

Advanced Computer Programming

TICT2134[P]

Introduction to Visual Studio.NET



It is an IDE provided by microsoft for developing .NET Applications.

The.NET Framework can be used to develop a wide range of applications.

- Desktop Applications
- Web Applications
- Mobile Applications

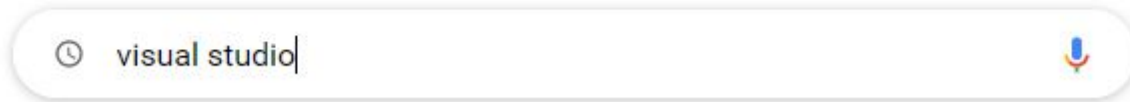
The.NET Framework supports over 30 languages, including C#, F#, and J#.

In .NET framework, every application will be called as project.

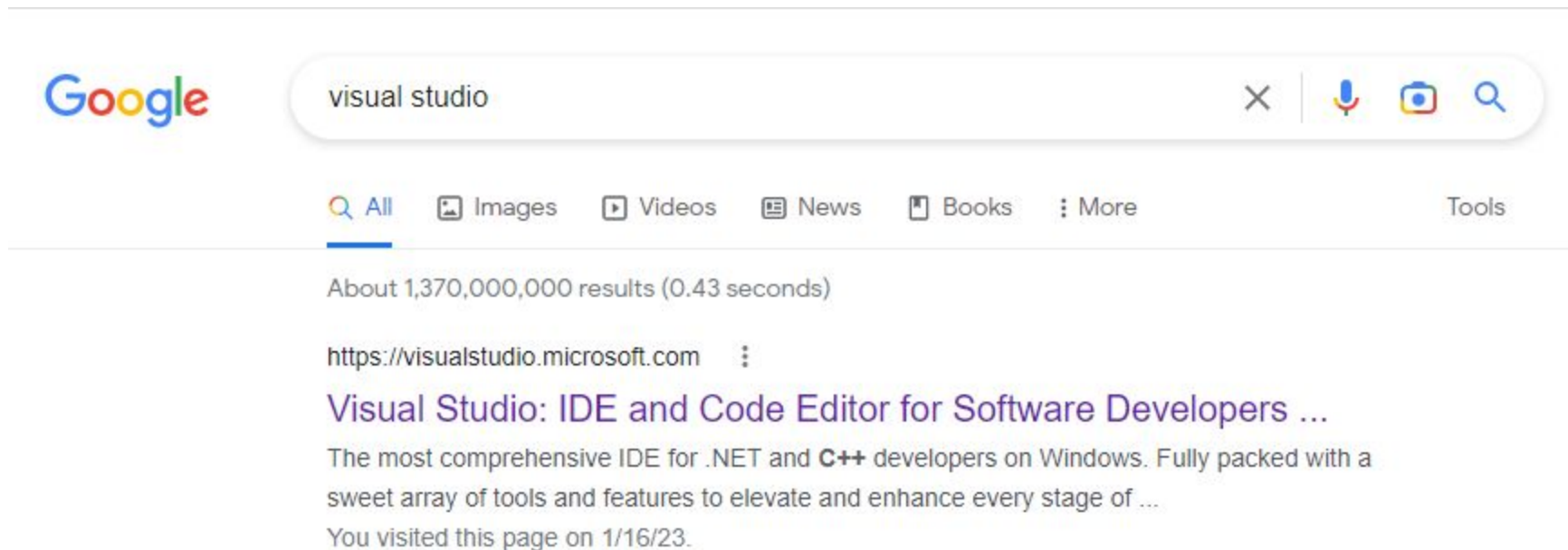
Download the visual studio.NET

Open up your browser.

Type "visual studio" in the search bar.

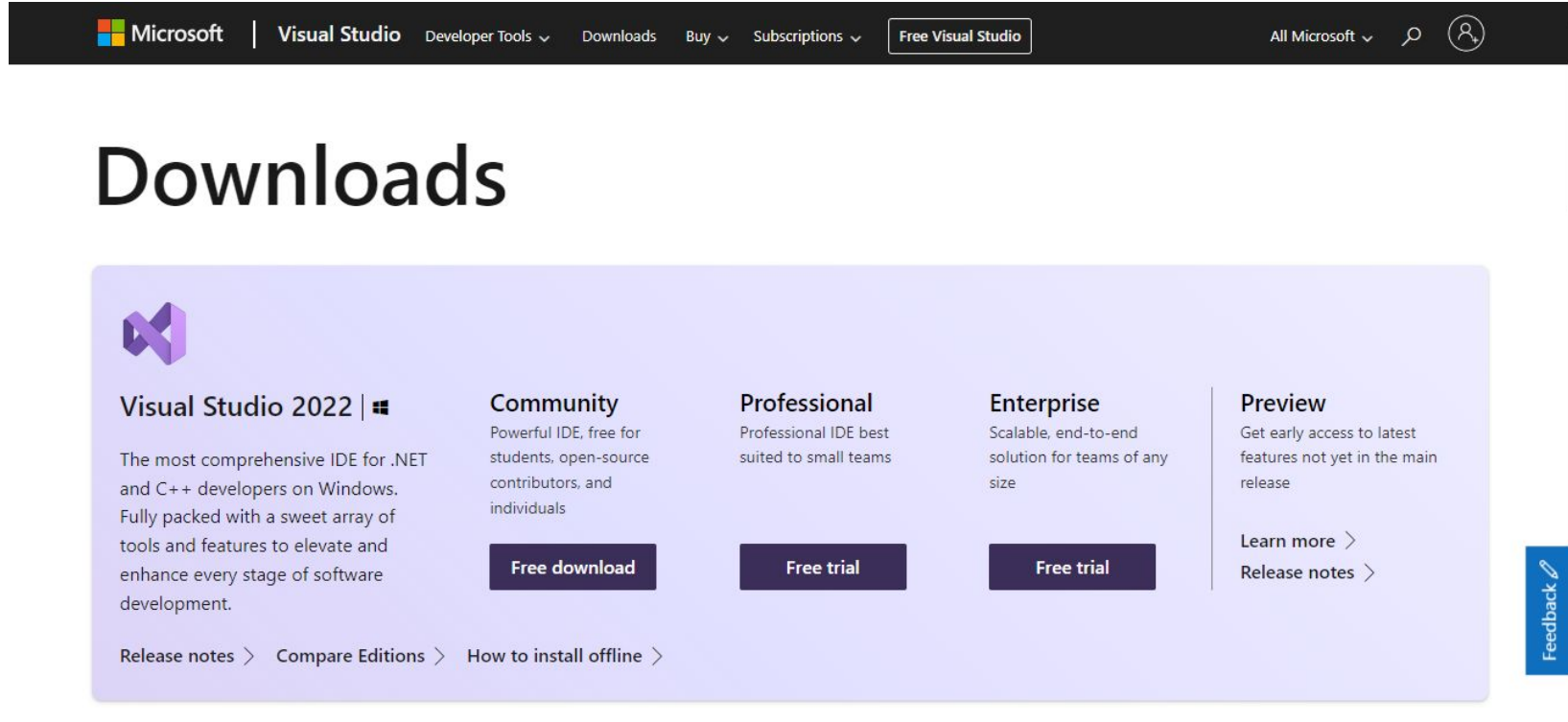


Click the link to the official Visual Studio website.




Click on **download** menu.

download the **community** edition for **windows operating system**.



The screenshot shows the top navigation bar of the Visual Studio website. It includes the Microsoft logo, 'Visual Studio', and links for 'Developer Tools', 'Downloads', 'Buy', and 'Subscriptions'. A 'Free Visual Studio' button is also present. On the right, there are links for 'All Microsoft', a search icon, and a user profile icon.

Downloads



Visual Studio 2022 |

The most comprehensive IDE for .NET and C++ developers on Windows. Fully packed with a sweet array of tools and features to elevate and enhance every stage of software development.

[Release notes >](#) [Compare Editions >](#) [How to install offline >](#)

Community

Powerful IDE, free for students, open-source contributors, and individuals

[Free download](#)

Professional

Professional IDE best suited to small teams

[Free trial](#)

Enterprise

Scalable, end-to-end solution for teams of any size

[Free trial](#)

Preview

Get early access to latest features not yet in the main release

[Learn more >](#)
[Release notes >](#)

[Feedback](#)

The Visual Studio setup file will start to download.



