HACKTHEBOX

INTRODUCTION TO C#

CHEAT SHEET

Cheat Sheet

Basic Syntax

Content	Description
Main Method	The main entry point for all C# programs. Defined as: `static void Main(string[] args) { }`
Case Sensitivity	C# is case-sensitive. For instance, `MyVariable`, `myvariable`, and `myVariable` would be three different identifiers.
Identifiers	Names given to entities such as variables, methods, etc. Must start with a letter (A-Z or a-z), an underscore (_), followed by zero or more letters, underscores, and digits (0-9).
Keywords	Predefined reserved words with special meanings that cannot be used as identifiers. Examples: `public`, `class`, `void`, etc.
The ;	In C#, the semicolon is a statement terminator. Each statement must end with a semicolon. Example: `int x = 10;`
Statements & Expressions	A statement performs an action, e.g., `x = 7;`. An expression is a construct comprising variables, operators, and method invocations evaluated to a single value, e.g., `x + 7`.
Blocks of Code	Blocks are used to group two or more C# statements and are defined by braces `{}`. Example: `{ int x = 7; Console.WriteLine(x); }`

Content	Description
Comments	Comments are used to explain code and are ignored by the compiler. Single-line comments start with `//`. Multi-line comments start with `/*` and end with `*/`.
Read Compiler Errors	Compiler errors indicate issues in your code that prevent it from compiling. They often include the line number and a description of the error, which can guide you towards resolving the issue.

Variables, Constants, and Data Types

Content	Description
Variables	Variables are storage locations, each defined with a specific data type. They are declared using the syntax: `dataType variableName; int num;`
Constants	Constants are similar to variables, but, as the name suggests, their value remains constant throughout the program. They are declared using the `const` keyword. `const double Pi = 3.14159;`
Enums	Enum is short for "enumerations", which are a distinct type consisting of a set of named constants. Declared using the `enum` keyword. `enum Days {Sun, Mon, Tue, Wed, Thu, Fri, Sat};`
Data Types	Data types specify the data type that a valid C# variable can hold. C# has several data types, including `int`, `double`, `char`, `bool`, and `string`. Each has its own range of values and behaviours.

Operators and Type Conversion

Content	Description
Arithmetic Operators	These include `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (modulus) and more.
Relational Operators	These include `==` (equal to), `!=` (not equal to), `<` (less than), `>` (greater than), `<=` (less than or equal to) and `>=` (greater than or equal to).
Logical Operators	These include `&&` (logical AND), `

Content	Description
Bitwise Operators	These perform operations on binary representations of numbers. They include `&` (AND), `
Assignment Operators	The assignment operator is `=`. There are also compound assignment operators like `+=`, `-=`, etc.
Unary Operators	These operators work with only one operand. They include `++`, `--`, and the logical negation operator `!`.
Ternary Operator	A shorthand for conditional statements. Syntax: `(condition) ? true_expression : false_expression`.
Null Conditional Operators	Used to simplify checking for null values, denoted as `?.`.
Null-coalescing Operator	Used to define a default value for nullable value types or reference types, denoted as `??`.
Implicit Type Conversion	Also known as widening conversion, it is done automatically by the compiler where no data loss is expected. Example: converting an integer to a float.
Explicit Type Conversion	Also known as narrowing conversion, the programmer must do it manually when there might be data loss. Example: converting a float to an integer.
Type Checking 'is'	The 'is' keyword checks if an object is of a certain type.
Type Checking 'as'	The 'as' keyword performs certain types of conversions between compatible reference types.

Namespaces

Content	Description

Content	Description
Creating and Organizing Code Using Namespaces	Namespaces are used to organise code and create globally unique types. Declare a namespace with `namespace` keyword followed by name and body enclosed in `{}`. `namespace MyNamespace { // code }`.
Importing and Using Namespaces in C# Programs	Use the `using` directive at the beginning of your code to include a namespace in your program. `using System;`
Resolving Naming Conflicts with Namespaces	If two namespaces contain types with the same name, fully qualify the name by including the namespace to avoid conflict. `System.Console.WriteLine("Hello, world!");`

Console I/O

Content	Description
Console.Read	Reads the next character from the standard input stream. Returns the ASCII value of the character read, or -1 if no more characters are available.
Console.ReadLine	Reads the next line of characters from the standard input stream. Returns a string containing the line read or null if no more lines are available.
Console.Write	Writes data to the standard output stream without a newline character at the end. Can take a string or other data types as argument(s). `Console.Write("Hello, world");`
Console.WriteLine	Similar to `Console.Write`, but appends a newline character (`\n`) at the end, causing subsequent output to appear on a new line. `Console.WriteLine("Hello, world");`

Control Statements and Loops

Content	Description

Content	Description
if	A control statement executes a block of code if a specified condition is `true`.
else	Used after an `if` statement. Its block of code executes if the `if` condition is `false`.
else if	Used after an `if` or another `else if` to test multiple conditions.
switch	A control statement that selects one of many code blocks to be executed.
for	A loop that repeats a block of code a certain number of times, defined at the start of the loop.
while	A loop that repeats a block of code as long as a specified condition is `true`.
do-while	Similar to the `while` loop, but checks the condition at the end of the loop. This means the loop will always run at least once.
break	Used to exit a loop or a `switch` statement prematurely.
continue	Skips the rest of the current iteration and moves directly to the next iteration of the loop.
goto	Transfers control to another part of the program marked with a label.

