Session: 2

Variables and Data Types

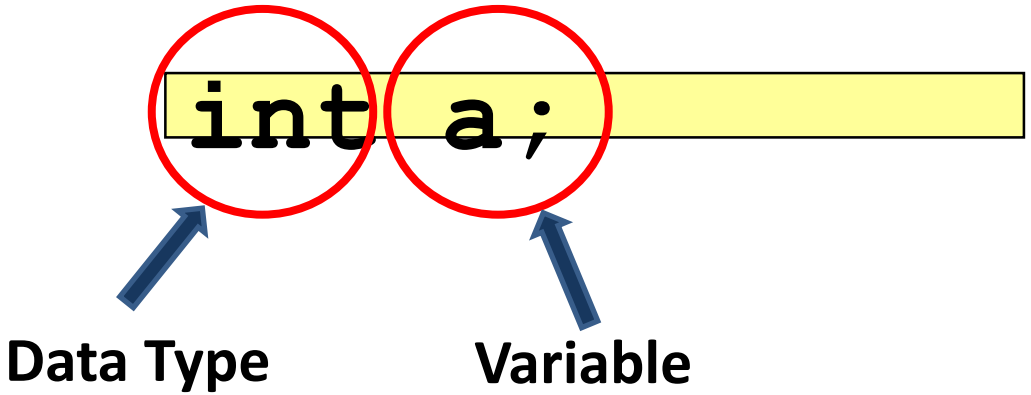
Objectives

- Define and describe variables and data types in C#
- Explain comments and XML documentation
- Define and describe constants and literals
- List the keywords and escape sequences
- Explain input and output

Variables

- A variable is used to store data in a program and is declared with an associated data type.
- A variable has a name and may contain a value.
- Syntax:

```
data_type <variableName> [= <value>];
```



- In C# data types are divided into two categories:

Value Types	Reference Types
• store actual values in a stack → faster memory allocation. • Most of the built-in data types are value types such as: int, float, double, char, and bool. • Include user-defined value types: struct and enum	• store the memory address of variables in a stack, store actual values in a heap. • These values can either belong to a built-in data type or a user-defined data type.

Pre-defined Data Types

- Are basic data types that have a pre-defined range and size.

Data Type	Size	Range
byte	Unsigned 8-bit integer	0 to 255
sbyte	Signed 8-bit integer	-128 to 127
short	Signed 16-bit integer	-32,768 to 32,767
ushort	Unsigned 16-bit integer	0 to 65,535
int	Signed 32-bit integer	-2,147,483,648 to 2,147,483,647
uint	Unsigned 32-bit integer	0 to 4,294,967,295
long	Signed 64-bit integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	Unsigned 64-bit integer	0 to 18,446,744,073,709,551,615
float	32-bit floating point with 7 digits precision	±1.5e−45 to ±3.4e38
double	64-bit floating point with 15-16 digits precision	±5.0e−324 to ±1.7e308
Decimal	128-bit floating point with 28-29 digits precision	±1.0 × 10e−28 to ±7.9 × 10e28
Char	Unicode 16-bit character	U+0000 to U+ffff
bool	Stores either true or false	true or false

- Declared for value types rather than for reference types.
- Have to initialize at the time of its declaration.
- The following syntax is used to initialize a constant:

```
const <data type> <identifier name> = <value>;
```

```
const float _pi = 3.14F;
float radius = 5;
float area = _pi * radius * radius;
Console.WriteLine("Area of the circle is " + area);
```

Types of Literals 1-2

- **Boolean Literal:** has two values, **`true`** or **`false`**.
  For example, `bool val = true;`

- **Integer Literal:** can be assigned to **`integer data`** types.
  Suffixes for integer literals include U, L, UL, or LU. U denotes **`uint`** or **`ulong`**, L denotes **`long`**. UL and LU denote **`ulong`**.
  For example, `long val = 53L;`

- **Real Literal:** is assigned to **`real`** data types.
  Suffix letters include F denotes **`float`**, D denotes **`double`**, and M denotes **`decimal`**.
  For example, `float val = 1.66F;`

- **Character Literal** is assigned to a char data type. A character literal is always enclosed in single quotes.
  For example, `char val = 'A';`

- **String Literal:** There are two types of string literals in C#, regular and verbatim.
  - regular string is a standard string.
  - verbatim string is prefixed by the '@' character.

  A string literal is always enclosed in double quotes.
  For example, `string mailDomain = "gmail.com";`

- **Null Literal:** The null literal has only one value, null.
  For example, `string email = null;`

Escape Sequence Characters in C#

Escape Sequence Characters	Non-Printing Characters
\'	Single quote, needed for character literals.
\"	Double quote, needed for string literals.
\\	Backslash, needed for string literals.
\0	Unicode character 0.
\a	Alert.
\b	Backspace.
\f	Form feed.
\n	New line.
\r	Carriage return.
\v	Vertical tab.
\t	Horizontal tab.
\?	Literal question mark.
\ooo	Matches an ASCII character, using a three-digit octal character code.
\xhh	Matches an ASCII character, using hexadecimal representation (exactly two digits). For example, \x61 represents the character 'a'.
\uhhhh	Matches a Unicode character, using hexadecimal representation (exactly four digits). For example, the character \u0020 represents a space.

