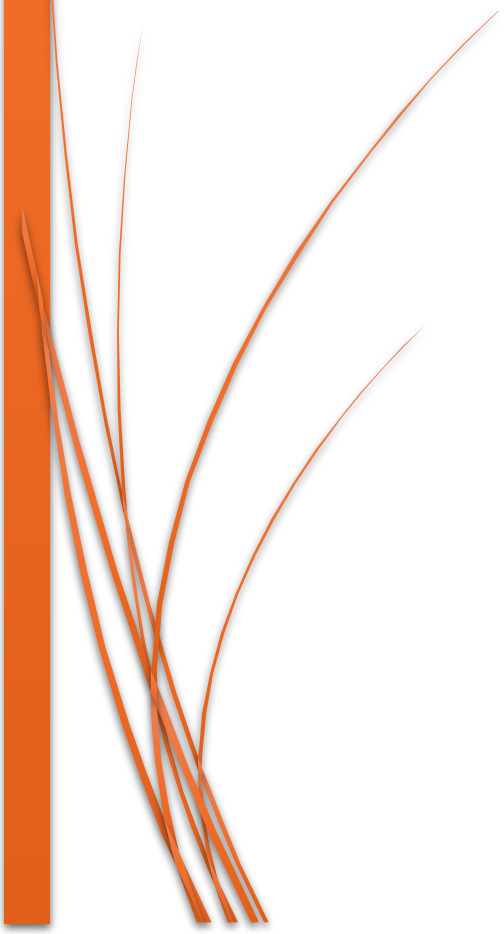
TechRacine
Integrations Simplified

C#.net Basics Guide



TechRacine
Integrations Simplified

Table of Contents

1. C#.net Basics

C# (pronounced "C-Sharp") is a modern, object-oriented programming language developed by Microsoft. It is part of the .NET ecosystem, which provides a wide variety of tools and libraries for building applications across multiple platforms, including Windows, web, cloud, and mobile. C# is powerful, versatile, and highly popular for enterprise-level applications, making it a great choice for beginners and experienced developers alike.

Key Features of C#

1. Object-Oriented: C# is designed with object-oriented principles, which means it supports concepts like classes, inheritance, polymorphism, and encapsulation. These principles make it easier to organize, reuse, and maintain code.

2. Type Safety: C# is a strongly typed language, meaning each variable has a defined type. This minimizes errors by enforcing type checks during compilation.

3. Rich Standard Library: The .NET framework and .NET Core provide an extensive set of libraries (called the Base Class Library, or BCL) for various tasks, including data manipulation, networking, file I/O, and user interface design.

4. Automatic Memory Management (Garbage Collection): C# includes a garbage collector that automatically manages memory, which helps prevent memory leaks and simplifies development.

5. Cross-Platform Support: With .NET Core and now .NET 5+ (the unified .NET platform), C# applications can run across Windows, macOS, and Linux. This is particularly useful for web and cloud applications.

6. Interoperability: C# can interact with code written in other languages and can use APIs from external sources, including native Windows APIs.

7. Language Features: C# has modern language features like async/await for asynchronous programming, LINQ (Language Integrated Query) for data manipulation, lambda expressions, pattern matching, and more.

2. Core concepts in C#.net

1. Namespaces: Namespaces are used to organize code and avoid naming conflicts. The 'System' namespace includes basic functionalities, such as console input and output.

2. Classes and Objects: Everything in C# is based on classes, which are blueprints for creating objects. Objects are instances of classes that hold data and methods to perform tasks.

```
class Car
{
    public string Color { get; set; }
    public void Drive()
    {
        Console.WriteLine("Driving...");
    }
```

```
}

// Usage
Car myCar = new Car();
myCar.Color = "Red";
myCar.Drive(); // Output: Driving...
```

3. Data Types: C# supports a wide range of data types, including 'int', 'double', 'char', 'string', 'bool', and custom types through classes and structs.

4. Control Structures: C# has standard control structures, such as 'if' statements, 'switch' cases, and loops ('for', 'while', 'do-while'), for controlling the flow of your program.

