**The City School**

**North Nazimabad Boys Campus**

 **Date: 05-11-2016 Class: 11 Subject: Computer Science Teacher: Lubna Tanweer**

**Console.** A console program has no graphics. It is text. Easy to develop, it uses few resources and is efficient. It will win no visual design awards.
**These programs,** however, will readily accomplish an analytical or processing task. We invoke WriteLine and Write. We can use ReadLine for input. Numbers and strings are handled.
**An example.** This program uses Console.WriteLine. It prints "Hello world" to the screen. The program is contained in a module named Module1. The Sub Main is the entry point of the program.

# First program

Module Module1

 Sub Main()

 ' Say hi in VB.NET.

 **Console.WriteLine**("Hello world")

 Console.ReadKey()

 End Sub

End Module

**ReadLine.** Writing is most common with the Console type. But we can also read lines, from the keyboard, with ReadLine. The ReadLine Function returns a String.

**Here:**This program repeatedly calls ReadLine. It tests the input after the return key is pressed.

[**If Then**](https://www.dotnetperls.com/if-vbnet)

**Info:**We see if the user typed "1" or "2" and pressed return. We also display the output.

**VB.NET program that uses ReadLine**

Module Module1

 Sub Main()

 While True

 ' Read value.

 Dim s As String = **Console.ReadLine**()

 ' Test the value.

 If s = "1" Then

 Console.WriteLine("One")

 ElseIf s = "2" Then

 Console.WriteLine("Two")

 End If

 ' Write the value.

 Console.WriteLine("You typed " + s)

 End While

 End Sub

End Module

**Output**

1

One

You typed 1

2

Two

You typed 2

3

You typed 3

## Data Types Available in VB.Net

Module DataTypes

 Sub Main()

 Dim b As Byte

 Dim n As Integer

 Dim si As Single

 Dim d As Double

 Dim da As Date

 Dim c As Char

 Dim s As String

 Dim bl As Boolean

 b = 1

 n = 1234567

 si = 0.12345678901234566

 d = 0.12345678901234566

 da = Today

 c = "U"c

 s = "Me"

 If ScriptEngine = "VB" Then

 bl = True

 Else

 bl = False

 End If

 If bl Then

 'the oath taking

 Console.Write(c & " and," & s & vbCrLf)

 Console.WriteLine("declaring on the day of: {0}", da)

 Console.WriteLine("We will learn VB.Net seriously")

 Console.WriteLine("Lets see what happens to the floating point variables:")

 Console.WriteLine("The Single: {0}, The Double: {1}", si, d)

 End If

 Console.ReadKey()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

U and, Me

declaring on the day of: 12/4/2012 12:00:00 PM

We will learn VB.Net seriously

Lets see what happens to the floating point variables:

The Single:0.1234568, The Double: 0.123456789012346

## Accepting Values from User

The Console class in the System namespace provides a function **ReadLine** for accepting input from the user and store it into a variable. For example,

Dim message As String

message = Console.ReadLine

The following example demonstrates it:

Module variablesNdataypes

 Sub Main()

 Dim message As String

 Console.Write("Enter message: ")

 message = Console.ReadLine

 Console.WriteLine()

 Console.WriteLine("Your Message: {0}", message)

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result (assume the user inputs Hello World):

Enter message: Hello World

Your Message: Hello World

# VB.Net - Constants and Enumerations

The **constants** refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.

Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well.

The constants are treated just like regular variables except that their values cannot be modified after their definition.

An **enumeration** is a set of named integer constants.

## Declaring Constants

Where, each constant name has the following syntax and parts:

constantname [ As datatype ] = initializer

* ***constantname***: specifies the name of the constant
* ***datatype***: specifies the data type of the constant
* ***initializer***: specifies the value assigned to the constant

For example,

'The following statements declare constants.'

Const maxval As Long = 4999

Public Const message As String = "HELLO"

Private Const piValue As Double = 3.1415

## Example

The following example demonstrates declaration and use of a constant value:

Module constantsNenum

 Sub Main()

 Const PI = 3.14149

 Dim radius, area As Single

 radius = 7

 area = PI \* radius \* radius

 Console.WriteLine("Area = " & Str(area))

 Console.ReadKey()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

Area = 153.933

# VB.Net - Modifiers

The modifiers are keywords added with any programming element to give some especial emphasis on how the programming element will behave or will be accessed in the program

For example, the access modifiers: Public, Private, Protected, Friend, Protected Friend, etc., indicate the access level of a programming element like a variable, constant, enumeration or a class.

|  |  |
| --- | --- |
| Private | Specifies that one or more declared programming elements are accessible only from within their declaration context, including from within any contained types. |
|  | Protected | Specifies that one or more declared programming elements are accessible only from within their own class or from a derived class. |
|  | Public | Specifies that one or more declared programming elements have no access restrictions. |
|  |  |  |

# VB.Net - Statements

A **statement** is a complete instruction in Visual Basic programs. It may contain keywords, operators, variables, literal values, constants and expressions.

Statements could be categorized as:

* **Declaration statements** - these are the statements where you name a variable, constant, or procedure, and can also specify a data type.
* **Executable statements** - these are the statements, which initiate actions. These statements can call a method or function, loop or branch through blocks of code or assign values or expression to a variable or constant. In the last case, it is called an Assignment statement.

## Declaration Statements

The declaration statements are used to name and define procedures, variables, properties, arrays, and constants. When you declare a programming element, you can also define its data type, access level, and scope.

The programming elements you may declare include variables, constants, enumerations, classes, structures, modules, interfaces, procedures, procedure parameters, function returns, external procedure references, operators, properties, events, and delegates.

