MIS124 Final Notes

Console.WriteLine("Name\nSurname");
Name
Surname

Console.WriteLine("My Age:\t25");
**My Age:	25**

Console.WriteLine("This sentence consist of a \"double quote character");
This sentence consist of a "double quote character

> string student_name; // variable declaration
> student_name = "zeki"; variable assignment
> int student_age = 39; both declaratio and assignment

wrong variable names
int 3name; // it mustn't start with a number
string $surName; //it mustn't start with a special character
string üğçöş; //correct because variables may consist of non-english characters

variable values can be changed later but constant values can't
const double pi = 3.14;
pi = 3.1418; // you can't change it later

uint number = 2112312; //**uint is unsigned integer data type**
int number = -1231231; //**int is integer data type that may have a sign**

Block Level Variable:

when you declare a variable inside a block "{}" then that variable can't be called outside that block. for example:

```
      { //OUTER BLOCK STARTS
                 int x = 1;
                 Console.WriteLine(x);
 {// INNER BLOCK STARTS
```

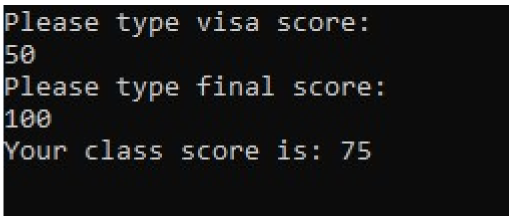
```
                int y = 5;
                Console.WriteLine(x);
                Console.WriteLine(y);
            }// INNER BLOCK ENDS
            Console.WriteLine(x);
            Console.WriteLine(y);

            {// INNER BLOCK2 STARTS
                Console.WriteLine(y);
            }

        }//OUTER BLOCK ENDS
```

this code will give an error because the y variable only exists in the inner block, when you use writeline outside that block it expects an "y" variable in the outer block or the inner block2.

A program for calculating class score

```
    Console.WriteLine("Please type visa score:");
            int visa_score = Int32.Parse( Console.ReadLine());
            Console.WriteLine("Please type final score:");
            int final_score = Int32.Parse(Console.ReadLine());

            double class_point = (visa_score + final_score) / 2;
            Console.WriteLine("Your class score is: " +
class_point);

            Console.ReadLine(); // to terminate the program (you
text somethings and program finishes)
```

output will be like this:



```
Please type visa score:
50
Please type final score:
100
Your class score is: 75
```

Double Divide //

whatever you write after double divide won't be executed as a program so you can put comments and notes. if you wanna give multi-line comment you ca use /* to open the comment section and */ to close it like this:

programming_things_etc
/*
*comment1
*comment2
*comment3
*/
programming_things_etc

C# is case sensitive

which means uppercase or lowercase makes difference (these are different variables: name, Name, nAme, NAME, nAmE...)

Console Methods

- Console.Beep(); // beep sound
- Console.Clear() // clears the text

Console Input

- Console.Read(); // reads one character(returns an int)

what we mean by returns an int for example if you enter something that is not integer the computer will save it as integer.
for example:

```
using System;

namespace ConsoleApp5
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int değişken = Console.Read();
```

```
                Console.WriteLine(değişken);

            }
        }
    }
```

if you enter "a" the console will write it as 97
if you enter "b" the console will write it as 98
because "Console.Read();" returns the input as integer.

- ConsoleReadkey();//reads only one key and shows it on console (returns a character)
- Console.Readline();//reads the whole line until typed ENTER key (returns a string) if you wanna enter a int data type you have to add Int32.Parse(); or double data type Double.Parse();
  for example:

  ```
  int st_number = Int32.Parse(Console.Readline());
  double any_number = Double.Parse(ConsoleReadline());
  ```

  .Parse will convert the data types from string to other things.

C# program location

In Windows, C# source code an program lacated in
C:\Users\your_user_name\source\repos\your_program_name\
Program.cs

In Windows, Your program lacated in
C:\Users\your_user_name\source\repos\your_program_name\bin\
Debug\net5.0\ your_program_name.exe

Console Outputs

- Console.Write(); //writes text to the console
- Console.WriteLine(); // writes the text and the things after this on console will be in a new line.

writing string

```
Console.WriteLine("Management Information Systems"); //this will
show whatever text is in quotation marks on the screen.
```

writing variable

```
string st_number = Console.ReadLine();
Console.WriteLine(st_number); // this will show the value of that
variable on the screen.
```

using string placeholder to write

single placeholder will be like:

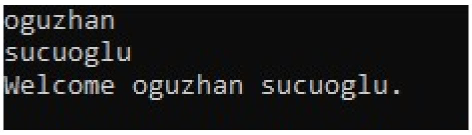
"string {indice}", variable
//indices start with 0

```
int st_number = Int32.Parse(Console.ReadLine());
Console.WriteLine("Your number is {0}.", st_number);
```

multiple placeholder will be like:

"string {indice1}, {indice2}, ", variable1, variable2);

```
string st_name = Console.ReadLine();
string st_surname = Console.ReadLine();
Console.WriteLine("Welcome {0} {1}.", st_name, st_surname);
```

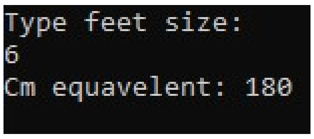
output:


```
oguzhan
sucuoglu
Welcome oguzhan sucuoglu.
```

A program for converting feet to cm

```
using System;

namespace calculate_feet_to_cm
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Type feet size:");

            double feet = Double.Parse(Console.ReadLine());

            double cm = feet * 30;

            Console.WriteLine("Cm equavelent: {0}", cm);
        }
    }
}
```

output:



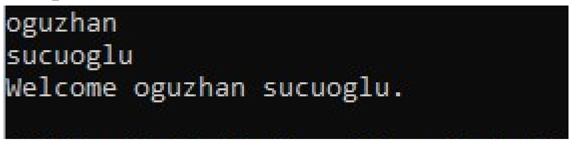
```
Type feet size:
6
Cm equavelent: 180
```

using string interpolation

The $ special character identifies a string literal as an interpolated string.
Structure:
$"String {variable1} {variable2}"
when writing variables, we use the name of the variable instead of the index. So we don't have to remember the order of variables. for example:

```
string st_name = Console.Readline();
string st_surname = Console.Readline();
Console.WriteLine($"Welcome {st_name} {st_surname}.");
```

output:



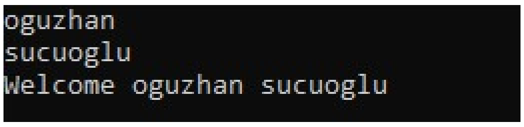
```
oguzhan
sucuoglu
Welcome oguzhan sucuoglu.
```

using string concatenation operator

"string " + variable;

```
                string st_name = Console.ReadLine();
                string st_surname = Console.ReadLine();
                Console.WriteLine("Welcome " + st_name + " " +
st_surname);
```

output:



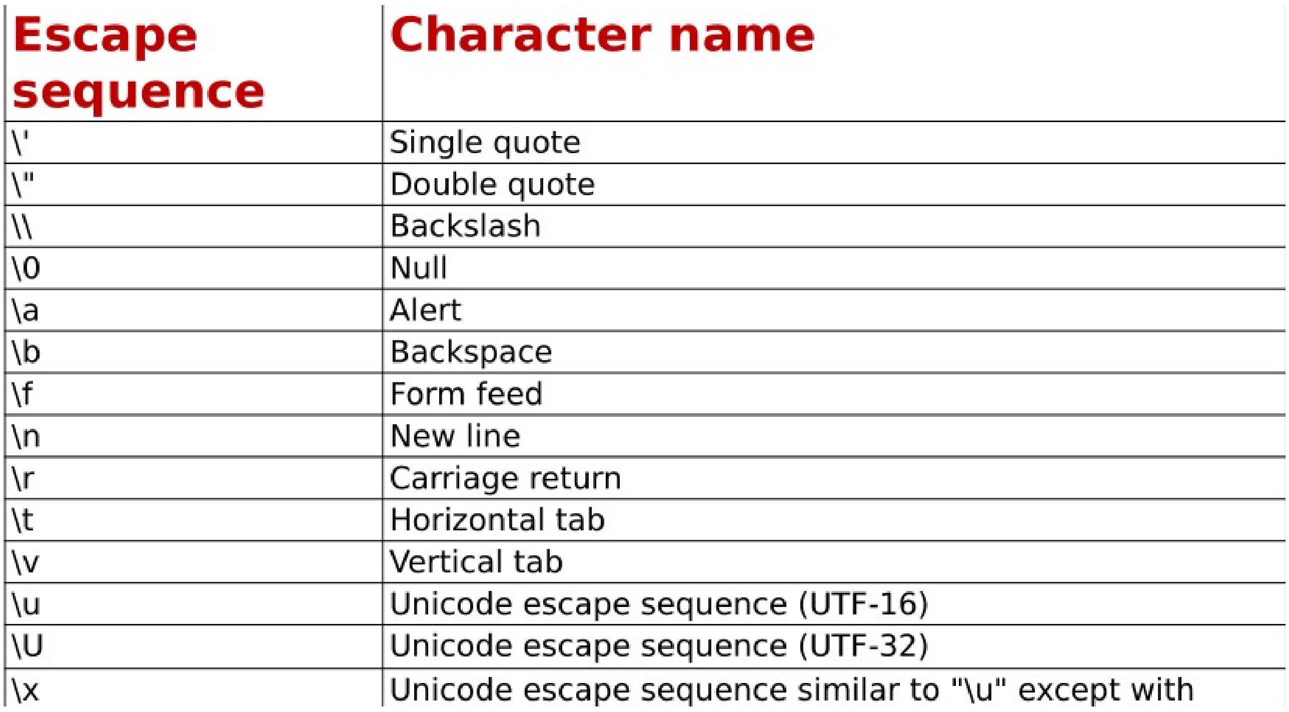
```
oguzhan
sucuoglu
Welcome oguzhan sucuoglu
```

Console Properties(Setting Functions)

- SetCursorPosition
- SetBufferSize
- SetWindowPosition
- SetWindowSize

C# Escape Characters

Escape characters are designed to ensure that special text in a string literal appears as it should. The most common usages are for single and double quotes, tabbed spaces, and new line characters. Every escape character begins with a backslash () and is followed by a value or token.

