What are delegates in C#?

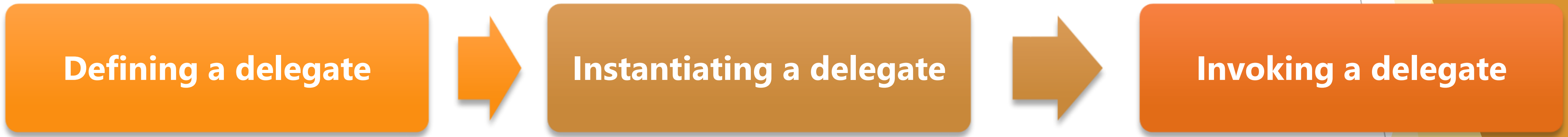
In simple words, we can say that the delegates in C# are the **Type-Safe Function Pointer.** It means they hold the reference of a method or function and then call that method for execution.

How many ways we can call a method in C#?

1.We can call the method using the object of the class if it is a non-static method or we can call the method through class name if it is a static method.

2.We can also call a method in C# by using delegates. Calling a C# method using delegate will be faster in execution as compared to the first process i.e. either by using an object or by using the class name.

How to invoke methods using Delegates in C#?



Step1: Define a Delegate in C#

The syntax to declare a delegate in C# is very much similar to the function declaration. In delegate, we need to use the keyword delegate. The syntax for defining a delegate:

```
<Access Modifier> delegate <return type> <delegate name> (arguments list);
public delegate void AddDelegate(int a, int b);
public void Add(int x, int y)
{
Console.WriteLine(@"The Sum of {0} and {1}, is {2} ", x, y, (x + y));
}
```

Step2: Instantiating the Delegate in C#.

Once we declare the delegate, then we need to consume the delegate. To consume the delegate, first, we need to create an object of the delegate and while creating the object the method you want to execute using the delegate should be passed as a parameter. The syntax to create an instance of a delegate is given below.

```
DelegateName ObjectName = new DelegateName (target function-name);
```

Example:

```
AddDelegate ad = new AddDelegate(obj.Add);
GreetingsDelegate gd = new GreetingsDelegate(Program.Greetings);
```

While binding the method with a delegate, if the method is non-static refer to it as the object of the class and if it is static refer to it with the name of the class.

Step3: Invoking the Delegate in C#.

Once we create an instance of a delegate, then we need to call the delegate by providing the required values to the parameters so that the methods get executed internally which is bound with the delegates. For example:

```
ad(100, 50);
ad.Invoke(200, 300);
string GreetingsMessage = gd(“Priyanka”);
string GreetingsMessage = gd.Invoke(“Anurag”);
```

At this point, the function that is referred to by the delegate will be called for execution.

Rules of using Delegates in C#:

1.A delegate in C# is a user-defined type and hence before invoking a method using a delegate, we must have to define that delegate first.

2.The **signature** of the delegate **must match** the signature of the **method**, the delegate points to otherwise we will get a compiler error. This is the reason why delegates are called type-safe function pointers.

3.A Delegate is similar to a class. This means we can create an instance of a delegate and when we do so, we need to pass the method name as a parameter to the delegate constructor, and it is the function the delegate will point to

4.Tip to remember delegate syntax: Delegates syntax looks very much similar to a method with a delegate keyword.

What are the Types of Delegates in C#?

Single Cast Delegate

Multicast Delegate

If a delegate is used for invoking a single method then it is called a single cast delegate or unicast delegate. In other words, we can say that the delegates that represent only a single function are known as single cast delegates.

If a delegate is used for invoking multiple methods then it is known as the multicast delegate. Or the delegates that represent more than one function are called Multicast delegates.

What is Multicast Delegate in C#?

A Multicast Delegate in C# is a delegate that holds the references of more than one function. When we invoke the multicast delegate, then all the functions which are referenced by the delegate are going to be invoked. If you want to call multiple methods using a delegate then all the method signatures should be the same.

```
public class Rectangle
{
public void GetArea(double Width, double Height)
{
Console.WriteLine(@"Area is {0}", (Width * Height));
}
public void GetPerimeter(double Width, double Height)
{
Console.WriteLine(@"Perimeter is {0}", (2 * (Width + Height)));
}
static void Main(string[] args)
{
Rectangle rect = new Rectangle();
rect.GetArea(23.45, 67.89);
rect.GetPerimeter(23.45, 67.89);
Console.ReadKey();
}
}
```

```
Rectangle rect = new Rectangle();
RectangleDelete rectDelegate = new RectangleDelete(rect.GetArea);
rectDelegate += rect.GetPerimeter;
rectDelegate(23.45, 67.89);
```

