FUNDAMENTALS OF PROGRAMMING

INTRODUCTION TO C#

Yunghans Irawan

yirawan@nus.edu.sg

Topics

- Introduction to C#
- Compiling and Running C# Program
- C# Syntax

What is C# (C sharp)

Object Oriented Programming Language

From Microsoft

Runs on .NET (dot net) Framework

We will not discuss too much of the object oriented concepts in this module (FOPCS) because it will be discussed in more details in the next module.

Advantage of C#

- Well supported and integrated with Visual Studio
- General purpose – can be used to build many different kind of software from web applications, games, mobile app and robotics
- Contains many of the feature of modern programming language
    - Garbage collection
    - Exception handling
    - Type safety
    - Versioning
- C++ heritage
- Interoperability with other .NET languages
    - VB.NET, F#
- Large and vibrant developer community

C# Language Prelude

- C# is truly object oriented
  - The object oriented topics are covered under the OOP module;
  - We concentrate only on programming constructs now, shielding to a large extent from OOP concepts and clearly developing your programming constructs skill. We may need to introduce some custom libraries (Developed by ISS) for this.

- C# is Case Sensitive
  - Unlike some languages like VB, C# is case sensitive so the words:

    TomAndJerry, tomandjerry, TOMANDJERRY, tomANDjerry

    mean different things or interpreted differently.
  - CAUTION: Mind your case/capitalisation.

C# Language Prelude

- C# is strongly typed:
  - The freedom of mixing and assigning values to variables is limited.
  - More details on later lesson.

- C# works only when .Net Framework is available in the computer.

First C# Program

- Problem Statement and Task Objectives:
  - This is perhaps the simplest programs.
  - The program should greet the executor (not the executioner!) with the words “Welcome to ISS”
  - The purpose of the program is to expose to the learner the Structure of the C# program.
  - Some frills have been deliberately trimmed to aid the ease of understanding.
  - With this program we would go through the screen captures to walk you through the compilation and execution process.
  - We would demonstrate a few errors by forcefully introducing them.

First C# Program

FirstProg.cs

```
/*   This program when executed, prints
     the message "Welcome to ISS" */

//   Program written by Venkat

class MyFirstProgram
{
   static void Main()
   {
       System.Console.WriteLine("Welcome to ISS");
   }
}
```

class human
object me
more specification

the startup class which
ha the main method ;if yo
have one ,the one is the
startup class but if you
ave mang class which has
he main method

Multiline Comment

Single line Comment

In C# every program is coded as a class (object)

This is where the execution begins

Braces: Used to group lines of code into a block

Prints out whatever is provided in brackets

Fundamental Concepts

- Concepts illustrated in our First Program:
  - Comments:
    - Comments are useful for improving readability & documentation
    - Single line comments are placed after //
    - Multi-line comments are enclosed between /* … and … */
  - Class:
    - Class is a blueprint of an Object *(More on this in OOP).*
    - At present (for FOP) write each of your program as a class.
    - Class has a name - give a name meaningful to what the program does.
  - Braces:
    - Braces are curly (aka, Flower) brackets.
    - Several lines of codes that need to be logically grouped together as a "block" is placed between braces { … and … }
    - Here we use braces to indicate beginning and end of Class.

Fundamental Concepts

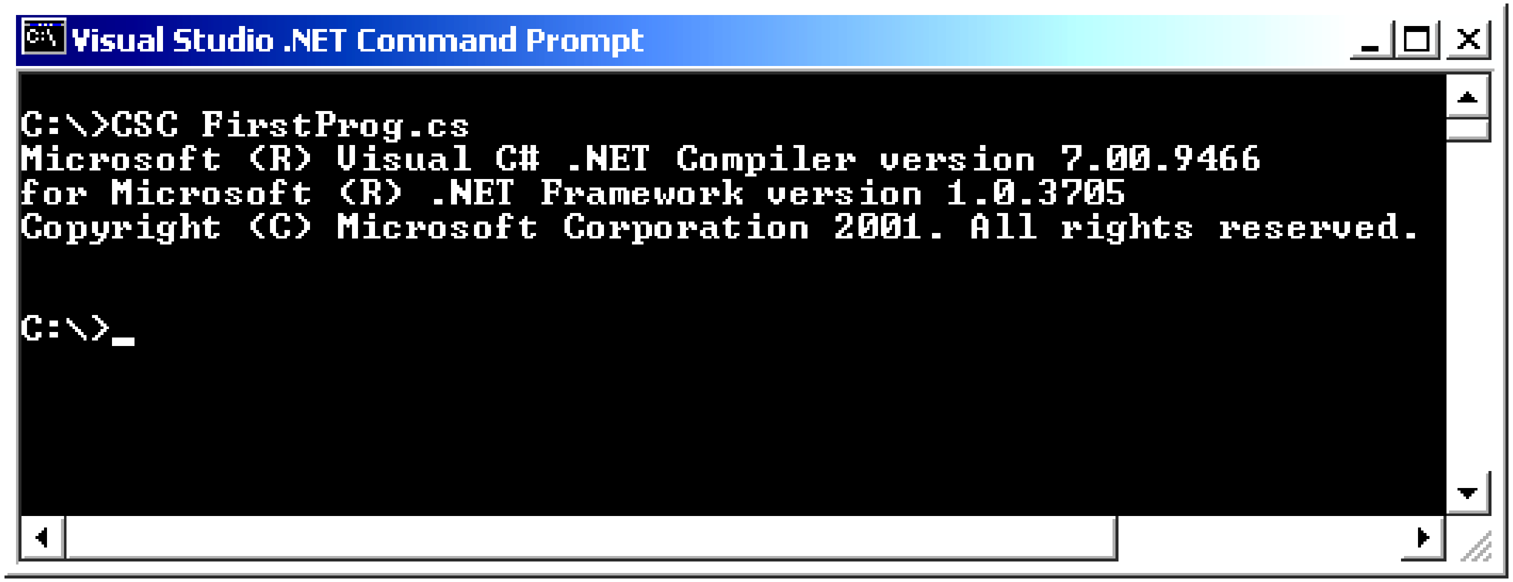
- Main method:
  - Method indicates an Action *(More on this in OOP).*
  - At present (for FOP) blindly follow the words
    `static void Main()`
  - The program always executes this first.
  - At present, place THE ENTIRE program codes here, i.e., between the braces that indicate the beginning and end of Main method.
  - In other words, at least for now, your entire flow chart would (should) be coded and placed under the Main method.

