The term ‘Lambda expression’ has derived its name from ‘lambda’ calculus which in turn is a mathematical notation applied for defining functions. Lambda expressions as a LINQ equation’s executable part translate logic in a way at run time so it can pass on to the data source conveniently. However, lambda expressions are not just limited to find application in LINQ only.

These expressions are expressed by the following syntax:

\[
inputparameters => \text{expression or statement block}
\]

Below is an example of lambda expression

\[
y => y \ast y
\]

The above expression specifies a parameter named \( y \) and that value of \( y \) is squared. However, it is not possible to execute a lambda expression in this form. Example of a lambda expression in C# is shown below.

**C#**

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace lambdaexample
{
    class Program
    {
        delegate int del(int i);
        static void Main(string[] args)
        {
            del myDelegate = y => y * y;
            int j = myDelegate(5);
            Console.WriteLine(j);
            Console.ReadLine();
        }
    }
}
```

**VB**

```vbnet
Module Module1
    Private Delegate Function del(ByVal i As Integer) As Integer
    Sub Main(ByVal args As String())
        Dim myDelegate As del = Function(y) y * y
        Dim j As Integer = myDelegate(5)
        Console.WriteLine(j)
        Console.ReadLine()
    End Sub
End Module
```

When the above code of C# or VB is compiled and executed, it produces the following result:

25

**Expression Lambda**

As the expression in the syntax of lambda expression shown above is on the right hand side, these are also known as expression lambda.
Async Lambdas

The lambda expression created by incorporating asynchronous processing by the use of async keyword is known as async lambda. Below is an example of async lambda.

```csharp
Func<Task<string>> getWordAsync = async () => “hello”;
```

Lambda in Standard Query Operators

A lambda expression within a query operator is evaluated by the same upon demand and continually works on each of the elements in the input sequence and not the whole sequence. Developers are allowed by Lambda expression to feed their own logic into the standard query operators. In the below example, the developer has used the ‘Where’ operator to reclaim the odd values from given list by making use of a lambda expression.

C#

```csharp
//Get the average of the odd Fibonacci numbers in the series...
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace lambdaexample
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] fibNum = { 1, 1, 2, 3, 5, 8, 13, 21, 34 };
            double averageValue = fibNum.Where(num => num % 2 == 1).Average();
            Console.WriteLine(averageValue);
            Console.ReadLine();
        }
    }
}
```

VB

```vbnet
Module Module1
    Sub Main()
    Dim fibNum As Integer() = {1, 1, 2, 3, 5, 8, 13, 21, 34}
    Dim averageValue As Double = fibNum.Where(Function(num) num Mod 2 = 1).Average()
    Console.WriteLine(averageValue)
    Console.ReadLine()
    End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
7.33333333333333
```

Type Inference in Lambda

In C#, type inference is used conveniently in a variety of situations and that too without specifying the types explicitly. However in case of a lambda expression, type inference will work only when each type has been specified as the compiler must be satisfied. Let’s consider the following example.

```csharp
delegate int Transformer (int i);
```

Here the compiler employ the type inference to draw upon the fact that x is an integer and this is done by examining the parameter type of the Transformer.
Variable Scope in Lambda Expression

There are some rules while using variable scope in a lambda expression like variables that are initiated within a lambda expression are not meant to be visible in an outer method. There is also a rule that a captured variable is not to be garbage collected unless the delegate referencing the same becomes eligible for the act of garbage collection. Moreover, there is a rule that prohibits a return statement within a lambda expression to cause return of an enclosing method.

Here is an example to demonstrate variable scope in lambda expression.

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace lambdaexample
{
    class Program
    {
        delegate bool D();
        delegate bool D2(int i);

        class Test
        {
            D del;
            D2 del2;
            public void TestMethod(int input)
            {
                int j = 0;
                // Initialize the delegates with lambda expressions.
                // Note access to 2 outer variables.
                // del will be invoked within this method.
                del = () => { j = 10; return j > input; };

                // del2 will be invoked after TestMethod goes out of scope.
                del2 = (x) => { return x == j; };

                // Demonstrate value of j:
                // The delegate has not been invoked yet.
                Console.WriteLine("j = {0}", j);  // Invoke the delegate.
                bool boolResult = del();

                Console.WriteLine("j = {0}. b = {1}", j, boolResult);
            }

        static void Main()
        {
            Test test = new Test();
            test.TestMethod(5);

            // Prove that del2 still has a copy of
            // local variable j from TestMethod.
            bool result = test.del2(10);

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

When the above code is compiled and executed, it produces the following result:

```
j = 0
j = 10. b = True
True
```
Expression Tree

Lambda expressions are used in Expression Tree construction extensively. An expression tree give away code in a data structure resembling a tree in which every node is itself an expression like a method call or can be a binary operation like x<y. Below is an example of usage of lambda expression for constructing an expression tree.

Statement Lambda

There is also statement lambdas consisting of two or three statements, but are not used in construction of expression trees. A return statement must be written in a statement lambda.

Syntax of statement lambda

```
(params) => {statements}
```

Example of a statement lambda

```csharp
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Linq.Expressions;

namespace lambdaexample
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] source = new int[] { 3, 8, 4, 6, 1, 7, 9, 2, 4, 8 };

            foreach (int i in source.Where(x =>
                {
                    if (x <= 3)
                        return true;
                    else if (x >= 7)
                        return true;
                    return false;
                }))
            {
                Console.WriteLine(i);
            }
            Console.ReadLine();
        }
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
3
8
1
7
9
2
8
```

Lambdas are employed as arguments in LINQ queries based on methods and never allowed to have a place on the left side of operators like is or as just like anonymous methods. Although, Lambda expressions are much alike anonymous methods, these are not at all restricted to be used as delegates only.

Important points to remember while using lambda expressions

- A lambda expression can return a value and may have parameters.
- Parameters can be defined in a myriad of ways with a lambda expression.
- If there is single statement in a lambda expression, there is no need of curly brackets whereas if there are multiple statements, curly brackets as well as return value are essential to write.
- With lambda expressions, it is possible to access variables present outside of the lambda expression block by a feature known as closure. Use of closure should be done cautiously to avoid any problem.
- It is impossible to execute any unsafe code inside any lambda expression.
- Lambda expressions are not meant to be used on the operator’s left side.