C# Cheat Sheet

A comprehensive cheat sheet for the C# programming language, covering syntax, data types, control flow, object-oriented programming, and more. This cheat sheet is designed to provide a quick reference for both beginners and experienced C# developers.

Basic Syntax and Data Types

Basic Structure

A basic C# program structure:

```
using System;

namespace MyNamespace
{
  class MyClass
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello, World!");
    }
  }
}
```

`using System;`: Imports the `System` namespace, which contains fundamental classes and base types.
`namespace MyNamespace`: Declares a namespace to organize code.
`class MyClass`: Defines a class, the fundamental building block of C# programs.
`static void Main(string[] args)`: The entry point of the program.

Data Types

int	32-bit integer. Example: `int age = 30;`
float	32-bit floating-point number. Example: `float price = 19.99f;`
double	64-bit floating-point number. Example: `double pi = 3.14159265359;`
bool	Boolean value (true or false). Example: `bool isAdult = true;`
char	Single Unicode character. Example: `char grade = 'A';`
string	Sequence of characters. Example: `string name = "John Doe";`
decimal	128-bit data type suitable for financial and monetary calculations. Example: `decimal salary = 50000.00m;`

Variables

Variable declaration:

```
dataType variableName = value;
```

Example:

```
int age = 30;
string name = "John Doe";
```

Control Flow

Conditional Statements

if Statement	`if (condition)`
	`{`
	`  // Code to execute if the condition is true`
	`}`

if-else Statement	`if (condition)`
	`{`
	`  // Code to execute if the condition is true`
	`}`
	`else`
	`{`
	`  // Code to execute if the condition is false`
	`}`

if-else if-else Statement	`if (condition1)`
	`{`
	`  // Code to execute if condition1 is true`
	`}`
	`else if (condition2)`
	`{`
	`  // Code to execute if condition2 is true`
	`}`
	`else`
	`{`
	`  // Code to execute if all conditions are false`
	`}`

switch Statement	`switch (expression)`
	`{`
	`  case value1:`
	`  // Code to execute if expression == value1`
	`  break;`
	`  case value2:`
	`  // Code to execute if expression == value2`
	`  break;`
	`  default:`
	`  // Code to execute if no case matches`
	`  break;`
	`}`

Loops

for Loop	`for (int i = 0; i < 10; i++)`
	`{`
	`  // Code to execute`
	`}`

while Loop	`while (condition)`
	`{`
	`  // Code to execute`
	`}`

do-while Loop	`do`
	`{`
	`  // Code to execute`
	`} while (condition);`

foreach Loop	`foreach (var item in collection)`
	`{`
	`  // Code to execute`
	`}`

Jump Statements

break	Terminates the loop or switch statement.
continue	Skips the current iteration of the loop and proceeds to the next iteration.
return	Exits the current method.
goto	Transfers control to a labeled statement (use with caution).

Object-Oriented Programming

Classes and Objects

Class definition:

```
class MyClass
{
  // Fields (variables)
  // Methods (functions)
}
```

Object creation:

```
MyClass obj = new MyClass();
```

Encapsulation

Access Modifiers	`public`: Accessible from anywhere. `private`: Accessible only within the class. `protected`: Accessible within the class and derived classes. `internal`: Accessible within the same assembly. `protected internal`: Accessible within the same assembly and derived classes.
Properties	Provide controlled access to class fields. `public int Age { get { return age; } set { age = value; } }`

Inheritance

Inheritance allows a class to inherit properties and methods from another class.

```
class DerivedClass : BaseClass
{
  // Additional properties and methods
}
```

Example:

```
class Animal
{
  public string Name { get; set; }
  public virtual void MakeSound() {
Console.WriteLine("Generic animal sound"); }
}

class Dog : Animal
{
  public override void MakeSound() {
Console.WriteLine("Woof!"); }
}
```

Polymorphism

Virtual Methods	Methods that can be overridden in derived classes. `public virtual void MyMethod() { ... }`
Abstract Classes and Methods	Abstract classes cannot be instantiated and may contain abstract methods (methods without implementation). `abstract class MyAbstractClass { public abstract void MyAbstractMethod(); }`
Interfaces	Define a contract that classes can implement. `interface IMyInterface { void MyMethod(); }`

Advanced Concepts

Delegates and Events

Delegates are type-safe function pointers. Events provide a way for a class to notify other classes when something of interest happens.

```
delegate void MyDelegate(string message);

event MyDelegate MyEvent;
```

Example:

```
public class MyClass
{
  public delegate void MyDelegate(string
message);
  public event MyDelegate MyEvent;

  public void DoSomething()
  {
    if (MyEvent != null)
    {
      MyEvent("Something happened!");
    }
  }
}
```

LINQ (Language Integrated Query)

LINQ provides a unified way to query data from various sources.

```
var result = from item in collection
             where item.Property > 10
             select item;
```

Or using method syntax:

```
var result = collection.Where(item =>
item.Property > 10);
```

Asynchronous Programming

async and await	Keywords for writing asynchronous code. `public async Task<int> MyMethodAsync() { await Task.Delay(1000); return 42; }`
Task	Represents an asynchronous operation. `Task.Run(() => { ... });`

Exception Handling

Exception handling using `try-catch` blocks.

```
try
{
  // Code that may throw an exception
}
catch (Exception ex)
{
  // Code to handle the exception
}
finally
{
  // Code that always executes, regardless of
exceptions
}
```