Visual Studio

Developer Tools ▾

Downloads

Buy ▾

Subscriptions ▾

Free Visual Studio

All Microsoft ▾



Sign in



Thank you for downloading Visual Studio

Your download will start shortly. If your download does not begin, [click here to retry](#).

Feedback



VisualStudioSetup....exe

0.5/2.0 MB, 5 secs left

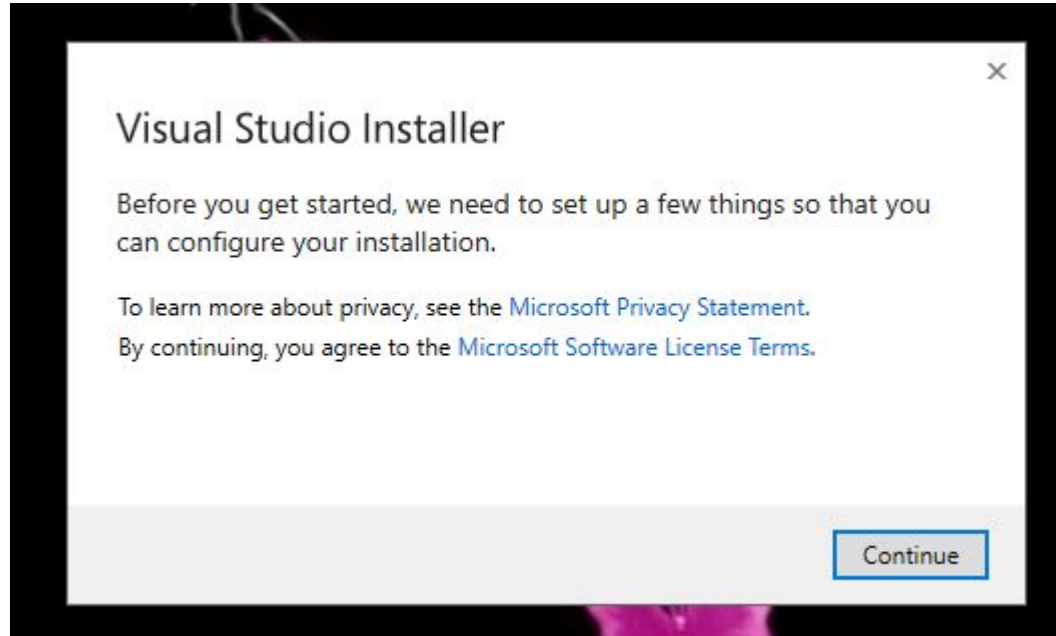


Show all



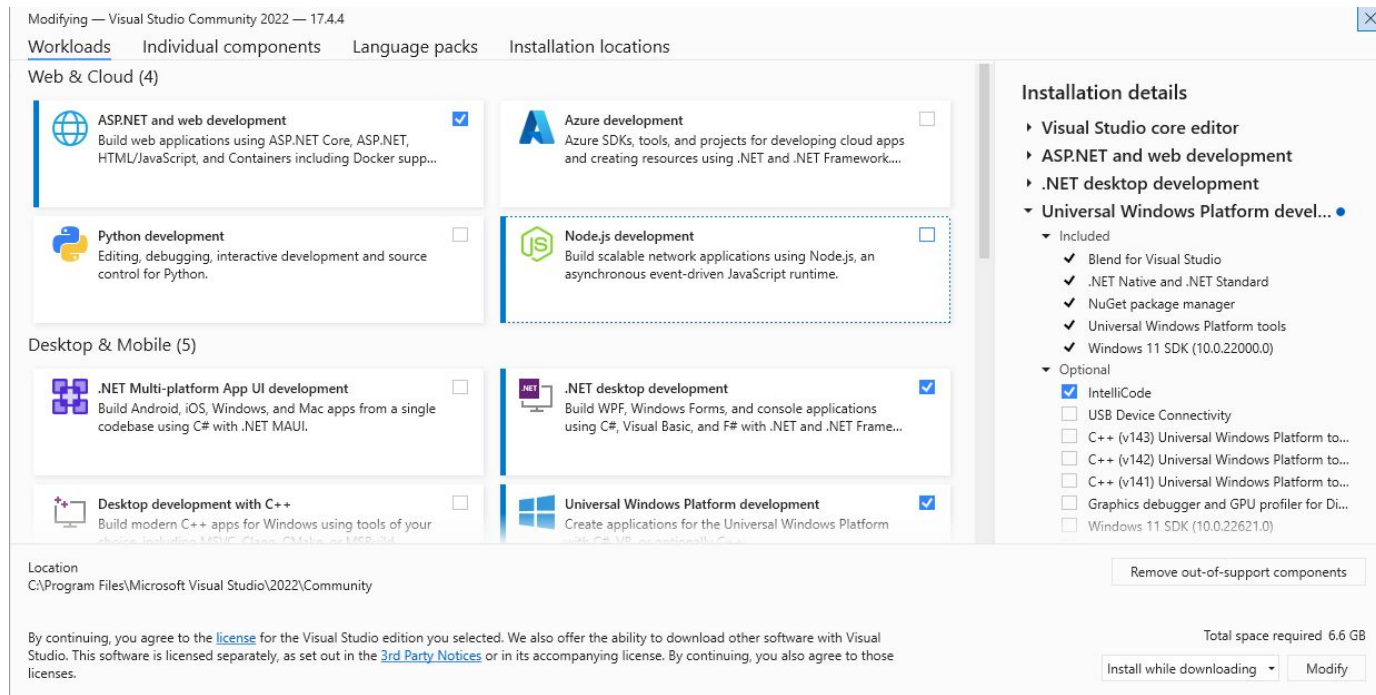
Install the visual studio.NET.

Double click on **visual studio setup** file and click on **continue**.



Under the workload menu, select the following options and click on "Install."

- ASP.NET and web development.
- .NET Desktop development.
- Universal windows platform development.



The Visual Studio installer will start to download and install the packages that you have selected.

Visual Studio Installer

Installed Available



Visual Studio Community 2022

Downloading and verifying: 342 MB of 1.24 GB

26%



The installation will start after all your packages have been downloaded.

[Release notes](#)

Pause

Developer News

[Introducing the Git Status Bar and Testing Improvements in Visual Studio for Mac 17.5](#)

Last week saw the release of the third preview in t...

25 January 2023

[Documents and tool windows unleashed](#)

We're continuing our video series with tips and tri...

24 January 2023

[Keyboard Shortcuts to Master Your Git Flow in Visual Studio](#)

One popular way for users to optimize their effici...

23 January 2023

[View more Microsoft developer news...](#)

Visual Studio Installer

Installed

Available



Visual Studio Community 2022

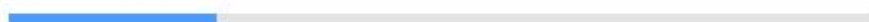
Pause

Downloaded



Installing: package 5 of 11

24%



Microsoft.VisualStudio.DotNetNative.CoreRuntime

[Release notes](#)

When it has finished installing, press the launch button.

Visual Studio Installer

Installed

Available

All installations are up to date.