- **Single-line Comments**: Begin with two forward slashes (//).

Snippet

```
// This block of code will add two numbers
int doSum = 4 + 3;
```

- **Multi-line Comments**: Begin with a forward slash followed by an asterisk (/*) and end with an asterisk followed by a forward slash (*/).

Snippet

```
/* This is a block of code that will multiply two
numbers, divide the resultant value by 2 and display
the quotient */
int doMult = 5 * 20;
int doDiv = doMult / 2;
Console.WriteLine("Quotient is:" + doDiv)
```

- In C#, all console operations are handled by the **Console** class of the **System** namespace.
  A namespace is a collection of classes having similar functionalities.

- There are two output methods that write to the standard output stream as follows:
  - **Console.Write()**
  - **Console.WriteLine()**

- Two methods that read data from standard input stream :
  - **Console.Read()**
  - **Console.ReadLine()**

- **Convert** class in the **System** namespace is used to convert one base data type to another base data type.

```
Console.Write("Enter your name: ");
string userName = Console.ReadLine();
Console.Write("Enter your age: ");
int age = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter the salary: ");
double salary = Convert.ToDouble(Console.ReadLine());
Console.WriteLine("Name: {0}, Age: {1}, Salary: {2} ",
userName, age, salary);
```

Output

```
Enter your name: David Blake
Enter your age: 34
Enter the salary: 3450.50
Name: David Blake, Age: 34, Salary: 3450.50
```

Using Numeric Format Specifiers

Format Specifier	Name	Description
`C or c`	Currency	The number is converted to a string that represents a currency amount.
`D or d`	Decimal	The number is converted to a string of decimal digits (0-9), prefixed by a minus sign in case the number is negative. The precision specifier indicates the minimum number of digits desired in the resulting string. This format is supported for fundamental types only.
`E or e`	Scientific (Exponential)	The number is converted to a string of the form '-d.ddd...E+ddd' or '-d.ddd...e+ddd', where each 'd' indicates a digit (0-9).

- The following code demonstrates the conversion of a numeric value using C, D, F and E format specifiers:

Snippet

```
int d = 100;
float f = 12.4566F;
Console.WriteLine("Currency format = {0:C6}",d);
Console.WriteLine("Decimal format ={0:D6}",d);
Console.WriteLine("Floating format={0:F2}",d);
Console.WriteLine("D = {0:E4}; F = {1,8:0.00}", d, f);
```

Output

```
Currency format = $100.000000
Decimal format = 000100
Floating format= 100.00
D = 1.0000E+002; f =   12.45
```

Standard Date and Time Format Specifiers

- is special characters displaying the date and time values in different formats.
- Syntax :

```
Console.WriteLine("{format specifier}", <datetime object>);
```

```
DateTime dt = DateTime.Now;
Console.WriteLine("Short date,time with seconds (G):{0:G}", dt);
Console.WriteLine("Month and day (m):{0:m}", dt);
Console.WriteLine("Short time (t):{0:t}", dt);
Console.WriteLine("Short time with seconds (T):v{0:T}", dt);
Console.WriteLine("Year and Month (y):{0:y}", dt);
```

```
Short date, time with seconds (G): 6/23/2018 6:01:24 PM
Month and day (m): June 23
Short time (t): 6:01 PM
Short time with seconds (T): 6:01:24 PM
Year and Month (y): June 2018
```

Custom DateTime Format Strings

Format	Description
ddd	abbreviated name of the day of the week
dddd	full name of the day of the week
FF	two digits of the seconds fraction
H	Hour, 0 to 23
HH	Hour, 00 to 23
MM	Month, 01 to 12
MMM	abbreviate name of month
s	Seconds, 0 to 59

```
Date is Sat Jun 23, 2018
Time is 06:19 PM
24 hour time is 18:19
Time with seconds: 18:19:28 PM
```

```
DateTime date = DateTime.Now;
Console.WriteLine("Date is {0:ddd MMM dd, yyyy}", date);
Console.WriteLine("Time is {0:hh:mm tt}", date);
Console.WriteLine("24 hour time is {0:HH:mm}", date);
Console.WriteLine("Time with seconds: {0:HH:mm:ss tt}", date);
```

Summary

- A variable is a named location in the computer's memory and stores values.
- Comments are used to provide detailed explanation about the various lines in a code.
- Constants are static values that you cannot change throughout the program execution.
- Keywords are special words pre-defined in C# and they cannot be used as variable names, method names, or class names.
- Escape sequences are special characters prefixed by a backslash that allow you to display non-printing characters.
- Console operations are tasks performed on the command line interface using executable commands.
- Format specifiers allow you to display customized output in the console window.