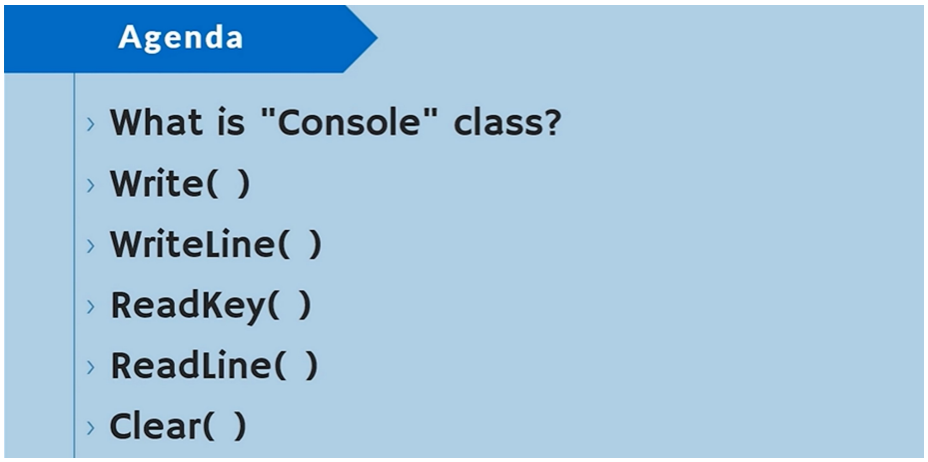
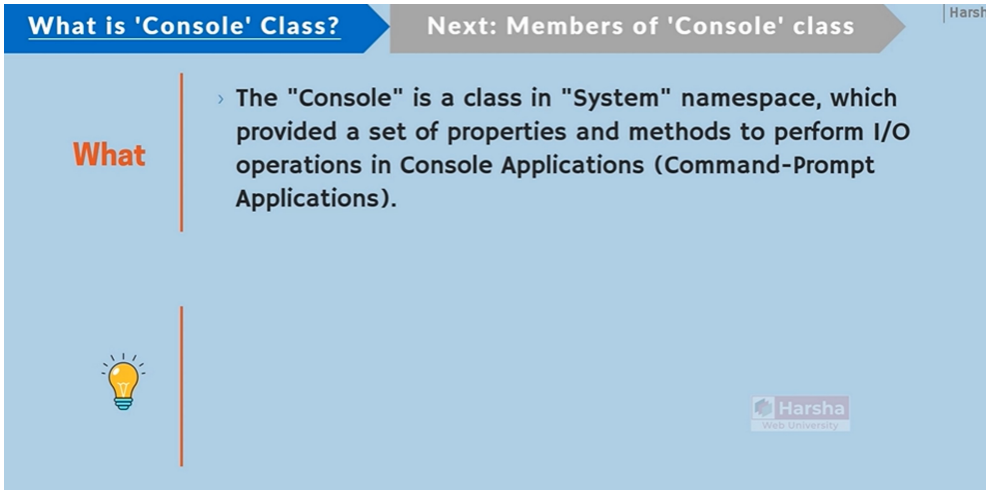
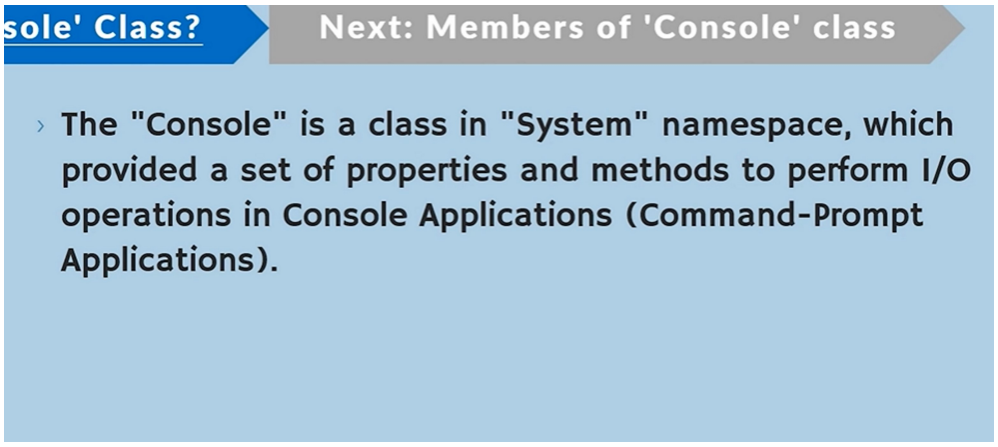
Section 2: C# Langauge Basics (Practical Starts Here)

1 . The System.Console Class





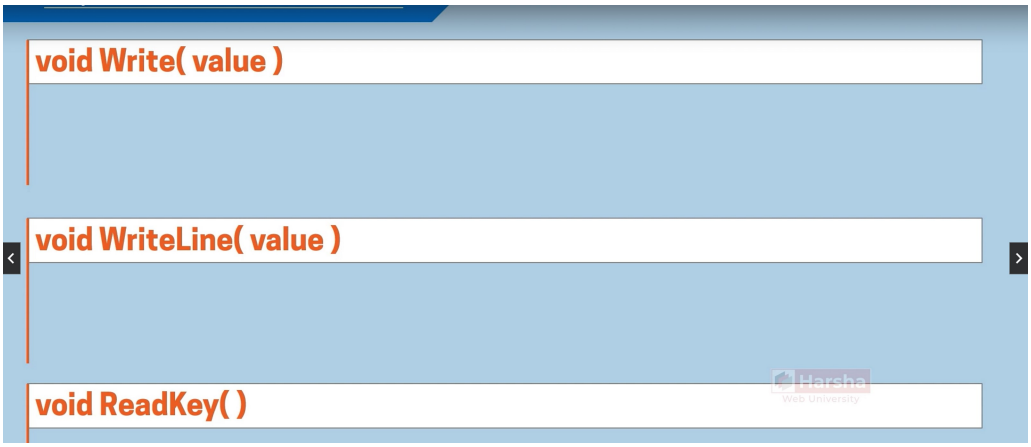


- It is a static class. So all the members of "Console" class are accessible without creating any object for the "Console" clas

printf() scanf()

- It is a static class. So all the members of "Console" class are accessible without creating any object for the "Console" class.
- The "Console" class is a part of BCL (Base Class Library).

Members of Console Class



void Write(value)

- It receives a value as parameter and displays the same value in Console (Command-Prompt window).

void WriteLine(value)

- It receives a value as parameter and displays the same value in Console also moves the cursor to the next line, after the value.

void ReadKey( )

- It waits until the user presses any key on the keyboard.
- It makes the console window wait for user's input.