- Output:
  - Use the phrase `System.Console.WriteLine( … )` to write something on the Console screen. Whatever needs to be written would be placed within the brackets (note that these are normal brackets).

- Statement Separator / Terminator:
  - Every C# statement ends with a semi-colon “ `;` ”

Compiling

- The program can be compiled from the console with the command
  `C:\> CSC FirstProg.cs`



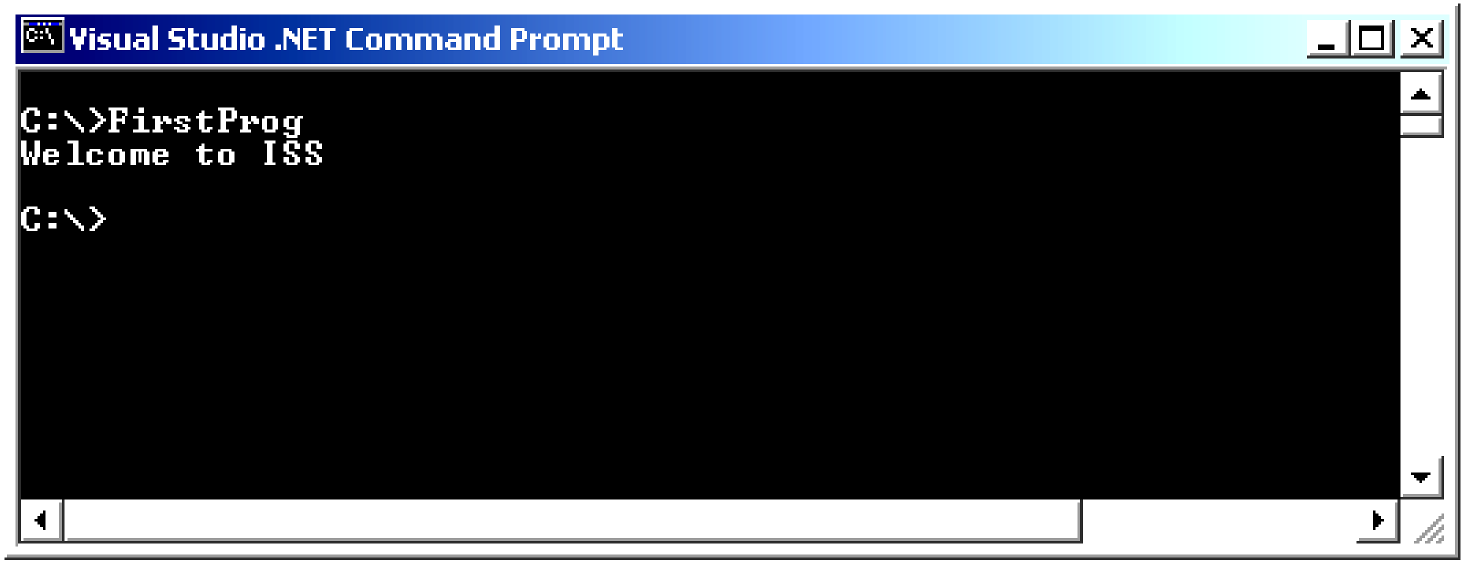
```
C:\>CSC FirstProg.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.


C:\>_
```

- Notice that there is no error message - if you use the **`Dir`** command, you would have the file **`FirstProg.exe`**

Executing

- The program can be executed from the console by the command

```
C:\> FirstProg
```



```
C:\>FirstProg
Welcome to ISS

C:\>
```