Visual Studio Community 2022

17.4.4

Powerful IDE, free for students, open-source contributors, and individuals

[Release notes](#)

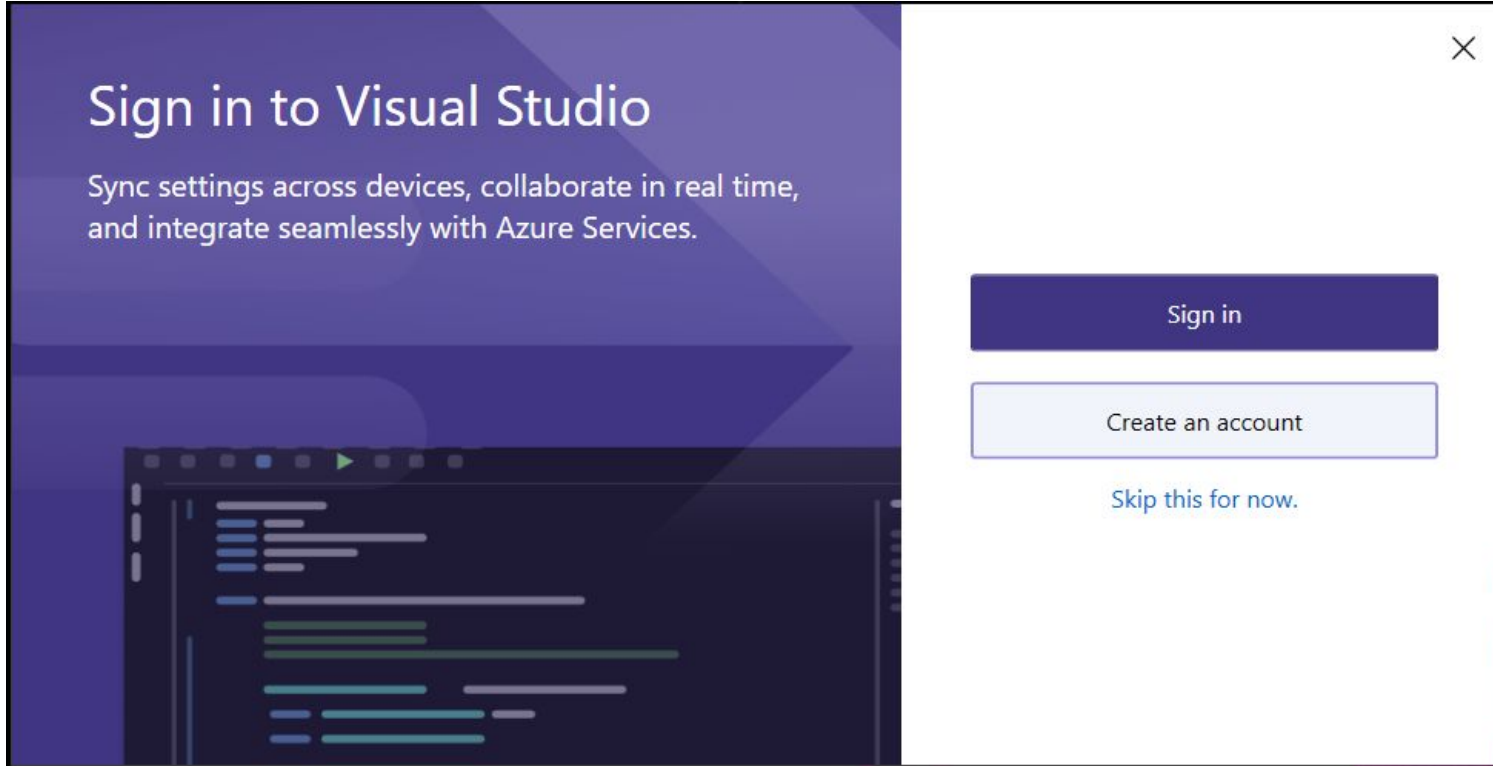
Modify

Launch

More ▼

Open the Visual Studio

Select “Skip this for now”.

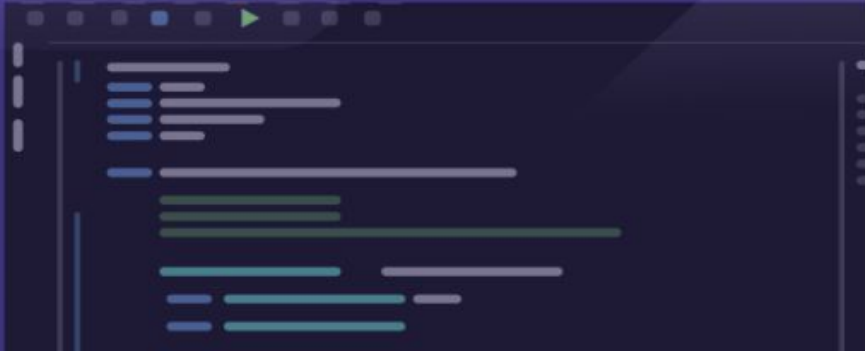


Choose your color theme and start visual studio.

Personalize your Visual Studio experience

Optimize the layout and keyboard shortcuts for your workflow. Select the color theme that fits your style.

You can always change these settings later.



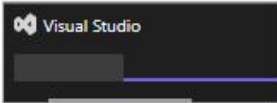
✕

Development settings

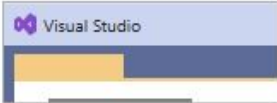
General ▾

Choose your color theme

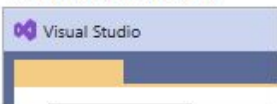
☒ Dark



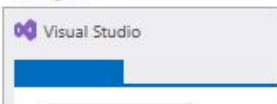
☐ Blue



☐ Blue (Extra Contrast)

