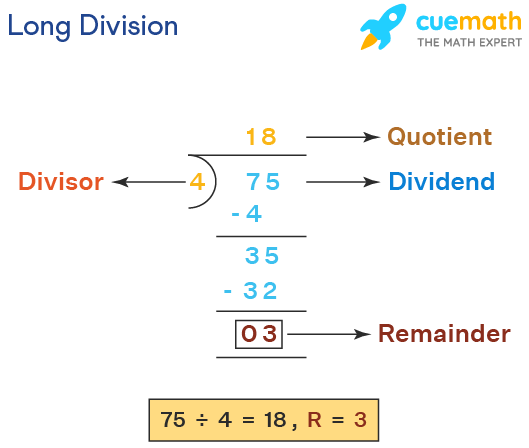
# Division and Mod Concept:



75 (Divident) / 4 (Divisor) = 18 (Quotient), 3 (Remainder/Mod Value)

75 (Divident) mod 4 (Divisor) = 3 (Remainder/Mod Value)

# Check If a Number is Odd or Even

10 % 2 = 0

11 % 2 = 1

# C# While Loop:

while loop loops through a block of code as long as a specified condition is True

while (*condition*)

{

*// code block to be executed*

}

**Example:**

int i = 0;

while (i < 5)

{

Console.WriteLine(i);

i++;

}

# C# Do/While Loop

-A variant of While Loop.

The loop will execute the **code block once**, **before checking if the condition is true**, then it will repeat the loop as long as the condition is true.

do

{

*// code block to be executed*

}

while (*condition*);

**Example:**

int i = 0;  
do

{

Console.WriteLine(i);

i++;

}

while (i < 5)

# Arrays:

-Multiple Values in a single variable of same datatype.

## Support Links:

[**https://www.w3schools.com/cs/cs\_arrays.php**](https://www.w3schools.com/cs/cs_arrays.php)

## Default Methods and Properties:

[**https://www.tutorialspoint.com/csharp/csharp\_array\_class.html**](https://www.tutorialspoint.com/csharp/csharp_array_class.html)

## Declaration:

string[] cars;

## Declaration and Initialization:

int[] carNo = {1,2,3,4,5};

string[] carNames = {“BMW”,”Audi”};

string[] cars = new string[4]; //// Creates array with size

string[] cars = new string[] {"Volvo", "BMW", "Ford", "Mazda"};

## Properties:

carNames.**Length** //// Returns the No of Elements in an Array

## Accessing an Array Value:

string[] cars = new string[] {"Volvo", "BMW", "Ford", "Mazda"};

### Type 1: Using Index

Console.WriteLine(cars[0]); //// Volvo

### Type 2: Using Index and For Loop

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

{

Console.WriteLine(cars[i]);

}

i < cars.Length //// Check the Array Elements Bound with Index

Example:

0 1 2

0<3, 1<3, 2<3, 3<3 FALSE STOP

### IndexOutOfRange Exception:

(i < cars.Length) ---CHECKING if the elements is inside the bounds of ARRAY to avoid IndexOutOfArange Exception

### Type 3: Using Each Element

foreach (string i in cars)

{

Console.WriteLine(i);

}

### Default Methods:

Array.Sort(arrayName); //// sorts the array elements in ascending order

Array.Reverse(arrayName); //// reverse an array order

Array.FindIndex(arrayName, instance => instance.Property > 1) //// Returns the Index of the Condition

Array.Find(arrayName, instance => instance.Property > 1) //// Returns the Person Object.

Array.IndexOf(arrayCollection, arrayInstance); //// Returns Index No

Array.Copy(Source Array, destination Array, Length) //// COPIES from one array to another array.

Example: int personDipakIndex = Array.FindIndex(people, people => people.Id == 1);

int[] arr = { 10, 20 ,30};

Array.Reverse(arr); //// Default Method to reverse an Array

int index = Array.FindIndex(arr, ar => ar == 10);

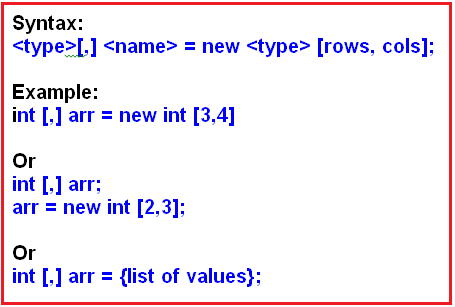
Types of Arrays in C#:  
Single dimensional array

Multi-dimensional array ---->

Jagged array: Whose rows and columns are not equal

Rectangular array: Whose rows and columns are equal

Rectangular Array: A two-dimensional array is an array in which each element is referred to by two indexes. Element in the 2D array is stored in the matrix form. The first index shows the row of the matrix and the second index shows the column of the matrix.

