What is delegate?

A delegate is a type safe function pointer. It holds reference to a function. The signature of delegate must match the signature of the function that it points to else we will get compilation error. Delegates are type safe pointers because it points to a function and holds the signature of the function.

How to Declare Delegates in C#?

Delegate is similar to a class. You can create an instance of it and you need to pass the function name as a parameter to the delegate constructor. All delegates are implicitly derived from the **System.Delegate** class.

Syntax: `Delegate<return type><delegate name><parameter list>`

```
public delegate void Message(string msg);
```

What are the uses of delegate?

- Proper usage of delegates can promote code re-usability and flexibility in your designs.
- Delegates can be used to define callback methods.
- Delegates can be chained together; for example, multiple methods can be called on a single event.
- Delegates allow methods to be passed as parameters.
- These are used to represent or refer to one or more functions.
- These can only be used to define call-back methods.
- In order to consume a delegate, we need to create an object to delegate.

What are the different types of Delegates?

1. **Single Delegate:** A delegate that can call a single method is called single delegate.
2. **Multicast Delegate:** A delegate which call multiple methods. + (pulse) and – (minus) operator use to subscribe and unsubscribe. A Multicast delegate is a delegate that has references to more than one function. When you invoke a multicast delegate, all the functions the delegate is pointing to, are invoked.
3. **Generic Delegate:** Generic delegate does not require an instance of delegate to be defined.

Single Delegate

```
using System;
namespace Delegate_Demo
{
    public delegate void Message(string msg);
    class Program
    {
        static void Main(string[] args)
        {
            Message msg = new Message(PrintMessage);
            msg("Welcome to delegate");
            Console.ReadLine();
        }
        public static void PrintMessage(string message)
        {
            Console.WriteLine(message);
        }
    }
}
```

Multicast Delegate

```
using System;
namespace Delegate_Demo
{
    class MulticastDelegate
    {
        public delegate void PrintMessage();
        static void Main()
        {
            //PrintMessage msg1, msg2, msg3, msg4;
            //msg1 = new PrintMessage(Greeting);
            //msg2 = new PrintMessage(Wish);
            //msg3 = new PrintMessage(Suggesion);
            //msg4 = msg1 + msg2 + msg3;
            //msg4();
            //Console.ReadLine();

            //PrintMessage msg = new PrintMessage(Greeting);
            //msg += Wish;
            //msg += Suggesion;
            //msg();
            //Console.ReadLine();

            PrintMessage msg = new PrintMessage(Greeting);
            msg += Wish;
            msg -= Suggesion;
            msg();
            Console.ReadLine();
        }

        public static void Greeting()
        {
            Console.WriteLine("Good Morning");
        }
        public static void Wish()
        {
            Console.WriteLine("happy Wedding Anniversary");
        }
        public static void Suggesion()
        {
            Console.WriteLine("You should take your wife for a tour.");
        }
    }
}
```

What are the type of generic delegates?

There are three types of generic delegates

1. **Action**
2. **Func**
3. **predicate**

Action is a delegate (pointer) to a method, that takes zero, one or more input parameters, but does not return anything.

```
using System;
namespace Delegate_Demo
{
    class ActionDelegate
    {
        static void Main()
        {
            //Action<int> yourAge = Age;
            //yourAge(30);

            //Action<int> yourAge = age => Console.WriteLine(age);
            //yourAge(30);

            Action<int> yourAge = new Action<int>(Age);
            yourAge(30);
            Console.ReadLine();
        }

        public static void Age(int age)
        {
            Console.WriteLine("Your Age is:"+ age);
        }
    }
}
```

Features:

1. Action delegate is similar to Func delegate except that it does not return anything. Return type must be void.
2. Action delegate can take 1 to 16 input parameters.
3. Action delegate can be used with anonymous methods or lambda expressions.

Func is a delegate (pointer) to a method, that takes zero, one or more input parameters, and returns a value (or reference).

```
using System;
namespace Delegate_Demo
{
    public delegate int Marks(int a,int b,int c,int d, int e, int f);
    class FunDelegate
    {
        static void Main()
        {
            Marks marks = CalculateMarks;
            int Result = marks(80, 75, 58, 72, 89, 60);
            Console.WriteLine("Total Marks:" + Result);
            Console.ReadLine();
        }

        public static int CalculateMarks(int hindi,int english, int physics, int
chemistery, int biology, int math)
        {
            return hindi + english + physics + chemistery + biology + math;
        }
    }
}
```

Features:

1. Func is built in type delegate.
2. Func must return a value.
3. Func can take 0 to 16 input parameter.
4. Func can be used with an anonymous method or lambda expression.

Predicate is a special kind of Func often used for comparisons.

```
using System;

namespace Delegate_Demo
{
    class Predicate
    {
        static void Main()
        {
            Predicate<int> voter = IsEligibleForVote;
            bool result=voter(18);
            Console.WriteLine("Eligible for voting: " + result);
            Console.ReadLine();
        }

        public static bool IsEligibleForVote(int age)
        {
            if (age >= 18)
                return true;
            else
                return false;
        }
    }
}
```

Features:

1. Predicate delegate takes one input parameter and Boolean return type.

2. Predicate delegate must contain same criteria to check whether supplied parameter meets those criteria or not.
3. Anonymous method and lambda expression can be assigned to the predicate delegate.