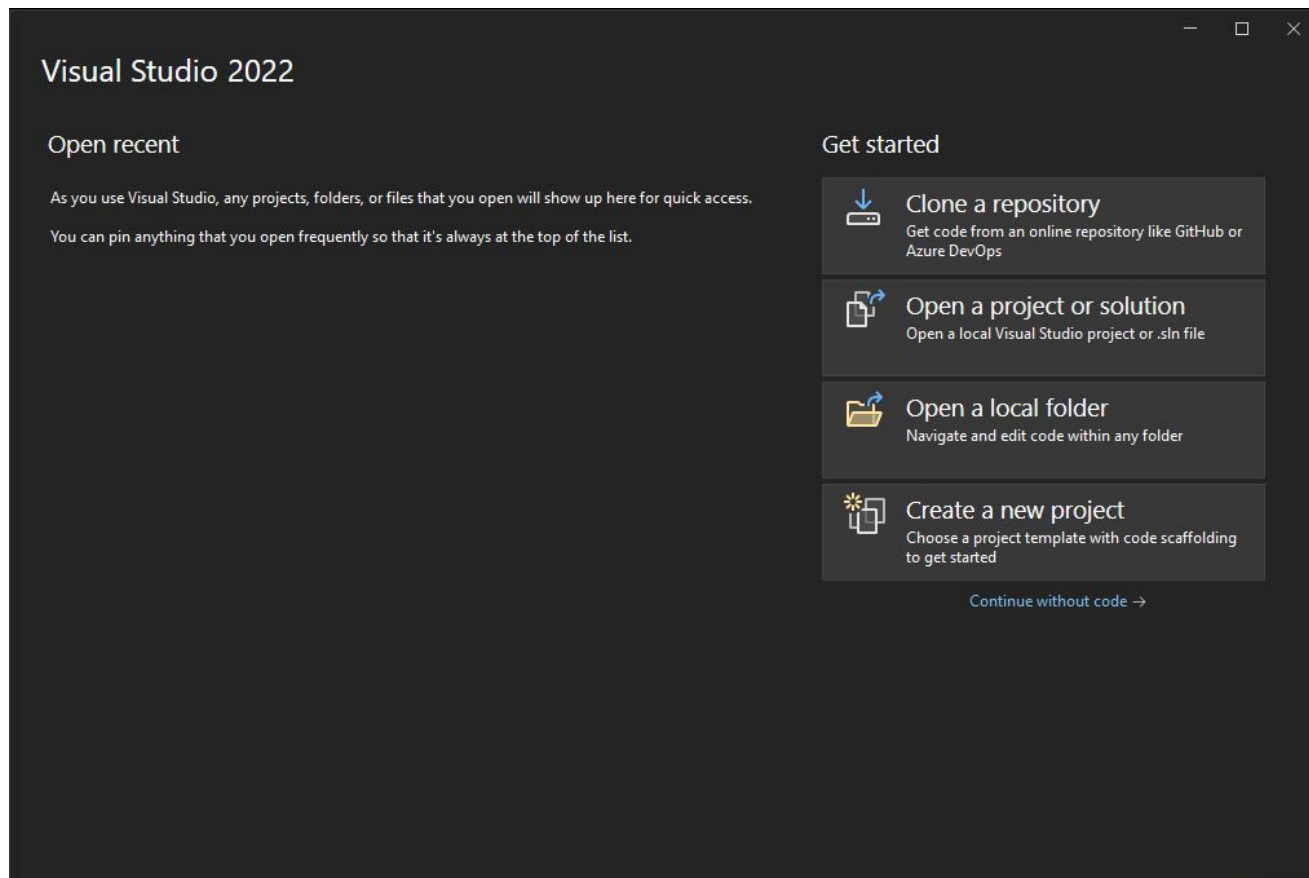


☐ Light



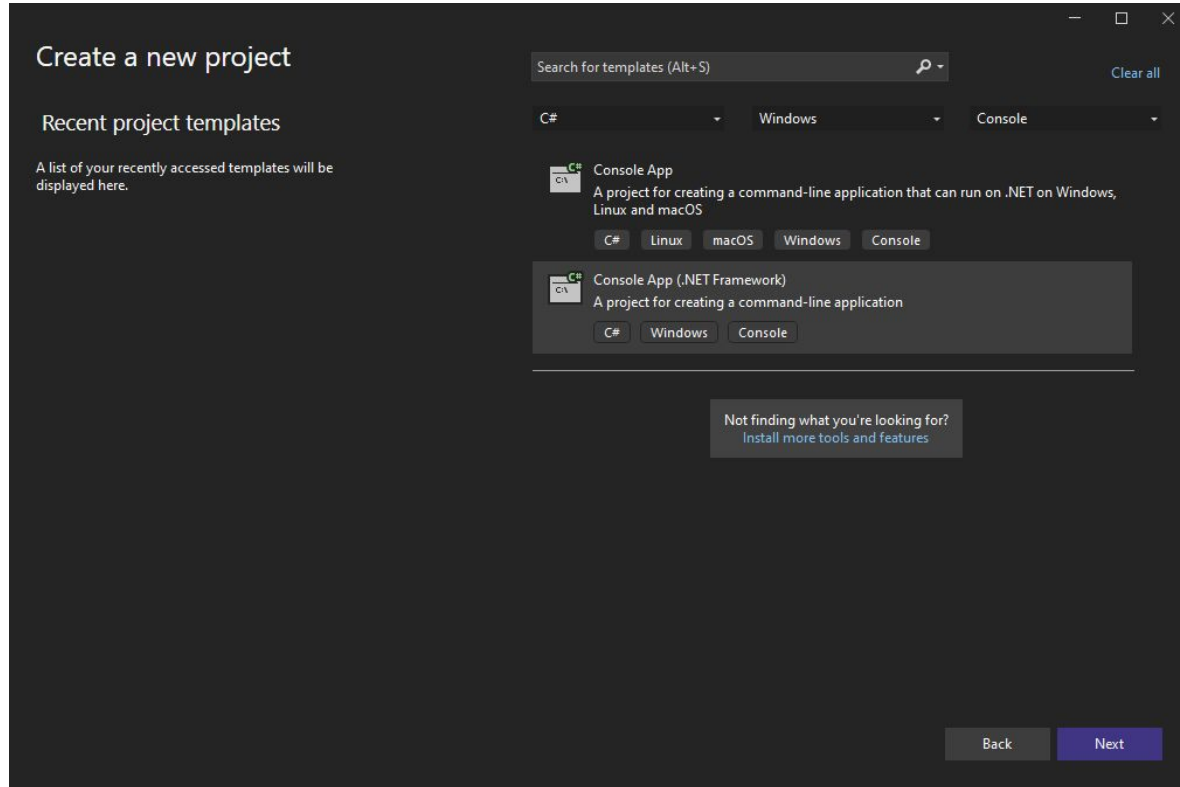
Start Visual Studio

To create a new project, click on “create a new project” option.

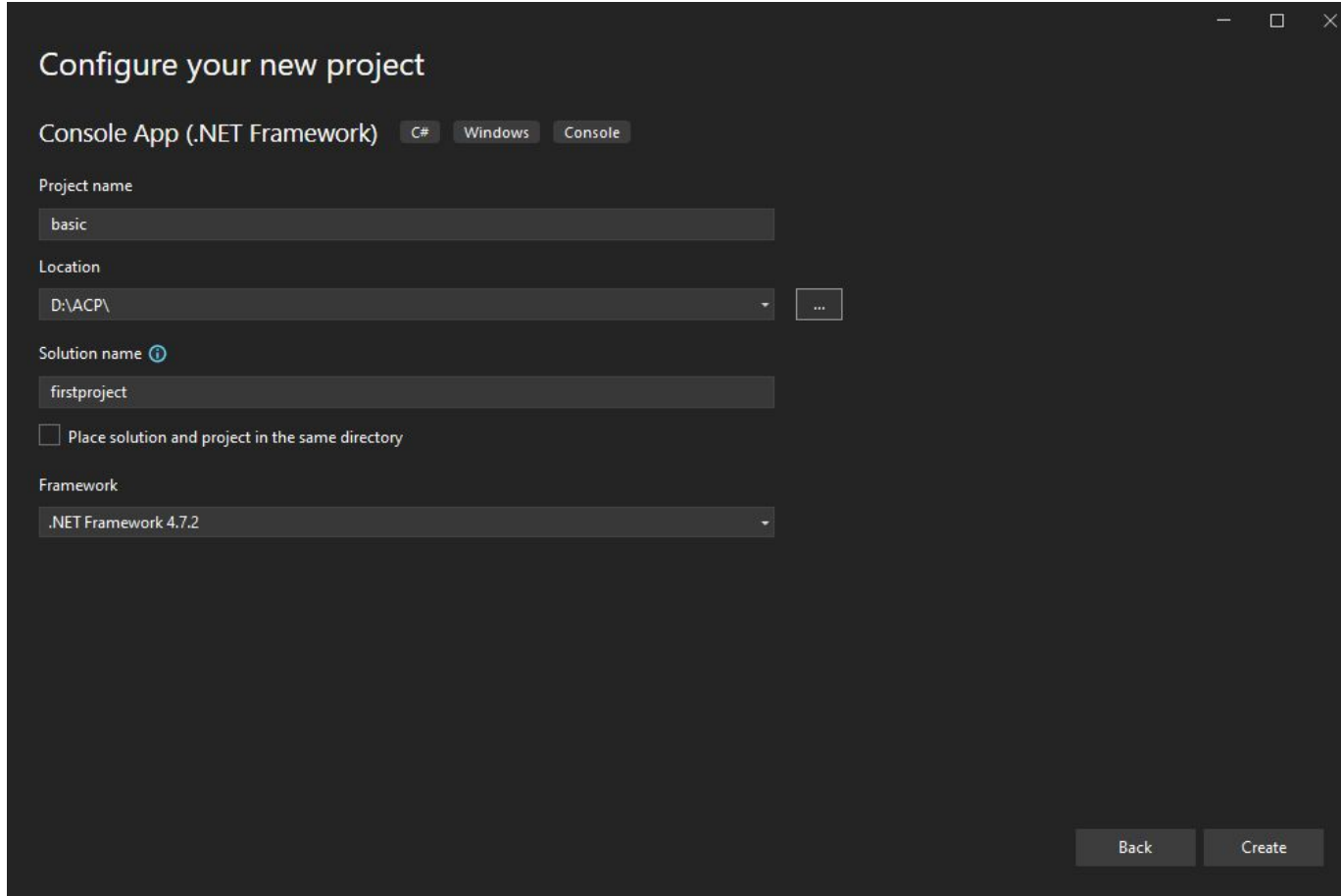


Select language as **C#**, Platform as **windows** and project types as **Console**.

Then click on “Next” Button.



Configure your new project and click on “Create” button.



The screenshot shows the 'Configure your new project' dialog box in Visual Studio. The title bar includes standard window controls (minimize, maximize, close). The main title is 'Configure your new project'. Below it, the project type is 'Console App (.NET Framework)', with tabs for 'C#' (selected), 'Windows', and 'Console'. The 'Project name' field contains 'basic'. The 'Location' field shows 'D:\ACP\' with a dropdown arrow and a browse button (...). The 'Solution name' field, which has a help icon (i), contains 'firstproject'. There is an unchecked checkbox labeled 'Place solution and project in the same directory'. The 'Framework' dropdown menu is set to '.NET Framework 4.7.2'. At the bottom right, there are 'Back' and 'Create' buttons.

