

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

1 . The *System.Console Class*

Agenda

- › What is "Console" class?
- › Write()
- › WriteLine()
- › ReadKey()
- › ReadLine()
- › Clear()

What is 'Console' Class?

Next: Members of 'Console' class

| Harsh

What

- › The "Console" is a class in "System" namespace, which provided a set of properties and methods to perform I/O operations in Console Applications (Command-Prompt Applications).



Harsha
Web University

'sole' Class?

Next: Members of 'Console' class

- › The "Console" is a class in "System" namespace, which provided a set of properties and methods to perform I/O operations in Console Applications (Command-Prompt Applications).

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

Web University

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).

Harsha
Web University

Members of Console Class

void Write(value)

void WriteLine(value)

void ReadKey()

Harsha
Web University

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

What is Variable?

Next: How to Create Variable

What

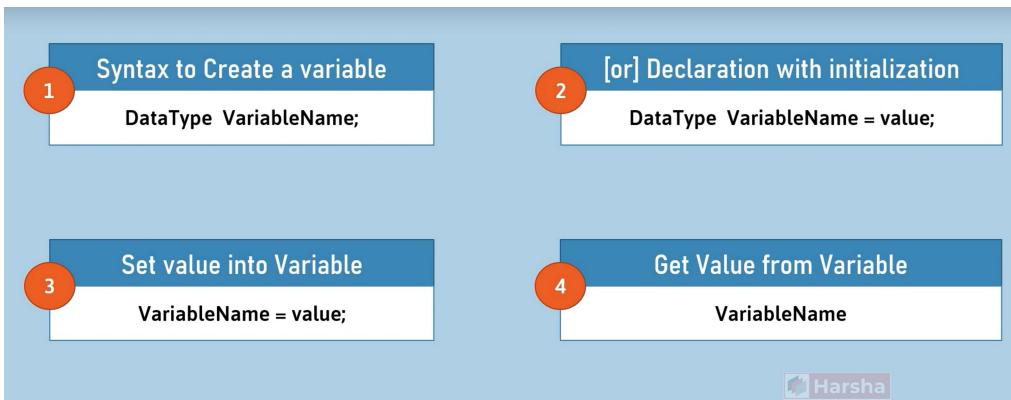
- Variable is a named memory location in RAM, to store a particular type of value, during the program execution.
- All Variables will be stored in Stack.
- For every method call, a new "Stack" will be created. 
- The variable's value can be changed any no. of times.



What

- Variable is a named memory location in RAM, to store a particular type of value, during the program execution.
- All Variables will be stored in Stack.
- For every method call, a new "Stack" will be created.
- The variable's value can be changed any no. of times.
- The variables must be declared before its usage.
- The variables must be initialized before reading its value.
- Variable's data type should be specified while declaring the variable; it can't be changed later.
- The stack (along with its variables) will be deleted automatically, at the end of method execution.





- Variable name should not contain spaces.

Student-Name

StudentName

- Variable name should not have special characters [except underscore].

Student#Name

StudentName

- Duplicate variable names are not allowed.