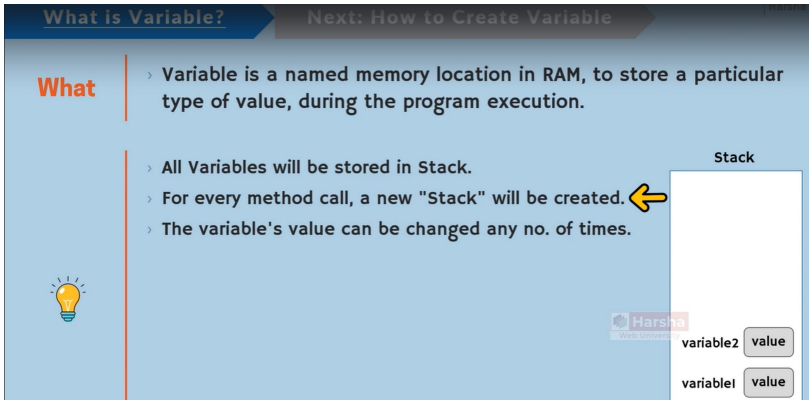
void Clear( )

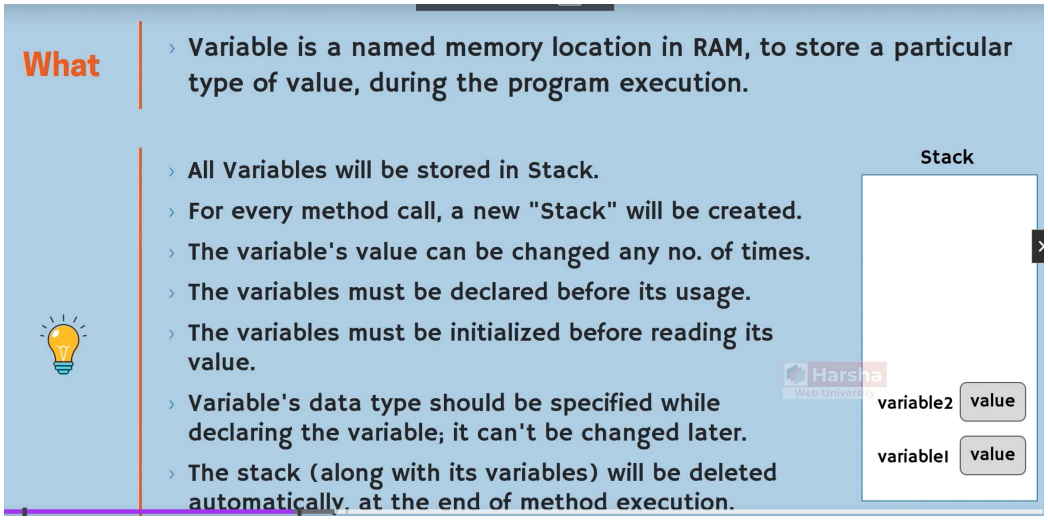
- It clears (make empty) the console window.
- After clearing the screen, you can display output again, using Write( ) or WriteLine( ) methods.

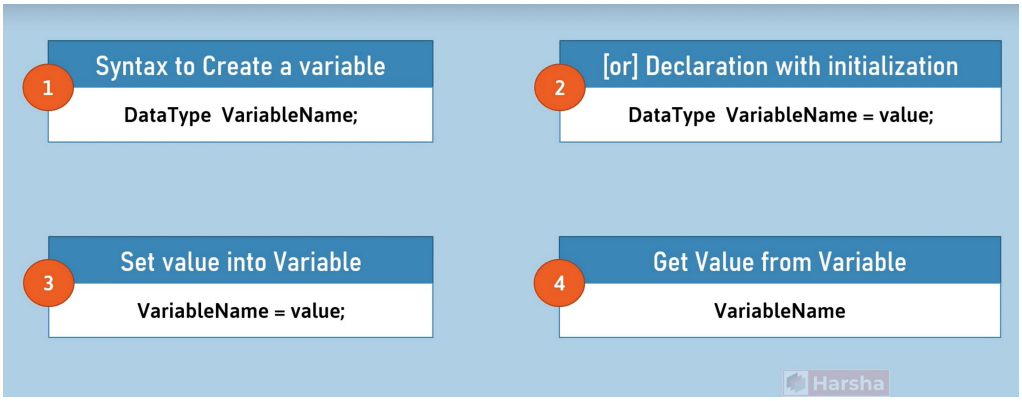
string ReadLine( )

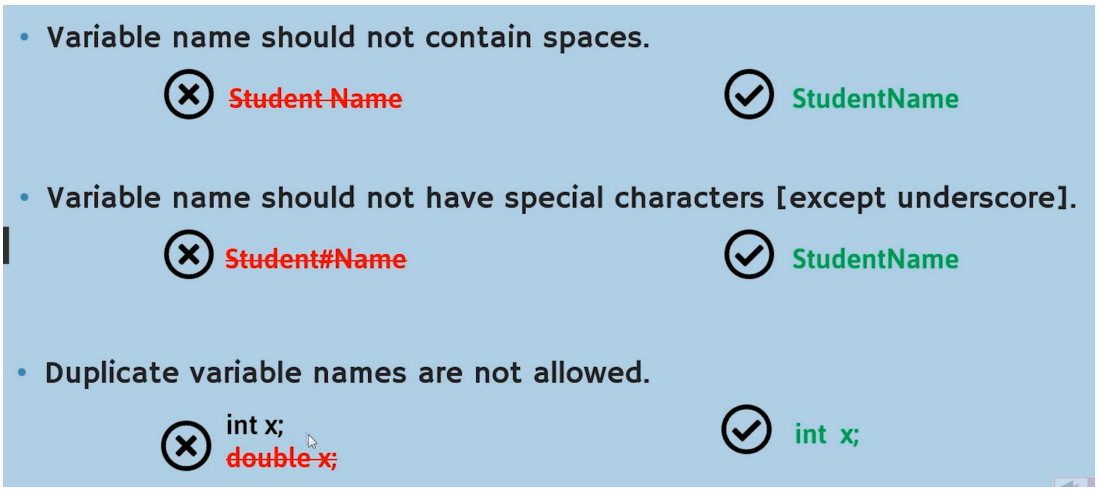
- It accepts a string value from keyboard (entered by user) and returns the same
- It always returns the value in "string" type only.

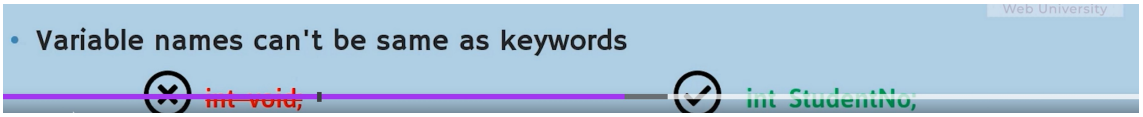
2. Variables





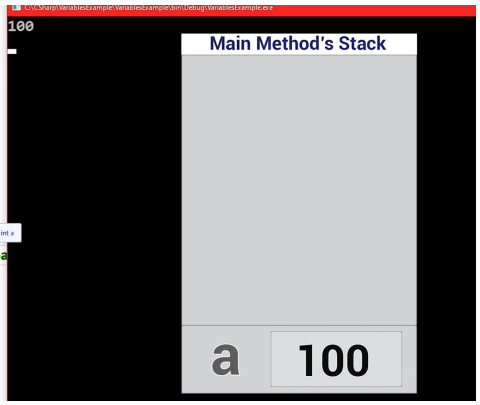







```
0 references
class Sample
{
    0 references
    static void Main()
    {
        //declare the variable
        int a = 100;

        //print value
        System.Console.WriteLine(a);
        //wait for pressing some key on keyboa
        System.Console.ReadKey();
    }
}
```



3. Primitive Types

C#

Harsha Vardhan

- UI Expert
- .NET Expert
- Module Lead
- Corporate Instructor

Harsha Web University

Primitive Types

What

- 'Type' specifies what type of value to be stored in memory.
- "Type" is a.k.a. "data type".
- Ex: int, string etc.

Primitive Types

(sbyte, byte, short, ushort,
int, uint, long, ulong,
float, double, decimal, char, bool)

- Strictly stores single value.
- Primitive Types are basic building blocks of non-primitive types

Non-Primitive Types

(string, Classes, Interfaces,
Structures, Enumerations)

- Stores one or more values.
- Usually contains multiple members.