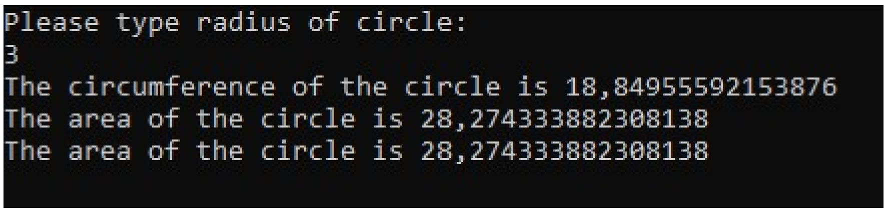


Escape sequence	Character name
\'	Single quote
\"	Double quote
\\	Backslash
\0	Null
\a	Alert
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\u	Unicode escape sequence (UTF-16)
\U	Unicode escape sequence (UTF-32)
\x	Unicode escape sequence similar to "\u" except with

A program for calculating circle

```
using System;

namespace circle_calculation
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Please type radius of circle:");
            int radius = Int32.Parse(Console.ReadLine());


            double circumference = 2 * Math.PI * radius;
            Console.WriteLine("The circumference of the circle is "
+ circumference);


            double area = Math.PI * radius * radius;
            // Math.Pow(base, exponent)
            double area2 = Math.PI * Math.Pow(radius, 2);
            Console.WriteLine("The area of the circle is " + area);
            Console.WriteLine("The area of the circle is " +
area2);


            Console.ReadLine();
        }
    }
}
```

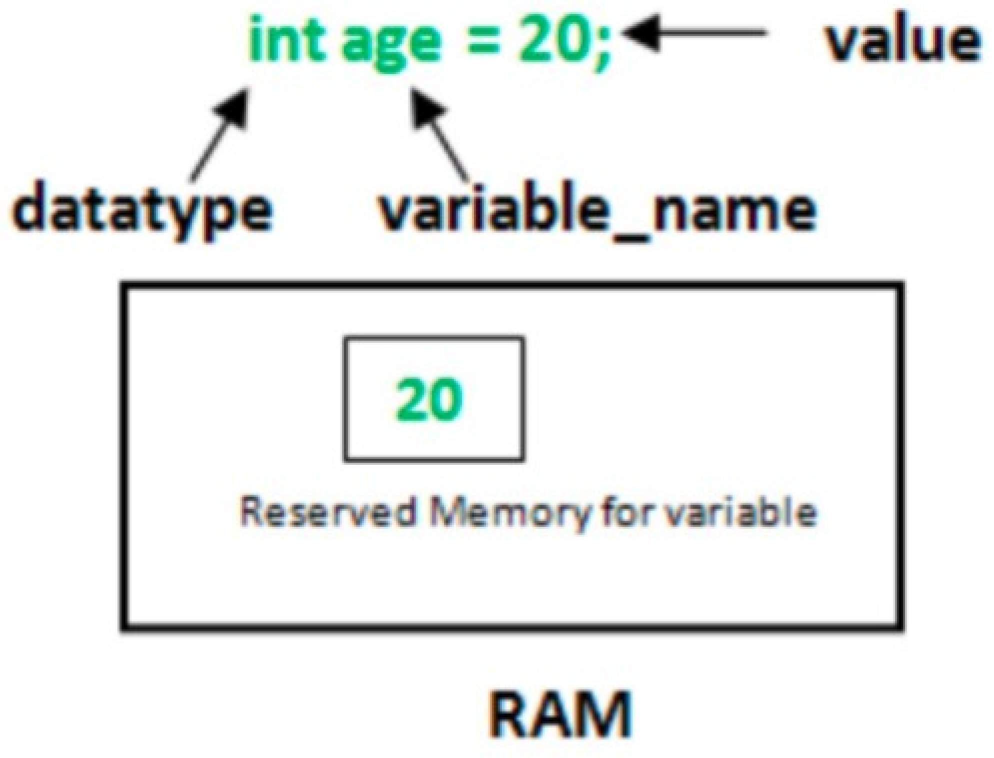
output:



```
Please type radius of circle:
3
The circumference of the circle is 18,84955592153876
The area of the circle is 28,274333882308138
The area of the circle is 28,274333882308138
```

Variables

They are the names you give to computer memory locations which are used to store values in a computer program.



Variable Structure

= ;

example:

```
string st_name = Console.ReadLine();
int st_number = 123456;
```

data type of the st_name variable is string which means it's a text and the value it gets will be the what the user types in the console.
st_number variable is an int type which is integer and value is 123456.

1. variable defining/declaring

;

example:

```
string st_name;
int st_number;
```

2. assigning a value to a variable

= value;

example:

```
st_name = "Zeki";
st_number = 123456;
```

3. defining a variable and assigning a value

= ;

example:

```
string st_name = "Zeki";
int st_number = 123456;
```

4. defining and assigning a value to multiple variable at once

More than one variable of the same data type can be defined at once.

example:

```
string st_name, st_surname, st_department;
int x = 10, y = 5;
```

Identifier

identifiers are the user-defined name of the program components. In C#, an identifier can be a class name, method name, variable name or label.
Example:

```
public class GFG {
    static public void Main ()
    {
        int x;
    }
}
```

There are 3 identifiers here: CFG (name of the class), Main (method name), x (variable name)

To use the following structures, we must give them a name: variable, constant, class, function, method, interface, struct, delegate, enum, member, namespace…

Identifier Naming Rules in C#

- The only allowed characters for identifiers are all alphanumeric characters(**[A-Z]**, **[a-z]**, **[0-9]**), '**_**' (underscore). For example "geek@" is not a valid C# identifier as it contain '@' – special character.
- Identifiers should not start with digits([0-9]). For example "123geeks" is a not a valid in C# identifier.
- Identifiers should not contain white spaces. (use underscore instead)
- Identifiers are not allowed to use as keyword unless they include @ as a prefix. For example, **@as** is a valid identifier, but "**as**" is not because it is a keyword.
- C# identifiers cannot contain more than 512 characters.
- C# identifiers allow Unicode Characters.
- C# identifiers are case-sensitive.
- Identifiers does not contain two consecutive underscores in its name because such types of identifiers are used for the implementation.

Naming conventions

There are some coding conventions:

- **camelCase:** first letter of the word always in lowercase and after that each word starts with uppercase (camelCase, oguzhanSucuoglu, oneTwoThreeFour, halicUniversityFirstYearStudents)
- **PascalCase:** every first letter of every word is uppercase. (PascalCase, OguzhanSucuoglu, HalicUniversityStudents)
- **_underScore:** start with underscore and the word after _ use camelCase terminology. (_underScore, _oguzhanSucuoglu, _halicUniversityStudents)

- **snake_case:** each word is lowercase with underscores separating words. (snake_case, oguzhan_sucuoglu, halic_university_students)

Constant

The value that never changes in the program. Constants are declared using the **const** keyword.
syntax:
const <data_type> <constant_name> = value;

```
const double pi = 3.14159;
```

Keywords

Keywords are predefined, reserved identifiers that have special meanings to the compiler. **They cannot be used as identifiers unless they inclue @ as a prefix.**
if is keyword. @if is a valid identifier.

Literals

All values and texts given for variables or outputs are called literals. Literal is a value that is used by the variables. The data we request from the user and the value we output to the screen are literals. Literals can be of the following types:

- integer
- floating-point
- character
- string
- null
- boolean

Integer Literals

Decimal Literals (Base 10): in the decimal type of literals 0-9 digits are allowed. No prefix is required for the decimal type of literals.

```
int x = 100; // decimal type
```

Octal Literals (Base 8): in the octal type of literals 0-7 digits are allowed. 0 is used as a prefix to specify the form of octal type literals.

```
int y = 072; // octal type
```

Hexa-decimal literals (base 16): digits from 0-9 and characters from A-f are allowed. Uppercase and lowercase both types of characters are allowed in this case. 0X or 0x is used as a prefix to specify the form of hexadecimal type of literals.

```
int z = 0x123f; // hexadecimal type
```

Binary literals (base 2): the allowed digits are only 1's and 0's. Binary number should be prefix with 0b.

```
int binary = 0b101;
```

Double literals: the f or F letter can be used in float literal.

```
float f = 3.8F;
```

The d or D letter can be used in double literal.

```
double d = -7.6D;
```

The m or M letter can be used in decimal literal.

```
decimal decimal_number = 4.5M; //decimal type
```

The underscore character is used to make it easier to read.

```
int million = 1_000_000;
```

output: 1000000

scientific notation: the e or E letter to indicate the power of 10 as exponent part of scientific notation with float, double or decimal. for example scientific = 3.8 * 10^5 would be like this:

```
double scientific = 3.8E+5;
```

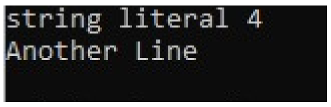
character literals: we use single quote literal ' ' to assign a value to char data type.

```
char ch = 'Z';
```

string literals: double-quote " "symbol as string literals. also add @ symbol for multiline string literals.

```
string department = "MIS";
```

```
string two_lines_text = @"string literal 4
Another Line";
```



```
string literal 4
Another Line
```

boolean literals
there are only two boolean literals.
true and **false**

```
bool b1 = true;
bool b2 = false;
```