Arrays

Content	Description
Arrays in C#	An array is a collection of elements of the same type stored in contiguous memory locations. It is declared with the type followed by square brackets `[]`. `int[] arr;`
Multidimensional Arrays in C#	C# supports multidimensional arrays, declared with commas in the square brackets. `int[,] arr;`
The Array Class	Provides various properties and methods to work with arrays. It is defined within the `System` namespace.

Content	Description
Array.Sort()	A method that sorts the elements in an entire one-dimensional Array. `Array.Sort(arr);`
Array.Reverse()	Reverses the sequence of the elements in the entire one-dimensional Array or in a portion of it. `Array.Reverse(arr);`
Array.IndexOf()	Returns the index of the first occurrence of a value in a one-dimensional Array or in a portion of it. `int index = Array.IndexOf(arr, value);`
Array.Clear()	Sets a range of elements in the Array to zero, to false, or to null, depending on the element type. `Array.Clear(arr, startIndex, length);`

Strings

Content	Description
String Declaration	In C#, a string is declared as: `string str = "Hello World";`
String Concatenation	Strings can be concatenated using the `+` operator. Example: `string str = "Hello" + " World";`
String Interpolation	Insert variables directly in a string with `{}`. Example: `string str = $"Hello {name}";`
Length Property	To get the length of a string, use the `Length` property. Example: `int length = str.Length;`
Indexing	Access individual characters in a string with an index, starting from 0. Example: `char ch = str[0];`
Substrings	Extract part of a string using the `Substring` method. Example: `string substr = str.Substring(startIndex, length);`
String Comparison	Compare two strings using the `==` operator or the `String.Equals` method.
String Case Conversion	Convert to uppercase or lowercase using the `ToUpper()` and `ToLower()` methods.

Content	Description
Trimming Strings	Remove whitespace from start/end of a string with `Trim()`, `TrimStart()`, or `TrimEnd()`.
Searching in Strings	Find a substring or character using the `IndexOf()` or `Contains()` methods.
Replacing in Strings	Replace a substring or character using the `Replace()` method.

Collections

Content	Description
Iterating through a collection	You can iterate through a collection using a `foreach` loop. `foreach(var item in collection) { // actions }`.
List	A list is an ordered collection of items that can contain duplicates. Use the `Add`, `Remove`, and `Sort` methods to manipulate a list.
Dictionary	A dictionary is a collection of key-value pairs where each key must be unique. Use the `Add`, `Remove`, and `TryGetValue` methods to manipulate a dictionary.
HashSet	A HashSet is an unordered collection of unique elements. It provides high-performance set operations like union, intersection, and difference.
List vs Dictionary vs HashSet	Lists are best for accessing elements by index or iterating in order. Dictionaries provide fast lookups for elements based on a unique key. HashSets provide fast lookups like dictionaries but only store individual values instead of key-value pairs.
Performance considerations	In general, Dictionaries and HashSets provide faster lookups than Lists, especially for large collections. However, the choice between these depends on the specific requirements of your program.

LINQ (Language Integrated Query)

Content	Description
LINQ Query Syntax	LINQ queries consist of three parts: `from clause`, `where clause`, and `select clause`. `var result = from s in source where s.condition select s.property;`
Where	Filters a collection based on a condition. `var result = data.Where(x => x > 5);`
Select	Projects each sequence element into a new form. `var result = data.Select(x => x * 2);`
OrderBy/OrderByDescending	Sorts the elements of a sequence in ascending/descending order. `var result = data.OrderBy(x => x);` or `var result = data.OrderByDescending(x => x);`
GroupBy	Groups the elements of a sequence according to a specified key selector function. Example: `var result = data.GroupBy(x => x.Key);`
Join	Joins two collections based on matching keys. `var result = list1.Join(list2, x => x.Key, y => y.Key, (x, y) => new { X = x, Y = y });`
Aggregate	Applies an accumulator function over a sequence. `var result = data.Aggregate((a, b) => a + b);`
Count/Sum/Average/Min/Max	Performs calculations on a sequence of values. `var count = data.Count();`, `var sum = data.Sum();`, `var avg = data.Average();`, `var min = data.Min();`, `var max = data.Max();`

Methods and Exception Handling

Content	Description
Creating a method	Methods are declared with a return type, name, and parameters. `public int Add(int x, int y) { return x + y; }`
Method Scope	The scope of a method is the region of code within which a method can be accessed. Typically defined by the access modifier (`public`, `private`, etc.).