Initialization:  


### Examples:

//// Initialization

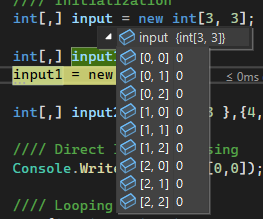
int[,] input = new int[3, 3];

int[,] input1;

input1 = new int[3, 3];

int[,] input2 = { {1,2,3 },{4,5,6 },{7,8,0 } };

Visualization of data inside the array: Row by Row



### Accessing the Array Elements:

int[,] input2 = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 0 } };

* //// Direct Index Accessing

Console.WriteLine(input[0, 0]);

* //// Looping Access

for (int i = 0; i < input2.GetLength(0); i++)

{

for (int j = 0; j < input2.GetLength(1); j++)

{

Console.WriteLine(input2[i, j]);

}

}

foreach(int item in input2)

{

Console.WriteLine(item);

}

### Properties:

arrayName.Length; //// Total elements that can be present inside the array.

Example:

input2.Length; or int[,] input = new int[3, 4]; int length = 3\*4;

### Methods:

arrayName.GetLength(0); //// 0-denotes row size and 1-denoted column size.

### Jagged Array

These are also two-dimensional arrays that will also store the data in the forms of rows and columns. But here in the jagged array, the column size will differ from row to row. That means if the first row contains 5 columns, then the second row may contain 4 columns while the third row may contain 10 columns. So, the point that you need to remember is if the column size varies from row to row then it is a jagged array. If the column size remains the same for all the rows, then it is a rectangular two-dimensional array.

Initialization:

int[][] input = new int[2][];

input[0] = new int[] { 1,2,3};

input[1] = new int[] { 4, 5 };

int[][] input1 = new int[][]

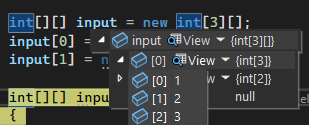
{

new int[] { 1,2},

new int[] { 3,4,5}

};

**Visualization:**



### Accessing the Array Elements:

//// Using Loops

for (int i = 0; i < input.GetLength(0); i++)

{

//arr[i].Length: Returns the Length of Each Row

for (int j = 0; j < input[i].Length; j++)

{

Console.Write(input[i][j] + " ");

}

}

foreach (int[] item in input)

{

//// item is SINGLE DIMENSIONAL using Direct Index Access

Console.Write(item[0]);

//// or

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

{

Console.Write(item[i]);

}

}

### Custom Array:

class Person {

public string Name {

get;

set;

}

public int Age {

get;

set;

}

}

Person[] people = {

new Person {

Name = "Dipak", Age = 25

},

new Person {

Name = "Karan", Age = 30

},

new Person {

Name = "Pankaj", Age = 22

}

};

# Brute Force-Algorithm:

https://studyalgorithms.com/theory/algorithmic-paradigms---brute-force/

-Try solving the problem with all possible combination approach. Even if it is time consuming. If the solution for the problem exisits then try newer approach to solve the problem.

If you were not able to solve using brute force approach then the solution to the problem does not exists.

# C# String

A collection of characters.

## Declaration:

string greeting = "Hello";

//// Length/Count of the String same as length

## Properties:

strVariableName.Length;

### Example:

Console.WriteLine("The length" + greeting.Length); //// OUTPUT: 2

## Default Methods:

### Syntax:

strVariableName.ToUpper();

strVariableName.ToLower();

### Example:

string txt = "Hello World";

Console.WriteLine(txt.ToUpper()); // Outputs "HELLO WORLD"

Console.WriteLine(txt.ToLower()); // Outputs "hello world"

## String Concatenation

The + operator can be used between strings to combine them. This is called concatenation or use default method

### Syntax:

string name = firstName + lastName;

string name = string.Concat(firstName, lastName);

Example:

string firstName = "John ";

string lastName = "Doe";

string name = firstName + lastName;

Console.WriteLine(name);

## String Interpolation

Another option of string concatenation, is string interpolation, which substitutes values of variables into placeholders in a string. Note that you do not have to worry about spaces, like with concatenation:

string firstName = "John"; string lastName = "Doe";

### Syntax:

string name = $"My full name is: {firstName} {lastName}";

## Access Strings:

### Type 1: Using Index

string myString = "Hello";

Console.WriteLine(myString[0]); // Outputs "H"

### Find Index of the Character Inside String Variable :

myString.IndexOf(“H”);

## Substring() Method

-Often combined with Index of to get the all the characters from the index

### Example:

string name = "John Doe";

int charPos = name.IndexOf("D"); // Location of the letter D

string lastName = name.Substring(charPos);

Console.WriteLine(lastName); Doe

## IsNullOrEmpty Method

-Checks if the string variable is empty or null

Example:

if(string.IsNullOrEmpty(sumOfAllArray))

## Converting Methods:

NameSpace: System

Convert.ToString(10);

Convert.ToInt32(“10”);

Convert.ToInt64(“10”);

To Analyze and To Do:

**Time Complexity** – O(1)  
**Space Complexity** – O(1)

Call By Value and Reference

Data types value and reference type

And mutable and immutable type

Datatypes which are call by value <https://dotnettutorials.net/lesson/call-by-value-and-call-by-reference-in-csharp/>

Access Modifiers

Syntax Boxes Align and Example Boxes Align seperately