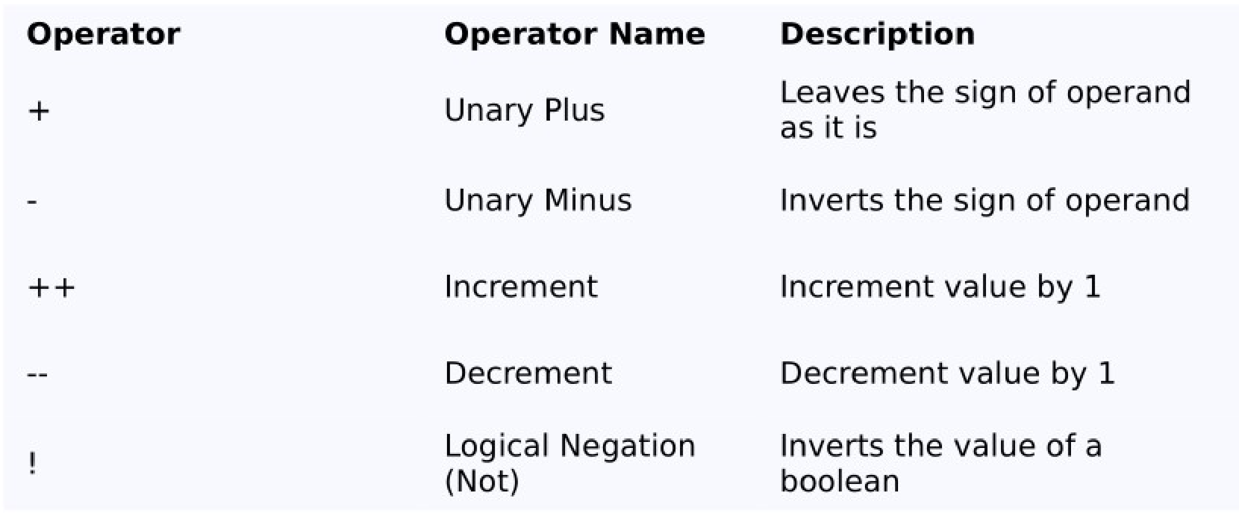
Operators and Operand

- operators allow us to perform different kinds of operations on operands.
- addition "+" operator adds two operands x+y
- x=3+5

  3 and 5 are operand. + is addition operator. = assignment operator.

assignment operator "="

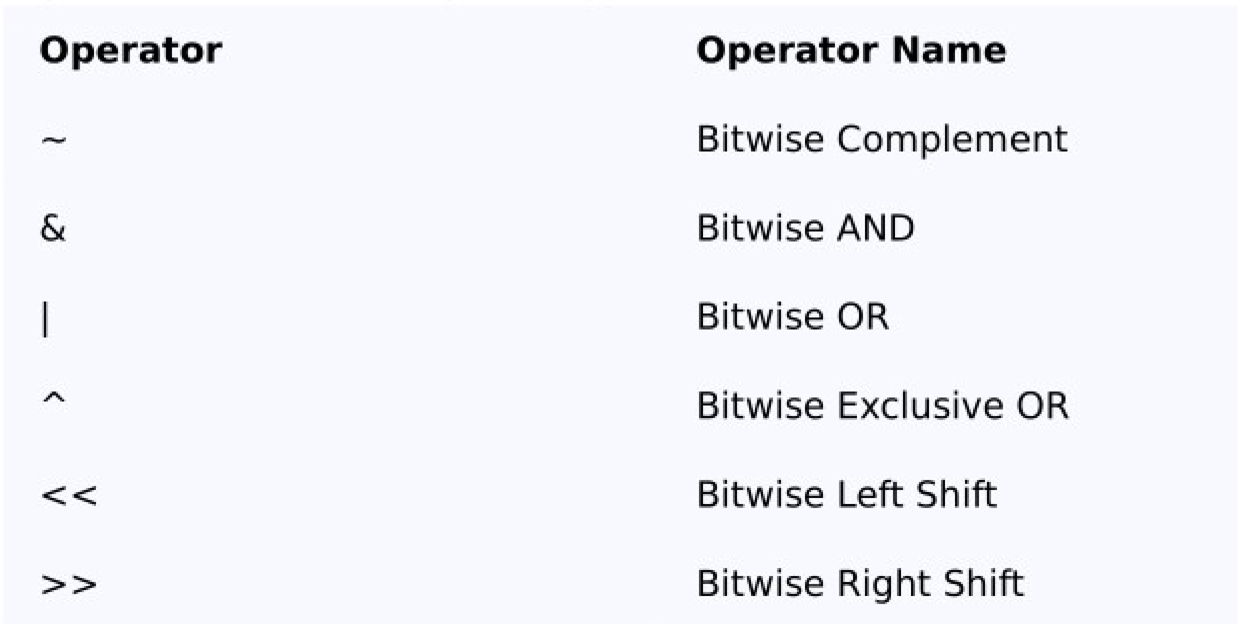
UNARY OPERATORS



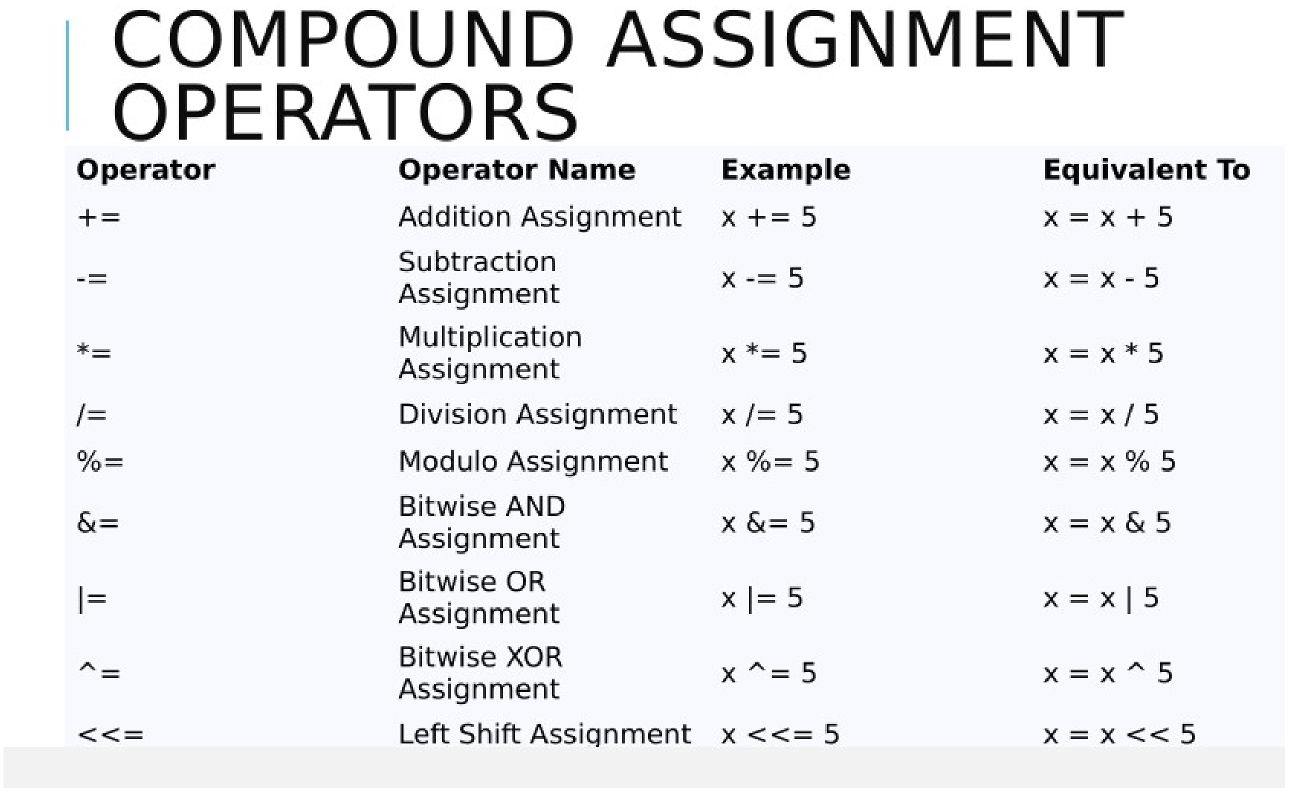
Operator	Operator Name	Description
+	Unary Plus	Leaves the sign of operand as it is
-	Unary Minus	Inverts the sign of operand
++	Increment	Increment value by 1
--	Decrement	Decrement value by 1
!	Logical Negation (Not)	Inverts the value of a boolean

