# Chapter 36: LINQ

LINQ (Language Integrated Query) is an expression that retrieves data from a data source. LINQ simplifies this situation by offering a consistent model for working with data across various kinds of data sources and formats. In a LINQ query, you are always working with objects. You use the same basic coding patterns to query and transform data in XML documents, SQL databases, ADO.NET Datasets, .NET collections, and any other format for which a LINQ provider is available.

## Section 36.1: Selecting from array with simple condition

```
Dim sites() As String = {"Stack Overflow", "Super User", "Ask Ubuntu", "Hardware
Recommendations"}
Dim query = From x In sites Where x.StartsWith("S")
' result = "Stack Overflow", "Super User"
```

Query will be enumerable object containing `Stack Overflow` and `Super User`. `x` in the query is iterating variable where will be stored each object checked by `Where` clause.

## Section 36.2: Mapping array by Select clause

```
Dim sites() As String = {"Stack Overflow",
                         "Super User",
                         "Ask Ubuntu",
                         "Hardware Recommendations"}
Dim query = From x In sites Select x.Length
' result = 14, 10, 10, 24
```

Query result will be enumerable object containing lengths of strings in input array. In this example this would be values 14, 10, 10, 24. x in the query is iterating variable where will be stored each object from the input array.

## Section 36.3: Ordering output

```
Dim sites() As String = {"Stack Overflow",
                         "Super User",
                         "Ask Ubuntu",
                         "Hardware Recommendations"}

Dim query = From x In sites
            Order By x.Length

' result = "Super User", "Ask Ubuntu", "Stack Overflow", "Hardware Recommendations"
```

OrderBy clause orders the output by the value returned from the clause. In this example it is Length of each string. Default output order is ascending. If you need descending you could specify `Descending` keyword after clause.

```
Dim query = From x In sites
            Order By x.Length Descending
```

## Section 36.4: Generating Dictionary From IEnumerable

```
' Just setting up the example
Public Class A
    Public Property ID as integer
    Public Property Name as string
```

```
    Public Property OtherValue as Object
End Class

Public Sub Example()
    'Setup the list of items
    Dim originalList As New List(Of A)
    originalList.Add(New A() With {.ID = 1, .Name = "Item 1", .OtherValue = "Item 1 Value"})
    originalList.Add(New A() With {.ID = 2, .Name = "Item 2", .OtherValue = "Item 2 Value"})
    originalList.Add(New A() With {.ID = 3, .Name = "Item 3", .OtherValue = "Item 3 Value"})

    'Convert the list to a dictionary based on the ID
    Dim dict As Dictionary(Of Integer, A) = originalList.ToDictionary(function(c) c.ID, function(c)
c)

    'Access Values From The Dictionary
    console.Write(dict(1).Name) ' Prints "Item 1"
    console.Write(dict(1).OtherValue) ' Prints "Item 1 Value"
End Sub
```

## Section 36.5: Projection

```
' sample data
Dim sample = {1, 2, 3, 4, 5}

' using "query syntax"
Dim squares = From number In sample Select number * number

' same thing using "method syntax"
Dim squares = sample.Select (Function (number) number * number)
```

We can project multiple result at once too

```
Dim numbersAndSquares =
    From number In sample Select number, square = number * number

Dim numbersAndSquares =
    sample.Select (Function (number) New With {Key number, Key .square = number * number})
```

## Section 36.6: Getting distinct values (using the Distinct method)

```
Dim duplicateFruits = New List(Of String) From {"Grape", "Apple", "Grape", "Apple", "Grape"}
'At this point, duplicateFruits.Length = 5

Dim uniqueFruits = duplicateFruits.Distinct();
'Now, uniqueFruits.Count() = 2
'If iterated over at this point, it will contain 1 each of "Grape" and "Apple"
```