- Notice that on execution the computer responds:

```
Welcome to ISS
```

Syntax Errors

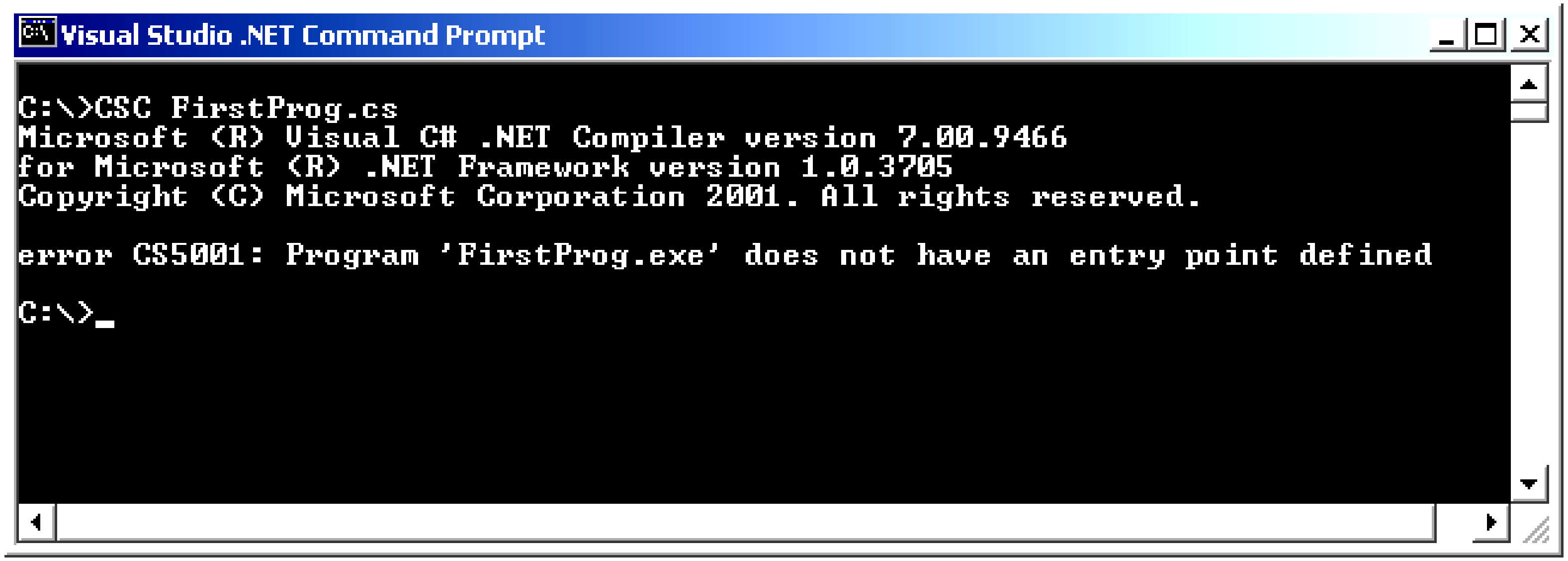
- To demonstrate some errors, we are going to mistype some things. Please note that instead of `Main` we type as `main` below

```
/*   This program when executed, prints
    the message "Welcome to ISS" */

//  Program written by Venkat

class MyFirstProgram
{
   public static void main()
   {
       System.Console.WriteLine("Welcome to ISS");
   }
}
```

Syntax Errors

- The program can be compiled from the console with the command

```
C:\>  CSC FirstProg.cs
```



Visual Studio .NET Command Prompt

```
C:\>CSC FirstProg.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

error CS5001: Program 'FirstProg.exe' does not have an entry point defined

C:\>_
```

- Error CS5001 occurs - the compiler does not find entry point; it expects Main() but does not find one.

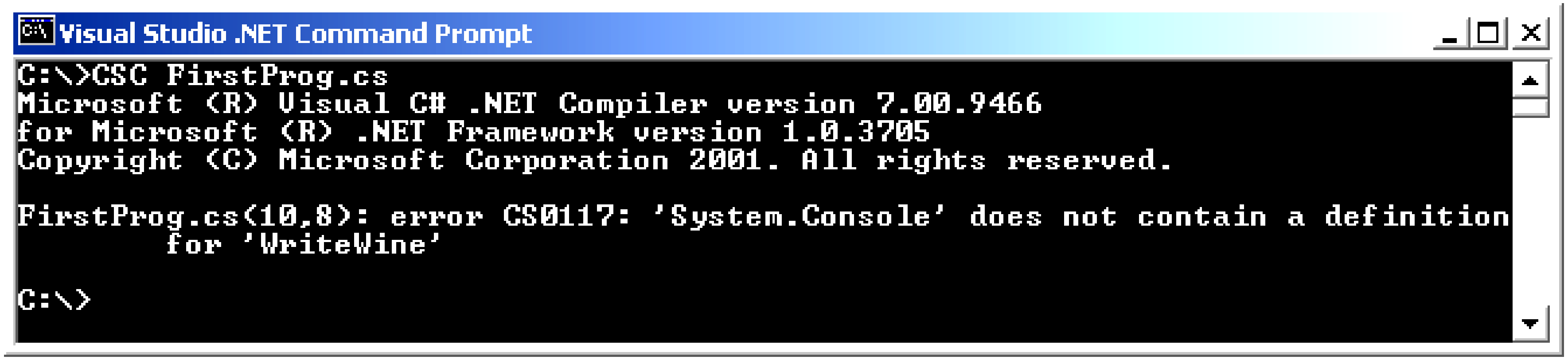
Syntax Errors

- To demonstrate some more errors, we mistype **`WriteLine`** as **`WriteWine`** below

```
Line
  1:  /*   This program when executed, prints
  2:       the message "Welcome to ISS" */
  3:
  4:  //   Program written by Venkat
  5:
  6:  class MyFirstProgram
  7:  {
  8:      public static void Main()
  9:      {
 10:           System.Console.WriteWine("Welcome to ISS");
 11:      }
 12:  }

Position:   012345678901234567890123456789012345678901234567890123
            0         1         2         3         4         5
```

Syntax Errors

- Compilation result:



```
C:\>CSC FirstProg.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

FirstProg.cs(10,8): error CS0117: 'System.Console' does not contain a definition
        for 'WriteWine'

C:\>
```

- Error CS0117 occurs:
  - The compiler is not able to find the method (or action) "WriteWine" in System.Console
  - The error occurs in Line 10 position 8 of FirstProg.cs.
    - Note even blank lines are counted.
    - Note I have placed 7 blank spaces before typing "System".