only needs one operand

BITWISE AND BIT SHIFT OPERATORS

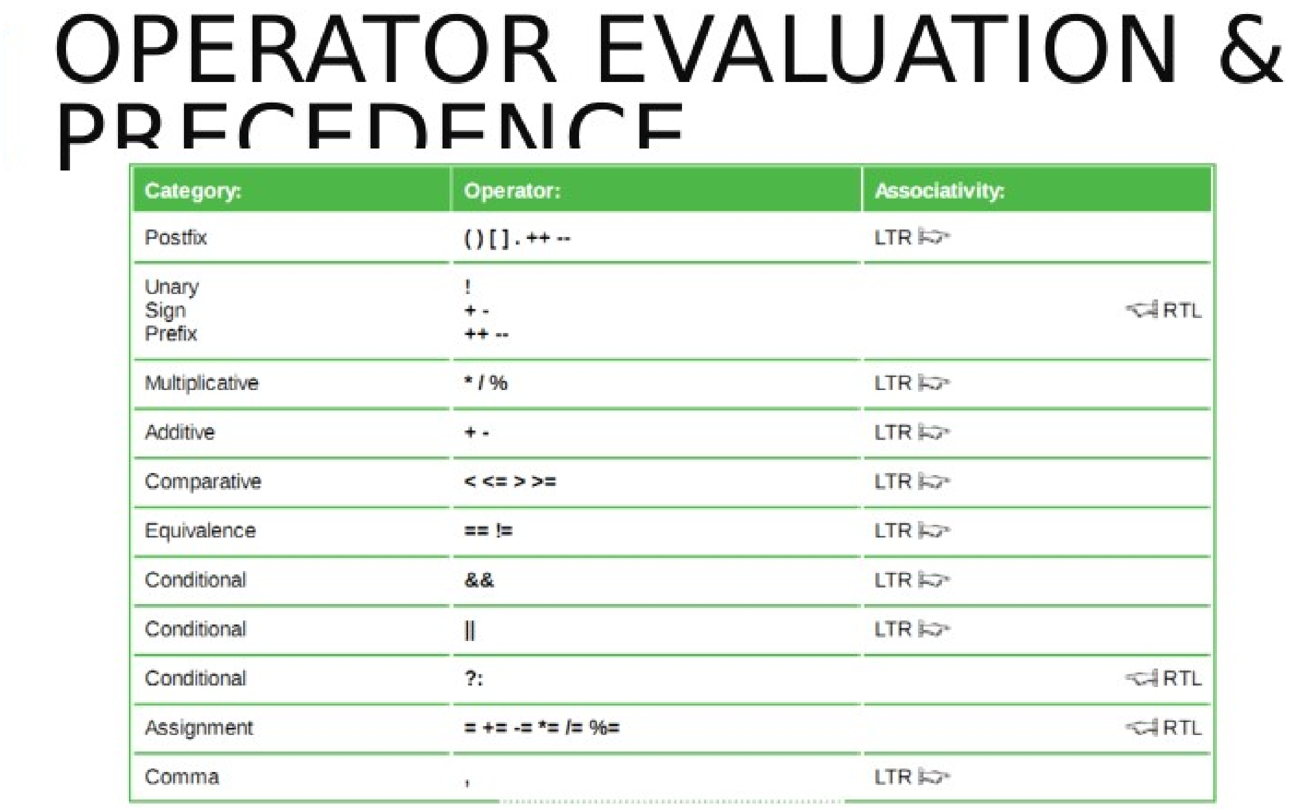


Operator	Operator Name
~	Bitwise Complement
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
<<	Bitwise Left Shift
>>	Bitwise Right Shift



COMPOUND ASSIGNMENT OPERATORS

Operator	Operator Name	Example	Equivalent To
+=	Addition Assignment	x += 5	x = x + 5
-=	Subtraction Assignment	x -= 5	x = x - 5
*=	Multiplication Assignment	x *= 5	x = x * 5
/=	Division Assignment	x /= 5	x = x / 5
%=	Modulo Assignment	x %= 5	x = x % 5
&=	Bitwise AND Assignment	x &= 5	x = x & 5
|=	Bitwise OR Assignment	x |= 5	x = x | 5
^=	Bitwise XOR Assignment	x ^= 5	x = x ^ 5
<<=	Left Shift Assignment	x <<= 5	x = x << 5



OPERATOR EVALUATION & PRECEDENCE

Category:	Operator:	Associativity:
Postfix	( ) [ ] . ++ --	LTR ☞
Unary Sign Prefix	! + - ++ --	☜ RTL
Multiplicative	* / %	LTR ☞
Additive	+ -	LTR ☞
Comparative	< <= > >=	LTR ☞
Equivalence	== !=	LTR ☞
Conditional	&&	LTR ☞
Conditional	||	LTR ☞
Conditional	?:	☜ RTL
Assignment	= += -= *= /= %=	☜ RTL
Comma	,	LTR ☞

OPERATOR PRECEDENCE

left to right

100 + 25 +35 = (100 + 25) + 35

45 – 3 + 9 = (45 – 3) + 9 // - is minus operator

right to left
100 * -25 = 100 * (-25) // - is sign operator
a = b = 100; Firstly b = 100 then a = 100

don't confuse additive operators and sign operators like + and -

Data Types

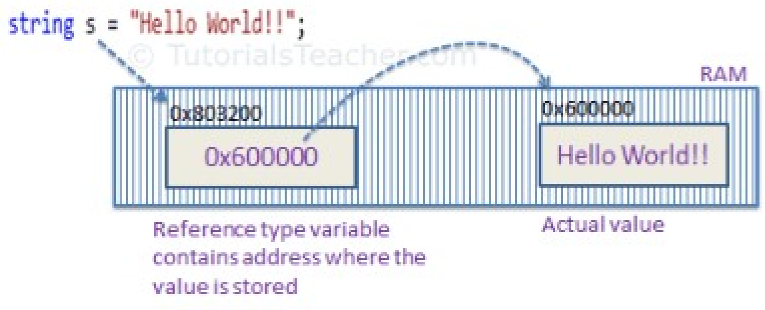
value types

a data type is a value type if it holds a data value within it's own memory space. variables of these data types directly contain values.

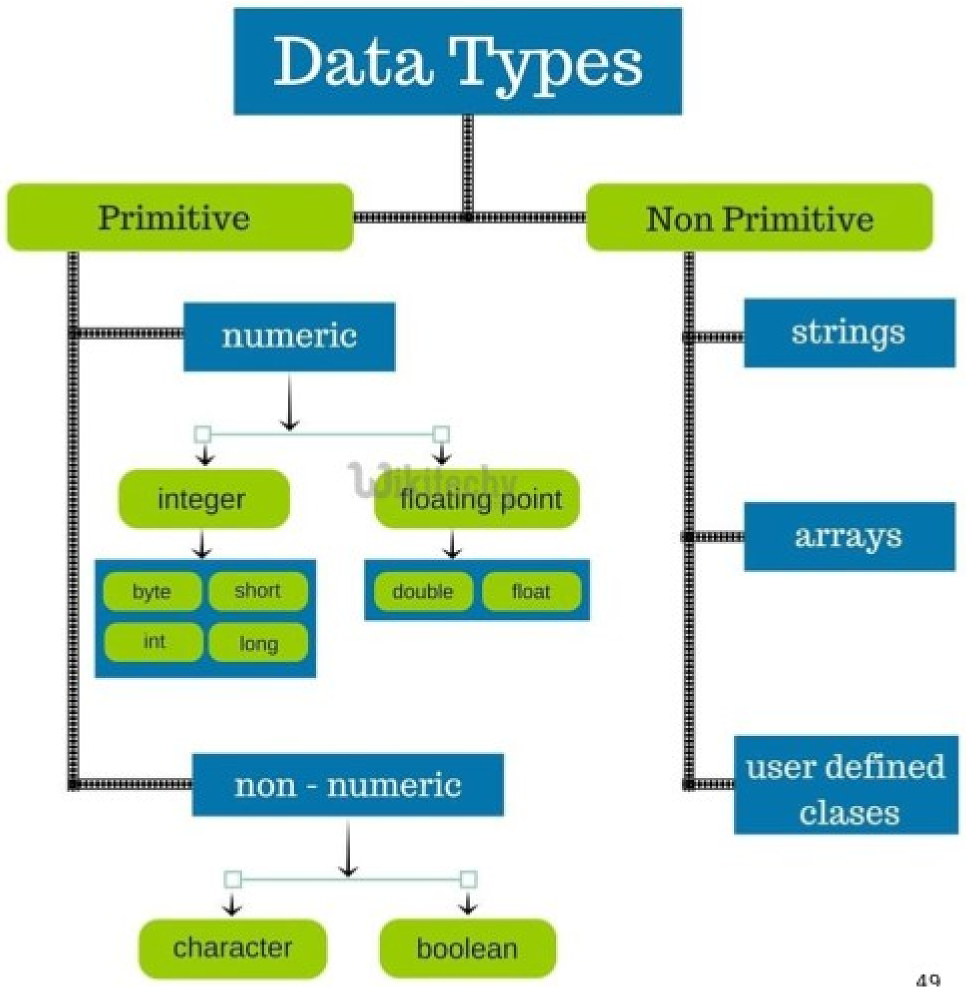
```
int i = 100;
```

reference types

unlike value types, a reference type doesn't store it's value directly. It stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data.


string s = "Hello World!!";
© TutorialsTeacher.com
RAM
0x803200
0x600000
0x600000
Hello World!!
Reference type variable contains address where the value is stored
Actual value

data types maximum minimum values and homework

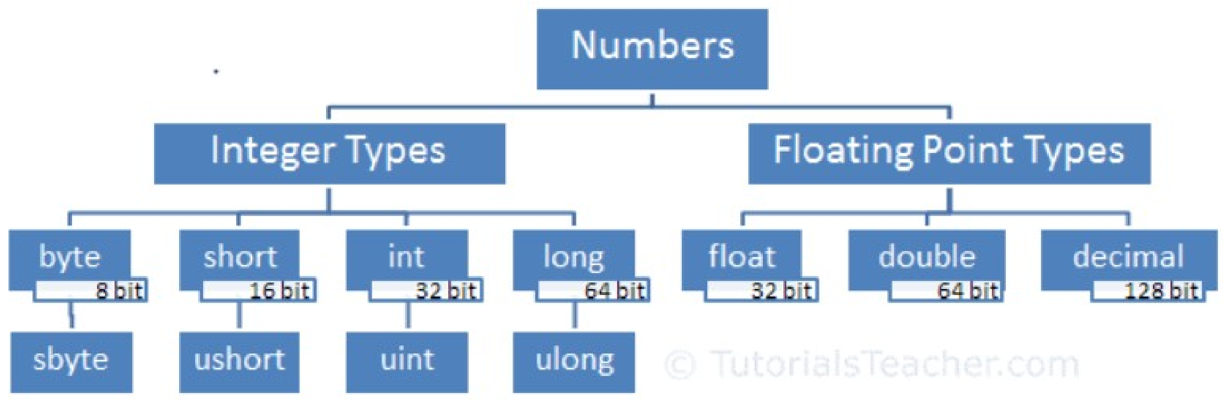


HOMEWORK2 – WEEK2

1. What are the differences between integer, decimal, and floating-point numbers? Explain and exemplify.

2. Explain the Floating-Point arithmetic.

3. Why does the value of numbers depend on the processor? What are the non-processor bound C# data types?



DATA TYPES AND THEIR DEFAULT, MINIMUM AND MAXIMUM VALUES

Data Types	Size (Bytes)	Default Value	Minimum Value	Maximum Value
`sbyte`	8-bit signed integer	0	-128	127
`byte`	8-bit unsigned integer	0	0	255
`short`	16-bit signed integer	0	-32768	32767
`ushort`	16-bit unsigned integer	0	0	65535
`int`	32-bit signed integer	0	-2147483648	2147483647
`uint`	32-bit unsigned integer	0u	0	4294967295

