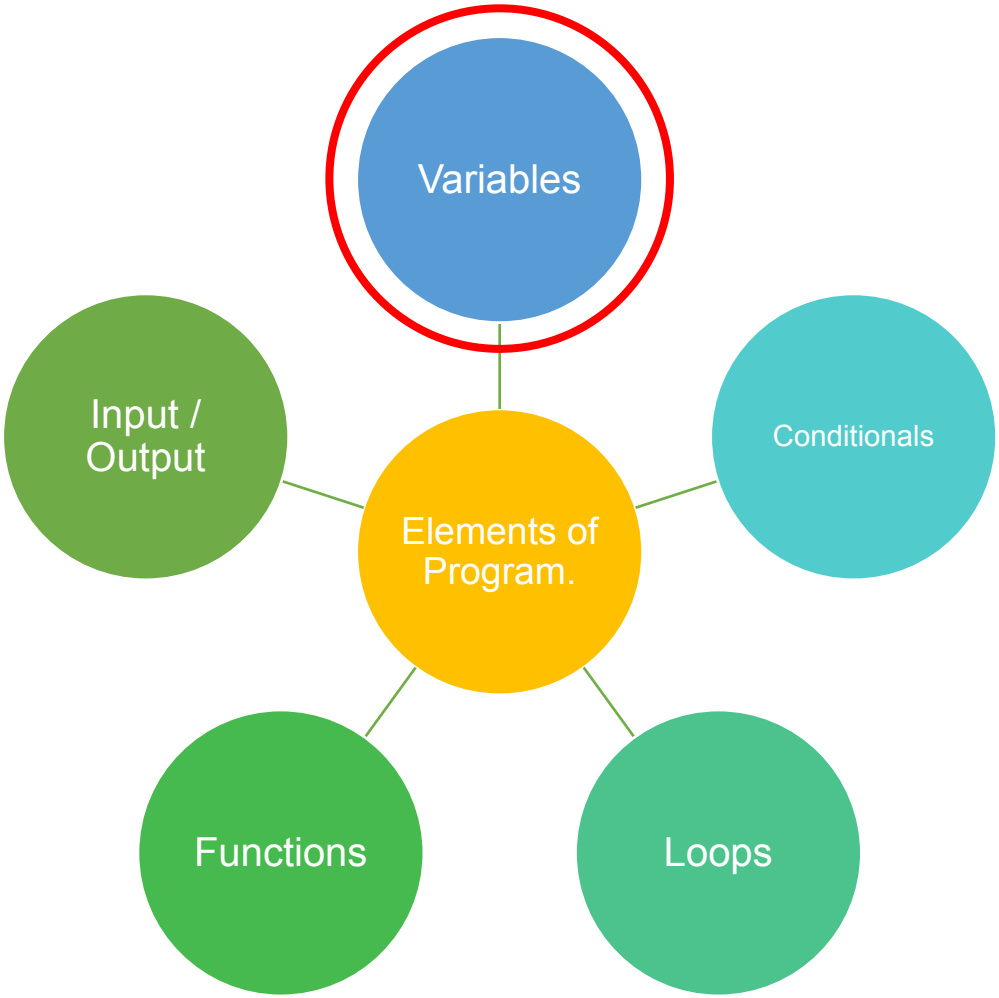
Variables

A4 – Constructs and Techniques and Their Implementation in Programming

Ikromkhujaev Ilyoskhuja

Elements

- To develop any instruction there are some elements needed or we can essentially present in all language.
- So any programming language is made up of 5 basic elements of the instructions.



Variables

Variables are containers for storing data values. In C#, there are different types of variables (defined with different keywords);

- **int** - stores integers (whole numbers), without decimals, such as 123 or -123
- **double** - stores floating point numbers, with decimals, such as 19.99 or -19.99
- **char** - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- **string** - stores text, such as "Hello World". String values are surrounded by double quotes
- **bool** - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Syntax

```
type variableName = value;
```

Where type is a C# type (such as int or string), and *variableName* is the name of the variable (such as **x** or **name**). The equal sign is used to assign values to the variable.

