**How to remove space between words**

static void Main(string[] args)

{

string text = " My name is Forget Code ";

Console.WriteLine(text.Replace(" ", string.Empty));

}

using System.Text.RegularExpressions;

string your\_String = "Hello@Hello&Hello(Hello)";

string my\_String = Regex.Replace(your\_String, @"[^0-9a-zA-Z]+", ",");

**SQL query to get salary between 10k to 15k**

List all the Employee Fname, Lname who is having salary between 30000 and 45000.

**SELECT Fname, Lname**

**FROM Employee**

**WHERE Salary**

**BETWEEN 30000 AND 45000;**

**How to print odd number below 1000**

static void Main(string[] args)

{

int i = 0;

Console.WriteLine("Even Numbers :");

for (i = 1; i <= 1000; i++)

{

if( i%2 == 0 )

{

Console.Write(i + " ");

}

}

Console.WriteLine("\nOdd Numbers :");

for (i = 1; i <= 1000; i++)

{

if (i % 2 != 0)

{

Console.Write(i + " ");

}

}

Console.WriteLine();

}

**first() & firstordefault() diff in LINQ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Name** | **Year** | **Address** | **Income** | **UserName** |
| 1 | PQR | 2010-2011 | C | 50000 | S123 |
| 2 | PQR | 2012-2013 | C | 180000 | S123 |
| 3 | XYZ | 2013-2014 | B | 200000 | S789 |
| 4 | ABC | 2013-2014 | A | 350000 | S253 |

**First()**

1. When we Use First() in LINQ in Query Syntax Or Method Syntax, At that Time If we Do not Get any Record Corresponding To Expression in Where Clause then It Will Throw You Exception as: *InvalidOperationException: Sequence contains no elements.*
   1. Var x=(from m **in** Member
   2. Where m.UserName=’S000’
   3. Select m.Name,m.Income,m.Year ).First()

In The Member table There Is no Such Record Which Will Match the expression so Above Query will Throw: *InvalidOperationException: Sequence contains no elements.*  
We Can Handle This Exception In C# by using try Catch like:

* 1. **try**
  2. {
  3. Var x=(from m **in** Member
  4. Where m.UserName=’S000’
  5. Select m.Name,m.Income,m.Year
  6. ).First()
  8. }
  9. **catch** (InvalidOperationException ex)
  10. {
  11. Console.WriteLine(ex.Message);
  12. }

1. When There Are Multiple Records Releted To TO Matching Expression and If You want only the First One Then You can Use **First()**.  
     
   i.e:
   1. Var x=(from m **in** Member
   2. Where m.UserName=’s123’
   3. Select m.Name,m.Income,m.Year).First()

Then You Will get The Result Like:

|  |  |  |  |
| --- | --- | --- | --- |
| PQR | 2010-11 | 50000 | S123 |

1. **First()** returns First Element Of Sequence.
2. **First()** throws Exception when There IS No element Presnt In Table

**FirstORDefault():**

1. When we Use **FirstORDefault ()** in LINQ in Query Syntax Or Method Syntax, At that Time If we Do not Get any Record Corresponding To Criteria in Where Clause then It Will return Some Default Value (Null).
   1. Var x=(from m **in** Member
   2. Where m.UserName=’S000’
   3. Select m.Name,m.Income,m.Year ).FirstOrDefault()

In The Member table There Is no Such Record Which Will Match the expression so Above Query will Return Default value as Null but Not An Exception.  
  
But when we use above code in c# then it might throw an Exception like:

* 1. Console.WriteLine(x.Name);

It Will Throw An Exception ( because x has Got Null Value because Of Absence Of Record In Table):  
*“object reference not set to an instance of an object”.*  
We Can Handle This Exception In C# by using try Catch like:

* 1. **try**
  2. {
  3. Var x=(from m **in** Member
  4. Where m.UserName=’S000’
  5. Select m.Name,m.Income,m.Year ). FirstOrDefault ()
  6. If(x!=**null**)
  7. {
  8. Console.WriteLine(x.Name);
  9. }
  10. **else**
  11. {
  12. Console.WriteLine("No Record Found");
  13. }
  14. }
  15. **catch** (Exception ex)
  16. {
  17. Console.WriteLine(ex.Message);
  18. }

