**C# - Action Delegate**

**Action is a delegate type defined in the System namespace. An Action type delegate is the same as**[**Func delegate**](https://www.tutorialsteacher.com/csharp/csharp-func-delegate)**except that the Action delegate doesn't return a value. In other words, an Action delegate can be used with a method that has a void return type.**

**For example, the following delegate prints an int value.**

**Example: C# Delegate**

**public delegate void Print(int val);**

**static void ConsolePrint(int i)**

**{**

**Console.WriteLine(i);**

**}**

**static void Main(string[] args)**

**{**

**Print prnt = ConsolePrint;**

**prnt(10);**

**}**

**Output:**

**10**

**You can use an Action delegate instead of defining the above Print delegate, for example:**

**Example: Action delegate**

**static void ConsolePrint(int i)**

**{**

**Console.WriteLine(i);**

**}**

**static void Main(string[] args)**

**{**

**Action<int> printActionDel = ConsolePrint;**

**printActionDel(10);**

**}**

[**Try it**](https://www.tutorialsteacher.com/codeeditor?cid=cs-Th00fz)

**You can initialize an Action delegate using the new keyword or by directly assigning a method:**

**Action<int> printActionDel = ConsolePrint;**

**//Or**

**Action<int> printActionDel = new Action<int>(ConsolePrint);**

**An Action delegate can take up to 16 input parameters of different types.**

**An Anonymous method can also be assigned to an Action delegate, for example:**

**Example: Anonymous method with Action delegate**

**static void Main(string[] args)**

**{**

**Action<int> printActionDel = delegate(int i)**

**{**

**Console.WriteLine(i);**

**};**

**printActionDel(10);**

**}**

[**Try it**](https://www.tutorialsteacher.com/codeeditor?cid=cs-hELcoU)

**Output:**

**10**

**A Lambda expression also can be used with an Action delegate:**

**Example: Lambda expression with Action delegate**

**static void Main(string[] args)**

**{**

**Action<int> printActionDel = i => Console.WriteLine(i);**

**printActionDel(10);**

**}**

[**Try it**](https://www.tutorialsteacher.com/codeeditor?cid=cs-xoInGo)

**Thus, you can use any method that doesn't return a value with Action delegate types.**

**Advantages of Action and Func Delegates**

1. **Easy and quick to define delegates.**
2. **Makes code short.**
3. **Compatible type throughout the application.**

** Points to Remember :**

1. **Action delegate is same as func delegate except that it does not return anything. Return type must be void.**
2. **Action delegate can have 0 to 16 input parameters.**
3. **Action delegate can be used with**[**anonymous methods**](https://www.tutorialsteacher.com/csharp/csharp-anonymous-method)**or**[**lambda expressions**](https://www.tutorialsteacher.com/linq/linq-lambda-expression)**.**

**C# - Func Delegate**

**C# includes built-in generic delegate types Func and Action, so that you don't need to define custom delegates manually in most cases.**

**Func is a generic delegate included in the System namespace. It has zero or more *input* parameters and one *out* parameter. The last parameter is considered as an out parameter.**

**The Func delegate that takes one input parameter and one out parameter is defined in the System namespace, as shown below:**

**Signature: Func**

**namespace System**

**{**

**public delegate TResult Func<in T, out TResult>(T arg);**

**}**

**The last parameter in the angle brackets <> is considered the return type, and the remaining parameters are considered input parameter types, as shown in the following figure.**

**[A diagram of a type of return

Description automatically generated](https://www.tutorialsteacher.com/Content/images/csharp/func-delegate.png)Func delegate**

**A Func delegate with two input parameters and one out parameters will be represented as shown below.**

**[A triangle with text and a line

Description automatically generated with medium confidence](https://www.tutorialsteacher.com/Content/images/csharp/func-delegate2.png)Func delegate**

**The following Func delegate takes two input parameters of int type and returns a value of int type:**

**Func<int, int, int> sum;**

**You can assign any method to the above func delegate that takes two *int* parameters and returns an *int* value.**

**Example: Func**

**class Program**

**{**

**static int Sum(int x, int y)**

**{**

**return x + y;**

**}**

**static void Main(string[] args)**

**{**

**Func<int,int, int> add = Sum;**

**int result = add(10, 10);**

**Console.WriteLine(result);**

**}**

**}**

[**Try it**](https://www.tutorialsteacher.com/codeeditor?cid=cs-2yWptk)

**Output:**

**20**

**A Func delegate type can include 0 to 16 input parameters of different types. However, it must include an out parameter for the result. For example, the following Func delegate doesn't have any input parameter, and it includes only an out parameter.**

**Example: Func with Zero Input Parameter**

**Func<int> getRandomNumber;**

**C# Func with an Anonymous Method**

**You can assign an anonymous method to the Func delegate by using the delegate keyword.**

**Example: Func with Anonymous Method**

**Func<int> getRandomNumber = delegate()**

**{**

**Random rnd = new Random();**

**return rnd.Next(1, 100);**

**};**

**Func with Lambda Expression**

**A Func delegate can also be used with a lambda expression, as shown below:**

**Example: Func with lambda expression**

**Func<int> getRandomNumber = () => new Random().Next(1, 100);**

**//Or**

**Func<int, int, int> Sum = (x, y) => x + y;**

[**Try it**](https://www.tutorialsteacher.com/codeeditor?cid=cs-CPmcNQ)

** Points to Remember :**

1. **Func is built-in delegate type.**
2. **Func delegate type must return a value.**
3. **Func delegate type can have zero to 16 input parameters.**
4. **Func delegate does not allow ref and out parameters.**
5. **Func delegate type can be used with an**[**anonymous method**](https://www.tutorialsteacher.com/csharp/csharp-anonymous-method)**or**[**lambda expression**](https://www.tutorialsteacher.com/linq/linq-lambda-expression)**.**

**C# - Predicate Delegate**

Predicate is the delegate like [Func](https://www.tutorialsteacher.com/csharp/csharp-func-delegate) and [Action](https://www.tutorialsteacher.com/csharp/csharp-action-delegate) delegates. It represents a method containing a set of criteria and checks whether the passed parameter meets those criteria. A predicate delegate methods must take one input parameter and return a boolean - true or false.

The Predicate delegate is defined in the System namespace, as shown below:

Predicate signature: public delegate bool Predicate<in T>(T obj);

Same as other delegate types, Predicate can also be used with any method, anonymous method, or lambda expression.

Example: Predicate delegate

static bool IsUpperCase(string str)

{

return str.Equals(str.ToUpper());

}

static void Main(string[] args)

{

Predicate<string> isUpper = IsUpperCase;

bool result = isUpper("hello world!!");

Console.WriteLine(result);

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-6jIeuz)

Output:

false

An anonymous method can also be assigned to a Predicate delegate type as shown below.

Example: Predicate delegate with anonymous method

static void Main(string[] args)

{

Predicate<string> isUpper = delegate(string s) { return s.Equals(s.ToUpper());};

bool result = isUpper("hello world!!");

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-GjU9b5)

A lambda expression can also be assigned to a Predicate delegate type as shown below.

Example: Predicate delegate with lambda expression

static void Main(string[] args)

{

Predicate<string> isUpper = s => s.Equals(s.ToUpper());

bool result = isUpper("hello world!!");

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-e0oBtk)

 Points to Remember:

1. Predicate delegate takes one input parameter and boolean return type.
2. [Anonymous method](https://www.tutorialsteacher.com/csharp/csharp-anonymous-method) and [Lambda expression](https://www.tutorialsteacher.com/linq/linq-lambda-expression) can be assigned to the predicate delegate.