5. Methods: Methods are blocks of code that perform specific tasks and can be called from other parts of the code.

6. Properties: Properties provide a way to get and set values in a controlled manner.

7. Error Handling: C# includes robust error handling with 'try', 'catch', 'finally' blocks. This helps manage unexpected errors gracefully.

```
try
{
   int num = int.Parse("abc"); // This will throw an exception
}
catch (FormatException e)
{
   Console.WriteLine("Invalid format: " + e.Message);
}
```

8. Collections and LINQ: C# provides various collections like arrays, lists, dictionaries, and more, along with LINQ (Language Integrated Query) to make querying and manipulating data easier.

9. Asynchronous Programming: C# offers support for asynchronous programming with 'async' and 'await' keywords, which is particularly useful for tasks like web requests or file operations that can take time.

Development Environment
To get started with C#, it's best to use Visual Studio or Visual Studio Code as your IDE. Visual Studio has extensive support for C# development, including debugging, code suggestions, and project templates.

3. Basic structure of C# program

Namespaces: Help organize code and prevent naming conflicts. System is a basic namespace.
Classes: Everything in C# is part of a class, which is a blueprint for creating objects.
Main Method: The entry point for the application. The program starts executing from the Main method.

The best way to start learning any programming language is by writing a simple "Hello World" program.

using System; // This includes the System namespace, which contains basic classes

```
namespace HelloWorldApp
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello, World!"); // Outputs "Hello, World!" to the console
    }
  }
}
```

To compile a C# program,

Using Visual Studio:
If you're using Visual Studio 2022 (or earlier), it provides a fully integrated development environment for C#. Here's how you can compile and run your program:

Create a Project:
Open Visual Studio.
Go to File > New > Project.
Select Console App (.NET) under the C# templates for a simple program, and name your project.

Write Your Code:
In the Program.cs file, write your C# code.

Build and Run:
Press F5 to compile and run the program in debug mode.
Alternatively, go to Build > Build Solution to compile it without running, then click Debug > Start Without Debugging to run.

Visual Studio will handle compilation automatically and show errors if there are issues in the code.

4. Variables and Datatypes in C#.net

In C#, variables are used to store data, and each variable has a specific data type that determines the kind of data it can hold. Understanding variables and data types is fundamental in C# because C# is a statically-typed language, meaning you need to declare the type of each variable when you create it.

Declaring Variables
To declare a variable in C#, specify its data type followed by the variable name:

```
int age = 25;
```

```
string name = "Alice";
```

Basic Data Types in C#
C# offers several built-in data types, which can be grouped into the following categories:

#1. Integer Types
- 'int': Stores whole numbers from -2,147,483,648 to 2,147,483,647.
- 'long': Stores larger whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
- 'short': Stores smaller whole numbers from -32,768 to 32,767.
- 'byte': Stores values from 0 to 255 (used for small numbers).

```
int age = 30;
long population = 7800000000;
short temperature = -5;
byte level = 100;
```

#2. Floating-Point Types
- 'float': Stores single-precision floating-point numbers (smaller range than 'double').
  - Needs an 'f' suffix (e.g., '3.14f').
- 'double': Stores double-precision floating-point numbers (more commonly used).
- 'decimal': Stores high-precision decimal numbers, mainly used for financial calculations.

```
float price = 19.99f;
double distance = 1234.56;
decimal salary = 7890.123m;
```

#3. Character Type
- 'char': Stores a single character, enclosed in single quotes. Examples: ''A'', ''1'', ''#''.

```
char grade = 'A';
char symbol = '#';
```

#4. Boolean Type
- 'bool': Stores 'true' or 'false', often used in conditional statements.

```
bool isLoggedIn = true;
bool isGameOver = false;
```

#5. String Type
- 'string': Stores sequences of characters, such as words or sentences. Strings are enclosed in double quotes.

```
string name = "Alice";
string greeting = "Hello, World!";
```

#6. DateTime Type
- 'DateTime': Stores dates and times, useful for handling calendar-related data.

```
DateTime today = DateTime.Now;
DateTime birthDate = new DateTime(1990, 5, 24);
```