Content	Description
Static vs Non-Static Methods	Static methods belong to the class itself and can be called without creating an instance of the class. Non-static methods belong to an instance of the class.
try catch finally	`try` contains code that might throw an exception. `catch` defines what to do if an exception is thrown in the try block. `finally` contains code that will always be executed, whether an exception is thrown or not.
throw	The `throw` keyword is used to throw an exception from within your code explicitly. `throw new Exception("An error occurred.");`

Lambda Expressions

Content	Description
Simple Lambda Expression	A lambda expression with no parameters, represented as: `() => SomeMethod();`
Lambda Expression with Parameters	A lambda expression with one or more parameters. `(param1, param2) => param1 + param2;`
Lambda Expression with Statement Block	A lambda expression with multiple statements enclosed in `{}`. `(param1, param2) => { var result = param1 + param2; return result; };`

Libraries

Content	Description
NuGet	NuGet is a package manager for .NET. It allows you to add third-party libraries to your project with ease. You can add a NuGet package using the Package Manager Console or the Manage NuGet Packages dialogue box in an IDE.
Manual Referencing	If a library isn't available on NuGet, or you have a local library that you want to use, you can manually add a reference to it in your project.

HTB ACADEMY CHEATSHEET

Object-Oriented Programming

Content	Description
Classes	A blueprint for creating objects. Defined with the `class` keyword.
Accessors	Methods that get and set the value of class properties (`get` and `set`).
Automatic Properties	C# allows you to define a property without specifying a field (also known as auto-implemented properties). `public string Name { get; set; }`
Structs	Similar to classes but are value types and don't support inheritance. Defined with the `struct` keyword.
Encapsulation	The process of hiding internal details and exposing only what's necessary. Achieved with access modifiers like `public`, `private`, etc.
Inheritance	The ability for one class to inherit properties and methods from another class. Defined using the `:` symbol. `public class ChildClass : ParentClass`
Single Inheritance	A class can inherit from one base class only.
Multilevel Inheritance	A chain of inheritance where a class inherits from a base class, which itself inherits from another base class, and so on.
base	The `base` keyword is used to access members of the base class from within a derived class. `base.MethodName()`

Polymorphism and Abstraction

Content	Description
Polymorphism	Allows objects of different types to be treated as objects of a common supertype. Enables us to write more generic and reusable code.
Method Overloading	The ability to define multiple methods in the same scope with the same name but different parameters.

HTB ACADEMY CHEATSHEET

Content	Description
Method Overriding	Allows a subclass to provide a specific implementation of a method that is already provided by its superclass. Achieved using `override` keyword.
Operator Overloading	The ability to redefine or overload most of the built-in operators available in C#. This allows using operators with user-defined types as well.
Property Overriding	Similar to method overriding but for properties. Allows a subclass to override a property in the base class.
Abstraction	Hiding complex details and providing a simpler interface. In C#, it's achieved through abstract classes and interfaces. Abstract classes contain abstract methods that have a declaration but no implementation.

Generics

Content	Description
Benefits of Generics	Generics increase the reusability of code, type safety, and performance by eliminating boxing and unboxing.
Generic Classes	A class that can be customized to work with a specified data type. `public class GenericClass<T> { }`
Generic Methods	Methods with a type parameter in its declaration. `public T GenericMethod<T>(T param) { return param; }`
Generic Constraints	Constraints are used to restrict the types that can be used as arguments for a type parameter in a generic class or method. `public class GenericClass<T> where T : IComparable { }`

File I/O

Content	Description
StreamReader	`StreamReader` is used for reading characters from a byte stream in a particular encoding. `StreamReader sr = new StreamReader(path);`

Content	Description
Reading Data with StreamReader	Use `sr.ReadToEnd();` to read all data.
StreamWriter	`StreamWriter` is used for writing characters to a stream in a particular encoding. `StreamWriter sw = new StreamWriter(path);`
Writing Data with StreamWriter	Use `sw.Write("Hello World");` to write data.

Network I/O

Content	Description
HttpClient	`HttpClient` is a class in .NET used for sending HTTP requests and receiving HTTP responses.
GetAsync	Sends a `GET` request to the specified Uri and returns the response. Example: `var response = await client.GetAsync(url);`
PostAsync	Sends a `POST` request to the specified Uri with a specified content. Example: `var response = await client.PostAsync(url, content);`
PutAsync	Sends a `PUT` request to the specified Uri with a specified content. Example: `var response = await client.PutAsync(url, content);`
DeleteAsync	Sends a `DELETE` request to the specified Uri and returns the response. Example: `var response = await client.DeleteAsync(url);`

Asynchronous Programming

Content	Description
async & await	`async` modifier indicates that a method, lambda expression, or anonymous method is asynchronous. `await` operator is applied to a task in an `async` method to suspend the execution of the method until the awaited task completes.

Content	Description
Tasks	A `Task` represents a single operation that does not return a value and that usually executes asynchronously. A `Task<TResult>` represents a single operation that returns a value.
Task Cancellation	The cooperative cancellation model provided by .NET allows you to cancel running tasks using `CancellationTokenSource` and `CancellationToken`.
Exception Handling with Async Code	In async methods, use `try-catch` blocks to handle exceptions. Exceptions are propagated when the task is awaited.

HTB ACADEMY CHEATSHEET