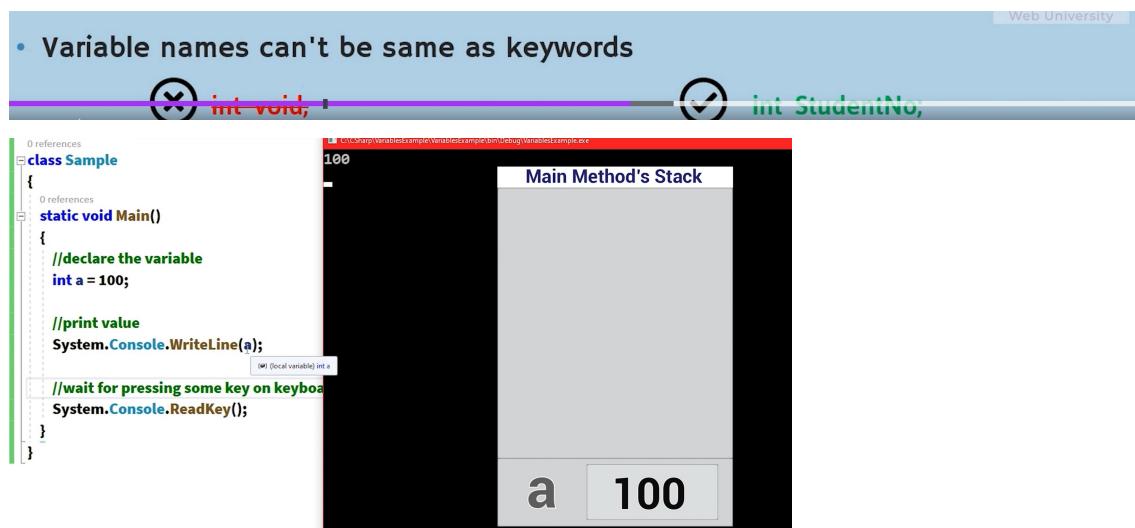
**int x;
double-x;**

int x;

- Variable names can't be same as keywords

int void;

int StudentNo;



3. Primitive Types

C#

Harsha Vardhan

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

Harsha

Primitive Types



Harsha Vardhan

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



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)

Non-Primitive Types

(string, Classes, Interfaces,
Structures, Enumerations)

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

- 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`

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`



<h3>3. short</h3> <ul style="list-style-type: none"> • 16-bit signed integer • Size: 2 bytes • Range: -32,768 to 32,767 • Default value: 0 • MinValue Command: <code>short.MinValue</code> • MaxValue Command: <code>short.MaxValue</code> 	<h3>4. ushort</h3> <ul style="list-style-type: none"> • 16-bit un-signed integer • Size: 2 bytes • Range: 0 to 65,535 • Default value: 0 • MinValue Command: <code>ushort.MinValue</code> • MaxValue Command: <code>ushort.MaxValue</code>
--	--

<h3>5. int</h3> <ul style="list-style-type: none"> • 32-bit signed integer • Size: 4 bytes • Range: -2,147,483,648 to 2,147,483,647 • Default value: 0 • MinValue Command: <code>int.MinValue</code> • MaxValue Command: <code>int.MaxValue</code> • By default, integer literals between -2,147,483,648 to 2,147,483,647 are treated as "int" data type. 	<h3>6. uint</h3> <ul style="list-style-type: none"> • 32-bit un-signed integer • Size: 4 bytes • Range: 0 to 4,294,967,295 • Default value: 0 • MinValue Command: <code>uint.MinValue</code> • MaxValue Command: <code>uint.MaxValue</code> • By default, integer literals between 2,147,483,648 to 4,294,967,295 are treated as "uint" data type.
--	---

<h3>7. long</h3> <ul style="list-style-type: none"> • 64-bit signed integer • Size: 8 bytes • Range: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 • Range: -2⁶³ to 2⁶³-1 • Default value: 0 • MinValue Command: <code>long.MinValue</code> • MaxValue Command: <code>long.MaxValue</code> • By default, integer literals between 4,294,967,296 and 9,223,372,036,854,775,807 are treated as "long" data type. 	<h3>8. ulong</h3> <ul style="list-style-type: none"> • 64-bit un-signed integer • Size: 8 bytes • Range: 0 to 18,446,744,073,709,551,615 • Default value: 0 • MinValue Command: <code>ulong.MinValue</code> • MaxValue Command: <code>ulong.MaxValue</code> • 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.
--	--

<h3>9. float</h3> <ul style="list-style-type: none"> • 32-bit signed floating-point number • Size: 4 bytes • Range: -34028230000000000000000000000000 to 34028230000000000000000000000000 • 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: <code>float.MinValue</code> • MaxValue Command: <code>float.MaxValue</code>

II. 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 forty-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 forty-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

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

What	› You can get the default value of respective type using the following syntax.
Default value	default(type)
Example	default(int) = 0

Harsha
Web University

4. Operators

C#

Operators

 Harsha Vardhan

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

What Harsha
Web University

- > 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

What

- > Used to perform arithmetical operations on the numbers

List of Operators

- | | |
|---|----------------|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Remainder |

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 |

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)

What

- > Used to compare two values and return true / false, based on the condition.

List of Operators

<code>==</code>	equal to
<code>!=</code>	not equal to
<code><</code>	less than
<code>></code>	greater than
<code><=</code>	less than or equal to
<code>>=</code>	greater than or equal to

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,
 even if left-hand operand returns true.