Configure your new project

Console App (.NET Framework) C# Windows Console

Project name

basic

Location

D:\ACP\

Solution name ⓘ

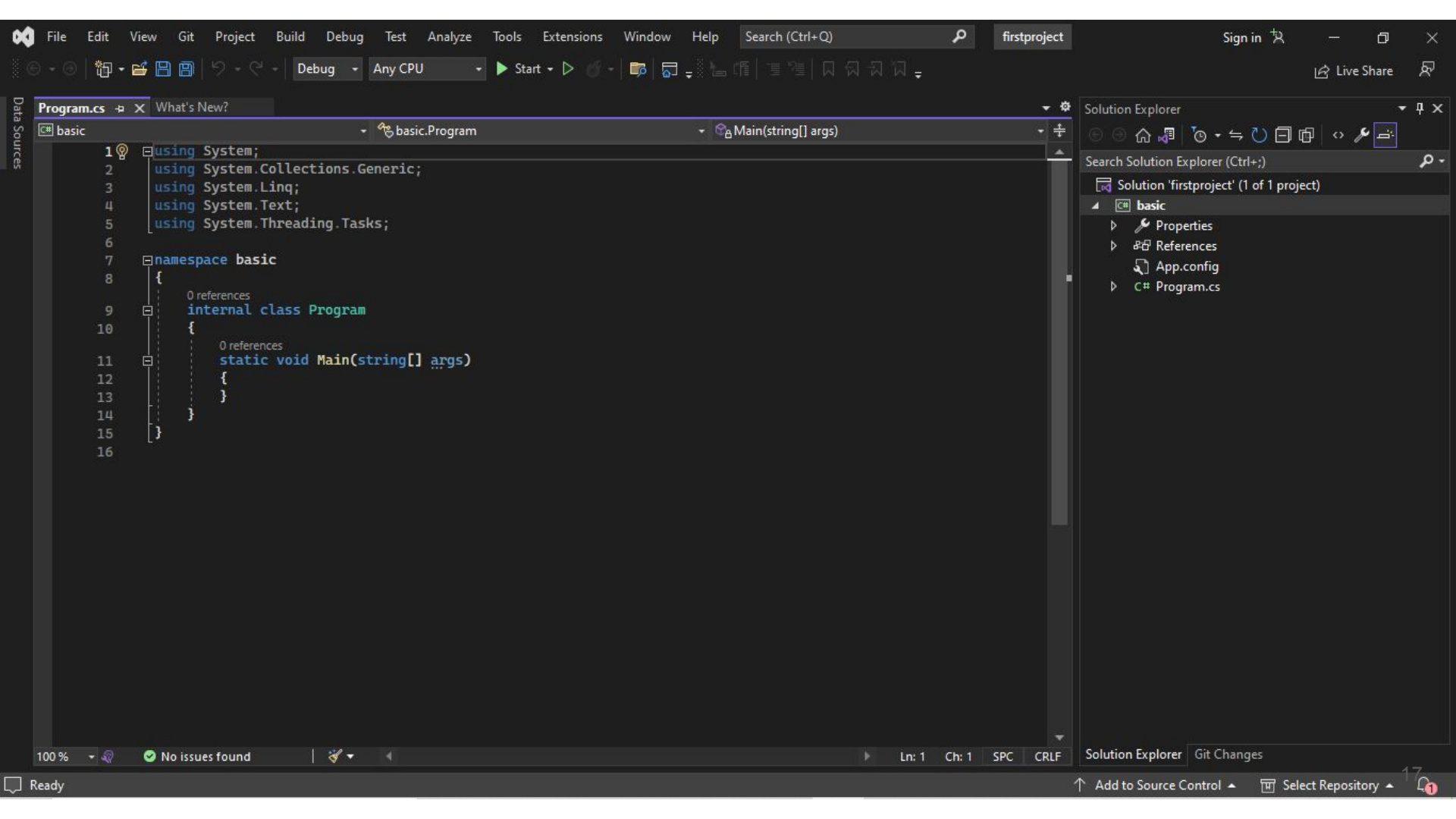
firstproject

☐ Place solution and project in the same directory

Framework

.NET Framework 4.7.2

Back Create



Difference between write and writeline function

WriteLine():

Print the statements in separate line.

```
Console.WriteLine("Hello");
```

```
Console.WriteLine("Everybody");
```

Output:

Hello
Everybody

Write():

Print the statements in same Line.

```
Console.Write("Hello");
```

```
Console.Write("Everybody");
```

Output:

Hello Everybody

Data types

Int : Stores Integers, without decimal.(123)

Double: Stores floating point numbers, with decimals.(19.99).

Char: Stores single characters.

Char values surrounded by single quotes.(‘a’, ‘b’).

String : Stores text.

String values surrounded by double quotes.(“hello world”).

Bool : Stores values with two statements:

True or false

Declare a variable

Syntax: data_type variable_name

```
Int num;
```

Declare and Initialize a variable

Syntax : data_type variable_name = value

```
Int num=52;
```

Constants

If you don't want to overwrite existing values, you can add the “const” keyword in front of the data type.

```
Const int num=15;
```

When you change the value of a constant variable, it will show an error message.

```
num=20;
```

You can not declare a constant variable without assigning the value.

```
const char letter;
```

Type conversion

Implicit casting : converting smaller type to a larger type size.(Automatically)

Char → int → long → float → double

```
Int myint=9;
```

```
Double mydouble=myint; //automatic casting
```

Explicit casting: converting a larger type to a smaller size type.(manually)

double → float → long → int → char

```
Double mydouble=9.78;
```

```
Int myint=(int)mydouble;
```

Type conversion methods

`Convert.ToBoolean();`

`Convert.ToDouble();`

`Convert.ToString();`

`Convert.ToInt32();` //for integer numbers

`Convert.ToInt64();` //for long numbers

Write a simple c# program to Convert a integer type value into a boolean datatype.

Get user input

```
Console.ReadLine();
```

Get the username from the user and print.

```
Console.WriteLine("Enter the username:");
```

```
String username=Console.ReadLine();
```

```
Console.WriteLine(" username is:"+username);
```

Get the user age from the user and print.

If statement

The if statement can cause other statements to execute only under certain conditions.

```
if(condition)
{
    statements
}
```

Exercises

01. Take two integer values from the user and find the summation.

02. The user enters three test scores and the program calculates their average.

If the average equals 100, the program congratulates the user on earning a perfect score.

03. The user enters the price of a product. If the value of that product is greater than \$500, then assign 0.2 to the price as a discount rate. Finally display the discount rate and the final selling price to the user.

04. Get the working hours from the employee. If the number of hours worked exceeds 40, multiply the pay rate by 1.5. (pay rate=42000).

05. A salesperson receives a commission for his sales in addition to a monthly salary of \$30,000. If the sales are greater than \$50,000, then a commission rate of 0.25% of his sales, and a bonus of \$250 to the salary. This month, he sells for \$62,000. Finally, calculate the salesman's commission rate and monthly salary.

If / else statement

The if / else statement will execute one set of statements when the if condition is true and another set when the condition is false.

```
if(condition)
```

```
{
```

```
    Statement set 1;
```

```
}
```

```
else
```

```
{
```

```
    Statement set 2;
```

```
}
```

Exercises

06. Get a number from the user.

Check if the given number is odd or even.

07. Write a program to test the value of a divisor before the division takes place. If the divisor is not equal to zero, then print the quotient. or else print the display message as "division by zero is not possible".

if/else if statement

The if / else if statement is a chain of if statements. They perform their tests, one after the other, until one of them is found to be true.

If (condition 1)

```
{  
    Statement set1;  
}
```

else if (condition 2)

```
{  
    Statement set 2;  
}
```

else if(condition n)

```
{  
    Statement set n;  
}
```

Exercises

08. Write a program to assign a letter grade of A, B,C,D or F to a numeric test score.

A trailing else is used to set if a negative value is entered.

Score ≥ 90 = A

Score ≥ 80 = B

Score ≥ 70 = C

Score ≥ 60 = D

Score ≥ 0 = F

Get a test score value from the user.