Delegates Real-time Example in C#

```
namespace DelegateRealtimeExample
{
public delegate bool EligibleToPromotion(Employee EmployeeToPromotion);
public class Employee
{
public int ID { get; set; }
public string Name { get; set; }
public string Gender { get; set; }
public int Experience { get; set; }
public int Salary { get; set; }
public static void PromoteEmployee(List<Employee> lstEmployees, EligibleToPromotion IsEmployeeEligible)
{
foreach (Employee employee in lstEmployees)
{
if (IsEmployeeEligible(employee))
{
Console.WriteLine("Employee {0} Promoted", employee.Name);
}
}
}
}
}
```

Anonymous Method in C# with examples

What is Anonymous Method in C#?

As the name suggests, an anonymous method in C# is a method without having a name. The Anonymous methods in C# can be defined using the keyword delegate and can be assigned to a variable of the delegate type.

```
public class AnonymousMethods
{
    public delegate string GreetingsDelegate(string name);
        static void Main(string[] args)
        {
            GreetingsDelegate gd = delegate(string name)
            {
            return "Hello @" + name + " Welcome to Dotnet Tutorials";
            };
            string GreetingsMessage = gd.Invoke("Pranaya");
            Console.WriteLine(GreetingsMessage);
            Console.ReadKey();
        }
}
```

- **What are the Limitations of Anonymous Methods in C#?**
- An anonymous method in C# cannot contain any jump statement like goto, break or continue.
- It cannot access the ref or out parameter of an outer method.
- The Anonymous methods cannot have or access the unsafe code.

- **Points to Remember while working with the Anonymous Methods in C#:**
- The anonymous methods can be defined using the delegate keyword
- An anonymous method must be assigned to a delegate type.
- This method can access outer variables or functions.
- An anonymous method can be passed as a parameter.
- This method can be used as event handler.

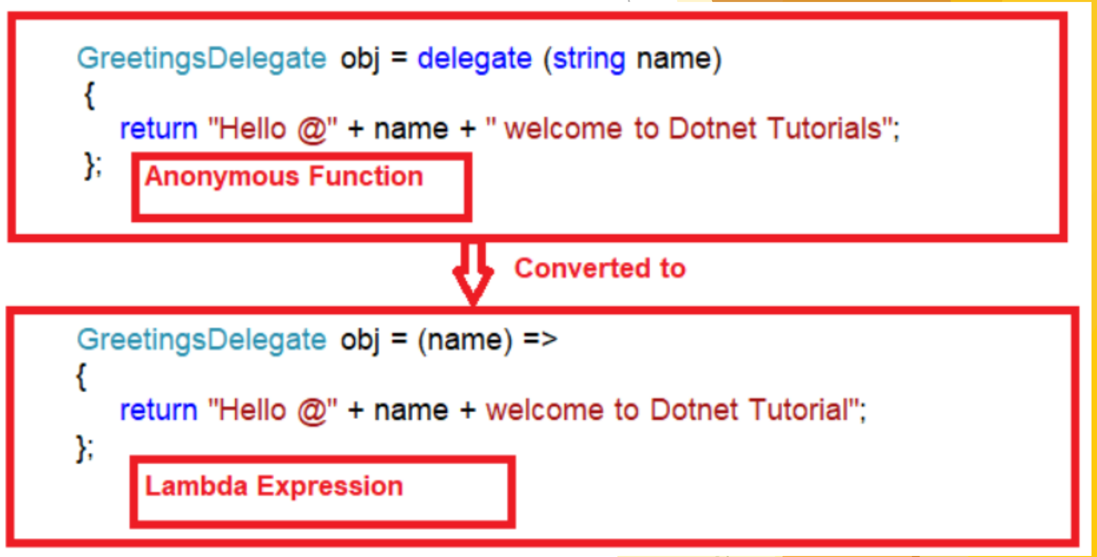
What are Lambda Expressions in C#?

The **Lambda Expression in C#** is the shorthand for writing the anonymous function. So we can say that the Lambda Expression in C# is nothing but to simplify the anonymous function in C#.

How to create Lambda Expression in C#?

