Cheat Sheet (Emmanuel D.)

C# 9.0

Top Level Program

```
using System;

Console.WriteLine("Hello World!");
```

Pattern matching enhancements

```
if (context is {IsReachable: true, Length: > 1 })
{
   Console.WriteLine(context.Name);
}
```

C# 9.0 introduces patterns corresponding to the relational operators <, <= and so on,

```
DeliveryTruck t when t.GrossWeightClass switch
{
   < 3000 => 8.00m,
   >= 3000 and <= 5000 => 10.00m,
   > 5000 => 15.00m,
}
```

Target-typed new expression

```
List<string> values = new();
```

Init-only properties

init accessor (variant of the set accessor) can only be called during object initialization.

```
struct Point
{
   public int X { get; init; }
   public int Y { get; init; }
}


var p = new Point() { X = 42, Y = 13 };
```

Records

Records define the whole object as immutable and behave more like a value than an object.

```
public data class Person
{
   public string FirstName { get; init; }
   public string LastName { get; init; }
}
```

With-expressions use object initializer syntax to state what's different in the new object from the old object. You can specify multiple properties.

```
var otherPerson = person with { LastName = "Hanselman" };
```

C# 8.0

Nullable reference types

```
string? name;

name!.Length; // null-forgiving operator
```

Null-coalescing assignment

```
List<int> numbers = null;
numbers ??= new List<int>();
```

Using declarations

Using declaration is now simplified and braces can be omitted

Asynchronous streams

Streams can be created and consumed asynchronously.

```
await foreach (var number in GenerateSequence())
{
   Console.WriteLine(number);
}
```

Indices and ranges

Indices and ranges provide a succinct syntax for accessing single elements or ranges in a sequence.

```
var words = new string[] {The", "quick", "brown, "fox"}

Console.WriteLine($"The last word is {words[^1]}");
var quickBrownFox = words[1..4];
var allWords = words[..];

Range phrase = 1..4;
var text = words[phrase];
```

Pattern matching enhancements

```
switch {
    Rainbow.Red => new RGBColor(0xFF, 0x00, 0x00),
    _ => => new RGBColor(0xFF, 0x00, 0x00),
};

location switch {
  { State: "WA" } => salePrice * 0.06M
  _ => 0M
};

(first, second)  switch {
  ("rock", "paper") => "Paper wins.",
  (_, _) => "tie"
};

point switch {
 (0, 0) => Quadrant.Origin,
 var (x, y) when x > 0 && y > 0 => Quadrant.One,
  _ => Quadrant.Unknown
};
```

C# 8.0

Default interface methods

```
interface IA {
    void M() { WriteLine("IA.M"); }
}
```

Readonly members

```
public readonly double Distance =>
                      Math.Sqrt(X * X + Y * Y);
```

Static local functions

```
int M()
{
  int y = 5; int x = 7;
  return Add(x, y);

  static int Add(int left, int right) => left + right;
}
```

C# 7.3

Language improvements

- Indexing fixed fields does not require pinning
- ref local variables may be reassigned
- stackalloc arrays support initializers
- More types support the fixed statement
- Enum or Delegate generic constraints
- Tuples support == and !=

C# 7.0

Tuples

```
(string Alpha, string Beta) letters = ("a", "b");
Console.WriteLine($"{letters.Alpha}, {letters.Beta}");

var alphabet = (Alpha: "a", Beta: "b");
Console.WriteLine ($"{alphabet.Alpha}, {alphabet.Beta}");

(int max, int min) = Range(numbers);
```

Pattern matching

```
switch (i) {
 case 0: break;
 case IEnumerable<int> childSequence:
   { foreach(var item in childSequence) sum += item : 0; break; }
 case int n when n > 0 : sum += n; break;
 case null: throw new NullReferenceException();
 default: throw new InvalidOperationException();
}
```

C# 7.0

Discard

```
var (_, _, _, pop1, _, pop2)
     = QueryCityDataForYears("NYC", 1960, 2010);
```

Generalized async return types

```
public async ValueTask<int> Func() {
  await Task.Delay(100);
  return 5;
}
```

Local Functions

```
public static IEnumerable<char>
                     AlphabetSubset3(char start, char end) {

  return alphabetSubsetImplementation();

 IEnumerable<char> alphabetSubsetImplementation() {
   for (var c = start; c < end; c++) yield return c;
  }
}
```

Throw expressions

throw are now supported on :
- Conditional operator (x ? y : throw ...)
- Null coalescing operator (x = y ?? throw ...)
- Expression-bodied lambda (int foo(x) => throw ..)

Ref locals and returns

```
public static ref int Find(int[,] matrix, Func<int, bool> predicate) {
   for (int i = 0; i < matrix.GetLength(0); i++)
     for (int j = 0; j < matrix.GetLength(1); j++)
       if (predicate(matrix[i, j]))
             return ref matrix[i, j];
   throw new InvalidOperationException("Not found");
}
ref var item = ref MatrixSearch.Find(matrix, (val) => val == 42);
```

C# default version by target framework

.NET Core	3.X	C# 8.0
.NET Core	2.x	C# 7.3
.NET Standard	2.1	C# 8.0
.NET Standard	2.0	C# 7.3
.NET Standard	1.0	C# 7.3
.NET Framework	all	C# 7.3

https://github.com/Manu06D/cheatsheet