**Ref and Out**

|  |  |
| --- | --- |
| Ref | Out |
| Should be initialised first and then used as a parameter | Not necessary to initialise before using it as a parameter |
| Not necessary to initialise the value of parameter before returning to calling method | Must initialise the value of parameter before returning to calling method. |
| useful when the called method also need to change the value of passed parameter. | declaring of parameter through out parameter is useful when a method return multiple values. |

**Partial methods**

that allows you to declare a method in one part of a partial class and define its implementation in another part.

Partial methods are implicitly private, and if there is no implementation provided, the compiler removes the method and all calls to it at compile time.

**Async and Await**

Allows you to write asynchronous code more easily and efficiently.

Asynchronous programming is used in tasks which take much time to complete like the I/0 operations, network requests, etc.

The async keyword is used to declare a method as asynchronous.

It indicates that the method contains an asynchronous operation, and it can be paused and resumed.

public async Task MyAsyncMethod()

{

// Asynchronous code here

}

An asynchronous method typically returns a Task or Task<T>.

Task : ongoing operation

T : represents an operation that produces a result of type <T>

The await keyword is applied to an asynchronous operation (typically a method returning Task or Task<T>).

It allows the method to await the completion of the operation without blocking the execution of the entire application.

**Params**

Allow a method to accept a variable number of arguments to the method.

void Sum(params int[] numbers)

{

int total = 0;

foreach (int num in numbers)

{

total += num;

}

Console.WriteLine(total);

}

// Usage

Sum(1, 2, 3); // Output: 6

Sum(1, 2, 3, 4, 5); // Output: 15

**Named Parameters**

allow you to specify the name of the parameters you are passing to a method or constructor. This can improve code readability, make it explicit which argument corresponds to which parameter.

Allow you to pass the parameters in a different order than they are defined in the method signature.

**Optional Parameters**

Have default values specified in the method.

If the user does not specify any value, then the default values will be considered.

void PrintNumber(int x = 0)

{

Console.WriteLine(x);

}

// Usage

PrintNumber(); // Output: 0

PrintNumber(5); // Output: 5

**Iterators**

iterators are implemented using the yield keyword. They allow you to create a sequence of values without having to generate the entire sequence at once.

are commonly used with foreach loops to iterate over a sequence of values lazily.

Iterators are useful when dealing with large datasets or when you want to generate values on-the-fly without consuming excessive memory.