1. sbyte

- 8-bit signed integer
- Size: 1 byte
- Range: -128 to 127
- Default value: 0
- MinValue Command: sbyte.MinValue
- MaxValue Command: sbyte.MaxValue

2. byte

- 8-bit un-signed integer
- Size: 1 byte
- Range: 0 to 255
- Default value: 0
- MinValue Command: byte.MinValue
- MaxValue Command: byte.MaxValue

Harsha

3. short

- 16-bit signed integer
- Size: 2 bytes
- Range: -32,768 to 32,767
- Default value: 0
- MinValue Command: short.MinValue
- MaxValue Command: short.MaxValue

4. ushort

- 16-bit un-signed integer
- Size: 2 bytes
- Range: 0 to 65,535
- Default value: 0
- MinValue Command: ushort.MinValue
- MaxValue Command: ushort.MaxValue

Harsha

3. short

- 16-bit signed integer
- Size: 2 bytes
- Range: -32,768 to 32,767
- Default value: 0
- MinValue Command: short.MinValue
- MaxValue Command: short.MaxValue

4. ushort

- 16-bit un-signed integer
- Size: 2 bytes
- Range: 0 to 65,535
- Default value: 0
- MinValue Command: ushort.MinValue
- MaxValue Command: ushort.MaxValue

5. int

- 32-bit signed integer
- Size: 4 bytes
- Range: -2,147,483,648 to 2,147,483,647
- Default value: 0
- MinValue Command: int.MinValue
- MaxValue Command: int.MaxValue
- By default, integer literals between -2,147,483,648 to 2,147,483,647 are treated as "int" data type.

6. uint

- 32-bit un-signed integer
- Size: 4 bytes
- Range: 0 to 4,294,967,295
- Default value: 0
- MinValue Command: uint.MinValue
- MaxValue Command: uint.MaxValue
- By default, integer literals between 2,147,483,648 to 4,294,967,295 are treated as "uint" data type.

7. long

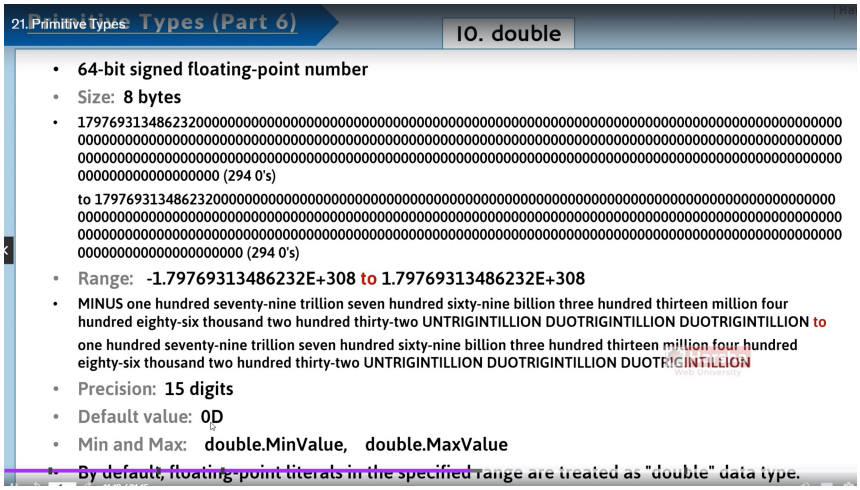
- 64-bit signed integer
- Size: 8 bytes
- Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- Range: -2^{63} to $2^{63}-1$
- Default value: 0
- MinValue Command: long.MinValue
- MaxValue Command: long.MaxValue
- By default, integer literals between 4,294,967,296 and 9,223,372,036,854,775,807 are treated as "long" data type.

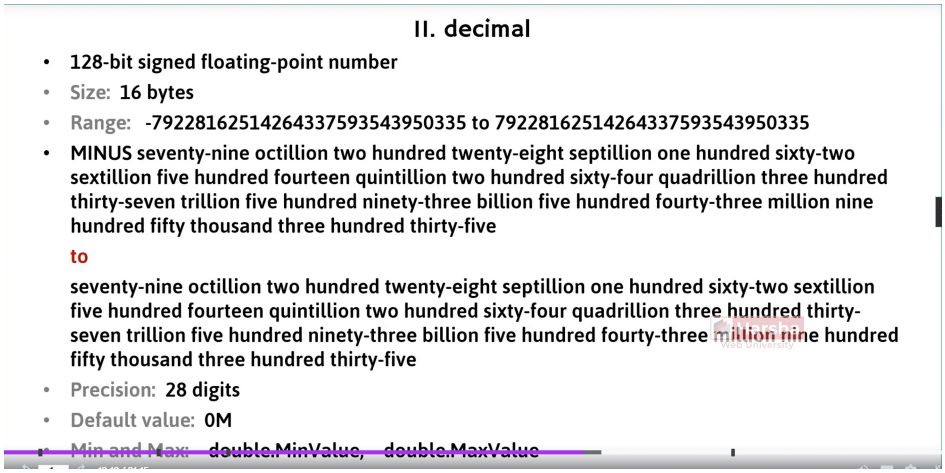
8. ulong

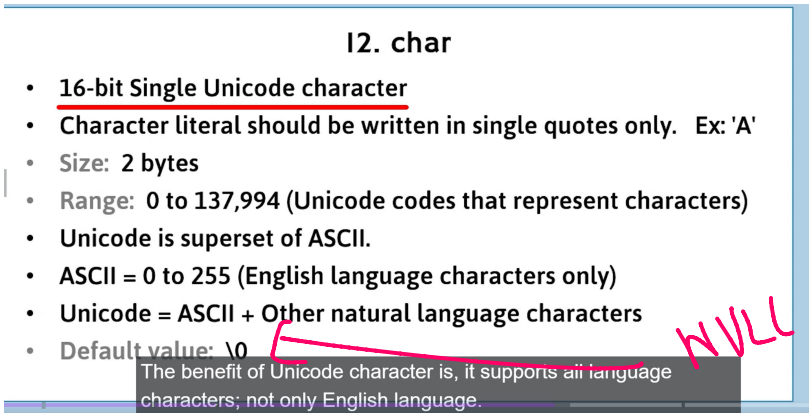
- 64-bit un-signed integer
- Size: 8 bytes
- Range: 0 to 18,446,744,073,709,551,615
- Default value: 0
- MinValue Command: ulong.MinValue
- MaxValue Command: ulong.MaxValue
- By default, integer literals between 9,223,372,036,854,775,808 and 18,446,744,073,709,551,615 are treated as "ulong" data type.

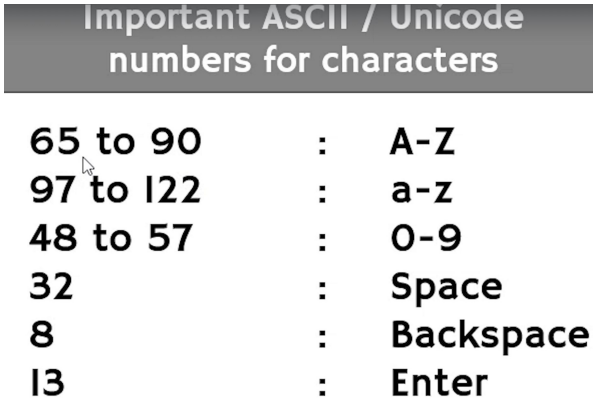
9. float

- 32-bit signed floating-point number
- Size: 4 bytes
- Range: -340282300000000000000000000000000000000 to 340282300000000000000000000000000000000
- Range: -3.402823E+38 to 3.402823E+38
- Range: MINUS three hundred fourty two hundred eighty-two three hundred nonillion to three hundred fourty two hundred eighty-two three hundred NONILLION
- Precision: 7 digits
- Default value: 0F
- MinValue Command: float.MinValue
- MaxValue Command: float.MaxValue


21. Primitive Types
Primitive Types (Part 6)
10. double
64-bit signed floating-point number
Size: 8 bytes
Range: -1.79769313486232E+308 to 1.79769313486232E+308
MINUS one hundred seventy-nine trillion seven hundred sixty-nine billion three hundred thirteen million four hundred eighty-six thousand two hundred thirty-two UNTRIGINTILLION DUOTRIGINTILLION DUOTRIGINTILLION to
one hundred seventy-nine trillion seven hundred sixty-nine billion three hundred thirteen million four hundred eighty-six thousand two hundred thirty-two UNTRIGINTILLION DUOTRIGINTILLION DUOTRIGINTILLION
Precision: 15 digits
Default value: 0D
Min and Max: double.MinValue, double.MaxValue
By default, floating-point literals in the specified range are treated as "double" data type.