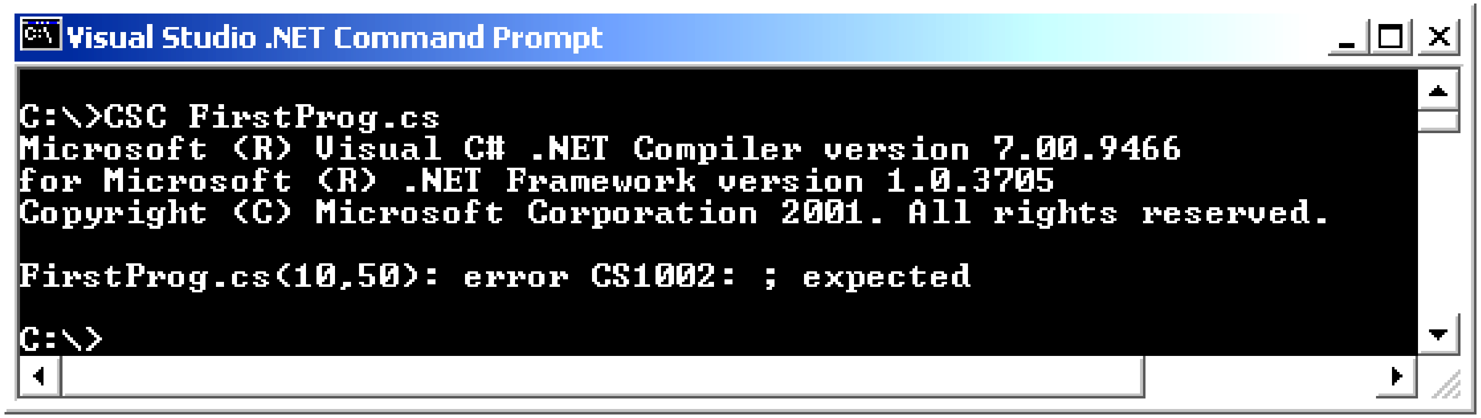
Syntax Errors

- In our third demonstration of syntax errors, we are going to miss the semi-colon after the WriteLine statement.

```
      Line
       1:   /*   This program when executed, prints
       2:       the message "Welcome to ISS" */
       3:
       4:   //  Program written by Venkat
       5:
       6:   class MyFirstProgram
       7:   {
       8:      public static void Main()
       9:      {
      10:         System.Console.WriteLine("Welcome to ISS")
      11:      }
      12:   }
Position:  012345678901234567890123456789012345678901234567890123
           0         1         2         3         4         5
```

Syntax Errors

- Compilation result:



```
C:\>CSC FirstProg.cs
Microsoft (R) Visual C# .NET Compiler version 7.00.9466
for Microsoft (R) .NET Framework version 1.0.3705
Copyright (C) Microsoft Corporation 2001. All rights reserved.

FirstProg.cs(10,50): error CS1002: ; expected

C:\>
```

- Error CS1002 occurs:
  - Compiler expects a semi-colon at position 50, line 10 of FirstProg.cs
  - Fortunately most of these are easily identified by compiler
  - Some times the error messages may not be so direct!