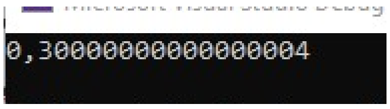
Data Types	Size (Bytes)	Default Value	Minimum Value	Maximum Value
`long`	64-bit signed type	0L	-9223372036854775808	9223372036854775807
`ulong`	64-bit unsigned type	0u	0	18446744073709551615
`float`	32-bit real floating-point type	0.0f	$\pm 1.5\times10^{-45}$	$\pm 3.4\times10^{38}$
`double`	64-bit real floating-point type	0.0d	$\pm 5.0\times10^{-324}$	$\pm 1.7\times10^{308}$

64-bit unsigned type means that the highest value is 2^64

Data Types	Size (Bytes)	Default Value	Minimum Value	Maximum Value
`decimal`	128-bit	0.0m	$\pm 1.0\times10^{-28}$	$\pm 7.9\times10^{28}$
`bool`		False	Two possible values: **`true`** and **`false`**	
`char`		'\u0000'	'\u0000'	'\uffff'
`object`		null	-	-
`string`		null	-	-

number precision at aritmetic operations

```
double sum = 0.1 + 0.2;
Console.WriteLine(sum);
```

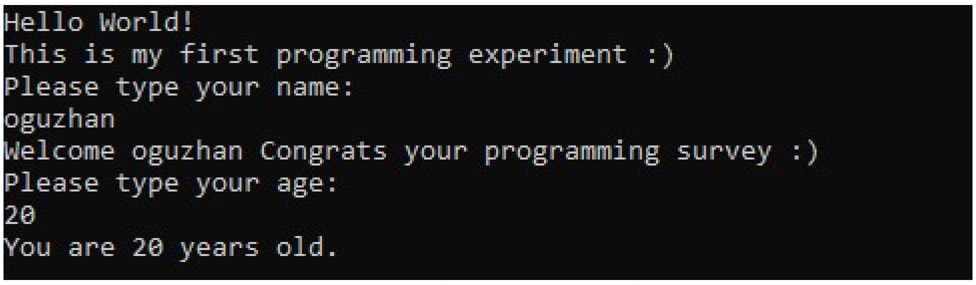


```
0,30000000000000004
```

Some programs

hello world

```
using System;

namespace Helloworld3
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.WriteLine("This is my first programming
experiment :)");

            Console.WriteLine("Please type your name:");
            string name = Console.ReadLine();
            Console.WriteLine("Welcome " + name + " Congrats your
programming survey :)");

            Console.WriteLine("Please type your age:");
            int age = Int32.Parse(Console.ReadLine());

            Console.WriteLine("You are " + age + " years old.");
        }
    }
}
```



```
Hello World!
This is my first programming experiment :)
Please type your name:
oguzhan
Welcome oguzhan Congrats your programming survey :)
Please type your age:
20
You are 20 years old.
```

Int32.Parse()

Console.Read(); expects you to input string when you enter int it gives an error like this:

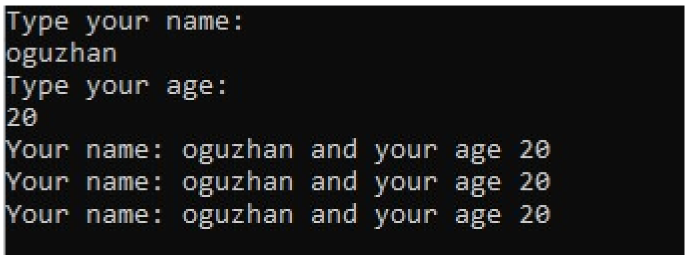
```
using System;

namespace console_read
{
    class Program
    {
        static void Main(string[] args)
        {
            int number = Console.Read();
            Console.WriteLine("You typed: " + number); // It prints
integer value as ASCII

            string character = Console.Read(); // It throws an
error
            Console.WriteLine("You typed: " + character);
        }
    }
}
```

so you should put Int32.Parse() before Console.Read() in order to input integer

```
using System;

namespace console_read
{
    class Program
    {
        static void Main(string[] args)
        {
            int number = Int32.Parse(Console.Read());
            Console.WriteLine("You typed: " + number);

            string character = Int32.Parse(Console.Read();
            Console.WriteLine("You typed: " + character);
        }
    }
                                        }
```

printing methods

```
using System;

namespace printing_methods
{
    internal class Program
    {
        static void Main(string[] args)
        {

            Console.WriteLine("Type your name:");
            string name = Console.ReadLine();
            Console.WriteLine("Type your age:");
            int age = Int32.Parse(Console.ReadLine());

            // printing two variables at once
            // we use + operator to concatanate string and
variabless
            Console.WriteLine("Your name: " + name + " and your age
" + age);

            // another method to printout something using
placeholder
            Console.WriteLine("Your name: {0} and your age {1}",
name, age);

            // this is the last method to printout using $ sign and
variable placeholder
            Console.WriteLine($"Your name: {name} and your age
{age}", name, age);

        }
    }
}
```



```
Type your name:
oguzhan
Type your age:
20
Your name: oguzhan and your age 20
Your name: oguzhan and your age 20
Your name: oguzhan and your age 20
```

printing variables

```
using System;

namespace printing_variables
{
    internal class Program
    {
        static void Main(string[] args)
        {
            // today we are going to learn print and input
functions
            // this code prints my name and age.
            // double divide operator puts one-line comment.
            Console.WriteLine("Type your name:");

            // C# is a case-sensitive language
            // it means that name and NAME are different variables

            // in comment line you can type non-english characters
            // just like üiçğşö
            string name = Console.ReadLine();
            string NAME = Console.ReadLine();




            Console.Write("Your name: ");
            Console.WriteLine(name);
            /*
             * this is
             * multi-line
             * comments block
             */
            Console.WriteLine("Type your age:");
            int age = Int32.Parse( Console.ReadLine());
            Console.Write("Your age: ");
            Console.WriteLine(age);

        }
    }
}
```

sphere volume calculation

```
using System;

namespace sphere_calculation
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Please type radius of sphere:");
            int radius = Int32.Parse(Console.ReadLine());

            double area = 2 * Math.PI * radius * radius;
            // Math.Pow(base, exponent)
            double area2 = 2 * Math.PI * Math.Pow(radius, 2);
            Console.WriteLine("The area of the sphere is " + area);
            Console.WriteLine("The area of the sphere is " +
area2);

            double volume = (4 / 3) * Math.PI * Math.Pow(radius,
3);
            Console.WriteLine("The volume of the sphere is " +
volume);

            Console.ReadLine();
        }
    }
}
```

volme of the cone

```
using System;

namespace volume_of_the_cone
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("This program calculates the volume 
of the cone.");

            Console.WriteLine("Please type the radius value of the 
Cone:");
            double radius = Double.Parse(Console.ReadLine());

            Console.WriteLine("Please type the height value of the 
Cone:");
            double h = Double.Parse(Console.ReadLine());

            // Calculation the volume of the Cone
            double volume = (Math.PI * radius * radius * h) / 3;
            //double volume2 = (Math.PI * Math.Pow(radius, 2) * h) 
/ 3;
            Console.WriteLine("The volume of the cone is " + 
volume);

            Console.ReadLine();
        }
    }
}
```

Variable Scopes

1. class-level scope

these are called fields or class members. Declared in front of all methods. Can be accessed by the non-static methods of the class. Can be accessed outside the class using by using access modifiers.

```
using System;

class student
{

    public static int student_id = 1;              //class level
variable
    public static string student_name = "Test";    //class level
variable

    public static void Main()
    {
        // accessing class level variables
        Console.WriteLine(student_id);
        Console.WriteLine(student_name);

        student_name = "Test2"; // class variables can be modified
later
        Console.WriteLine(student_name);

    }

    public void A()
    {
        // class level variables are visible in the entire class
        Console.WriteLine(student_id);
        Console.WriteLine(student_name);
    }

}
```

2. method-level scope

are not accessible outside the method. don't exist after method's execution is over.
example:

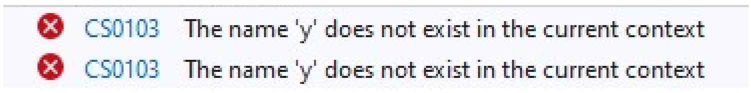
```
using System;

namespace method_level_variable
{
    class Program
    {
        public void A()
        {
            int number_A = 5; //this variable type is method-level
variable
            Console.WriteLine(number_A);

            // number_B variable is only visible in method B
            Console.WriteLine(number_B);
        }

        public void B()
        {
            int number_B = 10; //this variable type is method-level
variable
            Console.WriteLine(number_B);

            // number_A variable is only visible in method A
            Console.WriteLine(number_A);
        }

        static void Main(string[] args)
        {
            // number_A variable is only visible in method A
            Console.WriteLine(number_A);
            // number_B variable is only visible in method B
            Console.WriteLine(number_B);
        }
    }
}
```

3. block-level scope

a variable is only visible in the block in which it is defined.

```
using System;

namespace block_level_variable
{
    internal class Program
    {
        static void Main(string[] args)
        {
            { //OUTER BLOCK STARTS
                int x = 1;
                Console.WriteLine(x);

                {// INNER BLOCK STARTS
                    int y = 5;
                    Console.WriteLine(x);
                    Console.WriteLine(y);
                }// INNER BLOCK ENDS
                Console.WriteLine(x);
                Console.WriteLine(y);

                {// INNER BLOCK2 STARTS
                    Console.WriteLine(y);
                }

            }//OUTER BLOCK ENDS


        }
    }
}
```

when you try to run this code it'll give the following errors:



```
CS0103  The name 'y' does not exist in the current context
CS0103  The name 'y' does not exist in the current context
```

