C# Cheat Sheet

Learn How to Code Using C#: The Basics of Programming

udemy.com/learn-how-to-code-using-c-sharp-the-basics-of-programming

Data Types

Data Type	Size (bytes)	Stores
short	2	Whole numbers (≈ -32,000 to 32,000)
int	4	Whole numbers (≈ -2 billion to 2 billion)
long	8	Whole numbers (≈ -9 quintillion to 9 quintillion)
bool	1	True or false
char	2	Single characters
string	2 per char	Text
double	8	Decimal numbers

Variables

```
int x;
int y = 5;
x = y + 1;
```

Output

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

Order of Operations

HIGHER PRECEDENCE
++, -- (prefix)
*, /, %
+, -
=, +=, -=, *=, /=
++, -- (postfix)
LOWER PRECEDENCE

HIGHER PRECEDENCE
!
< > <= >=
== !=
&&
||
LOWER PRECEDENCE

Converting

```
myDouble = myInteger; // implicit conversion
myInteger = (int)myDouble; // explicit conversion
myInteger = Convert.ToInt32(myString);
```

Input

```
char userInput = Console.ReadKey().KeyChar;
string userInput = Console.ReadLine();
```

Boolean Operators

Symbol	Name
==	equality
!=	inequality
!	NOT
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
&&	logical AND
||	logical OR

Methods

```
static void Method1()
{
   // do this
}

static int Method2(char x)
{
   // do this
   return 7;
}
```

If-Else Statement

```
if (number >= 10)
   // do this
else if (number >= 20)
   // do this
else
   // do this
```

Switch Statement

```
switch (number)
{
   case 1:
      // do this
      break;
   case 2:
      // do this
      break;
   default:
      // do this
      break;
}
```

Loops

```
while (i <= 3)
{
   // do this
   i++;
}

do
{
   // do this
   i++;
}
while (i <= 3);
```

```
for (i = 1; i <= 3; i++)
{
   // do this
}

foreach (int n in nums)
   Console.WriteLine(n);
```

Strings

```
Console.Write($"There are {numApples} apples");
myString.Length
myString.ToLower()
myString.ToUpper()
myString.Contains("fun")
myString.IndexOf('i')
myString.Substring(5)
myString.Remove(5)
myString.Replace("fun","awesome")
```

Arrays

```
int[] myArray1 = { 2, 5, 7 };
int[] myArray2 = new int[3];
myArray2[1] =  myArray1[0];
myArray2 = myArray1; // both point to same array
myArray2 = (int[])myArray1.Clone(); // copy array
```

Lists

```
List<int> nums = new List<int>();
 nums.Add(9);                  nums.Count;
 nums.Add(5);                  nums.Contains(10);
 nums.Insert(0, 7);            nums.IndexOf(10);
 nums.RemoveAt(1);             nums.Sort();
 nums.Remove(5);               nums.Reverse();
 nums[0] = 10;                 nums.Clear();
```

Enumerations

```
enum Size { Small, Medium, Large };
Size sizeOfFries = Size.Medium;
```

Classes

```
class Lamp
{
   public string color;
   private bool isOn = false;
   static public int numLamps = 0;
   public Lamp(string color, bool isOn)
   { this.color = color;
     this.isOn = isOn; }
   public void TurnOn()
   { isOn = true; }
   public void TurnOff()
   { isOn = false; }
}
Lamp Lamp1 = new Lamp();
Lamp Lamp2 = new Lamp("red", true);
Lamp1.color = "green";
Lamp1.TurnOff();
Lamp.numLamps = 2;
```

Inheritance

```
class Car
{
   protected string color;
   public Car(string color)
   { this.color = color; }
   public string GetColor()
   { return color; }
   public virtual void DisplayInfo()
   { Console.Write($"The car is {color}\n"); }
}
class RaceCar : Car
{
   private int numNitros;
   public RaceCar(string color,
      int numNitros) : base(color
   { this.numNitros = numNitros; }
   public void UseNitro()
   { numNitros--; }
   public override void DisplayInfo()
   { Console.Write($"The racecar is {color} and" +
        $"has {numNitros} nitros\n"); }
}
class PickupTruck : Car
{
   private int bedLength;
   public PickupTruck(string color,
      int bedLength) : base(color)
   { this.bedLength = bedLength; }
   public override void DisplayInfo()
   { Console.Write($"The truck is {color} with" +
        $"a {bedLength}-inch bed\n"); }
}
Car myCar = new Car("red");
RaceCar myRaceCar = new RaceCar("green", 5);
PickupTruck myPickupTruck = new PickupTruck("white",
100);
myCar.DisplayInfo();
myRaceCar.DisplayInfo();
myPickupTruck.DisplayInfo();
Console.ReadKey();
```

Debugging in Visual Studio

Start with Debugging	F5
Start without Debugging	Ctrl+F5
Toggle Breakpoint	F9
Step Over	F10
Step Into	F11
Step Out	Shift+F11
Continue	F5

Error Handling

```
if (int.TryParse(myString, out parsedInteger))
{ // do this }
else
{ Console.Write($"{myString} is not an integer"); }

try
{
   num = Convert.ToInt32(myString); // exception risk
   if (num < 1 || num > 100)
      // throw your own exceptions for custom validation
      throw new Exception("Invalid input");
}
catch (Exception ex)
{ Console.Write(ex.Message + " Please try again."); }
```

Properties

```
class MyClass
{
   private int _myInteger;
   public int MyInteger
   {
      get { return _myInteger; }
      set { _myInteger = value; }
   }
   public int MyInteger2 { get; set; }
}
MyClass MyObject = new MyClass();
MyObject.MyInteger = 5; // calls set accessor
x = MyObject.MyInteger; // calls get accessor
MyObject.MyInteger2 = 5;
x = MyObject.MyInteger2;
```

Structures

```
struct Vector
{
   private int x;
   private int y;
   private int z;
   public Vector(int x, int y, int z)
   {
      this.x = x;
      this.y = y;
      this.z = z;
   }
   public void DisplayVector()
   { Console.Write($"({x},{y},{z})"); }
}

Vector MyVector = new Vector(1,2,3);
Vector MyVector2;
MyVector.DisplayVector();
MyVector2 = MyVector; // makes a copy of each field
```

Interfaces

```
interface IMyInterface
{ int SomeMethod(string someString); }
class MyClass : IMyInterface
{
   private int x, y;
   public int SomeMethod(string myString)
   { // this method is required }
   public void AnotherMethod()
   { // this method is not required }
}
IMyInterface MyInterface;
MyClass MyClassObject = new MyClass();
MyInterface = MyClassObject;
MyInterface.SomeMethod("hello");
```

Generics

```
class MyStack<T>
{
   private T[] array = new T[10];
   private int currentIndex = 0;
   public void Push(T item)
   {
     array[currentIndex] = item;
     currentIndex++;
   }
}
MyStack<string> stack = new  MyStack<string>();
stack.Push("hello");
```