11. decimal
128-bit signed floating-point number
Size: 16 bytes
Range: -79228162514264337593543950335 to 79228162514264337593543950335
MINUS seventy-nine octillion two hundred twenty-eight septillion one hundred sixty-two sextillion five hundred fourteen quintillion two hundred sixty-four quadrillion three hundred thirty-seven trillion five hundred ninety-three billion five hundred fourty-three million nine hundred fifty thousand three hundred thirty-five
to
seventy-nine octillion two hundred twenty-eight septillion one hundred sixty-two sextillion five hundred fourteen quintillion two hundred sixty-four quadrillion three hundred thirty-seven trillion five hundred ninety-three billion five hundred fourty-three million nine hundred fifty thousand three hundred thirty-five
Precision: 28 digits
Default value: 0M
Min and Max: double.MinValue, double.MaxValue


12. char
16-bit Single Unicode character
Character literal should be written in single quotes only. Ex: 'A'
Size: 2 bytes
Range: 0 to 137,994 (Unicode codes that represent characters)
Unicode is superset of ASCII.
ASCII = 0 to 255 (English language characters only)
Unicode = ASCII + Other natural language characters
Default value: \0
NULL
The benefit of Unicode character is, it supports all language characters; not only English language.

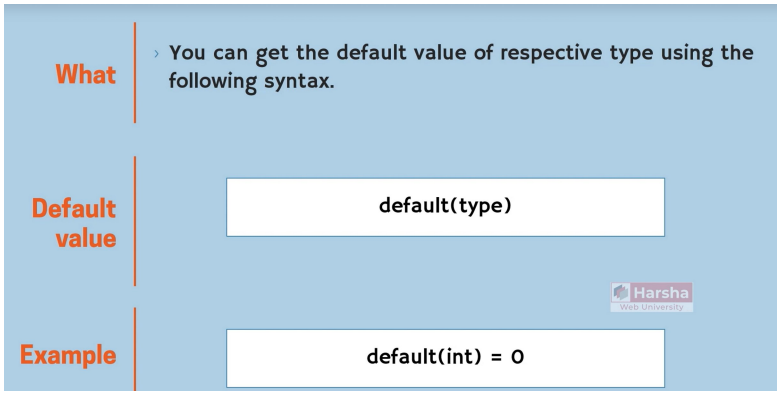

Important ASCII / Unicode numbers for characters
65 to 90 : A-Z
97 to 122 : a-z
48 to 57 : 0-9
32 : Space
8 : Backspace
13 : Enter

13. string

- Collection of Unicode characters
- String literal should be written in double quotes only. Ex: "Abc123"
- Size: Length * 2 bytes
- Range: 0 to 2 billion characters
- Default value: null

14. bool

- Stores logical value (true / false)
- Possible values: true, false
- Size: 1 bit
- Default value: false



4. Operators

What is Operator?

Next: Arithmetical Operators

What

- Operator is a symbol to perform operation.
- Ex: +, -, *, /, == etc.

Classification

- Arithmetical Operators
- Assignment Operators
- Increment and Decrement Operators
- Comparison Operators
- Logical Operators
- Concatenation Operator
- Ternary Operator

Arithmetical Operators

Next: Assignment Operators

What

- Used to perform arithmetical operations on the numbers

List of Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder

Assignment Operators

Next: Increment / Decrement Operators

What

- Used to perform arithmetical operations on the numbers

List of Operators

=	Assigns to
+=	Add and Assigns to
-=	Subtract and Assigns to
*=	Multiply and Assigns to
/=	Divide and Assigns to
%=	Remainder Assigns to

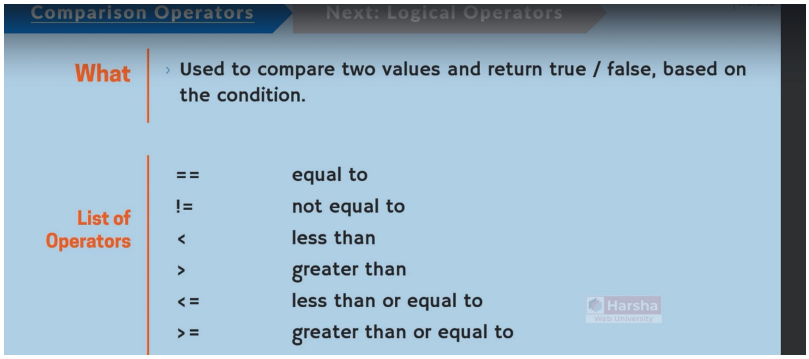
Increment / Decrement Operators

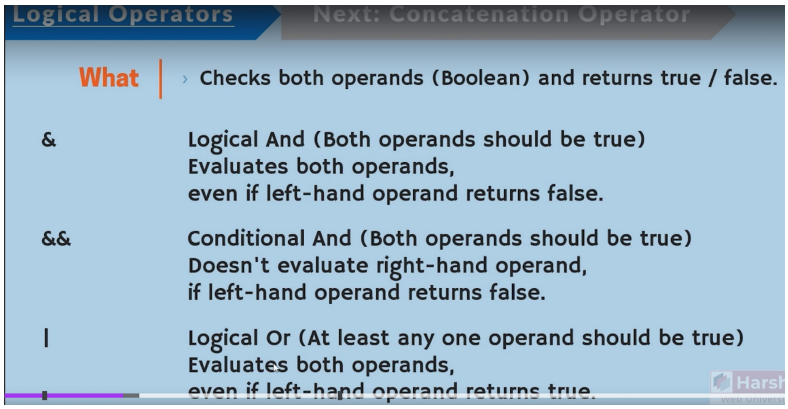
Next: Comparison Operators

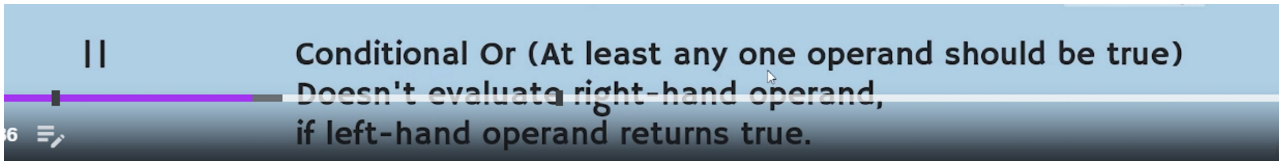
What

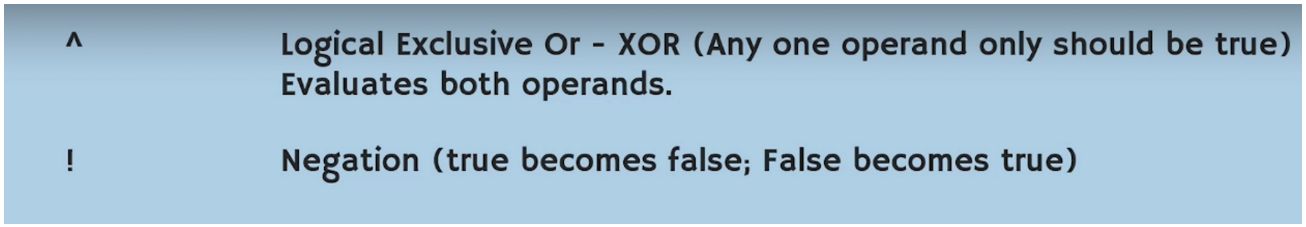
a = 5
a++
a = 6

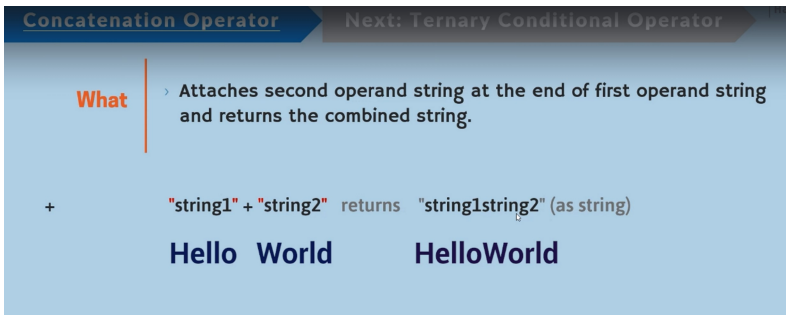
n++	Post-Incrementation (First it returns value; then increments)
++n	Pre-Incrementation (First it increments value; then returns)
n--	Post-Decrementation (First it returns value; then decrements)
--n	Pre-Decrementation (First it decrements value; then returns)