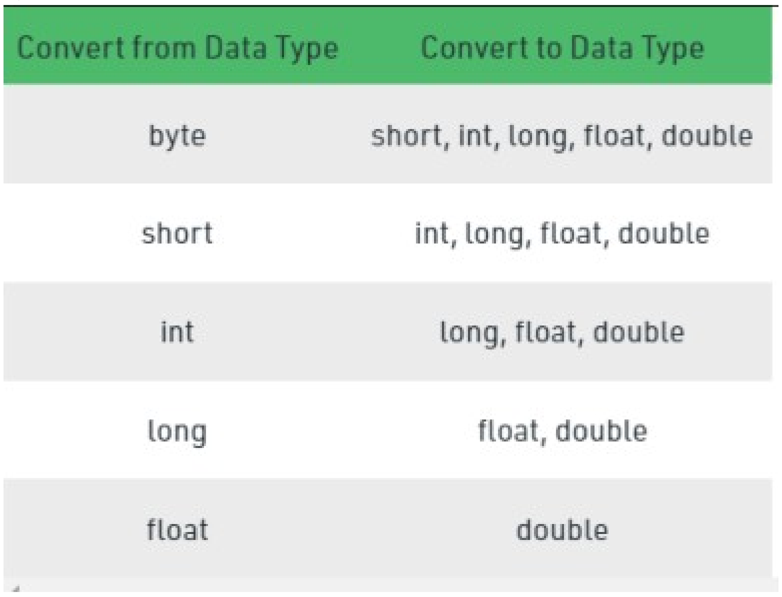
variable types

Global variable: accesible anywhere.
Local variable: accesible only it's defined block.

Built-in data type casting

Implicit conversion

does only occurs between compatible datatypes. (for example between integers). Compiler automatically performs but conversion can be only smaller data types to bigger data types.



Convert from Data Type	Convert to Data Type
byte	short, int, long, float, double
short	int, long, float, double
int	long, float, double
long	float, double
float	double

example:

```
using System;

namespace implicit_data_type_conversion
{
    class Program
    {
        static void Main(string[] args)
        {
            byte number = 18;
            Console.WriteLine("byte number =" + number);
            /*
             * The following conversion will be done automatically
             * by the compiler successfully.
             * Because the data type conversion is made
             * from the byte data type with a small number range
             * to the short data type with a large number range.
             * This conversion named as Implicit Data Type
Conversion.
            */
            short number2 = number;
            Console.WriteLine("short number =" + number2);



            short number3 = 18;
            Console.WriteLine("short number =" + number3);
            /*
             * The following conversion will not be possible.
             * Because data type conversion is made
             * from short data type with large number range
             * to byte data type with small number range.
             * so the compiler will throw an error.
            */
            byte number4 = number3;
            Console.WriteLine("byte number =" + number4);
        }
    }
}
```

Explicit conversion

between incompatible datatypes. compiler doesn't automatically performs. we should command to the compiler using some type conversion keywords such as (int) (double) (long) etc.

explicitly type conversion syntax
datatype1 var1 = value1; example: decimal x= 123.5M;
datatype var2 = (datatype2) var1; example: int y = (int) x;

example:

```
using System;

namespace explicitly_conversion_3
{
    class Program
    {
        static void Main(string[] args)
        {

            Console.WriteLine("Converting data type small number
range to large number range:");
            short number = -32767;
            Console.WriteLine("number=" + number);
            /*
             * We can convert data types explicitly
             * by typing data type inside parenthesis.
            */
            int number2 = (int)number;
            Console.WriteLine("number2=" + number2);




            Console.WriteLine();
            Console.WriteLine("Converting data type decimal numbers
to integers:");
            double number3 = -9898.45;
            Console.WriteLine("number3=" + number3);
            /*
             * But when converting from decimal numbers to
integers,
             * the decimal part of the number will be lost.
            */
            int number4 = (int)number3;
            Console.WriteLine("number4=" + number4);



            Console.WriteLine();
            Console.WriteLine("Converting data type large number
range to small number range:");
            ulong number5 = 9898454558798766646;
            Console.WriteLine("number5=" + number5);
            /*
             * Conversion from large number range data type
             * to small number range data type
             * will not preserve the number as desired.
            */
            short number6 = (short)number5;
            Console.WriteLine("number6=" + number6);

            Console.ReadLine();
        }
    }
}
```

floating point data types

double/float

mathematical functions

decimal

monetary calculations

incrementing and decrementing

pre-increment operator
int a = 3;
int b = ++a; // both a and b will now be 4

post-increment operator
int c = 3;
int d = c++; // the original value of 3 is assigned to "d", while "c" is now 4

a++ means "give me the value in a, then increment a," while ++a means, "increment a first, then give me the resulting value"

int a = 3;
a++;
int b = a;

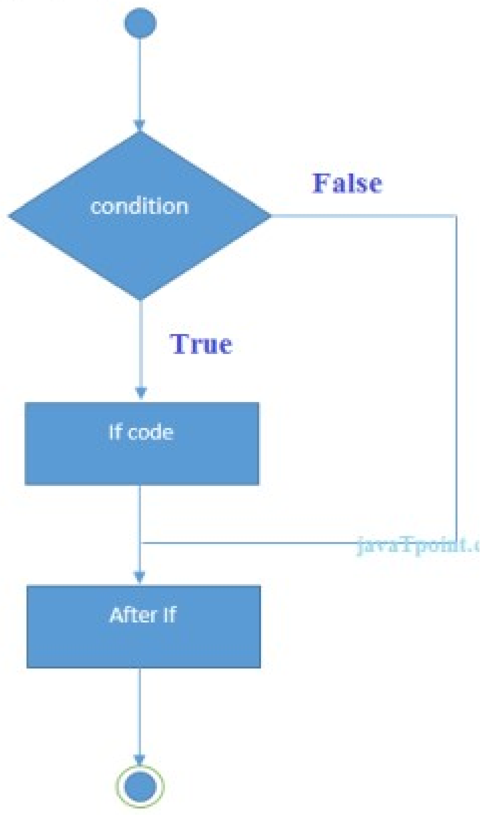
```
int c = 3;
int d = c;
c++;
```

Conditional Structures

the "IF" statement

if the condition is true then statement executes; otherwise is skipped (isn't executed)
the general IF syntax:

```
if (condition){
statement;
}
```



if the condition is false statement isn't executed and code flow continues next statement (other_statement).

the general IF syntax:

```
if (condition){
statement;
}
other_statement;
```

we don't have to use curly braces when there is only one line statement.

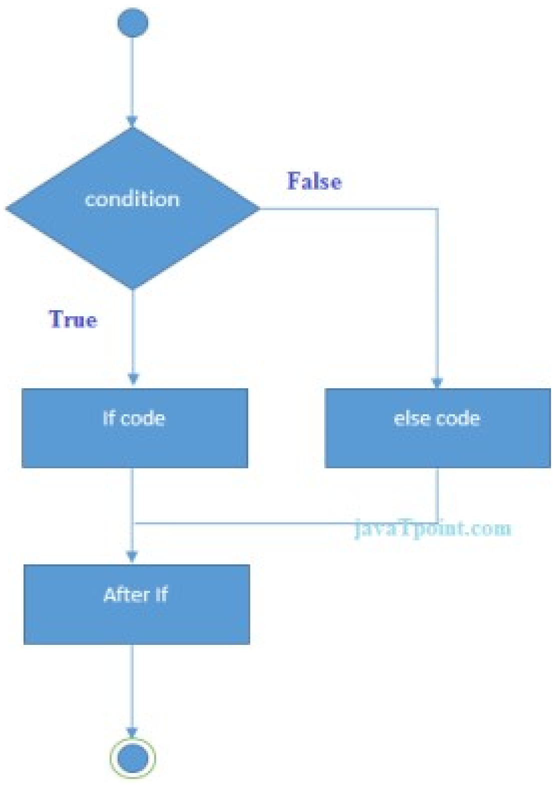
if (condition)
statement;

if(condition)
{
statement1;
statement2;
}
else
{
statement3;
statement4;
}

the "else" statement

If "if" condition is not true, compiler executes the "else" block.

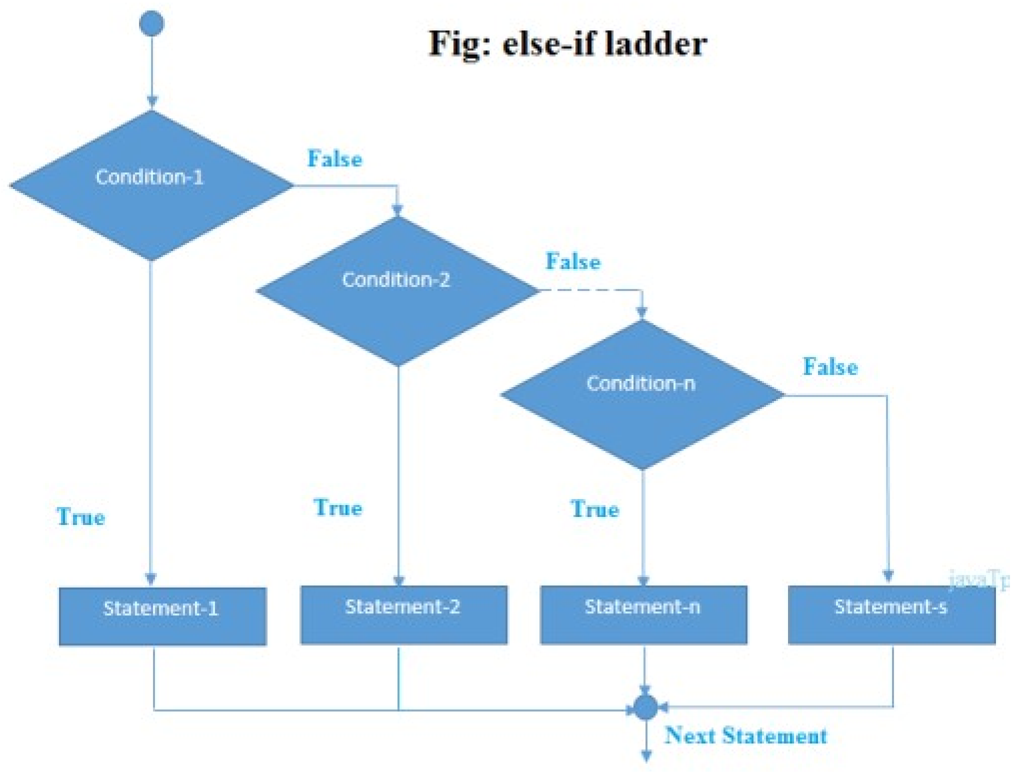
```
if(condition){
statement sequence1
}
else{
statement sequence2
}
```