#7. Object Type

- 'object': The base type of all other types in C#. Any type can be assigned to a variable of type 'object'. However, you must cast it back to its original type when retrieving it.

```
object value = 123; // Can hold any data type
value = "Hello";
```

Type Inference with 'var'
C# can infer the type of a variable based on the assigned value using the 'var' keyword. The type is determined at compile time and cannot change.

```
var count = 10;        // 'int'
var name = "Alice";    // 'string'
var isActive = true;   // 'bool'
```

Type Conversion
C# provides ways to convert between compatible data types:

- Implicit Conversion: When converting from a smaller to a larger type (e.g., 'int' to 'double'), which is safe.

```
int num = 10;
double bigNum = num; // Implicit conversion
```

- Explicit Conversion: For converting from larger to smaller types or between incompatible types, use casting or conversion methods.

```
double pi = 3.14;
int truncatedPi = (int)pi; // Cast to 'int'
```

```
string str = "123";
int number = int.Parse(str); // Convert 'string' to 'int'
```

Here's a simple program that demonstrates variable declaration, assignment, and type conversion:

```
using System;

namespace VariableExample
{
  class Program
  {
    static void Main(string[] args)
    {
      int age = 25;
      double height = 5.9;
      char initial = 'A';
      bool isStudent = true;
      string name = "Alice";
      DateTime currentDate = DateTime.Now;

      Console.WriteLine("Name: " + name);
      Console.WriteLine("Age: " + age);
      Console.WriteLine("Height: " + height);
      Console.WriteLine("Initial: " + initial);
      Console.WriteLine("Is Student: " + isStudent);
```

```
        Console.WriteLine("Current Date: " + currentDate);

        // Type conversion
        string ageString = age.ToString();
        int newAge = int.Parse(ageString);
        Console.WriteLine("Converted Age: " + newAge);
    }
  }
}
```

5. Operators in C#.net

In C#, operators are symbols that perform specific actions on operands. These operators are grouped based on the type of operations they perform, such as arithmetic, comparison, logical, and assignment.

1. Arithmetic Operators
Arithmetic operators are used to perform mathematical calculations.

Operator	Description	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (remainder)	a % b
++	Increment	a++ (post-increment) or ++a (pre-increment)
--	Decrement	a-- (post-decrement) or --a (pre-decrement)

2. Assignment Operators
Assignment operators are used to assign values to variables. They can also be combined with arithmetic operators for shorthand operations.

Operator	Description	Example
=	Simple assignment	a = 5
+=	Addition assignment	a += 3 (same as a = a + 3)
-=	Subtraction assignment	a -= 2 (same as a = a - 2)
*=	Multiplication assignment	a *= 4 (same as a = a * 4)
/=	Division assignment	a /= 2 (same as a = a / 2)
%=	Modulus assignment	a %= 3 (same as a = a % 3)

3. Comparison Operators
Comparison operators are used to compare two values and return a boolean result (true or false).

Operator	Description	Example
==	Equal to	a == b
!=	Not equal to	a != b
>	Greater than	a > b
<	Less than	a < b
>=	Greater than or equal to	a >= b
<=	Less than or equal to	a <= b

4. Logical Operators
Logical operators are used to perform logical operations on boolean values and expressions.

Operator	Description	Example
&&	Logical AND	(a > b) && (a > c)
!	Logical NOT (negation)	!(a > b)

5. Bitwise Operators
Bitwise operators operate on binary representations of integers.

Operator	Description	Example
&	Bitwise AND	a & b
^	Bitwise XOR	a ^ b
~	Bitwise NOT	~a
<<	Left shift	a << 2
>>	Right shift	a >> 2

6. Ternary Operator
The ternary operator is a shorthand for if-else statements.

```
condition ? expressionIfTrue : expressionIfFalse;
```

Example:

```
int a = 10;
int b = 5;

string result = (a > b) ? "a is greater" : "b is greater";
Console.WriteLine(result);  // Output: "a is greater"
```

6. Control structures in C#.net

Control structures in C# dictate the flow of a program, allowing you to make decisions and repeat code. These include conditional statements, loops, and branching structures. Here's a breakdown of the main control structures in C#:

1. Conditional Statements
Conditional statements let you execute specific blocks of code based on certain conditions.

'if' Statement

The 'if' statement executes a block of code only if the specified condition is true.

```
int age = 20;
if (age >= 18)
{
   Console.WriteLine("You are an adult.");
}
```

'if-else' Statement

The 'if-else' statement provides an alternative block of code if the condition is false.

```
int age = 15;
if (age >= 18)
{
   Console.WriteLine("You are an adult.");
}
else
{
   Console.WriteLine("You are a minor.");
}
```

'else if' Statement

Use 'else if' to check multiple conditions in sequence.

```
int age = 65;
if (age < 18)
{
   Console.WriteLine("You are a minor.");
}
else if (age >= 18 && age < 65)
{
   Console.WriteLine("You are an adult.");
}
else
{
   Console.WriteLine("You are a senior citizen.");
}
```

'switch' Statement

The 'switch' statement selects one of many code blocks to execute based on the value of an expression.

```
int day = 3;
switch (day)
{
   case 1:
      Console.WriteLine("Monday");
      break;
   case 2:
      Console.WriteLine("Tuesday");
```

```
    break;
  case 3:
    Console.WriteLine("Wednesday");
    break;
  default:
    Console.WriteLine("Other day");
    break;
}
```

2. Loops

Loops allow you to execute a block of code multiple times.

'for' Loop
The 'for' loop is commonly used when the number of iterations is known. It consists of an initializer, a condition, and an increment/decrement.

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine("Iteration " + i);
}
```

'while' Loop
The 'while' loop repeats as long as a specified condition is true. It checks the condition before each iteration.

```
int i = 0;
while (i < 5)
{
  Console.WriteLine("Iteration " + i);
  i++;
}
```

'do-while' Loop
The 'do-while' loop executes the code block at least once, then checks the condition at the end of each iteration.

```
int i = 0;
do
{
  Console.WriteLine("Iteration " + i);
  i++;
} while (i < 5);
```

'foreach' Loop
The 'foreach' loop iterates over each item in a collection (e.g., an array or a list).

```
string[] fruits = { "Apple", "Banana", "Cherry" };
foreach (string fruit in fruits)
{
  Console.WriteLine(fruit);
```

```
}
```

3. Branching Statements
Branching statements alter the flow of control in loops and conditional statements.

'break' Statement
The 'break' statement exits the loop or 'switch' statement immediately.

```
for (int i = 0; i < 10; i++)
{
   if (i == 5)
   {
      break;
   }
   Console.WriteLine(i);
}
// Output: 0 1 2 3 4
```

'continue' Statement
The 'continue' statement skips the current iteration and moves to the next iteration of the loop.

```
for (int i = 0; i < 10; i++)
{
   if (i % 2 == 0)
   {
      continue; // Skip even numbers
   }
   Console.WriteLine(i);
}
// Output: 1 3 5 7 9
```

'return' Statement
The 'return' statement exits a method and optionally returns a value.

```
int Sum(int a, int b)
{
   return a + b;
}
Console.WriteLine(Sum(3, 4)); // Output: 7
```

4. Exception Handling ('try-catch-finally')
Exception handling in C# is done using 'try', 'catch', and 'finally' blocks to handle runtime errors gracefully.

```
try
{
   int x = 5;
   int y = 0;
   int result = x / y; // This will cause a divide-by-zero exception
```

```
}
catch (DivideByZeroException ex)
{
   Console.WriteLine("Cannot divide by zero.");
}
finally
{
   Console.WriteLine("This will always execute.");
}
```

7. C#.net programs

Here are some programs that helps to improve your C# basic knowledge.

1. Simulate Simple Calculator
2. Even or Odd Checker from the array of given inputs
3. Find out the given number is Prime Number or not
4. Palindrome Checker
5. Temperature Converter (Celcius/Farenheit)
6. Simple To-Do List
7. BMI Calculator
8. Currency Converter
9. Random Password Generator - Combination of unique strings, numbers and special chars
10. Age Calculator