09. A student will not be allowed to sit in exam if his/her attendance is less than 75%.

Take following input from user

Number of classes held

Number of classes attended.

And print

percentage of class attended

Is student is allowed to sit in exam or not.

Attendance percentage = $\frac{\text{no. of days a student is present}}{\text{Total no. of days of attendance}} * 100$

10. Write a program to check whether a entered character is lowercase (a to z) or uppercase (A to Z).

11. A shop will give discount of 10% if the cost of purchased is more than 1000.

Ask user for quantity

Suppose, one unit will cost 100.

Judge and print total cost for user.

The switch statement

```
switch(Integer Expression)
{
    case constantExpression:
        //place one or more statements here
    case constantExpression:
        //place one or more statements here
    default:
        //place one or more statements here
}
```

The integer expression can be either of the following:

- A variable of any of the integer data types(including char).
- An expression whose value is of any of the integer data types.

The constant expression can be either an integer literal or an integer named constant.

An optional default section comes after all the case statements. This section branched to if none of the case expressions match the switch expression.(like trailing else in an if / else if statement).

Exercises

12. Write a program to calculate the membership rates based on the user's choice.

Membership rates:

ADULT_RATE=120.0

CHILD_RATE=60.0

SENIOR_RATE=100.0

Health club membership Menu:

1 - standard adult membership

2 - child membership

3 - senior citizen membership

4 - quit the program

Ask the user for how many months they need the membership.

Based on the months, calculate charges and display them to the user.

If the user's selection is not between one and four, display a warning message instructing the user to select the correct option.

Looping

A loop is a control structure that causes a statement or group of statements to repeat.

C# has three looping control structures:

- While loop
- do-while loop
- For loop

The difference between each of these is how they control the repetition.

The while loop

The while loop has two important parts:

1. An **expression** that is tested for a true or false value.
2. A **statement or block** that is repeated as long as the expression is true.

General format of the while loop:

```
while(condition) //no semicolon after the condition expression.
```

```
{
```

```
    Statement;
```

```
    Statement;
```

```
    //place as many statements
```

```
    //here as necessary
```

```
}
```

How the loop works

The condition expression is tested, and if it is true,

Each statement in the body of the loop is executed.

Then the condition is tested again. If it is still true,

Each statement is executed again.

This cycle repeats until the condition is false.


```
Int count=1;  
while(count <= 5)  
{  
    Console.WriteLine("Hello");  
    count++;  
}
```

Each execution of a loop is known as an **iteration**.

- In the above example, loop will perform five iterations.

A variable that controls the number of times a loop iterates is referred to as a **loop control variable**.

- In the above example, count is the loop control variable.

While is a **pretest loop**.

This means it tests its condition before each iteration.

infinite loops.

If a loop does not have a way of stopping, it is called an infinite loop.

infinite loops keep repeating until the program is interrupted.

A loop must include a way to terminate.

If there is only one statement repeated by the loop. It should appear on the line after the while statement and be indented one level.

If the loops repeats a block of statements, the block should begin on the line after the while statement, and each line inside the braces should be indented.

The while loop can be used to create input routines that repeat until acceptable data is entered.

```
Console.WriteLine("Enter a number in the range 1- 100: ");  
Int number=Convert.ToInt32(Console.ReadLine()); //102  
while(number<1 || number>100)  
{  
    Console.WriteLine("Enter a number in the range 1-      100: ");  
    Int number=Convert.ToInt32(Console.ReadLine()); //100  
}  
Console.WriteLine(number);
```

Exercises

Q01. Write a C# program to displays the numbers from 1 up to 5. The second displays the numbers from 5 down to 1.

Use increment and decrement operators to change the value.

Sample output:

```
1    2    3    4    5
5    4    3    2    1
```

Q02. Write a program to displays integer numbers and their square root, beginning with one and ending with whatever number the user requests.

Do-while loop

The do-while loop is a post-test loop, which means its expression is tested after each iteration.

```
do  
{  
    Statement;  
    statement;  
}  
while(condition);
```

The do-while loop **must be terminated with a semicolon** after the closing parentheses of the test expression.

```
Int x=1;
```

```
while(x<0)
```

```
    Console.WriteLine(x);
```

You should use the do-while loop when you want to make sure the loop executes at least once.

```
Int x=1;
```

```
do
```

```
    Console.WriteLine(x);    //1
```

```
while(x<0);
```

Exercise

Q03. Write a program to calculate the averages of 3 test scores.

Use the do-while loop that allows the code to repeat as many times as the user wishes.

The for loop

The for loop is a pretest loop that combines the initialization, testing, and updating of a loop control variable in a single loop header.

```
for(initialization; test; update)
```

```
{
```

```
    Statement;
```

```
    Statement;
```

```
}
```

- 1.it must initialize a counter variable to a starting value.
- 2.it must test the counter variable by comparing it to a final value. When the counter variable reaches its final value, the loop terminates.
- 3.it must update the counter variable during each iteration. This is usually done by incrementing the variable.

Exercises

Q1. create a C# program, to print the factorial up to number n.

Get the number n from the user

Q2. create a C# program, to print the Fibonacci series up to number n.

Get the number n from the user.

0, 1, 1, 2, 3, 5, 8, 13, 21...

Nested loops

A loop that is inside another loop is called a nested loop.

```
while(condition1)                                //beginning of the outer loop
{
    //statements
    while(condition2)                            //beginning of the inner loop
    {
        //statements
    }                                            //end of the inner loop
}
```

Exercise

Q3. Write a program to calculate the average of the scores.

Asks the user for the number of students and the number of test scores per student.

Use the nested loop.

Breaking out of a loop

C# provides ways to break out of a loop or out of a loop iteration early.

```
Int count=1;
```

```
while(count<=10)
```

```
{
```

```
    Console.WriteLine(count);
```

```
    Count++;
```

```
    if(count==6)
```

```
        break;
```

```
}
```

Exercise

Q4. Write a program to find the square root of 5 numbers entered by the user. However, if a negative number is entered an error message displays and break statement.

The continue statement

Sometimes you want to stay in a loop but cause the current loop iteration to end immediately.

This can be done with the continue statement.

When continue is encountered, all the statements in the body of the loop that appear after it are ignored.

And loop prepares for the next iteration.

```
int test_val=0;
while(test_val<10)
{
    test_val++; //1//2//3//4//5
    if(test_val==4)
        Continue; //terminate this iteration of the loop.
    Console.WriteLine(test_val+" "); //1//2//3//5
}
1 2 3 5 6 7 8 9 10
```

Exercise

Q5.

Write a program to calculate the charges for DVD rentals where current releases cost rs.35.00 and all others cost rs.25.00. If a customer rents several DVDs, every third one is free.

Arrays in C#.

Arrays are used to store multiple values in a single variable.

Create an array

```
string[] flowers;
```

Create an array with size.

```
string[] flowers = new string[4];
```

Create an array of three elements and add values

```
string[] fruits = new string[3] {"Mango", "Apple", "Orange"};
```

Create an array of four elements without specifying the size.

```
string[] city = new string[] {"vavuniya", "colombo", "kandy", "jaffna"};
```

Create an array of four elements, omitting the new keyword, and without specifying the size.

```
string[] mobile = {"samsung", "oppo", "windows", "redmi"};
```

Make an integer type array of size 5 and insert values into it.

Access the Elements of an Array

Access an array element by referring to the index number.

The following statement accesses the value of the first element in mobile:

```
string[] mobile = {"samsung", "oppo", "windows", "redmi"};
```

```
Console.WriteLine(mobile[0]);
```

Output?

Array indexes start with 0: [0] is the first element. [1] is the second element.

Print the "windows" as an output from the mobile array.

Change an Array Element

To change the value of a specific element, refer to the index number.

In the array `mobile`, change the value "Samsung" to "iPhone."

```
string[] mobile = {"samsung", "oppo", "windows", "redmi"};
```

```
mobile[0]="iphone";
```

Print the first element from the `mobile` array.

```
Console.WriteLine(mobile[0]);
```

Array Length

To find out how many elements an array has, use the **Length** property:

```
string[] city = {"vavuniya", "colombo", "kandy", "jaffna"};
```

```
Console.WriteLine(city.Length);
```

output?