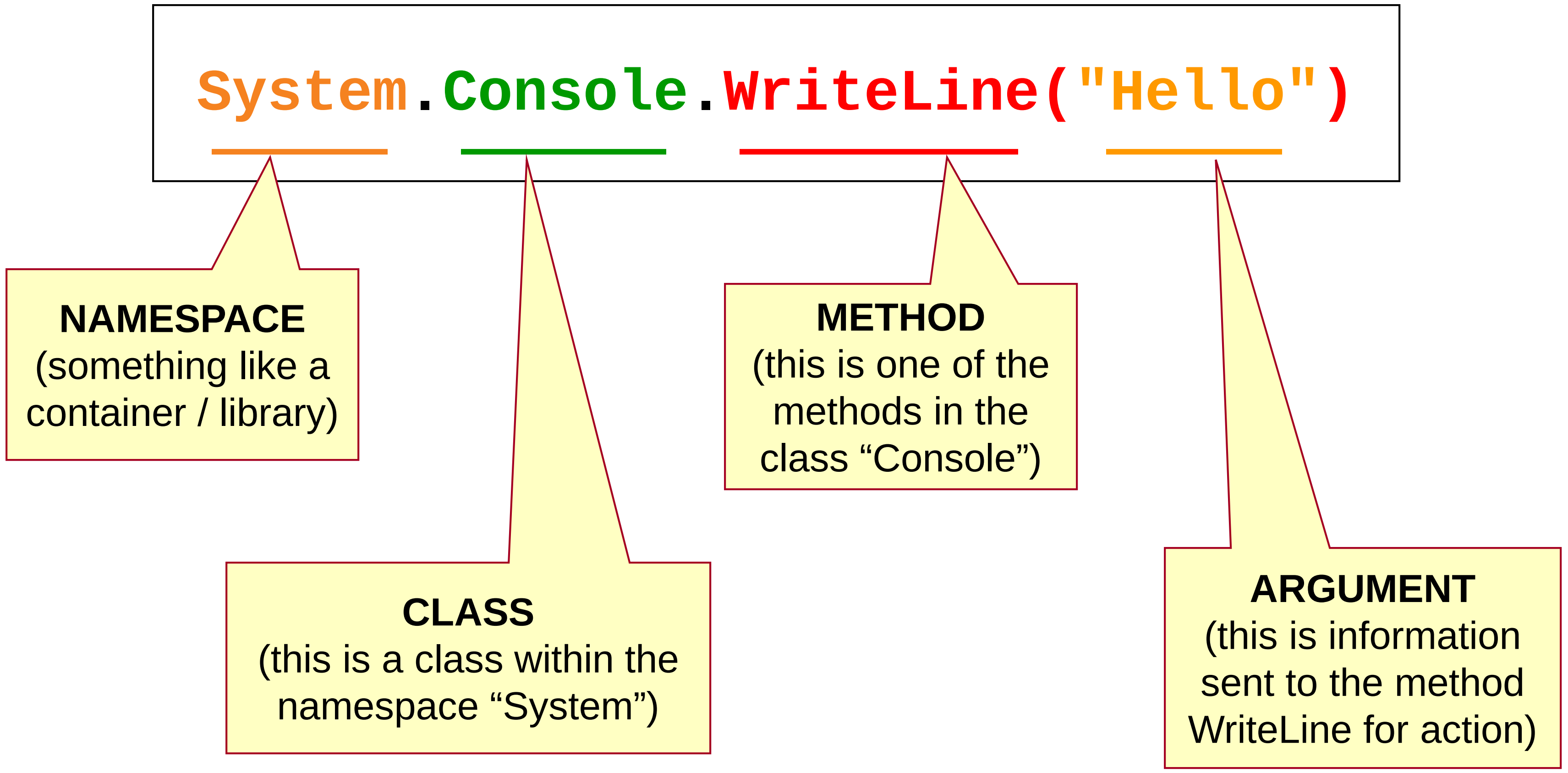
Syntax Summary

- Summary of key syntax characteristics in C#
  - Case Sensitivity
  - Statement Terminator
  - Syntax Errors
    - Displayed the way the compiler interprets it
      (may or may not be specific/correctly identified)
    - Displays: Line numbers and Position; Error Codes and Brief Description
    - Sometimes wrong location (line) may be indicated!!!
  - Multiple errors in a single program
  - Multiple errors in a single statement
    - All possible errors may be listed
    - Only one may be listed at a time!

  - The source codes are just text documents. Any text editor can be used to edit them.
    - IDE like Visual Studio can help make development easier

System.Console.WriteLine

- Understanding the components:



```
System.Console.WriteLine("Hello")
```

NAMESPACE
(something like a container / library)

CLASS
(this is a class within the namespace “System”)

METHOD
(this is one of the methods in the class “Console”)

ARGUMENT
(this is information sent to the method WriteLine for action)

Namespace

- What is a Namespace?
  - If book is an object (i.e., class); books are kept in libraries (i.e., namespaces). Thus namespaces can be thought of as containers where classes are placed in.
- Why should we have Namespaces?
  - To organise classes neatly.
- What are the periods (aka, dots)?
  - Separators
- In school libraries, we have sections, racks, shelves etc. to organise books, can we do the same with namespaces?
  - Yes of course; You may nest one namespace within another.
- How can I place “Welcome to ISS” program in a namespace? Typing the namespace makes program lines longer is there a simplification?
  - Yes; We will expand further on these now …

Creating Namespaces

- Do not create namespaces arbitrarily
  - Namespaces are an outcome of clear and robust design structure
  - Use primarily for large project environment
- Identify Namespace Hierarchy well before programming
- Use meaningful names for namespaces
- Enforce namespace consistently amongst all members

Creating Namespaces

- Purpose:
  - The purpose of this discussion at this stage is purely academic in nature and to place things in proper perspective as a preliminary understanding of namespace would enable ease of comprehending some of the further discussions.
  - What we present here is only a means of explaining how namespaces are coded in programs.
  - You need not use namespaces in Fundamentals of Programming Exercises / workshops.
  - More comprehensive discussion on namespace is included during the OOP lessons.

Creating Namespaces

- How do we place the "Welcome to ISS" program inside a namespace?
  - Let us assume you want to place this program (i.e., class) in a namespace called "MyPrograms".

FirstProg.cs :

```
 /*   This program when executed, prints
     the message "Welcome to ISS" */
 //  Program written by Venkat

namespace MyPrograms
{
   class MyFirstProgram
   {
      static void Main()
      {
        System.Console.WriteLine("Welcome to ISS");
      }
   }
}
```