"else if" statements

```
if(condition){
statement sequence
}
else if {
statement sequence
}
else if {
statement sequence
}
else {
statement sequence
}
```

Fig: else-if ladder



example: to see if number is negative or positive

```
Console.WriteLine("Please type a number:");
int number = Int32.Parse( Console.ReadLine());

if (number > 0)
{
    Console.WriteLine("The number is positive.");
}
else if (number < 0)
{
    Console.WriteLine("The number is negative.");
}
else if (number == 0)
    Console.WriteLine("The number is zero");
```

example:

```
Console.WriteLine("Type x value:");
double x = Double.Parse(Console.ReadLine());
int y = 2;
double fx = Math.Pow(x, 2) - 2 * x;

if (fx < 0)
    y = -1;
```

```
            else if (fx == 0)
                y = 0;
            else if (fx > 0)
                y = 1;
            Console.WriteLine("fx = {0}", fx);
            Console.WriteLine("y value: {0}", y);
            Console.ReadKey();
```

if-else if-else statements

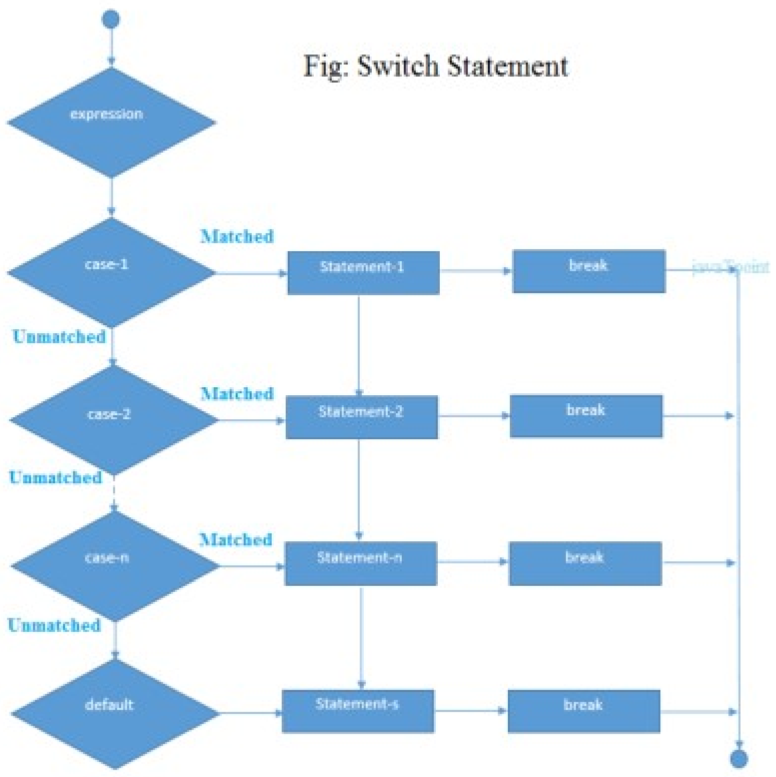
```
if(condition){
statement sequence
}
else if {
statement sequence
}
else if {
statement sequence
}
else {
statement sequence
}
```

nested ifs

```
if(condition){
        if(condition) {
    statement sequence
        }
statement sequence
}
```

switch statement

```
switch (variable-name)
{
case value1 : statement; break;
case value2 : statement; break;
case value3: statement; break;
default: statement;
}
```



how does switch works?
the switch expression is evaluated once
the value of the expression is compared with the values of each case
if there is a match, the associated block of code is executed.

some switch examples:
example: checks if the number is even or odd.

```
            Console.WriteLine("Please type a number:");
            int number = Int32.Parse(Console.ReadLine());
            int number_status = Math.Abs(number % 2);
            switch (number_status)
            {
                case 0:
                    Console.WriteLine("{0} is even number.",
number);
                    break;
                case 1:
                    Console.WriteLine("{0} is odd number.",
number);
                    break;
                default:
                    Console.WriteLine("Unknown number.");
                    break;
```

example: checks if it's a weekday or weekend (when you add .ToLower() after ReadLine() it will convert the input text to all lowercase)

```
            Console.WriteLine("Please type a day name:");
            string day = Console.ReadLine().ToLower();
            switch (day)
            {
                case "monday":
                case "tuesday":
                case "wednesday":
                case "thursday":
                case "friday":
                    Console.WriteLine("{0} is a weekday.", day);
                    break;
                case "saturday":
                case "sunday":
                    Console.WriteLine("{0} is a weekend.", day);
                    break;
                default:
                    Console.WriteLine("Unknown day!");
                    break;

                    /* a way to write this code without switch case
is this:
            Console.WriteLine("Please type a day name!");
            string day_name = Console.ReadLine().ToLower();

            if (day_name == "monday" || day_name == "thuesday" ||
day_name == "wednesday" || day_name == "thursday" || day_name ==
"friday")
            {
                Console.WriteLine(day_name + " is a week day.");
            } else if (day_name == "saturday" || day_name ==
"sunday")
```

```
                {
                    Console.WriteLine(day_name + " is a weekend day.");
                } else
                {
                    Console.WriteLine("Please type a validate day
name!");
                }
                        */
```

the break keyword

when c# reaches a break keyword, it breaks out of the switch block. Stops the execution of more code and case testing inside the block. when a match is found, and the job is done, it's time for a break. there is no need for more testing.

default keyword

the default keyword is optional and specifies some code to run if there is no case match.

```
switch (variable-name)
{
case value1 : statement ; break
;
case value2 : statement ; break
;
case value3 : statement ; break
;
default : statement ;
}
```

the GOTO keyword

jumps the another case clause

```
switch (score)
    {
        case 95:
            Console.WriteLine ("BA");
            break;
        case 98;
            goto case 95;
        default:
            console.WriteLine ("FF");
            break;
}
```

using multiple label in switch

```
switch (variable)
    {
        case value1:
        case value2:
            statement1;
            break;
        case value3:
        case value4:
    statement2;
            break;
    default:
            statement3;
            break;
}
```

good practices when using "switch-case"

- default keyword must be at the end
- first cases must be the common situations
- case labels should be in ascending order or alphabetical order.

Ternary Operator (if-else syntax)

variable = (condition) ? expressiontrue : expressionFalse;

example: shows if you passed or failed the class according your score (ternary operator replaces the long if statement code with a short line)

```
            int score = Int32.Parse(Console.ReadLine());
            string result;
            /*
            if (score < 50)
            {
                result = "You failed.";
            }
            else
            {
                result = "You passed.";
            }
            */
            // this is ternary operator
            // variable = (condition) ? true_statement :
false_statement;
            result = (score < 50) ? "You failed." : "You passed.";
            Console.WriteLine(result);
```

Loop Structures

- used for repeated statements.
- repeated code fragment until given condition is true or a fixed number of times.
- after loop structure, code continues with the next statement at the end of the loop.

for loop

used if the initial value and the end value is know.

```
for (initialization; condition; update)
{
loop body;
}
```

compiler checks the condition firstly, if it's true the body will be executed. if it's false the compiler exits the loop and runs after the last line of the loop statement.

for loop components

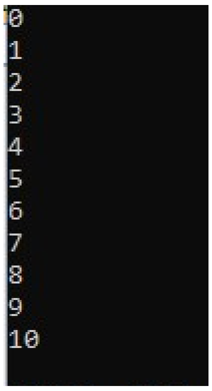
```
for (int number = 0; number < 100; number++)
{
// Loop body
}
```

int number = 0 is initial value
number < 100 is condition
number++ is update value

missing components

if there is an missing component it can be inside the loop body (which is the code piece that's been used repeatedly by the loop structure)

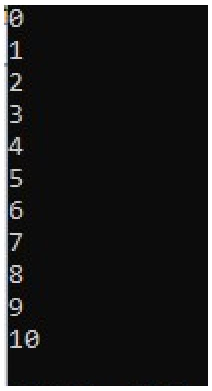
```
for (int number = 0; number <= 10; )
{
Console.WriteLine(number);
number++; // update component is inside loop body
}
```



```
0
1
2
3
4
5
6
7
8
9
10
```

condition components

```
bool status = true;
for (int number = 0; status ; number++ )
{
Console.WriteLine(number);
if (number >=10) status = false; // condition component changed
inside loop body
}
```



```
0
1
2
3
4
5
6
7
8
9
10
```

multiple indices in for loop

```
for (int x = 0, int y= 0; x < 5; x=x+1, y= y + x)
{
// Loop body
}
```

loops with no body

```
int sum = 0;
for (int x = 0; x < 5; sum+=x++) ;
Console.WriteLine(sum);
```

here the for loop doesn't have body so it'll just add x values from 0 to 4 and assign it to sum. the output will be 10 which is 1+2+3+4

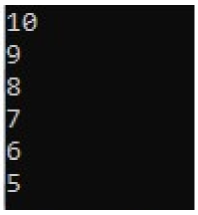
infinite loop

```
for (int x = 1; x >= 1; x++) {
Console.WriteLine(x);
}
```

when the condition will always be true the loop will never end.

decrease indices in for loop

```
for (int x = 10; x >= 5; x--)
{
Console.WriteLine(x);
}
```