1. When There Are Multiple Records Releted To TO Matching Expression and If You want only the First One Then You can Use **First().**  
   i.e
   1. Var x=(from m **in** Member
   2. Where m.UserName=’S123’
   3. Select m.Name,m.Income,m.Year ).FirstOrDefault()

Then You Will get The Result Like:

|  |  |  |  |
| --- | --- | --- | --- |
| PQR | 2010-11 | 50000 | S123 |

1. **FirstOrDefault ()** returns First Element Of Sequence.
2. **FirstOrDefault ()** does not throws Exception when There IS No element Present in Table

**IS vs AS OPERATOR?**

The is operator in C# is used to check the object type and it returns a bool value: **true** if the object is the same type and **false** if not.

For null objects, it returns false.

**bool isobject = (Object is Type);**

**namespace IsAndAsOperators**

**{**

**class Student**

**{**

**public int stuNo { get; set; }**

**public string Name { get; set; }**

**public int Age { get; set; }**

**}**

**class Employee**

**{**

**public int EmpNo { get; set; }**

**public string Name { get; set; }**

**public int Age { get; set; }**

**public double Salary { get; set; }**

**}**

**class Program**

**{**

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

**{**

**Student stuObj = new Student();**

**stuObj.stuNo = 1;**

**stuObj.Name = "Siva";**

**stuObj.Age = 15;**

**Employee EMPobj=new Employee();**

**EMPobj.EmpNo=20;**

**EMPobj.Name="Rajesh";**

**EMPobj.Salary=100000;**

**EMPobj.Age=25;**

**// Check Employee EMPobj is Student Type**

**bool isStudent = (EMPobj is Student);**

**System.Console.WriteLine("Empobj is a Student ?: {0}", isStudent.ToString());**

**// Check Student stiObj is Student Typoe**

**isStudent = (stuObj is Student);**

**System.Console.WriteLine("Stuobj is a Student ?: {0}", isStudent.ToString());**

**stuObj = null;**

**// Check null object Type**

**isStudent = (stuObj is Student);**

**System.Console.WriteLine("Stuobj(null) is a Student ?: {0}", isStudent.ToString());**

**System.Console.ReadLine();**

**}**

**}**

**AS**

The as operator does the same job of is operator but the difference is instead of bool, it returns the **object** if they are compatible to that type, else it returns null.

 T[*ypecasting*](https://www.geeksforgeeks.org/c-type-casting/)is an unavoidable thing. In many cases, developer need to convert an Object(Type) into another Object(Type) and sometimes he/she may get InvalidCastException. So, to overcome such types of exception C# provides **as** operator keyword

**Type obj = Object as Type;**

namespace IsAndAsOperators

{

class Student

{

public int stuNo { get; set; } public string Name { get; set; }

public int Age { get; set; }

}

class Employee

{

public int EmpNo { get; set; } public string Name { get; set; }

public int Age { get; set; }

public double Salary { get; set; }

}

class Program

{

static void Main(string[] args)

{

Student stuObj = new Student();

stuObj.stuNo = 1;

stuObj.Name = "Praveen";

stuObj.Age = 15;

Employee EMPobj=new Employee();

EMPobj.EmpNo=20;

EMPobj.Name="Rajesh";

EMPobj.Salary=100000;

EMPobj.Age=25;

System.Console.WriteLine("Empobj is a Student ?: {0}", CheckAndConvertobject(EMPobj));

System.Console.WriteLine("StuObj is a Student ?: {0}", CheckAndConvertobject(stuObj));

System.Console.ReadLine();

}

public static string CheckAndConvertobject(dynamic obj)

{

// If obj is Type student it asign value to Stuobj else it asign null

Student stuobj = obj as Student;

if (stuobj != null)

return "This is a Student and his name is " + stuobj.Name;

return "Not a Student";