To create a lambda expression in C#, we need to specify the input parameters (if any) on the left side of the lambda operator **=>**, and we need to put the expression or statement block on the other side.

```
public delegate string GreetingsDelegate(string name);
static void Main(string[] args)
{
GreetingsDelegate obj = (name) =>
{
return "Hello @" + name + " welcome to Dotnet Tutorials";
};
string GreetingsMessage = obj.Invoke("Pranaya");
Console.WriteLine(GreetingsMessage);
Console.ReadKey();
}
```



```
GreetingsDelegate obj = delegate (string name)
{
    return "Hello @" + name + " welcome to Dotnet Tutorials";
};
```

```
GreetingsDelegate obj = (name) =>
{
    return "Hello @" + name + welcome to Dotnet Tutorial";
};
```

What are Generic Delegates in C#?

The Generic Delegates in C# were introduced as part of **.NET Framework 3.5** which doesn't require defining the delegate instance in order to invoke the methods. To understand the Generic Delegates in C# you should have the basic knowledge of **Delegates**.

Func

Action

Predicate

Points to remember while working with C# Generic Delegates:

1.Func, Action, and Predicate are generic inbuilt delegates that are present in the System namespace which is introduced in C# 3.

2.All these three delegates can be used with the method, **anonymous method**, and l**ambda expressions**.

3.The Func delegates can contain a maximum of 16 input parameters and must have one return type.

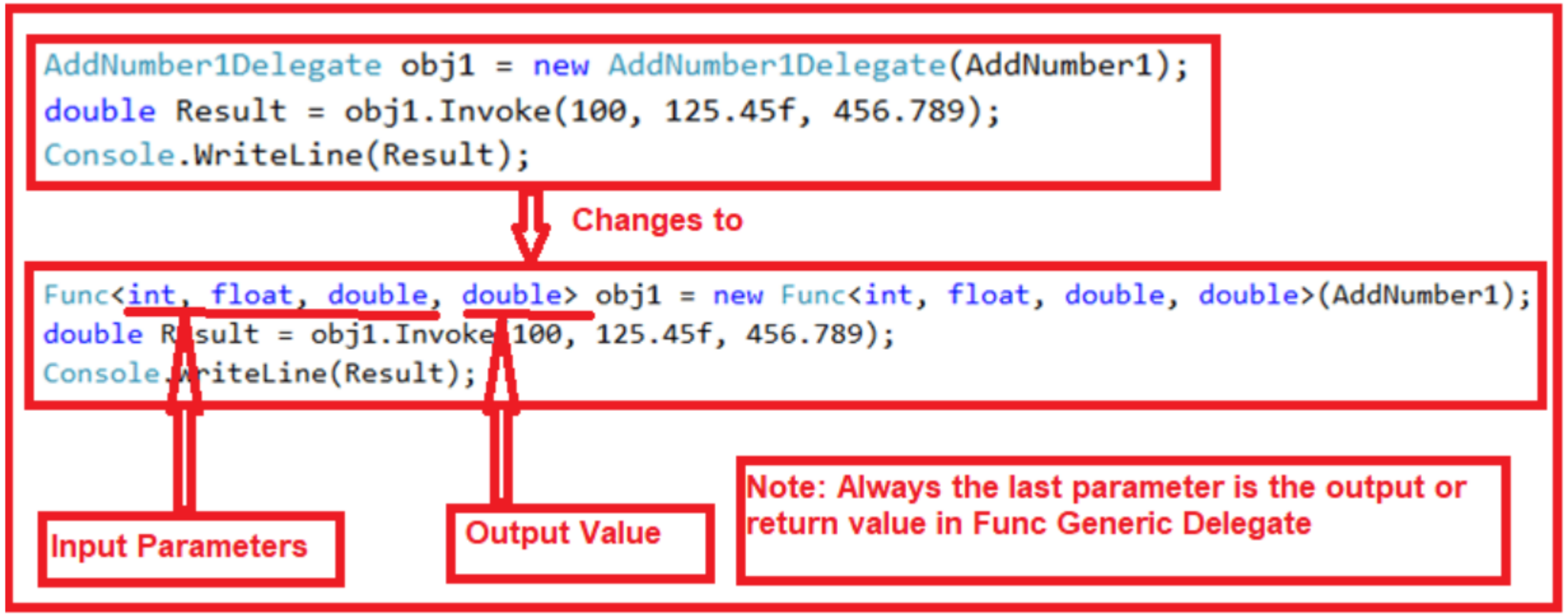
4. Action delegate can contain a maximum of 16 input parameters and does not have any return type.

5.The Predicate delegate should satisfy some criteria of a method and must have one input parameter and one Boolean return type either true or false which is by default. This means we should not have to pass that to the Predicate

What is Func Generic Delegate in C#?

The **Func Generic Delegate in C#** is present in the **System** namespace. This delegate takes one or more input parameters and returns one out parameter. The last parameter is considered as the return value. The Func Generic Delegate in C# can take up to 16 input parameters of different types. It must have one return type. The return type is mandatory but the input parameter is not.

