C# 7 Cheat Sheet

05.0.2017

Ref locals and returns

Return references to a defined variable

```
var array = new[] { 1, 2, 3, 4, 5};
ref int GetFirstItem(int[] arrayParam, int index) => ref arrayParam[index];
// Retrieve the reference to the first item from the array (1)
ref int firstItem = ref GetFirstItem(array, 0);
firstItem = -100;
// Now the array =  {-100, 2, 3, 4, 5};
```

Expression-bodied

Expression-bodied expanded

+ Constructors
+ Finalizers
+ Indexers

```
public ExpressionMembers(string text) => this.Text = text;
~ExpressionMembers() => Console.Error.WriteLine("Finalized!");
public string Text
{
   get => text;
   set => this.text = value ?? "Default Text";
}
```

Generalized async return

Define custom return types on async methods

+ ValueTask
+ Designed for the very scenario

```
// In this scenario the return is more efficient
public static async ValueTask<int> ValueTask(int[] numbers)
{
  if (!numbers.Any())
  {
    return 0;
  }
  else
  {
    return await Task.Run(() => numbers.Sum());
  }
}
```

Bassam Alugili
https://github.com/alugili/CSharp7Features

Binary Literals and Digit Separators

New langauge Syntax to improve readability
for numeric constants

```
int Sixteen =   0b0001_0000;
b = Binrary Literal
_ = Digit Separator
```

Local Functions

Nesting functions inside other functions to limit their scope and visibility

+ Better performance

```
public static void BasicExample()
{
   // Defining the func
   void EmptyLocalFunction()
   {
    Console.WriteLine("I'm Local");
   };
  // Calling the func
  EmptyLocalFunction();
}
```

Throw expressions

Throw exceptions in

+ Null coalescing expressions
+ Some lambda expressions
+ Expression-bodied

```
public static double Divide(int numerator , int denominator)
{
  return y != 0 ? numerator % denominator
                  : throw new DivideByZeroException();
}
// Excpetion Here!
var result = Divide(10, 0);
```

Out variables

Declare out variablie inline

```
public static void OutVarParse(string @int)
{
   int.TryParse(@int, out int tmp);
  Console.WriteLine(tmp);
}
```

Pattern Matching

Similar to a switch statement that works on the shape of the data

+ Constant patterns
+ Type patterns
+ Var patterns

```
var myText = "Type matched!";
var constatPattern = myText is null;
var typePattern = myText is string x ? x : "not a string";

object varPattern = 42;
switch (varPattern )
{
  case int i when i > 40:
  Console.WriteLine($"{varPattern  is an {(i % 2 == 0 ? "even" : "odd")} int");
  break;
  ....
 . ...
}
```

Tuples

A tuple is a data structure that has a specific number and sequence of Elements

+ ValueTuple
+ Immutable
+ Readability

```
(int, int) SwapValues((int x, int y) value)
{
 return (value.y, value.x);
}

(int xSwapped, int ySwapped) values = SwapValues((5, 6));
```

Destruction

Destruct an object to a tuple

```
public class CustomDateTime
{
  public int Hour { get; }
  public int Minute { get; }
  public int Second { get; }
  public CustomDateTime(int hour, int minute, int second)
 { this.Hour = hour; this.Minute = minute; this.Second = second; }
  public void Deconstruct(out int hour, out int minute, out int second)
 { hour = this.Hour; minute = this.Minute; second = this.Second; }
}

(int hour, int minute, int second) = new CustomDateTime(10, 5, 3);
```