Print the array elements.

Declare and initialize an array of available departments at the University of Vavuniya.

display all the departments that you have mentioned in the array.

// Declare an array

```
String[] departments;
```

// Add values, using new

```
departments = new string[]  
{ "ICT", "Bio-Science", "Physical-science", "HRM", "PM", "Accounting", "Marketing" };
```

//print the array elements

```
for (int i = 0; i < departments.Length; i++)
```

```
{
```

```
    Console.WriteLine(departments[i]);
```

```
}
```

The foreach Loop

which is used exclusively to loop through elements in an array.

```
foreach (data-type variable-Name in array-Name)
```

```
{
```

```
    // code block to be executed
```

```
}
```

```
foreach (string i in departments)
```

```
{
```

```
    Console.WriteLine(i);
```

```
}
```


Sort an Array

sorts an array alphabetically or in an ascending order.

Sort a string array.

```
string[] subjects = {"Maths", "Science", "History", "ICT","English","commerce"};  
Array.Sort(subjects);  
foreach (string i in subjects)  
{  
    Console.WriteLine(i);  
}
```

Sort an int array

```
int[] my_Numbers = {5, 1, 8, 9};
```

```
Array.Sort(my_Numbers);
```

```
foreach (int i in my_Numbers)
```

```
{
```

```
    Console.WriteLine(i);
```

```
}
```

Array methods

array methods, such as Min, Max, and Sum, can be found in the **System.Linq** namespace.

```
int[] my_Numbers = {5, 1, 8, 9};
```

```
Console.WriteLine(my_Numbers.Max()); // returns the largest value
```

```
Console.WriteLine(my_Numbers.Min()); // returns the smallest value
```

```
Console.WriteLine(my_Numbers.Sum()); // returns the sum of elements
```

Finding index of an array element

```
int[] arr = new int[6] { 5, 8, 9, 25, 0, 7 };
```

```
Console.WriteLine("\nIndex position of 25 is "+Array.IndexOf(arr,25));
```

Coping first array to empty array

```
int[] arr2 = new int[6];
```

```
Array.Copy(arr, arr2, arr.Length);
```

```
Console.WriteLine("Second array elements: ");
```

```
foreach(int i in arr2)
```

```
{
```

```
    Console.Write(i+"\t");
```

```
}
```

Display the array elements in reverse order

```
int[] arr = new int[6] { 5, 8, 9, 25, 0, 7 };  
Console.WriteLine("Original array: ");  
    foreach (int i in arr)  
    {  
        Console.Write(i + " ");  
    }  
Console.WriteLine(" ");  
Console.WriteLine("Reverse array: ");  
Array.Reverse(arr);  
foreach(int i in arr)  
{  
    Console.Write(i+" ");  
}
```

Exercises

Q01. Take the array length from the user.

Declare an array with the array length.

Get the values from the user and store them in an array.

Finally, print them on screen.

Q02. write a program to count the number of odd and even elements on an array shown in figure 1.

7	2	8	5	10	1	4	9	3	17
---	---	---	---	----	---	---	---	---	----

Figure 1

Sample output:

Number of odd elements: 6

Number of even elements: 4

Q03. write a c# program to sort an array in descending order.

Q04. C# program to insert an element in array.

Get the element and position from the user.

Q05. C# program to delete element from an array

Get the element from the user that they want to delete.

C# Multidimensional Arrays

A multidimensional array is basically an array of arrays.

Arrays can have any number of dimensions. The most common are two-dimensional arrays (2D).

if you want to store data as a tabular form, you need to get familiar with multidimensional arrays.

Create a 2D array

```
int[,] numbers = { {1, 4, 2}, {3, 6, 8} };
```

The single comma [,] specifies that the array is two-dimensional.

Numbers array in tabular form

	Column-01	Column-02	Column-03
Rows-01	1	4	2
Rows-02	3	6	8

Create a 2D integer array with 3 rows and 4 columns. Insert a value into that array as you wish.

Print the 2D array elements

```
int[,] numbers = { {1, 4, 2}, {3, 6, 8} };
```

```
foreach (int i in numbers)
```

```
{
```

```
    Console.WriteLine(i);
```

```
}
```

```
int[,] numbers = new int[3, 4] { { 1, 4, 7, 8 }, { 2, 5, 8, 9 }, { 3, 6, 9, 5 } };
```

```
for (int i = 0; i < 3; i++) //for (int i = 0; i < numbers.GetLength(0); i++)  
{  
    for (int j = 0; j < 4; j++)//for (int j = 0; j < numbers.GetLength(1); j++)  
    {  
        Console.Write(numbers[i, j]+" ");  
    }  
    Console.WriteLine();  
}
```

Access Elements of a 2D Array

```
int[,] numbers = { {1, 4, 2}, {3, 6, 8} };
```

```
Console.WriteLine(numbers[0, 2]);
```

Output?

Change Elements of a 2D Array

```
int[,] numbers = { {1, 4, 2}, {3, 6, 8} };
```

```
numbers[0, 0] = 5; // Change value to 5
```

```
Console.WriteLine(numbers[0, 0]); // Outputs 5 instead of 1
```

Exercise

You are requested to perform the following tasks to write c# program using array in order to find a smallest and largest element of a given matrix.

Declare a two-dimensional array.

Store the array values in a matrix and print it.

finally , print the smallest and largest element.

Sample input and output:

Enter no.of rows: 2

Enter no.of columns: 2

Enter the elements to matrix below:

Enter a [0][0] element: 1

Enter a [0][1] element: 2

Enter a [1][0] element: 3

Enter a [1][1] element: 4

The given matrix is:

1 2

3 4

Smallest element: 1

Largest element: 4

Methods in C#

A method is a **block of code** which only runs when it is called.

You can pass data, known as **parameters**, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods?

To reuse code.

Types of functions in C#

Basically, there are two types of Functions in C#

Built-in functions

The function which is **already defined** in the framework and available to be used by the developer or programmer is called a built-in function.

```
int number = 25;
```

```
double square_Root = Math.Sqrt(number);
```

User defined functions

if the function is **defined by the developer or programmer** explicitly then it is called a user-defined function.

Create a user-defined functions.

Static function

If you declare the method with a static modifier, then you can access the method directly without creating an instance.

Return type function

A return statement can also return a value to the calling function.

It cannot return multiple values but it can take multiple values as parameters.

If the function is **not returning any value**, then the return type should be **void**.

<Access specifier><modifier><return type>function-name(parameter list)

{

 //function body

}

Exercise

Create functions for the following operations to perform with two integers.

- o Addition
- o Subtraction
- o Multiplication
- o Division
- o Modulus

To do the above operations, take two integer values from the user and pass those values in as arguments to the function.

Get the operator from the user, and based on that operator, display the results.

Refer to the sample output in Figure 1.

```
Enter the operator that you want to perform:
+ : for summation
- : for subtraction
* : for multiplication
/ : for division
% : for modulus
*
Enter the first value:
5
Enter the second value:
6
multiplication of 5 and 6 is 30
```

```
Enter the operator that you want to perform:
+ : for summation
- : for subtraction
* : for multiplication
/ : for division
% : for modulus
1
select the correct operator
```

Figure: 1

Assignment: 01

C# OOP

OOP stands for Object-Oriented Programming.

object-oriented programming is about creating objects that contain both data and methods.

Example: Mobile phone

Methods: call, message, music player

Attributes: brands, colour.

Everything in C# is associated with **classes** and **objects**.

Class : A class is a blueprint to create an objects.

To create a class, use the **class** keyword

```
class mobile
```

```
{
```

```
}
```

Objects: An object is an instance of a class.

When the individual objects are created, they inherit all the variables and methods from the class.

To create an object, specify the class name, followed by the object name, and use the keyword **new**.

```
mobile obj=new mobile();
```

```
class mobile
```

```
{
```

```
    string brand="samsung";
```

```
    Public static void main(string[] args)
```

```
    {
```

```
        mobile obj = new mobile();
```

```
        Console.WriteLine(obj.brand);
```

```
    }
```

```
}
```

oop.cs

Constructors in C#

It is a **special method** present inside a class.

