all people who know the language. Similarly, programming languages also have a vocabulary, which is referred to as the set of keywords of that language, and a grammar, which is referred to as the syntax.

Consider a scenario, where one person speaks in English with the other person. However, the person who is listening understands Chinese and not English. Therefore, he calls a translator who can translate the words of English to Chinese. Similarly, computers cannot directly understand the instructions written in a programming language. Therefore, you need a translator to convert the instructions written in a programming language to machine language.

A compiler is a special program that processes the statements written in a particular programming language and converts them into machine language. Like everything else in the computer, the compiler also follows the Input-Process-Output (I-P-O) cycle. It takes the programming language instructions as input. It processes these instructions to convert them to machine language. These instructions can then be executed by the computer. This process of conversion is called compilation.

The following figure shows the working of a compiler:

![Working of a Compiler](image)

For each programming language, there is a different compiler available. For example, to compile a program written in the C language, you require a C compiler. For a Java program, you require a Java compiler. For C# programs, you will use the csc compiler.
Using C# for Writing Programs

Similar to the various programming languages, C# also has some predefined keywords that can be used for writing programs. For example, `class` is a keyword in C# that is used to define classes. Keywords are reserved words that have a special meaning. Further, the syntax for C# defines rules for grammatical arrangement of these keywords.

For example, the syntax of declaring a class in C# is:

```csharp
    class <class name>
    {
    
    }
```

In the preceding syntax, the braces, known as delimiters, are used to indicate the start and end of a class body.
In addition to using the predefined classes, you can also define your own classes. Consider the following code, which defines a class named Hello:

```csharp
public class Hello
{
    public static void Main(string[] args)
    {
        System.Console.WriteLine("Hello, World! \n");
    }
}
```

The preceding class declaration includes the method, `Main()` that will display the message, “Hello, World!” on your screen. The preceding code includes the following components:

- The class keyword
- The class name
- The `Main()` method
- The `System.Console.WriteLine()` method
- Escape sequences

Let us discuss these components in detail.

**The class Keyword**

The class keyword is used to declare a class. In the preceding code, the `class` keyword declares the class, `Hello`.

**The Class Name**

The class keyword is followed by the name of the class. In the preceding code, `Hello` is the name of the class defined by using the `class` keyword.

**Class Naming Conventions in C#**

Class names should follow certain naming conventions or guidelines. A class name:
Let's Practice

Michelle wrote the following code snippet to print the personal details for a resume:

```csharp
using System;
public class Hello
{
    public static void Main(string[] args)
    {
        //Insert code here
    }
}
```

She wants to print the following output:

Michelle Gracias   Email: michelle2012@abc.net
type can be used to store a value of any type. A variable of this data type is declared by using the `dynamic` keyword.

Consider the following statement:

```csharp
dynamic id;
```

In the preceding statement, the variable `id` is declared as a dynamic variable. This means that it can be used to store any type of value, as shown in the following statements:

- `id = 1;`  //id is considered as int
- `id = “A001”;`  //id is considered as string

In the preceding statements, the variable, `id` inherits the data type of the assigned value. In the first assignment statement, `id` is assigned an `int` value. Therefore, it is treated as an integer. In the second assignment statement, it is assigned a string value. Therefore, it is considered a string.

Dynamic variables can use the methods associated with the data type they are resolved to. For example, a dynamic variable containing a string value can access all the string related attributes and methods, as shown in the following code snippet:

```csharp
dynamic d = "Sample";
int len = d.Length;
```

The `Length` attribute of a string stores the length of a string.

For dynamic variables, type checking is done at run time and not at compile time. This is known as duck typing. For example, consider the following code snippet:

```csharp
dynamic var = 1;
int len = var.Length;
```

In the preceding code snippet, no error will be thrown at compile time because dynamic variables are allowed to access members associated with any type. However, at run time the code will generate an error, as `var` is resolved as an `int` type and `length` property is not associated with `int` data type.
The Member Variables

Data members of a class are called member variables. In the given code, the Car class has two member variables, Engine and NoOfWheels. These variables are used to store the data provided by the user. The variables are declared as private. This means that the variables are accessible only inside the class in which they are declared. Any code outside the class cannot access the private variables declared in the class.

The Member Functions

Member methods are also known as member functions. The code for the Car class contains two member functions, InputDetails() and DisplayDetails(). Both the methods are declared as public methods. This means that the methods are accessible from any other class outside the Car class.

The InputDetails() and DisplayDetails() methods are declared with the return type, void. This means that the methods do not return any value. The GetEngine() method is declared with the return type, string. This means that the method returns a string value.

The InputDetails() method prompts the user to enter the engine model and number of wheels in the vehicle. It then stores these details in the respective variables. The DisplayDetails() method displays the values stored in the member variables.

You will learn more about methods in the subsequent chapters.

The Instantiating Class

The MainClass class contains the Main() method. This class is used to instantiate the Car class. The first line of code that a C# compiler looks for is the Main() method.

To implement the functionality of a class, you need to create an object of the class.

Consider a class named Car. To create an object named MyCar of the Car class, the following statement can be used in the Main() method:

```csharp
Car MyCar = new Car();
```