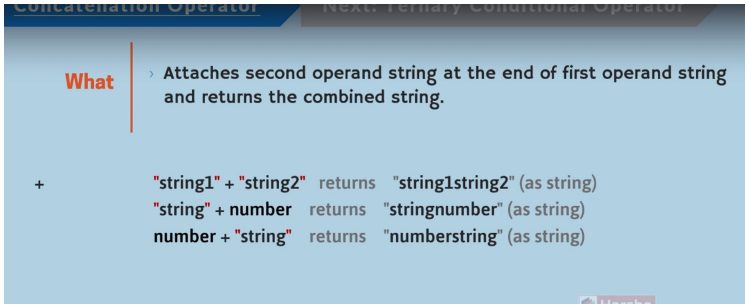

Comparison Operators
Next: Logical Operators
What
Used to compare two values and return true / false, based on the condition.
List of Operators
== equal to
!= not equal to
< less than
> greater than
<= less than or equal to
>= greater than or equal to

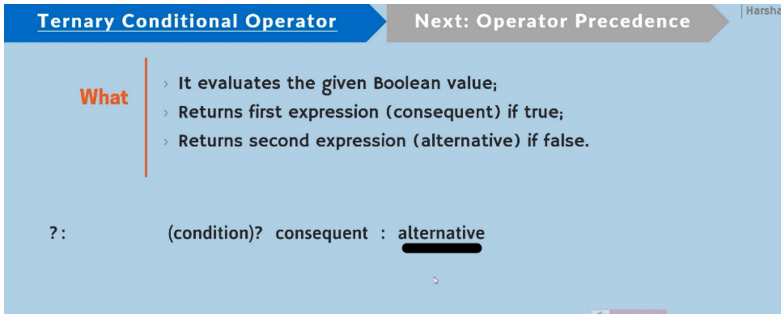

Logical Operators
Next: Concatenation Operator
What
Checks both operands (Boolean) and returns true / false.
&
Logical And (Both operands should be true)
Evaluates both operands,
even if left-hand operand returns false.
&&
Conditional And (Both operands should be true)
Doesn't evaluate right-hand operand,
if left-hand operand returns false.
|
Logical Or (At least any one operand should be true)
Evaluates both operands,

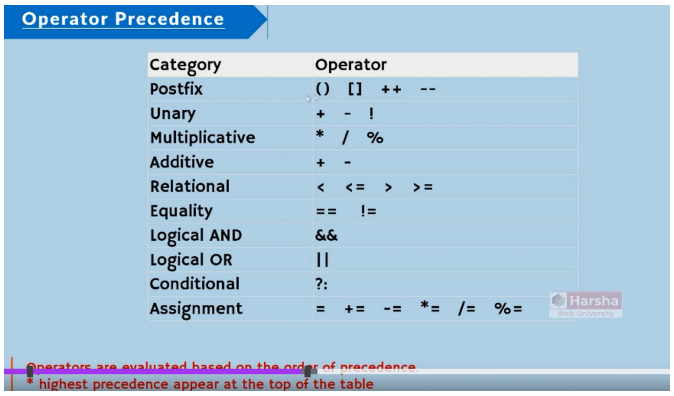

||
Conditional Or (At least any one operand should be true)
Doesn't evaluate right-hand operand,
if left-hand operand returns true.

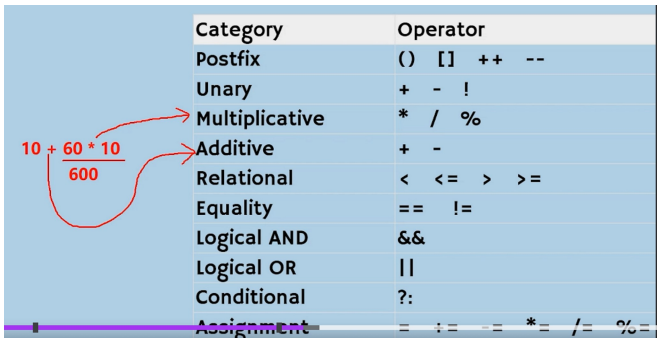

^
Logical Exclusive Or - XOR (Any one operand only should be true)
Evaluates both operands.
!
Negation (true becomes false; False becomes true)


Concatenation Operator
Next: Ternary Conditional Operator
What
Attaches second operand string at the end of first operand string and returns the combined string.
+
"string1" + "string2" returns "string1string2" (as string)
Hello World
HelloWorld



