# Introduction to C#

## Create First C# Program: 'Hello World'

C# programs start with a 'Main' method which serves as the entry point for the application. Here's a simple 'Hello World' example:  
  
```csharp  
using System;  
  
class Program  
{  
 static void Main()  
 {  
 Console.WriteLine("Hello, World!");  
 }  
}  
```  
  
- `using System;`: Allows usage of the System namespace.  
- `Console.WriteLine`: Prints the message to the console.

## Understanding C# Program Structure

C# program structure typically includes the following components:  
  
1. `Namespaces`: Used to organize code.  
2. `Class`: Encapsulates data and methods.  
3. `Methods`: Contain the code to be executed.  
  
Example:  
```csharp  
using System;  
  
namespace MyApp  
{  
 class Program  
 {  
 static void Main()  
 {  
 Console.WriteLine("C# Program Structure");  
 }  
 }  
}  
```

## Working with Code Files, Projects, and Solutions

- \*\*Code Files\*\*: C# source code files have a `.cs` extension.  
- \*\*Projects\*\*: Collection of code files, references, and resources.  
- \*\*Solutions\*\*: Container for multiple related projects.  
  
You can use Visual Studio to create and manage projects and solutions easily.

## Datatypes and Variables with Conversion

C# has several built-in datatypes such as `int`, `float`, `double`, `string`, `bool`, etc.  
  
Example of variable declaration and conversion:  
```csharp  
int number = 42;  
double convertedNumber = (double)number; // Explicit conversion  
string strNumber = number.ToString(); // Conversion to string  
```

## Operators and Expressions

C# supports various operators like arithmetic, logical, relational, etc.  
  
Example:  
```csharp  
int a = 10, b = 5;  
int sum = a + b; // Arithmetic operator  
bool isEqual = a == b; // Relational operator  
```

## Statements

C# statements include declarations, control statements, loops, and conditionals.  
  
Example:  
```csharp  
int x = 10;  
if (x > 5)  
{  
 Console.WriteLine("x is greater than 5");  
}  
```

## Understanding Arrays

Arrays store multiple values of the same type.  
  
Example:  
```csharp  
int[] numbers = { 1, 2, 3, 4 };  
foreach (int number in numbers)  
{  
 Console.WriteLine(number);  
}  
```

## Defining and Calling Methods

Methods are blocks of code that perform a specific task.  
  
Example:  
```csharp  
class Program  
{  
 static void Greet(string name)  
 {  
 Console.WriteLine("Hello, " + name);  
 }  
  
 static void Main()  
 {  
 Greet("Alice");  
 }  
}  
```

## Understanding Classes and OOP Concepts

Classes are the blueprint for objects in C#. They encapsulate data and behavior.  
  
Example:  
```csharp  
class Person  
{  
 public string Name { get; set; }  
 public void Greet()  
 {  
 Console.WriteLine("Hello, " + Name);  
 }  
}  
  
class Program  
{  
 static void Main()  
 {  
 Person person = new Person { Name = "Alice" };  
 person.Greet();  
 }  
}  
```

## Interfaces and Inheritance

Inheritance allows a class to inherit from another class. Interfaces define a contract for classes.  
  
Example:  
```csharp  
interface IAnimal  
{  
 void Speak();  
}  
  
class Dog : IAnimal  
{  
 public void Speak()  
 {  
 Console.WriteLine("Bark");  
 }  
}  
```

## Scope and Accessibility Modifiers

Modifiers define the visibility of classes and members.  
  
Example:  
```csharp  
public class Example  
{  
 private int hiddenValue;  
 public int VisibleValue;  
}  
```

## Namespaces and .NET Libraries

Namespaces group related classes and methods.  
  
Example:  
```csharp  
using System;  
namespace MyNamespace  
{  
 class MyClass  
 {  
 public void Display()  
 {  
 Console.WriteLine("In MyNamespace");  
 }  
 }  
}  
```

## Creating and Adding References to Assemblies

References allow using libraries. In Visual Studio, you can add references via Solution Explorer.  
  
Example:  
1. Right-click the project -> Add Reference -> Select the library.

## Working with Collections

Collections like List, Dictionary, and ArrayList store data.  
  
Example:  
```csharp  
List<int> numbers = new List<int> { 1, 2, 3 };  
numbers.Add(4);  
```

## Enumerations

Enums define named constants.  
  
Example:  
```csharp  
enum Days { Sunday, Monday, Tuesday }  
Days today = Days.Monday;  
Console.WriteLine(today);  
```

## Data Tables

DataTable is used to manage in-memory data.  
  
Example:  
```csharp  
DataTable table = new DataTable();  
table.Columns.Add("Name");  
table.Rows.Add("Alice");  
```

## Exception Handling

C# uses try-catch for error handling.  
  
Example:  
```csharp  
try  
{  
 int x = 10 / 0;  
}  
catch (DivideByZeroException ex)  
{  
 Console.WriteLine(ex.Message);  
}  
```

## Different Project Types

C# supports various project types like Console, Windows Forms, and Web applications. Each type serves different development needs.

## Working with the String Class

String class provides methods for string manipulation.  
  
Example:  
```csharp  
string name = "Alice";  
string upperName = name.ToUpper();  
Console.WriteLine(upperName);  
```

## Working with the DateTime Class

DateTime is used for handling dates and times.  
  
Example:  
```csharp  
DateTime now = DateTime.Now;  
Console.WriteLine(now.ToString("yyyy-MM-dd"));  
```

## Basic File Operations

C# provides classes for file operations.  
  
Example:  
```csharp  
using System.IO;  
File.WriteAllText("test.txt", "Hello World");  
string content = File.ReadAllText("test.txt");  
Console.WriteLine(content);  
```