`||`

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)

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

Concatenation Operator

What

- Attaches second operand string at the end of first operand string and returns the combined string.

+

- "string1" + "string2" returns "string1string2" (as string)
- "string" + number returns "stringnumber" (as string)
- number + "string" returns "numberstring" (as string)

Ternary Conditional Operator

What

- It evaluates the given Boolean value;
- Returns first expression (consequent) if true;
- Returns second expression (alternative) if false.

? : (condition)? consequent : alternative

Operator Precedence	
Category	Operator
Postfix	() [] ++ --
Unary	+ - !
Multiplicative	* / %
Additive	+ -
Relational	< <= > >=
Equality	== !=
Logical AND	&&
Logical OR	
Conditional	?:
Assignment	= += -= *= /= %=

Operators are evaluated based on the order of precedence.
Highest precedence appear at the top of the table

Category	Operator
Postfix	() [] ++ --
Unary	+ - !
Multiplicative	* / %
Additive	+ -
Relational	< <= > >=
Equality	== !=
Logical AND	&&
Logical OR	
Conditional	?:
Assignment	= += -= *= /= %=

10 + 60 * 10
600

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

C#



Harsha Vardhan

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

Control Statements

Agenda

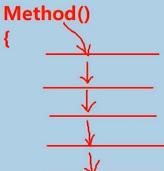
- 1 Introducing Control Statements
- 2 if
- 3 switch-case
- 4 while
- 5 do-While
- 6 for

Harsha
www.tutorialspoint.com

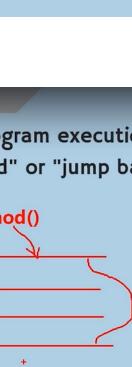
Control Statements? **Next: simple-if**

Control Statements are used to control the program execution flow.
Used to make the execution flow "jump forward" or "jump backward".

Conditional Control Statements



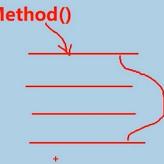
Looping Control Statements



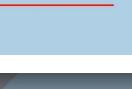
Control Statements? **Next: simple-if**

Control Statements are used to control the program execution flow.
Used to make the execution flow "jump forward" or "jump backward".

Conditional Control Statements



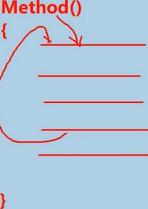
Looping Control Statements



Control 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

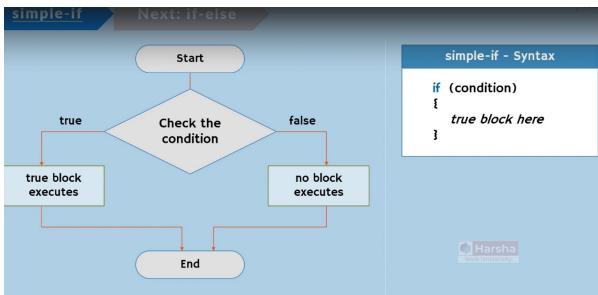


Looping Control Statements

What are Control Statements? **Next: simple-if**

What	<ul style="list-style-type: none"> > Control Statements are used to control the program execution flow. > Used to make the execution flow "jump forward" or "jump backward".
Classification	<ul style="list-style-type: none"> > Conditional Control Statements <ul style="list-style-type: none"> > if (simple-if, if-else, else-if, nested-if) > switch-Case > Looping Control Statements <ul style="list-style-type: none"> > while > do-While > for > Jumping Control Statements <ul style="list-style-type: none"> > goto > break > continue

Harsha
www.tutorialspoint.com



simple-if - Example

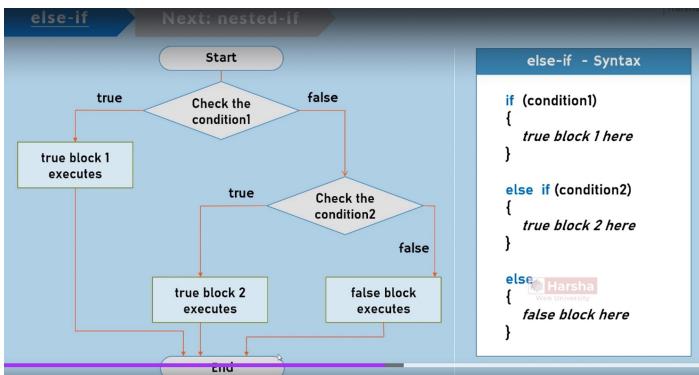
```

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

if-else - Example