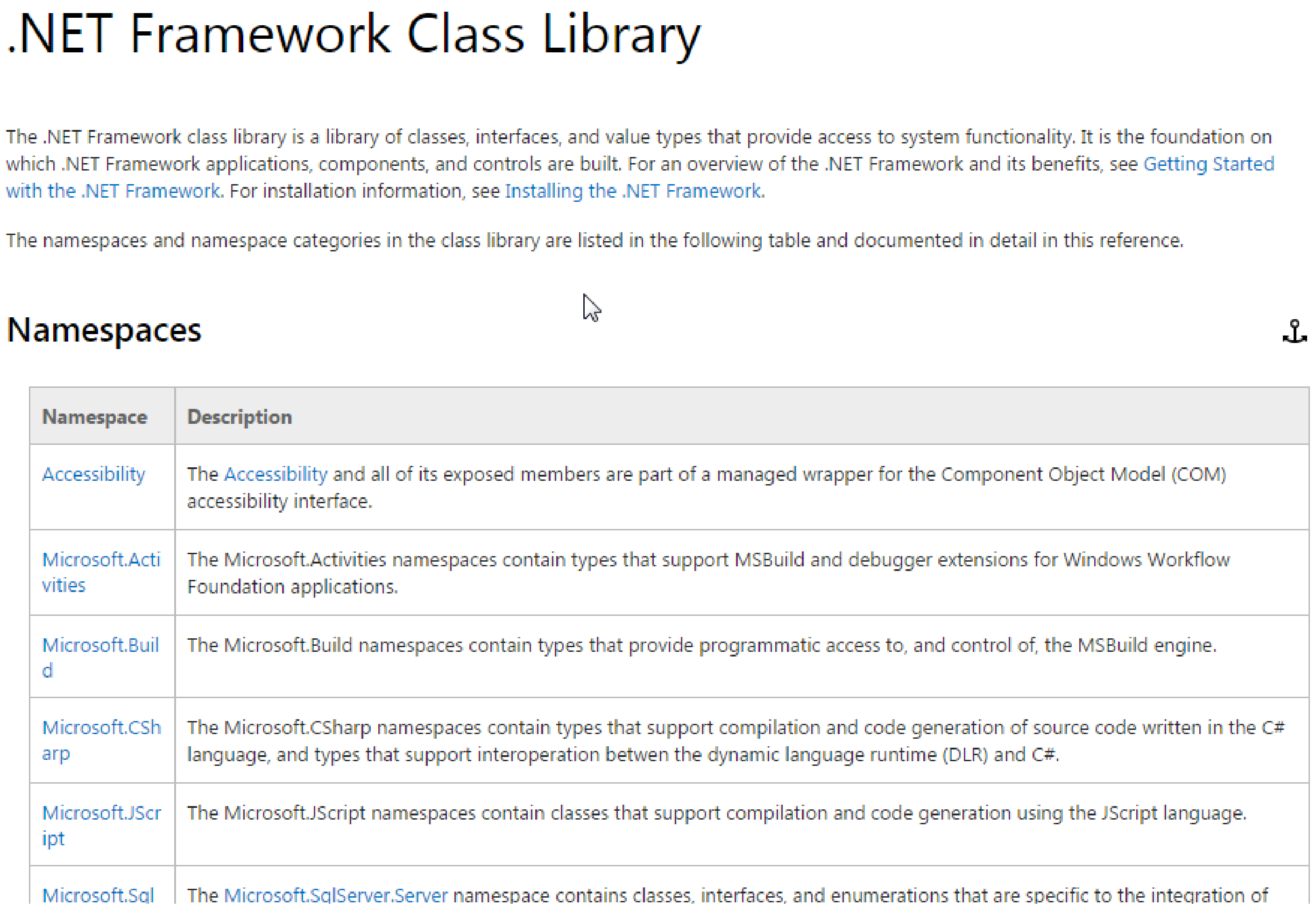
Impact of Using Namespace

- Does using Namespace in the "Welcome to ISS" program alter the way we compile and execute the program?
  - For this program there is NO difference in compilation/execution.

- Then where is the impact felt?
  - When this program is referred from another program (i.e., outside this namespace).

- Can I have an Example please?
  - Refer Microsoft's **`System.Console.WriteLine()`**
    - Microsoft has a Namespace called System, which contains a class called Console which, in turn, has a method called WriteLine. Since we are (or more precisely, our program statement is) outside the System namespace we should call WriteLine method by tracing the entire lineage.

Namespace in .NET Framework



.NET Framework Class Library

The .NET Framework class library is a library of classes, interfaces, and value types that provide access to system functionality. It is the foundation on which .NET Framework applications, components, and controls are built. For an overview of the .NET Framework and its benefits, see Getting Started with the .NET Framework. For installation information, see Installing the .NET Framework.

The namespaces and namespace categories in the class library are listed in the following table and documented in detail in this reference.

Namespaces

Namespace	Description
Accessibility	The Accessibility and all of its exposed members are part of a managed wrapper for the Component Object Model (COM) accessibility interface.
Microsoft.Activities	The Microsoft.Activities namespaces contain types that support MSBuild and debugger extensions for Windows Workflow Foundation applications.
Microsoft.Build	The Microsoft.Build namespaces contain types that provide programmatic access to, and control of, the MSBuild engine.
Microsoft.CSharp	The Microsoft.CSharp namespaces contain types that support compilation and code generation of source code written in the C# language, and types that support interoperation betwen the dynamic language runtime (DLR) and C#.
Microsoft.JScript	The Microsoft.JScript namespaces contain classes that support compilation and code generation using the JScript language.
Microsoft.Sql	The Microsoft.SqlServer.Server namespace contains classes, interfaces, and enumerations that are specific to the integration of

https://msdn.microsoft.com/en-us/library/mt472912(v=vs.110).aspx

Further Refinement

- Purpose of this discussion:

  Q: Is there a way of avoid retyping namespaces every time a class is referred? Especially troublesome in long namespaces…

  A: Yes; We can rewrite the FirstProg.cs as below:

```
/*    This program when executed, prints
      the message "Welcome to ISS" */
//   Program written by Venkat

using System;
class MyFirstProgram
{
   static void Main()
   {
      Console.WriteLine("Welcome to ISS");
   }
}
```

Indicates that this program “uses” the namespace called System

Notice that we **no longer qualify** Console class with it’s namespace viz., System

Discussion on "using”

- How does the program work?
  - When the compiler hits on the word Console, it looks for standard classes in the existing namespaces; if not found it looks for in the System namespace.
- Can I use more than one “using” in a single program?
  - Yes
- What happens if there are similar class names in two different namespaces that I have used?
  - Perfectly possible; but highly undesirable.
  - Compilation error (CS0104) will result “Ambiguous Reference”!
- Does that mean I cannot use a class name used by anyone else?
  - Guaranteed uniqueness in class names is impossible to achieve.
  - If CS0104 occurs, use the <u>fully qualified name</u> in your current (referring) program to remove the ambiguity.