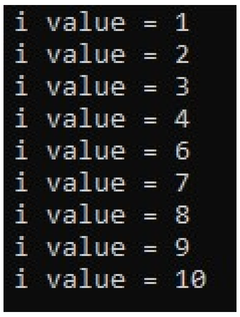


```
10
9
8
7
6
5
```

continue keyword

skips the current iteration of the loop
"do nothing just go on"

```
for (int i = 1; i <= 10; i++)
{
if (i == 5)
{
continue;
}
Console.WriteLine(" i value = " + i);
}
```

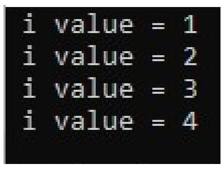


```
i value = 1
i value = 2
i value = 3
i value = 4
i value = 6
i value = 7
i value = 8
i value = 9
i value = 10
```

break keyword

the break keyword terminates the loop
"stop the loop go out of it"

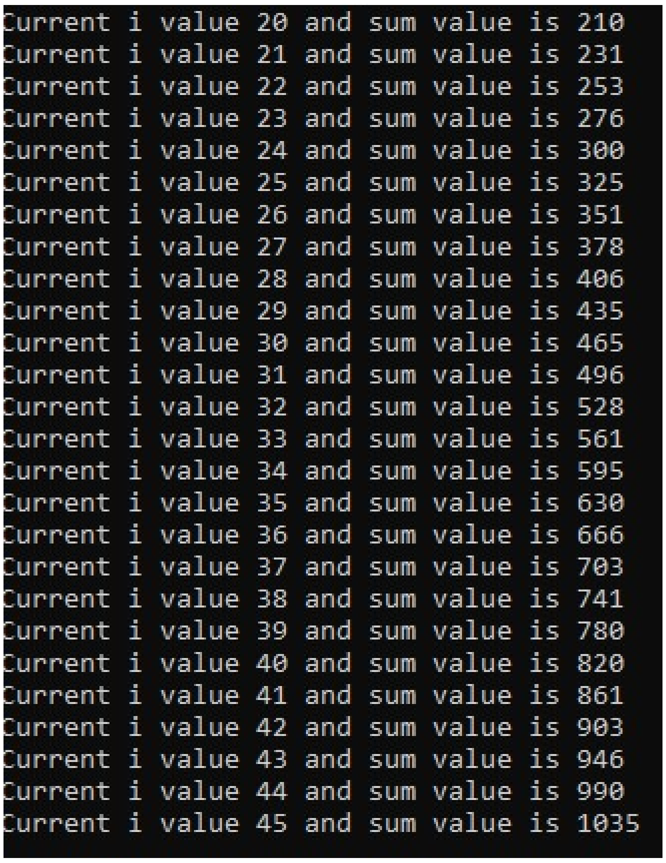
```
for (int i = 1; i <= 10; i++)
{
if (i >= 5)
{
break;
}
Console.WriteLine(" i value = " + i);
}
```



```
i value = 1
i value = 2
i value = 3
i value = 4
```

example: from 1 to 100 take the sum of integer numbers until the sum is greater or equal to 1000

```
            int sum = 0;
            for (int i = 1; i <= 100; i++)
            {
                sum = sum + i;

                Console.WriteLine($"Current i value {i} and sum
value is {sum}");
                if (sum >= 1000)
                    break;
```

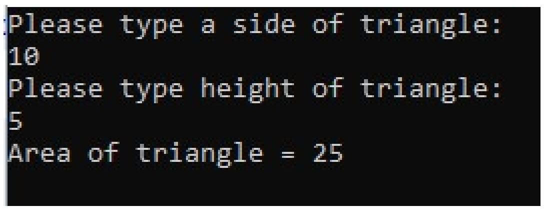


```
Current i value 20 and sum value is 210
Current i value 21 and sum value is 231
Current i value 22 and sum value is 253
Current i value 23 and sum value is 276
Current i value 24 and sum value is 300
Current i value 25 and sum value is 325
Current i value 26 and sum value is 351
Current i value 27 and sum value is 378
Current i value 28 and sum value is 406
Current i value 29 and sum value is 435
Current i value 30 and sum value is 465
Current i value 31 and sum value is 496
Current i value 32 and sum value is 528
Current i value 33 and sum value is 561
Current i value 34 and sum value is 595
Current i value 35 and sum value is 630
Current i value 36 and sum value is 666
Current i value 37 and sum value is 703
Current i value 38 and sum value is 741
Current i value 39 and sum value is 780
Current i value 40 and sum value is 820
Current i value 41 and sum value is 861
Current i value 42 and sum value is 903
Current i value 43 and sum value is 946
Current i value 44 and sum value is 990
Current i value 45 and sum value is 1035
```

Some coding exercises

Area of the triangle

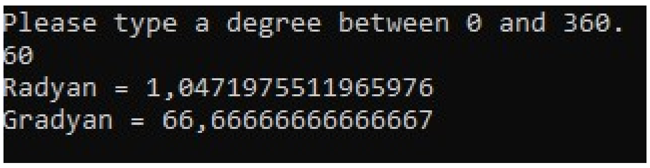
```
using System;

namespace area_of_triangle
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Please type a side of triangle:");
            double a = Double.Parse(Console.ReadLine());

            Console.WriteLine("Please type height of triangle:");
            double height = Double.Parse(Console.ReadLine());


            double area = a * height / 2;
            Console.WriteLine("Area of triangle = " + area);


            Console.ReadLine();
        }
    }
}
```



```
Please type a side of triangle:
10
Please type height of triangle:
5
Area of triangle = 25
```

converting degree to radian (radyan) gradian (gradyan)

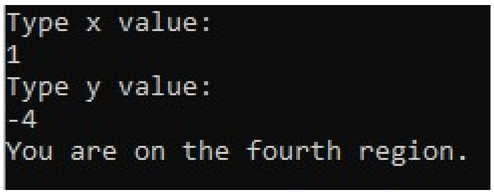
```
using System;

namespace degree_to_radyan_gradyan
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Please type a degree between 0 and 
360.");

            double degree = Double.Parse(Console.ReadLine());

            double radyan = degree * Math.PI / 180;
            double gradyan = degree * 200 / 180;


            Console.WriteLine("Radyan = " + radyan);
            Console.WriteLine("Gradyan = " + gradyan);

            Console.ReadLine();
        }
    }
}
```



```
Please type a degree between 0 and 360.
60
Radyan = 1,0471975511965976
Gradyan = 66,6666666666667
```

determining the region of coordinates

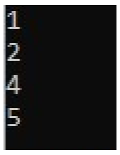
```
using System;

namespace determine_coordinate_region
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Type x value:");
            int x = Int32.Parse(Console.ReadLine());

            Console.WriteLine("Type y value:");
            int y = Int32.Parse(Console.ReadLine());

            string desc = "";

            if (x > 0 && y > 0)
                desc = "You are on the first region.";
            else if (x < 0 && y > 0)
                desc = "You are on the second region.";
            else if (x < 0 && y < 0)
                desc = "You are on the third region.";
            else if (x > 0 && y < 0)
                desc = "You are on the fourth region.";

            Console.WriteLine(desc);
        }
    }
}
```



```
Type x value:
1
Type y value:
-4
You are on the fourth region.
```

number from 1 to 5 except 3

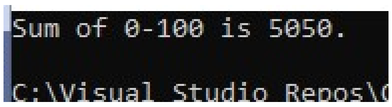
```
            for (int i = 1; i <= 5; i++)
            {
                if (i == 3)
                    continue;
                else
                    Console.WriteLine(i);
```



```
1
2
4
5
```

Sum of 0-100

```
            int sum = 0;
            for (int x = 0; x <= 100; x++)
            {
                sum = sum + x;
                //sum+=x;
                //Console.WriteLine(sum);
            }
            Console.WriteLine($"Sum of 0-100 is {sum}.");
```



```
Sum of 0-100 is 5050.

C:\Visual Studio Repos\
```

sum of even numbers from 0 to 100

```
            int sum = 0;
            for (int i = 0; i <= 100; i = i + 1)
            {
                if (i % 2 == 0)
                {
                    sum = sum + i;
                    //sum+=x;
                    Console.WriteLine($"Current i is {i} and sum is
{sum}.");
```

calculating area of a piece of circle

```
using System;

namespace part_of_area_a_circle
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Please type an angle of circle");
            double alfa = Double.Parse(Console.ReadLine());

            Console.WriteLine("Please type raius of circle:");
            double r = Double.Parse(Console.ReadLine());


            double area = alfa / 360 * Math.PI * r * r;
            double area2 = alfa / 360 * Math.PI * Math.Pow(r, 2);

            Console.WriteLine("Area = " + area2);
        }
    }
}
```

typing a name again and again

```
using System;

namespace repeated_task
{
    internal class Program
    {
        static void Main(string[] args)
        {

            Console.WriteLine("Please type your name:");

            string name = Console.ReadLine();

            for (int i=10;    i>=100;    i--)
            {
                Console.WriteLine(name);
            }
            /* instead of doing this
            Console.WriteLine(name);
            Console.WriteLine(name);
            Console.WriteLine(name);
            Console.WriteLine(name);
            Console.WriteLine(name);
            Console.WriteLine(name);
            Console.WriteLine(name);
            Console.WriteLine(name);
            Console.WriteLine(name);
            Console.WriteLine(name);
            */
        }
    }
}
```

is it weekday or weekend?

```
            Console.WriteLine("Please type a day name!");
            string day_name = Console.ReadLine().ToLower();

            if (day_name == "monday" || day_name == "thuesday" ||
day_name == "wednesday" || day_name == "thursday" || day_name ==
"friday")
            {
                Console.WriteLine(day_name + " is a week day.");
            } else if (day_name == "saturday" || day_name ==
"sunday")
            {
                Console.WriteLine(day_name + " is a weekend day.");
            } else
            {
                Console.WriteLine("Please type a validate day
name!");
            }
```