Note: Whenever your delegate returns some value, whether by taking any input parameter or not, you need to use the Func Generic delegate in C#.



```
AddNumber1Delegate obj1 = new AddNumber1Delegate(AddNumber1);
double Result = obj1.Invoke(100, 125.45f, 456.789);
Console.WriteLine(Result);
```

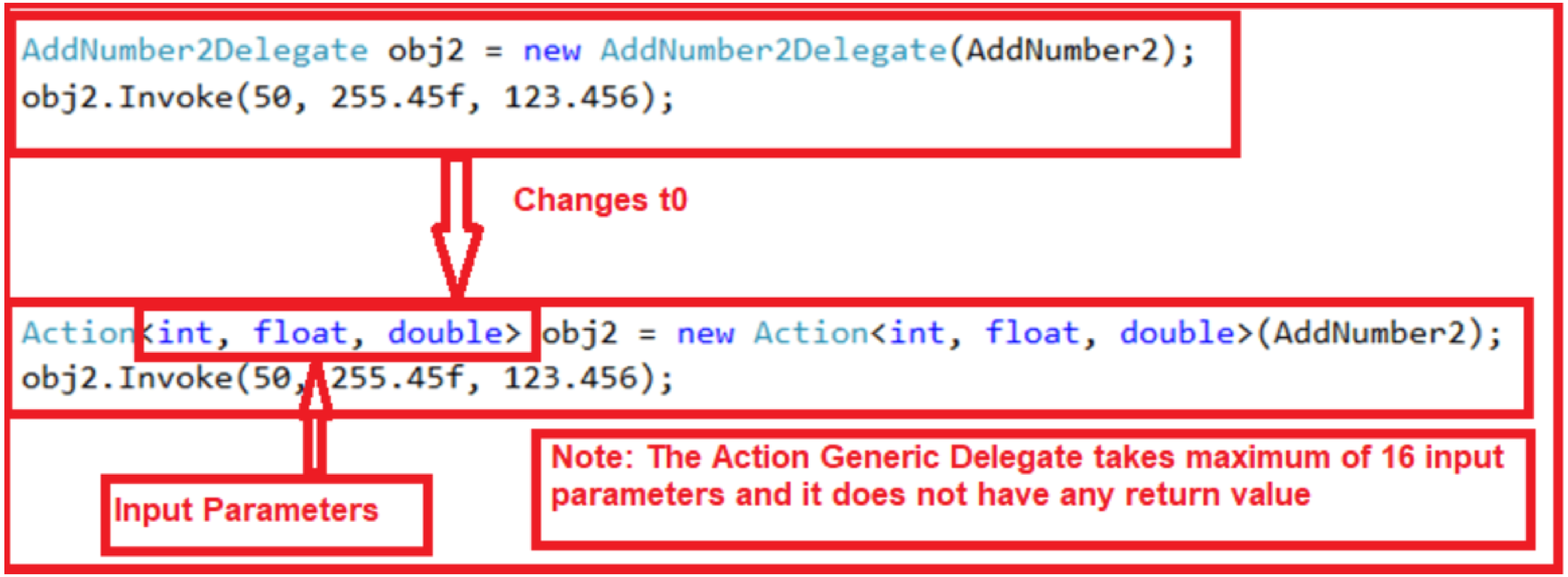
```
Func<int, float, double, double> obj1 = new Func<int, float, double, double>(AddNumber1);
double Result = obj1.Invoke(100, 125.45f, 456.789);
Console.WriteLine(Result);
```

Note: Always the last parameter is the output or return value in Func Generic Delegate

What is Action Generic Delegate in C#?

The **Action Generic Delegate in C#** is also present in the **System** namespace. It takes one or more input parameters and returns nothing. This delegate can take a maximum of **16 input parameters** of the different or same type

Note: Whenever your delegate does not return any value, whether by taking any input parameter or not, then you need to use the Action Generic delegate in C#.



```
AddNumber2Delegate obj2 = new AddNumber2Delegate(AddNumber2);
obj2.Invoke(50, 255.45f, 123.456);
```

Changes t0

```
Action<int, float, double> obj2 = new Action<int, float, double>(AddNumber2);
obj2.Invoke(50, 255.45f, 123.456);
```