WriteLine in Detail

- Revisiting the WriteLine statement

```
System.Console.WriteLine("Hello")
```

- What is an argument?
  - Argument is something (loosely speaking) that is sent (supplied) to a method for performing an action using it (the argument).
- Why is the word *Hello* placed between quotes?
  - Text information is included in double quotes
    - Conversely if the word *Hello* is written without quotes it is understood by the compiler as a variable name.
- Can WriteLine use an argument other than text?
  - Yes

WriteLine: Output Examples

Caution: In the examples below we provide only the relevant program lines.

To execute the line you should place these inside the Main( ) method.

Program:

```
System.Console.WriteLine("Welcome");
System.Console.WriteLine("to ISS");
```

Output:

```
Welcome
to ISS
```

Program:

```
System.Console.Write("Welcome");
System.Console.WriteLine(“ to");
System.Console.Write("ISS");
```

Output:

```
Welcome to
ISS
```

WriteLine: Concatenation

Program:

```
System.Console.WriteLine("Welcome"," to ISS");
System.Console.WriteLine();
```

Output:

```
Welcome
```

This is a logical error - the word “to ISS” does not appear.

Program:

```
System.Console.WriteLine("Welcome"+" to ISS");
System.Console.WriteLine();
```

Output:

```
Welcome to ISS
```

The symbol plus indicates concatenation (joining) if one of the argument is text

WriteLine: Numeric Values

Program:

```
System.Console.WriteLine(5.2);
System.Console.WriteLine("Done");
```

Output:

```
5.2
Done
```

WriteLine can take numeric arguments.

Program:

```
int A = 6;
System.Console.WriteLine(A);
System.Console.WriteLine("Done");
```

Output:

```
6
Done
```

We can assign the number 6 to a integer variable and use the variable as argument

WriteLine: Variations

Program:

```
double A = 5.2;
System.Console.WriteLine(A);
System.Console.WriteLine("Done");
```

Output:

```
5.2
Done
```

We can assign the number 5.2 to a double (non-integer) variable and use this as argument

---

Program:

```
double A = 5.2;
System.Console.WriteLine("A");
System.Console.WriteLine("Done");
```

Output:

```
A
Done
```

Since A is within quotes, the letter A rather than value of A is printed

WriteLine: Variations

Program:

```
double A = 5.2; double B = 3.9;
System.Console.WriteLine(A + B);
System.Console.WriteLine("Done");
```

Output:

```
9.1
Done
```

The value of A and B are added and printed. The + sign acts as addition for numeri

---

Program:

```
double A = 5.2; double B = 3.9;
System.Console.WriteLine("A" + "B");
System.Console.WriteLine("Done");
```

Output:

```
AB
Done
```

Since A and B are in quotes the word AB is printed not the value.

WriteLine: Variations

Program:

```
double A = 5.2; double B = 3.9;
System.Console.WriteLine(A + "B");
System.Console.WriteLine("Done");
```

Output:

```
5.2B
Done
```

The plus symbol indicates concatenation (joining) *even* if one of the argument is text

Program:

```
double A = 5.2; double B = 3.9;
System.Console.WriteLine("A" + B);
System.Console.WriteLine("Done");
```

Output:

```
A3.9
Done
```

The plus symbol indicates concatenation (joining) *even* if one of the argument is text

Escape Sequence

- A Text used as an argument to WriteLine can have some special characters for formatting purposes. These are called Escape Sequences.
- Some common Escape Sequences:

Escape Sequence	Meaning
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal Tab. Move the screen cursor to the next tab stop.
\\	Backslash. Used to print backslash character.
\”	Double Quote. Used to print double quote character.

WriteLine: Esc Sequences

Program:

```
Console.WriteLine("Welcome \n to ISS");
```

Output:

```
Welcome
 to ISS
```

The words following the \n character are printed in a new line

Program:

```
Console.WriteLine("Welcome \t to ISS");
Console.WriteLine();
```

Output:

```
Welcome          to ISS
```

The words following the \t character are printed from the next tab stop position

WriteLine: Placing a " inside

TASK: To print: He said, “ Thanks! " and bowed.

Program:

```
Console.WriteLine("He said, " Thanks! " and bowed");
```

Text 1

???

Text 2

This above is wrong. Syntax Error.

Program:

```
Console.WriteLine("He said, \" Thanks! \" and bowed");
```

Output:

```
He said, "Thanks" and bowed.
```

The letter after \ is 'escaped' thus the character " is not interpreted as end of Text

WriteLine: Placeholder

Program:

```
double Wt = 71.6; double Ht = 177;
Console.WriteLine
 ("He weighs " + Wt + " kgs and is " + Ht + " cms tall");
```

Output:

```
He weighs 71.6 kgs and is 177 cms tall
```

We have seen this before: achieving output through concatenation.

Program:

```
double Wt = 71.6; double Ht = 177;
Console.WriteLine
 ("He weighs {0} kgs and is {1} cms tall", Wt, Ht);
```

Output:

```
He weighs 71.6 kgs and is 177 cms tall
```

Format symbol { } is used as place holder for variables given as subsequent arguments.
First argument is always text to print, Second argument is always {0}, third is {1} etc.