```

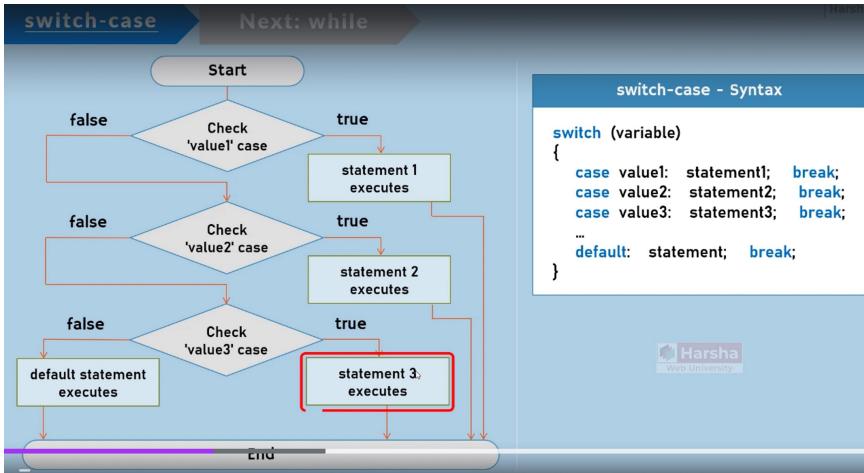
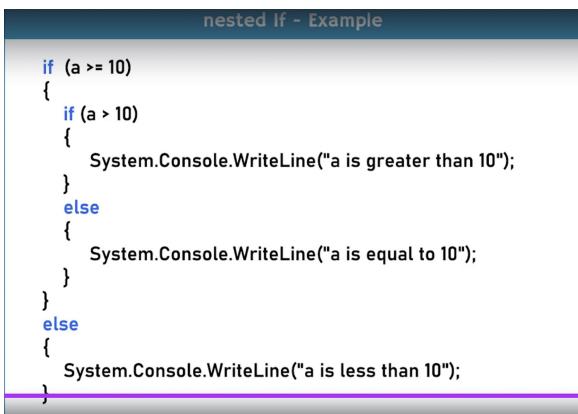
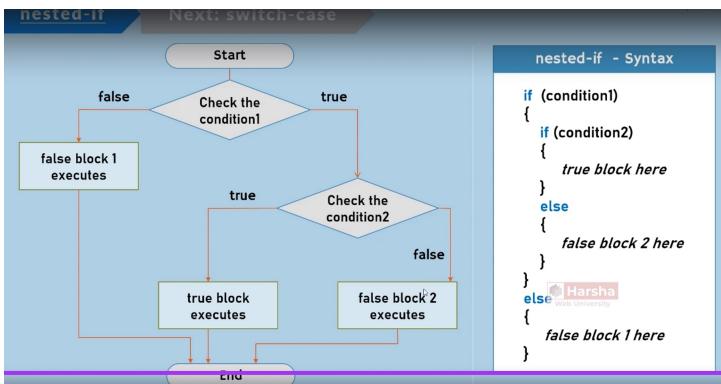
if (x > 10)
{
    System.Console.WriteLine("x is larger than 10");
}
else
{
    System.Console.WriteLine("x is smaller than or equal to 10");
}
  
```



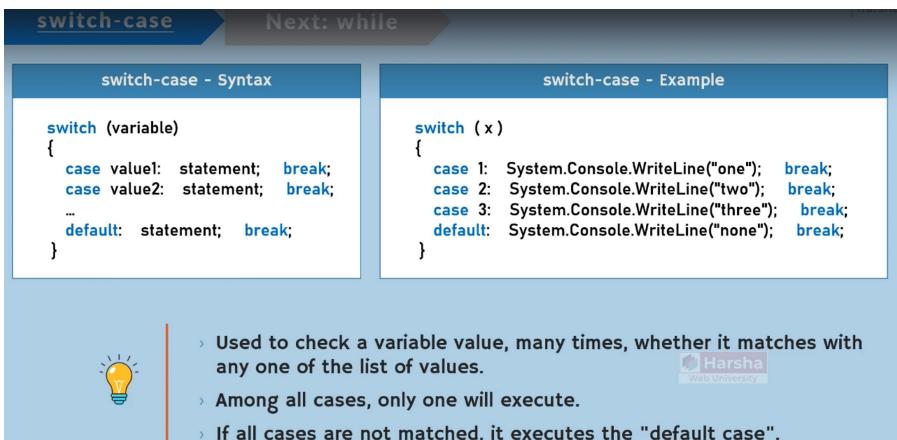
else If - Example

