Ref vs Out

Ref- It passes arguments by reference, i.e. the changes made to the arg in the method will return the same change to the method it gets called.

Out- out also passes the arguments as reference, it is same as ref.

Difference between ref and out

|  |  |
| --- | --- |
| Ref | out |
| We must initialize the parameter before sending it to ref | It is not necessary to initialize. |
| It is not necessary to initialize the value of a parameter before returning method call | It is necessary |
| Useful when called method should also modify the pass parameter | Useful when multiple values need to be returned from a method |
| Not req to initialize parameter before using in function call | Must be initialized within the method called |
| Bi directional | Uni directional |

Partial Keyword

Partial Class- we can split a class using partial keyword. Giving the same name to a class but defining it in parts using the partial key word.

Eg public partial class a{}

Public partial Class a{}

It can be used for classes, struct or interface.

All parts should be defined using the keyword partial

All should have the same level of accessibility

If any part is abstract the whole part is considered abstract

If one part any part is base , the whole part inherits that class

At compile type the parts get merged as one

All the partial def should be under the same assembly and same module

Partial Method- partial class/struct can have partial methods, where a part has the signature of the method and the implementation of that method is in the same or other part.

If implementation is not provided then the method and method calls are removed at compilation time

We need not have an implementation of partial methods if-

If there is no accessibility modifiers (even private)

Returns void

Doesnot have out parameters

Doesn’t have virtual, override, seal, new and extern modifiers

While creating partial method the signature in all parts should match

We can make delegate to a partial method

Named Parameters-

We pass argument name while function calling. By using named parameters it helps us to specify the to which parameter we are passing the value.

Eg

Public static void helloworld(string hello, string world){

// some method

}

While calling the method hello world we can

Call by order- helloworld(“hello”, “world”)

By the name of the argument- helloworld(hello:”hello”, world:”world”)

Optional Arguments

We need not pass an argument in optional arguments. We have default value given in the functional definition thus while functional calling its not necessary.

Eg

Public static doadd(int b, int a=1)

Method calling – doadd(2,3) // a gets assigned to 3

doadd(2) // b gets 2 and a is 1

so b is a required parameter and a is an optional parameter.

We must always define required parameter first and then followed by optional parameter.

Async and await

Using asynchronous programming indicates that a method can execute without waiting for another method to complete. Using async and await, we can run the methods above parallelly.

The async keyword to mark a method as asynchronous and with await, we can wait for an asynchronous operation in such a way that the original thread is not blocked.

Parms

We use this keyword when we want to give variable number of arguments. If we give parms we cannot give any additional parms. If no arguments have been passed it will be taken as default 0.

Yield

We use it for iteration to the next item or to show the end of an iteration. Yield is used in two ways

Yield return- to get to the next value

Yield break- to explicitly signal the end of iteration

Iterator

It is used to travers through the elements present in the collection. Using this we can perform action on each item. We can enumerate a custom collection

We can iterate using foreach

We use Enumerablecollection and yield return to iterate