WriteLine: Placeholder

Program:

```
double Wt = 71.6; double Ht = 177;
Console.WriteLine
 ("He weighs {1} kgs and is {0} cms tall", Ht, Wt);
```

Output:

```
 He weighs 71.6 kgs and is 177 cms tall
```

This also works - but more confusing!

WriteLine: Placeholder

Program:

```
double Wt = 71.6; double Ht = 177;
Console.WriteLine
 ("He weighs {0} kgs and is {1} cms tall", Ht, Wt);
```

Output:

```
He weighs 177 kgs and is 71.6 cms tall
```

Output is obtained - But is this what you wanted?

Program:

```
double Wt = 71.6; double Ht = 177;
Console.WriteLine
 ("He weighs {1} kgs and is {0} cms tall", Wt, Ht);
```

Output:

```
He weighs 177 kgs and is 71.6 cms tall
```

Again: Output is obtained - But is this what you wanted?

ADVICE: To avoid problem always try to go from left to right and 0, 1, 2 etc.

WriteLine: Number Formatting

- Placeholders can be used with formatting information
- Numeric formatting is usually done with # symbol.
- The 0 (zero) and . (decimal point) may also be used in conjunction

Program:

```
double Wt = 71.6; double Ht = 177;
Console.WriteLine
("He weighs {0:###.#} kgs and is {1:####} cms tall",Wt,Ht);
```

Output:

```
He weighs 71.6 kgs and is 177 cms tall
```

Each # accommodates a digit of number.

Extra # s are ignored.

WriteLine: Number Formatting

- Numeric formatting where fewer spaces are given for significant digits
- Example below give two # s for height to accommodate a three digit number:

Program:

```
double Wt = 71.6; double Ht = 177;
Console.WriteLine
("He weighs {0:###.#} kgs and is {1:##} cms tall",Wt,Ht);
```

Output:

```
He weighs 71.6 kgs and is 177 cms tall
```

This situation is called overflow;

Computer does not generate errors; Computer does not truncate digits.

Result still valid (safety is kept in mind by the computer).

WriteLine: Number Formatting

- Numeric formatting where fewer spaces are given for least significant digit (portion to right of decimal point)
- Example below less # s for height and weight on the fractional side of the decimal point.

Program:

```
double Wt = 71.63; double Ht = 177.7;
Console.WriteLine
("He weighs {0:##.#} kgs and is {1:###} cms tall",Wt,Ht);
```

Output:

```
He weighs 71.6 kgs and is 178 cms tall
```

The least significant digits are dropped.

"Rounding" takes place -

for weight rounded to one decimal place, for height rounding to nearest integer.

WriteLine: Number Formatting

Consider the following statement

Program:

```
double Cost = 0.60;
Console.WriteLine
    ("The chocolate costs {0:###.##} dollars",Cost);
```

Output:

```
 The chocolate costs .6 dollars
```

Ugly! Can we improve it? Yes see below.

Program:

```
double Cost = 0.60;
Console.WriteLine
  ("The chocolate costs {0:##0.00} dollars",Cost);
```

Output:

```
The chocolate costs 0.60 dollars
```

That's Neat!

WriteLine: Number Formatting

- Numeric formatting can use Zeros in place of #
  - As with # prints one digit wherever a 0 exists.
  - helps in forcing zeros to be printed instead of digits being omitted
  - This is applicable to both leading zeros and trailing zeros.

- Sometimes (like in cheques) you would like to force zeros in front also. This is how it is done:

Program:

```
double Amount = 71.52;
Console.WriteLine
     ("Please pay ${0:00000.00} only" , Amount);
```

Output:

```
Please pay $00071.52 only
```

References on Formatting

- https://docs.microsoft.com/en-us/dotnet/standard/base-types/formatting-types
- .NET Framework provides standard formatting (based on commonly used formatting) and custom which allows developer to have more control over the formatting

Reading Input from Console

- To read input from console, we can use `System.Console.ReadLine` method.

Program:

```
Console.Write
    ("Please enter your name: ");
string name = Console.ReadLine();
Console.WriteLine("Your name is {0}", name);
```

Output:

```
Please enter your name: John
Your name is John
```

Converting string to number

- System.Console.ReadLine() always returns a string.
- Sometimes we need to convert this string into number so that we can use it for calculation.
- To do that we can use the following methods
  - Convert.ToInt32()
  - Integer.Parse()

Convert.ToInt32() and Integer.Parse()

Program:

```
Console.Write
    (“Please enter a number: “);
int number = Convert.ToInt32(Console.ReadLine());
Console.WriteLine(“{0} + 1 = {1}”, number, number + 1);
```

Output:

```
Please enter a number: 15
15 + 1 = 16
```

Program:

```
Console.Write
    (“Please enter a number: “);
int number = Integer.Parse(Console.ReadLine());
Console.WriteLine(“{0} + 1 = {1}”, number, number + 1);
```

Output:

```
Please enter a number: 15
15 + 1 = 16
```

Conversion and Parsing for other data types

Data Type	Example

double	`double value = Double.Parse(input);`
	`double value = Convert.ToDouble(input);`

int	`int value = Int32.Parse(input);`
	`int value = Convert.ToInt32(input);`

bool	`bool value = Boolean.Parse(input);`
	`bool value = Convert.ToBoolean(input`

Summary

- What method did we cover that can produce an output to the screen?
- What method did we cover to get an input from user?
- What is the purpose of escape sequences?
- What are some of the symbols used for number formatting and its meaning?