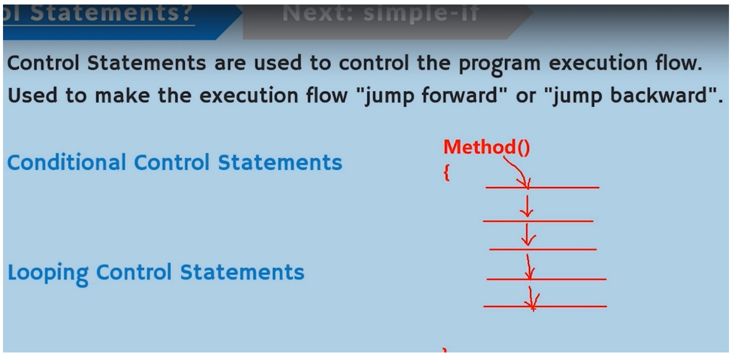


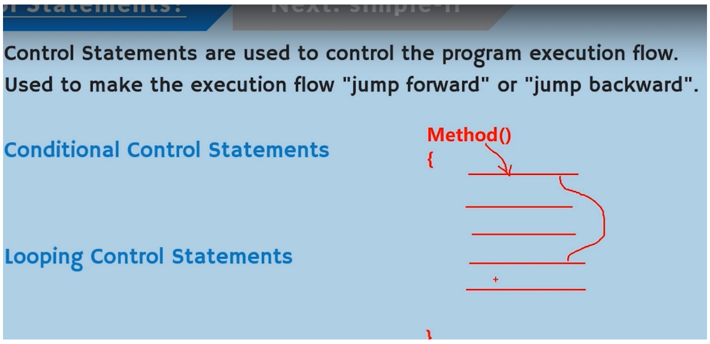


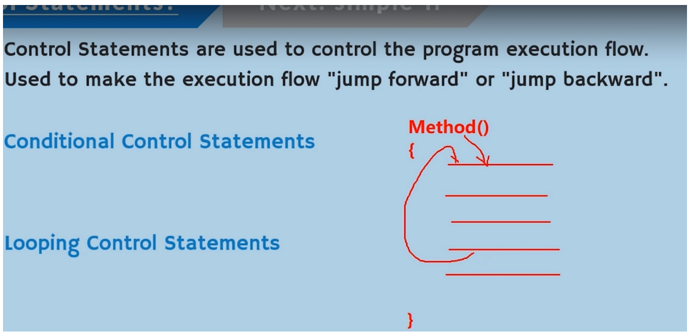


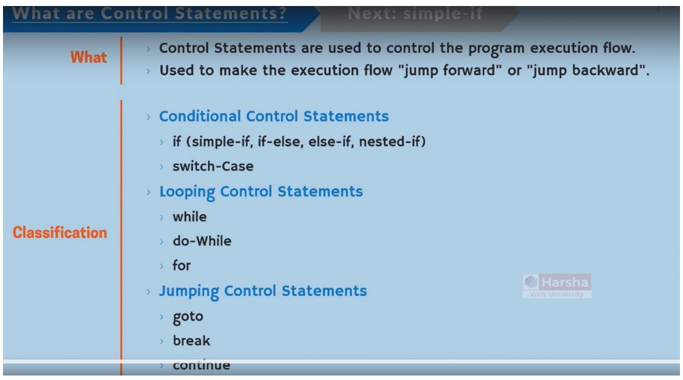
5. IF-else, else-if, nested-if

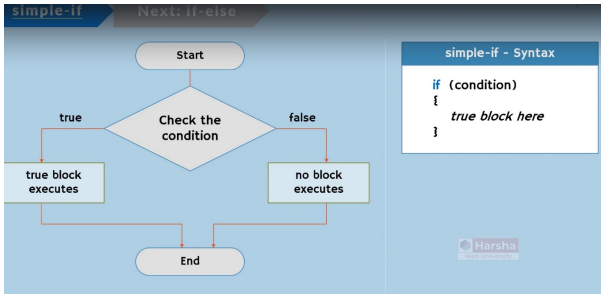

Agenda
1 Introducing Control Statements
2 if
3 switch-case
4 while
5 do-While
6 for
Harsha


ol Statements?
Next: simple-if
Control Statements are used to control the program execution flow.
Used to make the execution flow "jump forward" or "jump backward".
Conditional Control Statements
Looping Control Statements
Method()
{


Control Statements are used to control the program execution flow.
Used to make the execution flow "jump forward" or "jump backward".
Conditional Control Statements
Looping Control Statements
Method()
{


Control Statements are used to control the program execution flow.
Used to make the execution flow "jump forward" or "jump backward".
Conditional Control Statements
Looping Control Statements
Method()
{
}


What are Control Statements?
Next: simple-if
What
Control Statements are used to control the program execution flow.
Used to make the execution flow "jump forward" or "jump backward".
Classification
Conditional Control Statements
if (simple-if, if-else, else-if, nested-if)
switch-Case
Looping Control Statements
while
do-While
for
Jumping Control Statements
goto
break
continue
Harsha

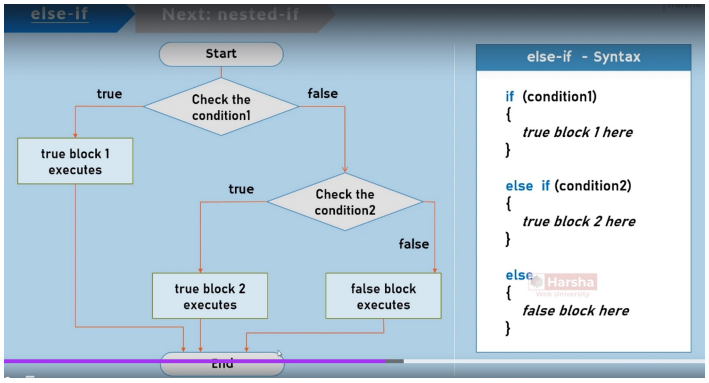


simple-if - Example

```
if (x < 10)
{
   System.Console.WriteLine("x is smaller than 10");
}
```

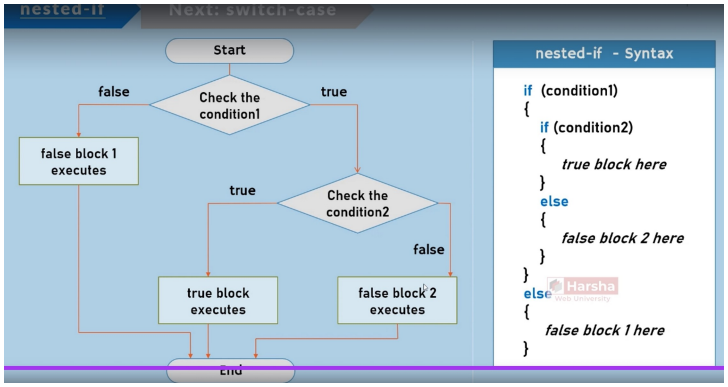
if-else - Example

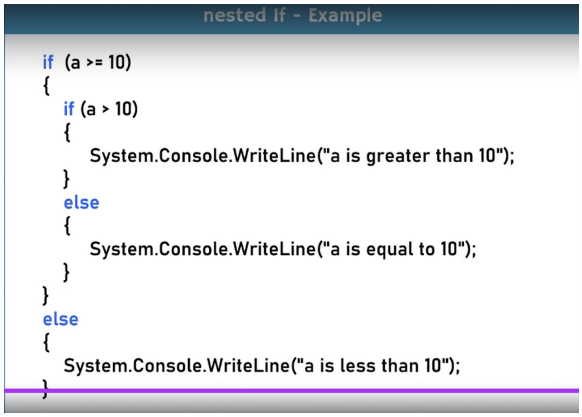
```
if (x > 10)
{
   System.Console.WriteLine("x is larger than 10");
}
else
{
   System.Console.WriteLine("x is smaller than or equal to 10");
}
```

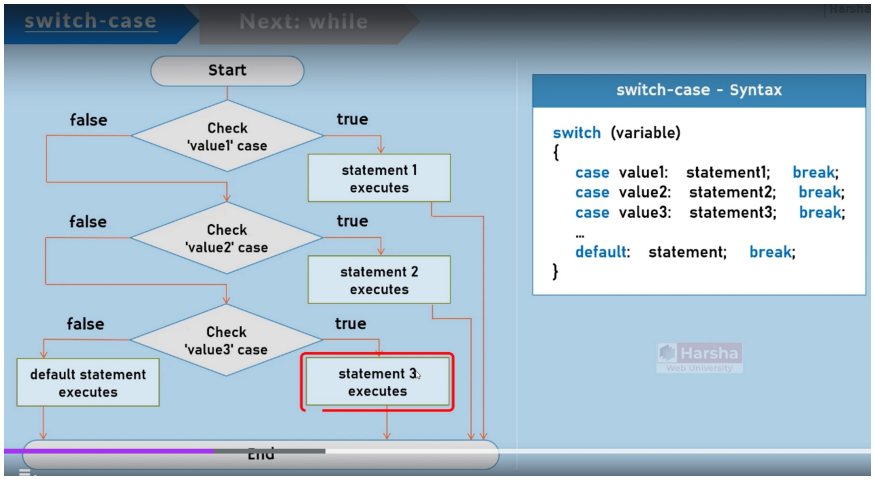


else if - Example

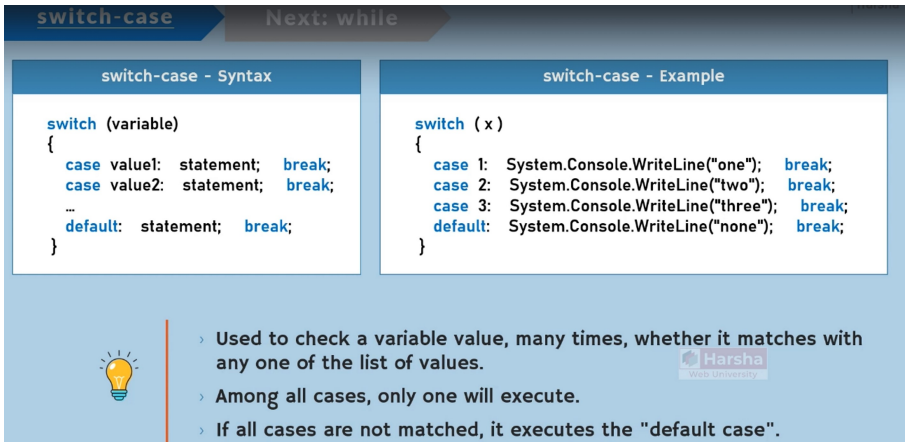
```
if (a > 10)
{
   System.Console.WriteLine("a is greater than 10");
}
else if (a < 10)
{
   System.Console.WriteLine("a is less than 10");
}
else
{
    System.Console.WriteLine("a is equal to 10");
}
```