```

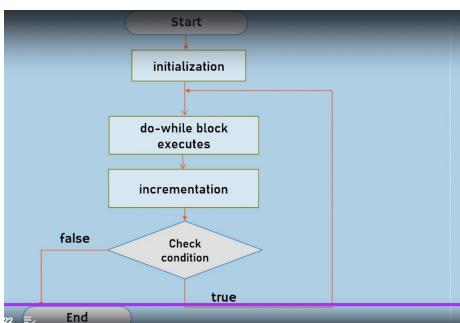
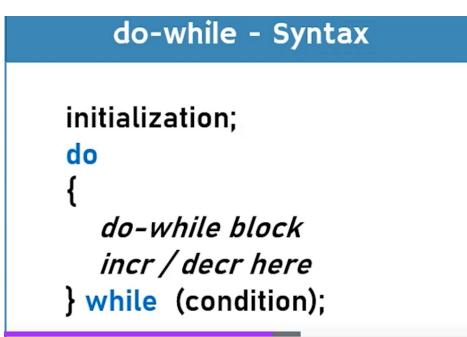
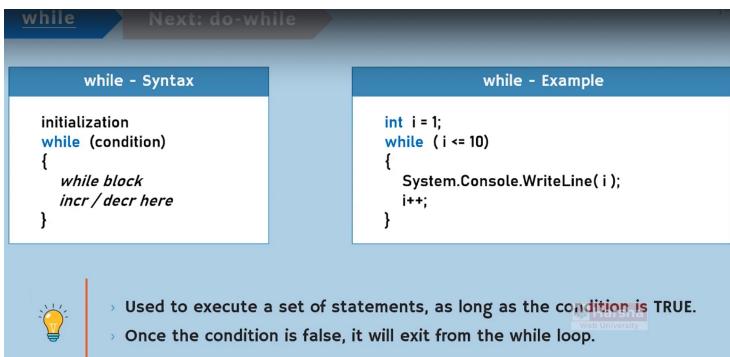
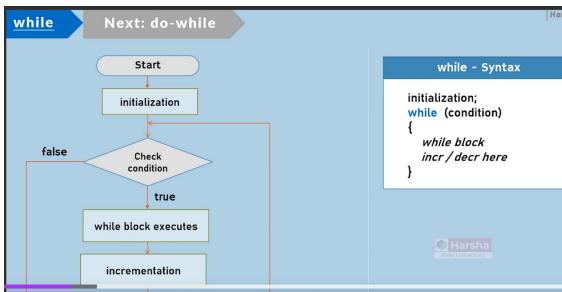
if (a > 10)
{
    System.Console.WriteLine("a is greater than 10");
}
else if (a < 10)
{
    System.Console.WriteLine("a is less than 10");
}
else
{
    System.Console.WriteLine("a is equal to 10");
}
  
```



6. Switch



7. While, Do-While



do-while Next: **for**

do-while - Syntax

```
do
{
    do-while block
    incr / decr here
} while (condition);
```

do-while - Example

```
int i = 1;
do
{
    System.Console.WriteLine(i);
} while (i <= 10);
```

Used to execute a set of statements, as long as the condition is TRUE.
 Once the condition is false, it will exit from the while loop.
 It is same as "While loop"; but the difference is:
 It executes at least one time even though the condition is false, because it doesn't check the condition for the first time.
 Second time onwards, it is same as "while" loop.

8. For Loop

for - Syntax

```
for (initialization; condition; incrementation)
{
    for block
}
```

for Next: **break**

```

graph TD
    Start([Start]) --> Initialization[initialization]
    Initialization --> Check{Check condition}
    Check -- true --> Execute[for block executes]
    Execute --> Increment[incrementation]
    Increment --> Check
    Check -- false --> END([END])
    
```

for - Syntax

```
for (initialization; condition; incrementation)
{
    for block
}
```

A screenshot of the Visual Studio IDE showing a C# program named 'Program.cs'. The code contains a Main() method with a for loop that prints numbers from 1 to 10 to the console. Annotations with arrows point to specific parts of the code: 'Initialization' points to the 'int i = 1' part, 'Condition' points to the 'i <= 10' part, and 'Incrementation' points to the 'i++' part. Below the code, the output window shows the numbers 0 to 9.

