Overview

In this article we are going to explain the basics of Action and Func delegates in C#. Action and Func are built-in generic delegate type in C#, and they are in the System namespace.
Since Action and Func are delegates, we will start with a brief explanation for the basics of delegates in C#.

We are going to divide this article into the following sections:

- Basics of delegates in C#.
- Func delegate.
- Action delegate
- Summary.

Basics of delegates in C#:

Delegate type is a function reference, used where we need to pass a function as a parameter,
or handle callback functions or event handler.
-We can use delegate in the following sequence:

1-Declare delegate:

[access modifier] delegate [return type] [delegate name] ([parameters list])
example:

In this example we declare a delegate, which name is MyDel, and returns int and takes two input parameters (a, b) of int datatype.

```
public delegate int MyDel(int a, int b);
```

2-Set target method:

The target method must has a signature matches the delegate signature.

```
public int Sum(int x, int y)
```

```
{ return (x +y); }
```

here we declared the function Sum which takes two parameters of type int (x, y), and returns int.

3-Invoke delegate:

```
MyDel = Sum; // assign the method Sum to the delegate MyDel.
MyDel(5, 10);
// or:
MyDel.Invoke(5, 10); // after assigning the method Sum to MyDel we invoked MyDel using Invoke.
// or using new keyword:
MyDel dl = new MyDel(sum);
// or using lambda expression:
MyDel dl = (5, 10) => 5+ 10;
dl(5, 10);
```

Note that the Sum method signature is the same as MyDel signature.

Func delegate:

Func delegate can has zero input parameter or more input parameters up to 16 input parameter and must has one out parameter , the last parameter is the out parameter.

We can use Func delegate as follows:

1-Declare Func delegate:

```
Func<T1, T2, ..., Tn> [Func name] = Tn [method name] (T1, T2, ..., Tn-1);
```

n number of parameters,

16 >= n > 0,

T1, T2, ..., Tn-1 are input parameters datatype, and Tn return type.

2-Set target method:

Example:

In this example we declare the function area which takes two input parameters with int datatype and returns double datatype, we also declare Func delegate which name is calcArea, you can note that the function area has the same signature as the Func delegate calcArea (input parameters and return parameter are the same)

Then we assign the function area to the Func delegate calcArea.

```
class Program{

public double area (int x, int y) { return x * y;}

static void Main(string[] args) {

// Declare Func that takes two input parameters and returns double.

Func<int, int, double> calcArea = area;

double res = calcArea(5, 10);

Console.WriteLine(res); // displays 50.

  }

}
```

Using Func with lambda expression:

```
Func<int, int, double> area = (a, b) => a* b;
```

```
Console.WriteLine(area(5, 10));
```

Using Func with an anonymous method (anonymous method does not have a name):

That can be done using the keyword delegate:

```
Func<string, string> display = delegate(string name){ return "(Hello " + name);};
```

```
Console.WriteLine(display("World"));
```

Action Delegate:

Action delegate is the same as Func delegate except that Action delegate does not return a value.

Example:

```
class Program {
static void Display(string msg){ Console.WriteLine(msg)); }
static void Main(string[] args)
{
Action<string> print = Display;
//or using the new keyword.
Action<string> print = new Action<string>(Display);
// using Action delegate with an anonymous method.
Action<string> print = delegate (string msg){ Console.WriteLine(msg); };
//With lambda expression.
Action<string> print = (msg) => Console.WriteLine(msg);
print("Hello everyone");
 }
}
```

Summary:

-Func delegate and Action delegate are built-in generic delegate types in the System namespace.

-They can have 0 to 16 input parameter.

-We can use Func and Action delegates with both anonymous methods and lambda expression.

-Func delegate must return a value.

-Func delegate does not allow ref or out parameters.

-Action delegate does not return any value.