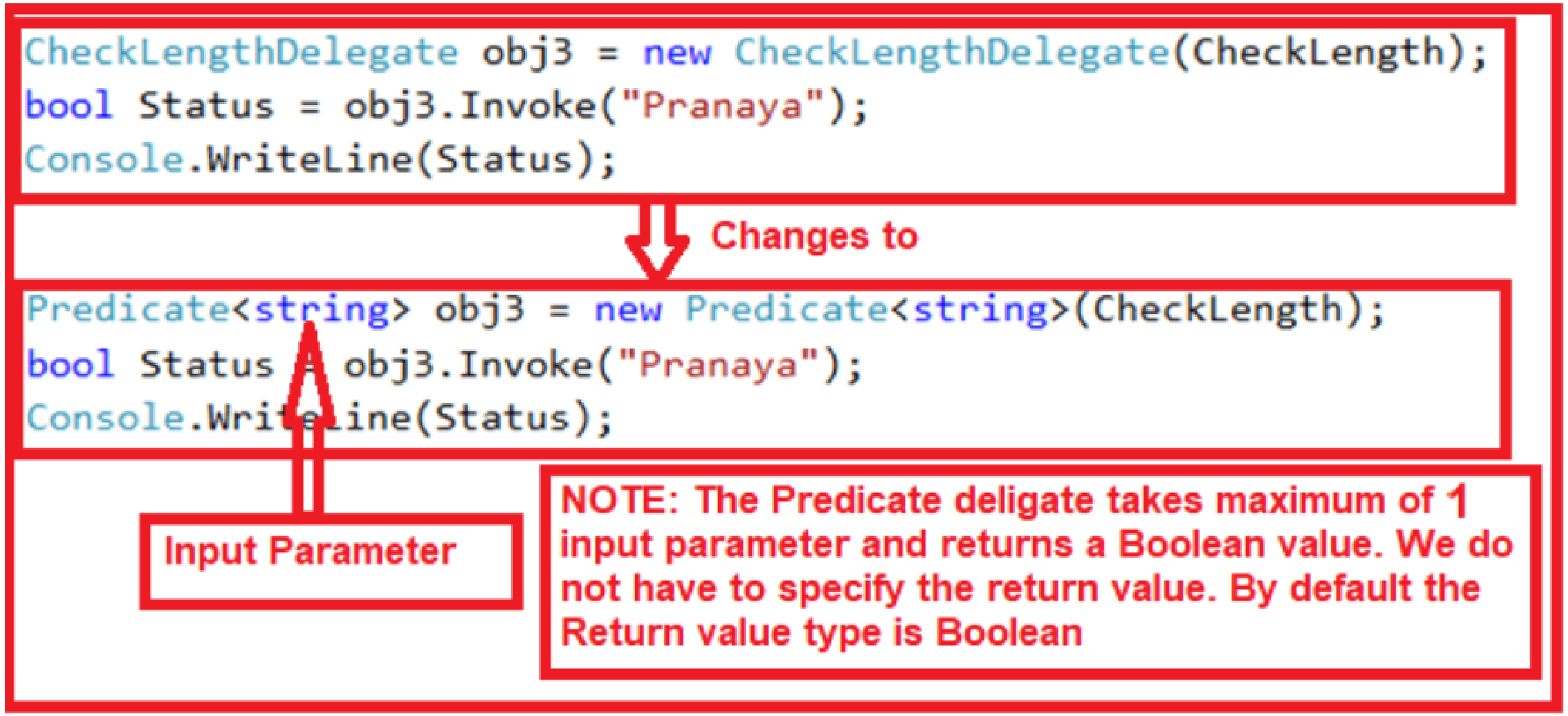
Input Parameters

Note: The Action Generic Delegate takes maximum of 16 input parameters and it does not have any return value

What is Predicate Generic Delegate in C#?

The **Predicate Generic Delegate in C#** is also present in the **System** namespace. This delegate is used to verify certain criteria of the method and returns the output as Boolean, either True or False. It takes one input parameter and always returns a Boolean value which is mandatory. This delegate can take a maximum of **1 input parameter** and always return the value of the Boolean type.

Note: Whenever your delegate returns a Boolean value, by taking one input parameter, then you need to use the Predicate Generic delegate in C#.



```
CheckLengthDelegate obj3 = new CheckLengthDelegate(CheckLength);
bool Status = obj3.Invoke("Pranaya");
Console.WriteLine(Status);
```

Changes to

```
Predicate<string> obj3 = new Predicate<string>(CheckLength);
bool Status = obj3.Invoke("Pranaya");
Console.WriteLine(Status);
```

Input Parameter

NOTE: The Predicate deligate takes maximum of 1 input parameter and returns a Boolean value. We do not have to specify the return value. By default the Return value type is Boolean