```

class Program
{
    static void Main()
    {
        //1 to 10
        for (int i = 1; i <= 10; i++)
        {
            System.Console.WriteLine(i + " ");
        }
        System.Console.WriteLine();
        //0 to 9
        i = 0;
    }
}

```

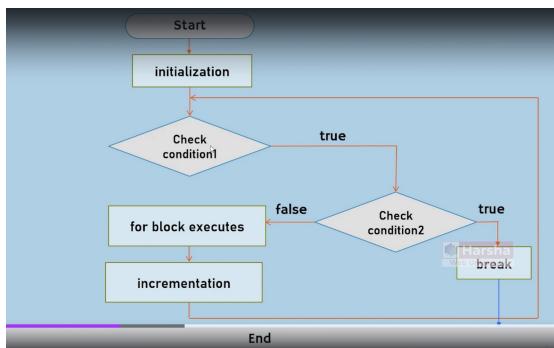
9. Break

Agenda

- 7 **break**
- 8 **continue**
- 9 **nested-for**
- 10 **goto**

break - Syntax

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



break Next: continue

break - Syntax	break - Example
<pre>for (initialization; condition1; incrementation) { if (condition2) { break; } for block code here }</pre>	<pre>for (int i = 0; i <= 10; i++) { if (i == 6) { break; } System.Console.WriteLine(i); }</pre> <p>//Output: 0, 1, 2, 3, 4, 5</p>

Tip:

- Used to stop the execution of current loop.
- It is recommended to keep the "break" statement, inside "if" statement.
- It can be used in any type of loop (while, do-while, for).

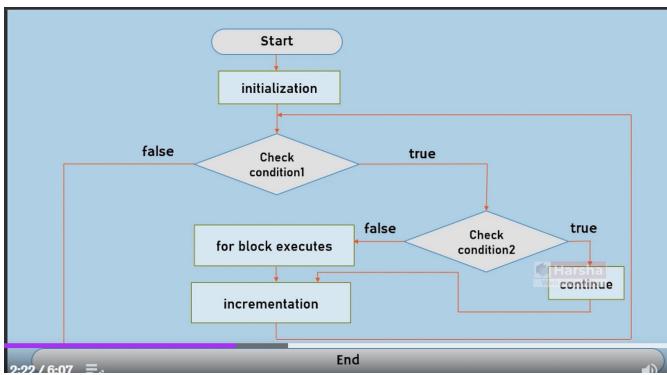
Agenda

- 7 break
- 8 continue
- 9 nested-for
- 10 goto

10. Continue

continue - Syntax

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



continue Next: [goto](#)

continue - Syntax

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

continue - Example

```
for (int i = 0; i <= 10; i++)
{
  if (i == 6)
  {
    continue;
  }
  System.Console.WriteLine(i);
}
//Output: 0, 1, 2, 3, 4, 5, 7, 8, 9, 10
```

💡 Used to skip the execution of current iteration; and jump to the next iteration.
It is recommended to keep the "continue" statement, inside "if" statement.
It can be used in any type of loop (while, do-while, for).

11. Nested For Loop

Agenda

- 7 [break](#)
- 8 [continue](#)
- 9 **nested-for**
- 10 [goto](#)

12. Go-To Statement

goto - Syntax

```
statement1;
statement2;
labelname: <-----+
statement3;
statement4;
goto labelname;
```

Goto - Example

```
System.Console.WriteLine("one");
System.Console.WriteLine("two");
mylabel: <-----+
System.Console.WriteLine("three");
System.Console.WriteLine("four");
goto mylabel;
System.Console.WriteLine("five");
```

```
{  
    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; j = 1  
    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");  
    i++;  
    if (i <= 5)  
    {  
        goto mylabel;  
    }  
    System.Console.WriteLine("Nepal");  
    System.Console.WriteLine("Dubai");  
  
    System.Console.ReadKey();
```