspx** : *For ASP.NET Web Pages*
- **.cs :** *For C# Program*.
- **.json** *: standard plan-Text file for representing Data*
- **.dll** : It *contains precompiled library.*
- **.config** : *configuration file*
- **.sln :** *represents a Visual Studio solution file.*
- **.csproj :** *C# project file.*

6. Understanding data types & variables with conversion

6.1 Base datatype

- *Like C++ and Java C# is also a Strongly-Typed language. That means we must have to specify the type of any variable.*
- There are main Two Types of Data Type in C#
  - **Value Type:** *This Type of Data type directly hold its value in its memory location.*
    - These are some Predefined datatype of Value Type:
      - *bool, byte, byte, char, short, short, int, int, float, double, long, long, decimal*
    - *User-defined datatype: struct, enum*
  - **Reference type:** *The reference types do not contain the actual data stored in a variable, but they contain a reference to the variables. In other words, they refer to a memory location.*
    - *Predefined Reference type Datatype:*
      - *String, array*
    - *User-defined reference type Data type:*
      - *Classes Interface*

6.2 Datatype Conversion

- *Datatype Conversion is used for converting one type of data into another type of data.*
- There are two types of conversion in C#.

Implicit Type Conversion:

- In this conversion, we can compiler automatically convert datatype into another.
- This conversation supports only if conversion performs from smaller to large integral type and derived to base class conversion.

Explicit conversion:

- These conversions are explicitly performed by the user. For doing this conversion we can call the inbuilt method accordingly.
- These are built-in methods provided in C#: ToBoolean, ToChar, ToDateTime, ToDouble, ToDouble,ToInt16, ToInt32, ToString etc.

6.3 Boxing/Unboxing

Boxing:

Converting implicitly Value type into reference Type is called Boxing.

Unboxing:

Explicitly conversion from a Reference type to Value type is called Unboxing.

7. Understanding Decision making & statements

7.1 If else, switch

IF:

- *If statements are used to control the flow of the program. If condition contains a specific condition and block of code. This block of code will only execute if the condition is true.*

Syntax:

```
 If(condition)
{
   Lines of code ….
}
```

Else:

- *Else statement is always used after If Statement. If above if Statement evaluates to false then and only that time block of else statements executes.*

```
If(condition)
{
   Lines of codes …
}
Else
{
   Lines of codes …
}
```

Switch Statement:

- *The switch can be used instead of multiple If-else Stem, ents for reducing lines of code and more easy readability.*

- *Switch contains condition or exception and one or multiple case statements.*
- *Case statements contain different cases with their line of codes.*
- *Then after which case is matched with switch condition or expression, only that case statements will be executed.*
- *Switch also contains default case where if anyone of case statements going to match then defaults case going to execute.*

Syntax:

```
Switch(condition)
{
   Case constant:       //lines of code ;
                break;
   Case constant2:      //lines of code;
                break;
   Default:             //lines of code;
                break;
}
```