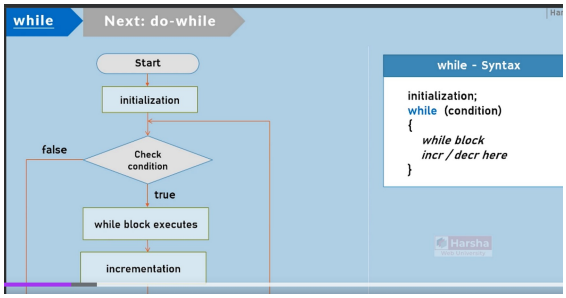
6. Switch

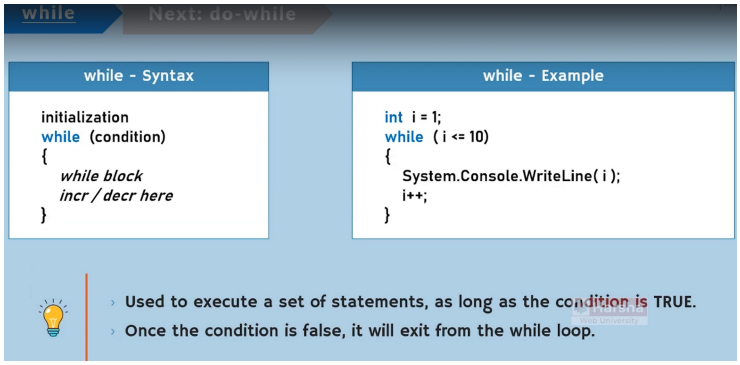


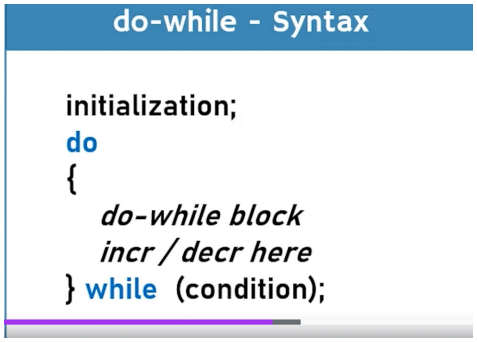
7. While, Do-While

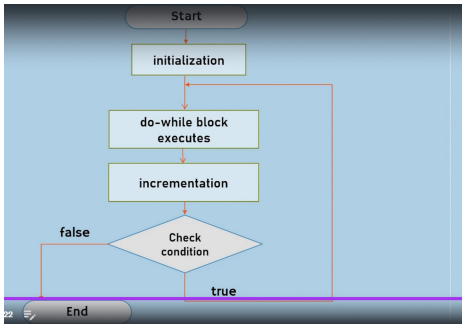
Agenda

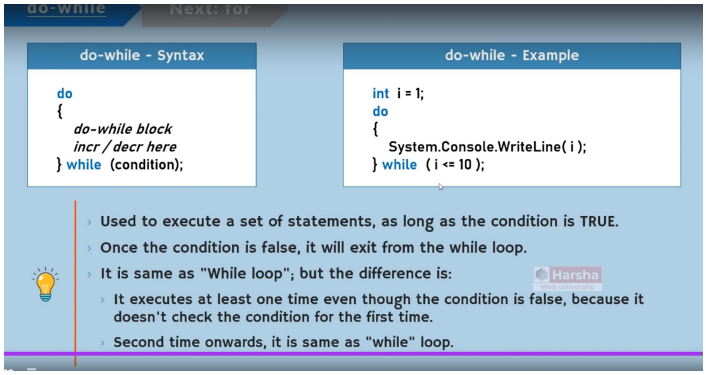
1. Introducing Control Statements
2. if
3. switch-case
4. while
5. do-While
6. for



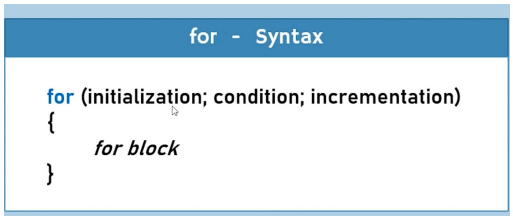


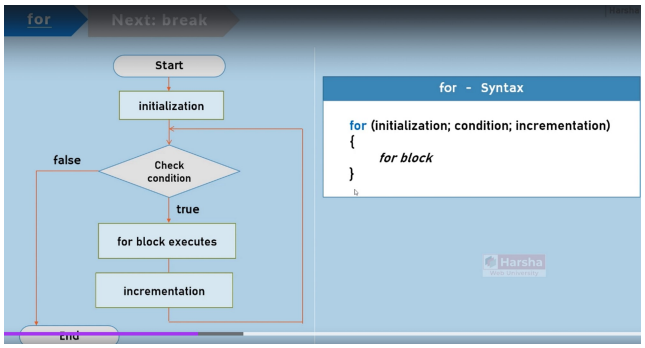


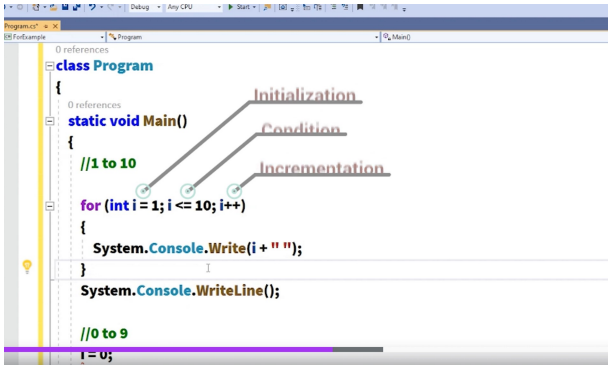




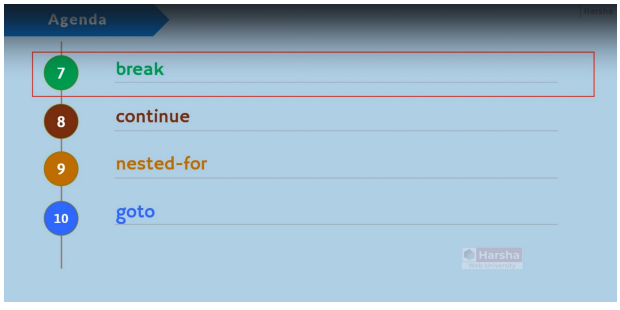
8. For Loop





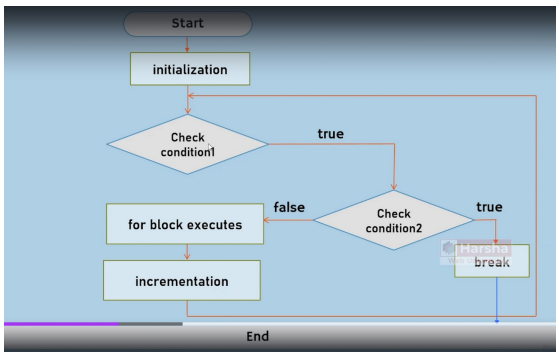


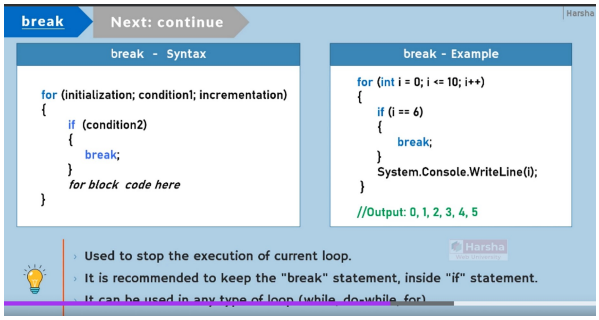
9. Break



break - Syntax

```
for (initialization; condition1; incrementation)
{
    if (condition2)
    {
        break;
    }
    for block code here
}
```