Declaring (Creating) Variables

To create a variable that should store text, look at the following example:

Example

Create a variable called **name** of type `string` and assign it the value "**John**":

```
string name = "John";
Console.WriteLine(name);
```

Declaring (Creating) Variables

To create a variable that should store a number, look at the following example:

Example

Create a variable called **myNum** of type `int` and assign it the value **15**:

```
int myNum = 15;
Console.WriteLine(myNum);
```

Declaring (Creating) Variables

You can also declare a variable without assigning the value, and assign the value later:

Example

```
int myNum;
myNum = 15;
Console.WriteLine(myNum);
```

Declaring (Creating) Variables

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

Change the value of `myNum` to 20:

```
int myNum = 15;
myNum = 20; // myNum is now 20
Console.WriteLine(myNum);
```

Declaring (Creating) Variables

A demonstration of how to declare variables of other types:

Example

```
int myNum = 5;
double myDoubleNum = 5.99D;
char myLetter = 'D';
bool myBool = true;
string myText = "Hello";
```

Declaring (Creating) Variables

If you don't want others (or yourself) to overwrite existing values, you can add the const keyword in front of the variable type.

Example

```
const int myNum = 15;
myNum = 20; // error
```

The const keyword is useful when you want a variable to always store the same value, so that others (or yourself) won't mess up your code. An example that is often referred to as a constant, is PI (3.14159...).

Note: You cannot declare a constant variable without assigning the value. If you do, an error will occur: A const field requires a value to be provided.

Displaying Variables

The WriteLine() method is often used to display variable values to the console window.

To combine both text and a variable, use the + character:

Example

```
string name = "John";
Console.WriteLine("Hello " + name);
```

Displaying Variables

You can also use the + character to add a variable to another variable:

Example

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName + lastName;
Console.WriteLine(fullName);
```

Displaying Variables

For numeric values, the + character works as a mathematical operator (notice that we use int (integer) variables here):

Example

```
int x = 5;
int y = 6;
Console.WriteLine(x + y); // Print the value of x + y
```

Declaring Multiple Variables

To declare more than one variable of the **same type**, use a comma-separated list:

Example

```
int x = 5, y = 6, z = 50;
Console.WriteLine(x + y + z);
```

Declaring Multiple Variables

You can also assign the same value to multiple variables in one line:

Example

```
int x, y, z;
x = y = z = 50;
Console.WriteLine(x + y + z);
```

Identifiers for Variables

- All C# variables must be identified with unique names.
- These unique names are called identifiers.
- Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Example

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

Identifiers for Variables

The general rules for naming variables are:

- Names can contain letters, digits and the underscore character (_)
- Names must begin with a letter
- Names should start with a lowercase letter and it cannot contain whitespace
- Names are case sensitive ("myVar" and "myvar" are different variables)
- Reserved words (like C# keywords, such as int or double) cannot be used as names

Data Types of Variables

As explained in the variables chapter, a variable in C# must be a specified data type:

Example

```
int myNum = 5;               // Integer (whole number)
double myDoubleNum = 5.99D;  // Floating point number
char myLetter = 'D';         // Character
bool myBool = true;          // Boolean
string myText = "Hello";     // String
```

Data Types of Variables

It is important to use the correct data type for the corresponding variable;

- To avoid errors,
- To save time and memory.
- To make your code more maintainable and readable.

Data Types of Variables

The most common data types are:

Data Type	Size	Description
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
bool	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter, surrounded by single quotes
string	2 bytes per character	Stores a sequence of characters, surrounded by double quotes

Numbers

Number types are divided into two groups:

- **Integer types** stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are int and long. Which type you should use, depends on the numeric value.

- **Floating point types** represents numbers with a fractional part, containing one or more decimals. Valid types are float and double.

Even though there are many numeric types in C#, the most used for numbers are int (for whole numbers) and double (for floating point numbers).

Integer Types - Int

The int data type can store whole numbers from -2147483648 to 2147483647. In general, and in our tutorial, the int data type is the preferred data type when we create variables with a numeric value.

Example

```
int myNum = 100000;
Console.WriteLine(myNum);
```

Integer Types - Long

The long data type can store whole numbers from -9223372036854775808 to 9223372036854775807. This is used when int is not large enough to store the value. Note that you should end the value with an "L":

Example

```
long myNum = 15000000000L;
Console.WriteLine(myNum);
```

Floating Point Types

You should use a floating point type whenever you need a number with a decimal, such as 9.99 or 3.14515. The float and double data types can store fractional numbers. Note that you should end the value with an "F" for floats:

Float Example