Responsible for **initializing the variables** of the class.

The name of the constructor method is exactly the **same name** as the class in which it was present.

if your class name is Student, then the constructor name is also going to be Student.

The constructor method **does not return any value**.

Implicitly Defined Constructors:

parameterless constructors.

Also known as Default Constructors-because they are used to initialize the variables with default values.(Numeric types:0,string:null,boolean:false).

Implicitly Defined Constructors are public.

Explicit Constructor:

If a programmer defines the constructor, that will be called an "explicit constructor."

Explicit Constructor can be **parameterless** and **parameterized** also.

<Modifiers><class name>(parameter list)

{

 //statements

}

[implicitly-constructor.cs](#)

Types of Constructors in C#

There are five types of constructors available in C#.

1. Default or Parameter Less Constructor
2. Parameterized Constructor
3. Copy Constructor
4. Static Constructor
5. Private Constructor

Default or Parameter Less Constructor.

If a constructor method does not take any parameters, then we call that a Default or Parameter Less Constructor.

System Defined default constructor.

User defined default constructor.

```
using System;

namespace ConstructorDemo
{
    class Employee
    {
        public int Id, Age;
        public string Address, Name;
        public bool IsPermanent;
    }
}
```

```
class Test
```

```
{  
    static void Main(string[] args)  
    {  
        Employee e1 = new Employee();  
        Console.WriteLine("Employee Id is: " + e1.Id);  
        Console.WriteLine("Employee Name is: " + e1.Name);  
        Console.WriteLine("Employee Age is: " + e1.Age);  
        Console.WriteLine("Employee Address is: " + e1.Address);  
        Console.WriteLine("Is Employee Permanent: " + e1.IsPermanent);  
        Console.ReadKey();  
    }  
}
```

```
using System;

namespace ConstructorDemo
{
    class Employee
    {
        public int Id, Age;

        public string Address, Name;

        public bool IsPermanent;

        //User Defined Default Constructor

        public Employee()
        {
            Id = 100;

            Age = 30;

            Address = "vavuniyar";

            Name = "prarthana";

            IsPermanent = true;
        }
    }
}
```

```
public void Display()

{

    Console.WriteLine("Employee Id is: " + Id);

    Console.WriteLine("Employee Age is: " + Age);

    Console.WriteLine("Employee Address is: " + Address);

    Console.WriteLine("Employee Name is: " + Name);

    Console.WriteLine("Is Employee Permanent: " + IsPermanent);

}

}

class Program

{

    static void Main(string[] args)

    {

        Employee e1 = new Employee();

        e1.Display();

        Console.ReadKey();

    }

}

}
```

Parameterized Constructor.

If a constructor method is defined with parameters, we call it a Parameterized Constructor in C#.

With the help of a Parameterized constructor, we can initialize each instance of the class with a different set of values.

Defined by the programmers.

```
using System;
```

```
namespace ConstructorDemo
```

```
{
```

```
public class ParameterizedConstructor
```

```
{
```



```
public ParameterizedConstructor(int i)
{
    Console.WriteLine($"Parameterized Constructor is Called: {i}");
}
}

class Program
{
    static void Main(string[] args)
    {
        ParameterizedConstructor obj = new ParameterizedConstructor(10);
        ParameterizedConstructor obj2 = new ParameterizedConstructor(20);
        Console.ReadKey();
    }
}
```

Copy Constructor in C#

If we want to create multiple instances with the same values then we need to use the copy constructor in C#.

```
using System;
```

```
namespace ConstructorDemo
```

```
{
```

```
    public class CopyConstructor
```

```
{
```

```
    int x;
```

```
//Parameterized Constructor
```

```
public CopyConstructor(int i)
```

```
{
```

```
    x = i;
```

```
}
```

```
//Copy Constructor
```

```
public CopyConstructor(CopyConstructor obj)
```

```
{
```

```
    x = obj.x;
```

```
}
```

```
public void Display()
```

```
{
```

```
    Console.WriteLine($"Value of X = {x}");
```

```
}
```

```
}
```

```
class Program
```

```
{
```

```
    static void Main(string[] args)
```

```
    {
```

```
        CopyConstructor obj1 = new CopyConstructor(10);
```

```
        obj1.Display();
```

```
        CopyConstructor obj2 = new CopyConstructor(obj1);
```

```
        obj2.Display();
```

```
        Console.ReadKey();
```

```
    }
```

```
}
```

```
}
```

Static Constructor in C#.

- If a constructor is declared explicitly by using the static modifier, then it is called a static constructor in C#.
- There can be only one static constructor in a class.
- The static constructor should be without any parameters.
- It can only access the static members of the class.
- There should not be any access specifiers in the static constructor definition.
- If a class is static then we cannot create the object for the static class.
- It is called automatically to initialize the static members.
- Static constructor will be invoked only once.

```
using System;

namespace ConstructorDemo
{
    public class StaticConstructor
    {
        int i;

        Static int j;

        static StaticConstructor()
        {
            j=100;//allowed

            i=50;//not allowed

            Console.WriteLine("Static Constructor Executed!");
        }
    }
}
```

```
public StaticConstructor
{
    i=500;
    j=250;//allowed
}

static void Main(string[] args)
{
    Console.WriteLine("Main Method Exceution Started...");
    Console.ReadKey();
}
}
```

Private Constructor in C#.

The constructor whose accessibility is private is known as a private constructor.

We need to use the private constructor in C# when the class contains only static members.

Using a private constructor is not possible to create an instance from outside the class.

So, private constructors are used to create an object for the class within the same class.


```
using System;

namespace ConstructorDemo
{
    class Program
    {
        private Program()
        {
            Console.WriteLine("This is private constructor");
        }

        static void Main(string[] args)
        {
            Program p = new Program();

            Console.WriteLine("Main method");

            Console.ReadKey();
        }
    }
}
```

```
public class Person
{
    private string last;
    private string first;

    public Person(string lastName, string firstName)
    {
        last = lastName;
        first = firstName;
    }

    // Remaining implementation of Person class.
}
```

C# Access Specifiers

Public : The type or member can be accessed by any other code in the same assembly or another assembly that references it.

Private: The type or member can be accessed only by code in the same class.

Protected: The type or member can be accessed only by code in the same class, or in a class that is derived from that class.

Internal: The type or member can be accessed by any code in the same assembly, but not from another assembly.

Protected internal: The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly.

Private internal: The type or member can be accessed by types derived from the class that are declared within its containing assembly.

Criteria	private	public	protected	internal	Protected Internal	Private Protected
With the Class	yes	yes	yes	yes	yes	yes
Derived Class in Same Assembly	no	yes	yes	yes	yes	yes
Non-Derived Class in Same Assembly	no	yes	no	yes	yes	no
Derived Class in Other Assemblies	no	yes	yes	no	yes	no
Non-Derived Class in Other Assemblies	no	yes	no	no	no	no

Class mobile

```
{  
    Public string brand="samsung";  
    Private int price=52000;  
}
```

Class testing

```
    Public static void main(string[] args)  
    {  
        mobile obj = new mobile();  
        Console.WriteLine(obj.brand);  
        Console.WriteLine(obj.price);  
    }  
}
```

Basic principles of object-oriented programming

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Encapsulation

Encapsulation Hides the internal data and functionality of an object and only allows access through a public set of functions.

The process of binding or grouping the Data Members and Member Functions together into a single unit (i.e. class, interface, struct, etc).

The Encapsulation Principle ensures that the Data and Functions of a unit cannot be accessed directly from other units.



[Encapsulation_demo.cs](#)

Implementing Data Encapsulation in C# using Properties:

The Properties are a new language feature introduced in C#.

Properties in C# help in protecting a methods or variable of a class by reading and writing the values to it.

Data Encapsulation in C# can be accomplished much smoother with properties.

[Encapsulation.cs](#)

Exercise

Q01.

Create a class by the name calculator which will have 2 methods.

First method will perform multiplication of that two values.

Second method will display the result.

Write a program that should take two input values from a user and perform multiplication of these two values.(take input from main function)

Make the data and multiplication method as private.

Use c# properties to access the data.

[calculator.cs](#)

Abstraction