} } }

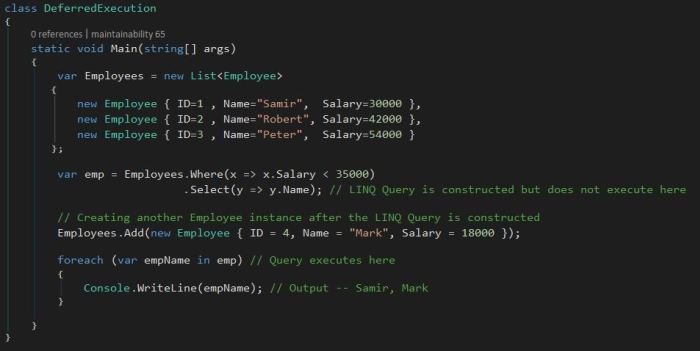
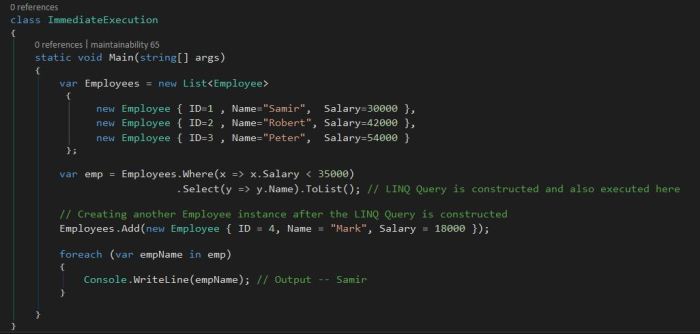
**Lambda Expressions**

* A lambda expression is a convenient way of defining an anonymous (unnamed) function that can be passed around as a variable or as a parameter to a method call
* Func<int, int> multiplyByFive = num => num \* 5; // Returns 35
* int result = multiplyByFive(7);
* The => operator is called the "lambda operator". In this example, num is an input parameter to the anonymous function, and the return value of this function is num \* 5. So when multiplyByFive is called with a parameter of 7, the result is 7 \* 5, or 35
* Notice that the num parameter doesn't explicitly specify a data type
* Notice also that there is no return statement. Single-line lambda expressions don't need to explicitly use the return keyword to return a value

**HTTP Request & HTTP Response**

<https://www.tutorialsteacher.com/webapi/consume-web-api-post-method-in-aspnet-mvc>

**Execution methods in LINQ**

* Deferred and Immediate execution
* **Deferred** - Query is not executed at the point of its declaration
* By default, LINQ uses deferred execution.
* When we write a LINQ query, it does not execute by itself. It executes, when we access the query results.
* In other words, execution of the query is deferred until the query variable is iterated over in a for each loop
* It avoids unnecessary query execution and hence improves performance
* 
* In the above example, we first create a list of 3 Employees. We then construct the LINQ Query. Once done, we create a new instance of the Employee object.
* Now had the query been executed when the LINQ Query was constructed, the result would have been only ‘Samir’. However, that is not the right result in this case.
* The Output of the program is Samir, Mark
* This is because the execution of the query was deferred until the query variable was iterated using a for each loop
* **Immediate –** immediate execution, a query is executed at the point of its declaration
* Immediate execution is the reverse of deferred execution. It forces the LINQ query to execute and gets the result immediately
* To force immediate execution of a query that does not produce a singleton value, we can call the ToList method, the ToDictionary method, or the ToArray method on a query or query variable
* 
* The Output of the above program is only ‘Samir’
* This is because the execution of the query was done immediately, when it was constructed – since we used the ‘ToList’ operator

**DBContext & DBSet in Entity Framework**

* **DBContext** – DbContext is an important class in Entity Framework
* It is a bridge between your domain or entity classes and the database.
* DbContext APIs simplify your application interaction with the database.
* DbContext corresponds to your database (or a collection of tables and views in your database) whereas a DbSet corresponds to a table or view in your database
* **Querying**: Converts LINQ-to-Entities queries to SQL query and sends them to the database.
* **Change Tracking**: Keeps track of changes that occurred on the entities after querying from the database.
* **DBSet** - Any changes you make to the contents of a DbSet will only be committed to the database if the **SaveChanges** method of the DbContext is called
* The DbSet class exposes a number of methods that enable you to perform basic CRUD (Create, Read, Update, Delete) operations against entities

**LINQ to SQL vs LINQ to Entity**

* LINQ to SQL only works with SQL Server.
* LINQ to SQL uses the Data Context class to interact with a database
* LINQ to Entity only works with Multiple Databases.
* LINQ to Entity uses the Data Context class to interact with a database