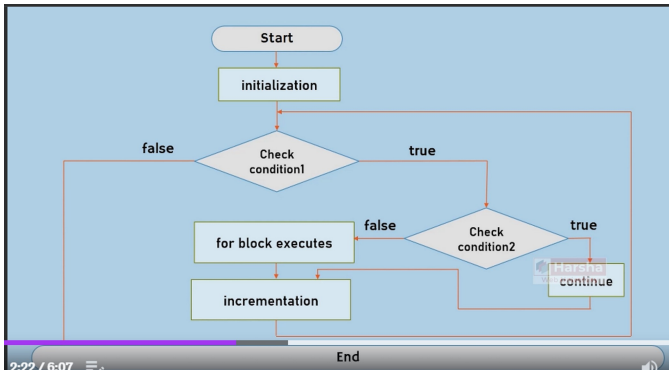


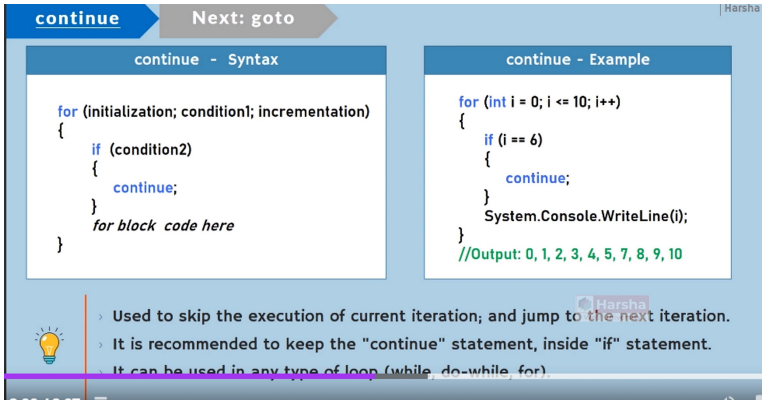


10. Continue

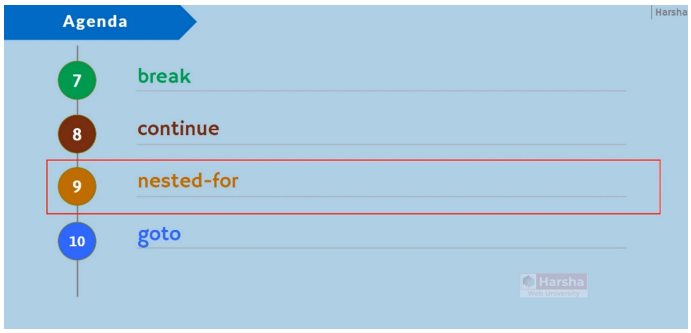
continue - Syntax

```
for (initialization; condition1; incrementation)
{
    if (condition2)
    {
        continue;
    }
    for block code here
}
```

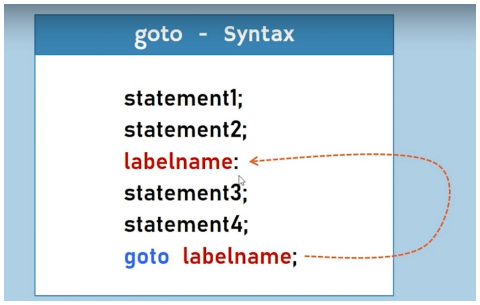


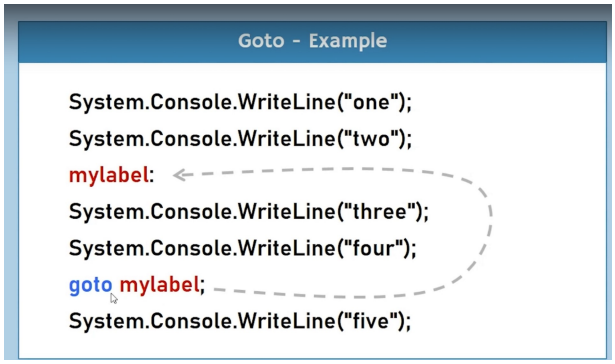


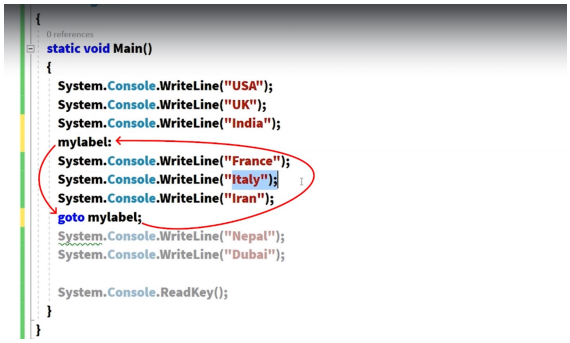
11. Nested For Loop



12. Go-To Statement

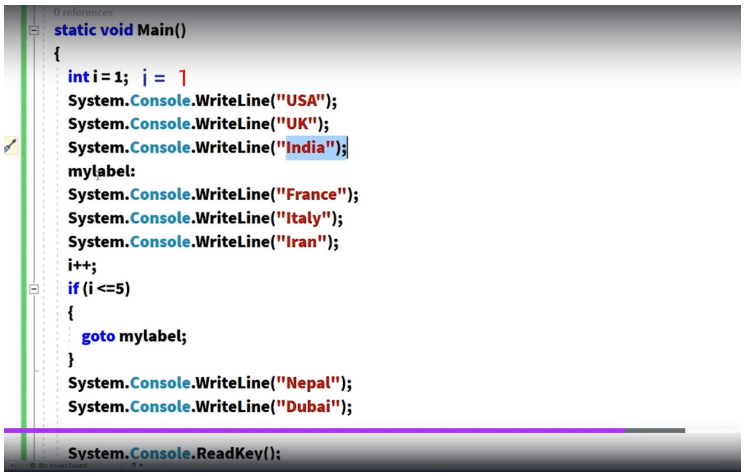






```
{
    0 references
    static void Main()
    {
        System.Console.WriteLine("USA");
        System.Console.WriteLine("UK");
        System.Console.WriteLine("India");
        mylabel:
        System.Console.WriteLine("France");
        System.Console.WriteLine("Italy");
        System.Console.WriteLine("Iran");
        goto mylabel;
        System.Console.WriteLine("Nepal");
        System.Console.WriteLine("Dubai");

        System.Console.ReadKey();
    }
}
```



```
0 references
static void Main()
{
    int i = 1;
    mylabel:
    System.Console.WriteLine("USA");
    System.Console.WriteLine("UK");
    System.Console.WriteLine("India");
    System.Console.WriteLine("France");
    System.Console.WriteLine("Italy");
    System.Console.WriteLine("Iran");
    i++;
    if (i <=5)
    {
        goto mylabel;
    }
    System.Console.WriteLine("Nepal");
    System.Console.WriteLine("Dubai");

    System.Console.ReadKey();
```