```
float myNum = 5.75F;
Console.WriteLine(myNum);
```

Double Example

```
double myNum = 19.99D;
Console.WriteLine(myNum);
```

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations:

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	x++
--	Decrement	Decreases the value of a variable by 1	x--

Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Modulus va Tegishli tushuntiruv (%=):

Misol:

```
int a = 20;
a %= 7; // a = a % 7; ga teng
// Natija: a = 6 (20 ni 7 ga bo'lishda qoldiq 6)
```

Bitwise AND va Tegishli tushuntiruv (&=):

Misol:

```
int b = 12;
b &= 5; // b = b & 5; ga teng
// Natija: b = 4 (12 ni 5 ga bitwise AND qilganda 4 hosil bo'ladi)
```

Bitwise OR va Tegishli tushuntiruv (|=):

Misol:

int c = 3;

c |= 8; // c = c | 8; ga teng

// Natija: c = 11 (3 ni 8 ga bitwise OR qilganda 11 hosil bo'ladi)

Bitwise XOR va Tegishli tushuntiruv (^=):

Misol:

int d = 15;

d ^= 7; // d = d ^ 7; ga teng

// Natija: d = 8 (15 ni 7 ga bitwise XOR qilganda 8 hosil bo'ladi)

Chapga o'girish va Tegishli tushuntiruv (<<=):

Misol:

int e = 4;

e <<= 3; // e = e << 3; ga teng

// Natija: e = 32 (4 ni 3 marta 2 ga o'sganda 32 hosil bo'ladi)

O'ngga o'girish va Tegishli tushuntiruv (>>=):

Misol:

int f = 64;

f >>= 2; // f = f >> 2; ga teng

// Natija: f = 16 (64 ni 2 marta 2 ga bittadan o'ngga o'girganda 16 hosil bo'ladi)

Comparison Operators

Comparison operators are used to compare two values:

Note: The return value of a comparison is either True or False.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Comparison Operators

Comparison operators are used to compare two values:

Note: The return value of a comparison is either True or False.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Comparison Operators

Comparison operators are used to compare two values:

Note: The return value of a comparison is either True or False.

Example

```
int x = 5;
int y = 3;
Console.WriteLine(x > y); // returns True because 5 is greater than 3
```

Logical Operators

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns True if both statements are true	x < 5 && x < 10
||	Logical or	Returns True if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns False if the result is true	!(x < 5 && x < 10)

Booleans

A boolean data type is declared with the bool keyword and can only take the values true or false:

Example

```
bool isCSharpFun = true;
bool isFishTasty = false;
Console.WriteLine(isCSharpFun);   // Outputs True
Console.WriteLine(isFishTasty);   // Outputs False
```

Boolean values are mostly used for conditional testing,

Boolean Expression

A Boolean expression is a an expression that returns a Boolean value: True or False. You can use a comparison operator, such as the **greater than** (>) operator to find out if an expression (or a variable) is true:

Example

```
int x = 10;
int y = 9;
Console.WriteLine(x > y); // returns True, because 10 is higher than 9
```

Boolean Expression

A Boolean expression is a an expression that returns a Boolean value: True or False. You can use a comparison operator, such as the **greater than** (>) operator to find out if an expression (or a variable) is true:

Example

```
int x = 10;
int y = 9;
Console.WriteLine(x > y); // returns True, because 10 is higher than 9
```

```
Console.WriteLine(10 > 9); // returns True, because 10 is higher than 9
```

Boolean Expression

In the examples below, we use the **equal to** (==) operator to evaluate an expression:

Example

```
int x = 10;
Console.WriteLine(x == 10); // returns True, because the value of x is equal to 10
```

```
Console.WriteLine(10 == 15); // returns False, because 10 is not equal to 15
```

Characters

The char data type is used to store a **single** character. The character must be surrounded by single quotes, like 'A' or 'c':

Example

```
char myGrade = 'B';
Console.WriteLine(myGrade);
```

String

The string data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:

Example

```
string greeting = "Hello World";
Console.WriteLine(greeting);
```

Next lecture

- In the next lecture we continue focusing on «Constructs and techniques and their implementation»
- “Microsoft Visual C# Step by Step” Microsoft Press