Following are the declaration statements in VB.Net:

|  |  |  |
| --- | --- | --- |
| **S.N** | **Statements and Description** | **Example** |
| 1 | **Dim Statement**Declares and allocates storage space for one or more variables. | Dim number As IntegerDim quantity As Integer = 100Dim message As String = "Hello!" |
| 2 | **Const Statement** Declares and defines one or more constants. | Const maximum As Long = 1000Const naturalLogBase As Object = CDec(2.7182818284) |
| 3 | **Enum Statement** Declares an enumeration and defines the values of its members. | Enum CoffeeMugSize Jumbo ExtraLarge Large Medium SmallEnd Enum  |
| 4 | **Class Statement**Declares the name of a class and introduces the definition of the variables, properties, events, and procedures that the class comprises. | Class BoxPublic length As DoublePublic breadth As Double Public height As DoubleEnd Class |
| 5 | **Structure Statement**Declares the name of a structure and introduces the definition of the variables, properties, events, and procedures that the structure comprises. | Structure BoxPublic length As Double Public breadth As Double Public height As DoubleEnd Structure |
| 6 | **Module Statement**Declares the name of a module and introduces the definition of the variables, properties, events, and procedures that the module comprises. | Public Module myModuleSub Main()Dim user As String = InputBox("What is your name?") MsgBox("User name is" & user)End Sub End Module |
| 7 | **Interface Statement**Declares the name of an interface and introduces the definitions of the members that the interface comprises. | Public Interface MyInterface Sub doSomething()End Interface  |
| 8 | **Function Statement**Declares the name, parameters, and code that define a Function procedure. | Function myFunction(ByVal n As Integer) As Double  Return 5.87 \* nEnd Function |
| 9 | **Sub Statement**Declares the name, parameters, and code that define a Sub procedure. | Sub mySub(ByVal s As String) ReturnEnd Sub  |
| 10 | **Declare Statement**Declares a reference to a procedure implemented in an external file. | Declare Function getUserNameLib "advapi32.dll" Alias "GetUserNameA" ( ByVal lpBuffer As String,  ByRef nSize As Integer) As Integer  |
| 11 | **Operator Statement**Declares the operator symbol, operands, and code that define an operator procedure on a class or structure. | Public Shared Operator +(ByVal x As obj, ByVal y As obj) As obj Dim r As New obj' implemention code for r = x + y Return r End Operator  |
| 12 | **Property Statement**Declares the name of a property, and the property procedures used to store and retrieve the value of the property. | ReadOnly Property quote() As String  Get  Return quoteString End Get End Property |
| 13 | **Event Statement**Declares a user-defined event. | Public Event Finished() |
| 14 | **Delegate Statement**Used to declare a delegate. | Delegate Function MathOperator(  ByVal x As Double,  ByVal y As Double ) As Double  |

## Executable Statements

An executable statement performs an action. Statements calling a procedure, branching to another place in the code, looping through several statements, or evaluating an expression are executable statements. An assignment statement is a special case of an executable statement.

**Example**

The following example demonstrates a decision making statement:

Module decisions

 Sub Main()

 'local variable definition '

 Dim a As Integer = 10

 ' check the boolean condition using if statement '

 If (a < 20) Then

 ' if condition is true then print the following '

 Console.WriteLine("a is less than 20")

 End If

 Console.WriteLine("value of a is : {0}", a)

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

a is less than 20;

value of a is : 10

# VB.Net - Decision Making

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages:



VB.Net provides the following types of decision making statements. Click the following links to check their details.

|  |  |
| --- | --- |
| **Statement** | **Description** |
| [**If ... Then statement**](https://www.tutorialspoint.com/vb.net/vb.net_if_statements.htm) | An **If...Then statement** consists of a boolean expression followed by one or more statements. |
| [**If...Then...Else statement**](https://www.tutorialspoint.com/vb.net/vb.net_if_else_statements.htm) | An **If...Then statement** can be followed by an optional **Else statement**, which executes when the boolean expression is false. |
| [**nested If statements**](https://www.tutorialspoint.com/vb.net/vb.net_nested_if_statements.htm) | You can use one **If** or **Else if** statement inside another **If** or **Else if** statement(s). |
| [**Select Case statement**](https://www.tutorialspoint.com/vb.net/vb.net_select_case_statements.htm) | A **Select Case** statement allows a variable to be tested for equality against a list of values. |

# VB.Net - Loops

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:



VB.Net provides following types of loops to handle looping requirements. Click the following links to check their details.

|  |  |
| --- | --- |
| **Loop Type** | **Description** |
| [**Do Loop**](https://www.tutorialspoint.com/vb.net/vb.net_do_loops.htm) | It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement. |
| [**For...Next**](https://www.tutorialspoint.com/vb.net/vb.net_fornext_loops.htm) | It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes. |
| [**For Each...Next**](https://www.tutorialspoint.com/vb.net/vb.net_foreachnext_loops.htm) | It repeats a group of statements for each element in a collection. This loop is used for accessing and manipulating all elements in an array or a VB.Net collection. |
| [**While... End While**](https://www.tutorialspoint.com/vb.net/vb.net_while_loops.htm) | It executes a series of statements as long as a given condition is True. |

# VB.Net - If...Then Statement

It is the simplest form of control statement, frequently used in decision making and changing the control flow of the program execution. Syntax for if-then statement is:

If condition Then

[Statement(s)]

End If

Where, *condition* is a Boolean or relational condition and Statement(s) is a simple or compound statement. Example of an If-Then statement is:

If (a <= 20) Then

 c= c+1

End If

If the condition evaluates to true, then the block of code inside the If statement will be executed. If condition evaluates to false, then the first set of code after the end of the If statement (after the closing End If) will be executed.

## Flow Diagram:



## Example:

Module decisions

 Sub Main()

 'local variable definition

 Dim a As Integer = 10

 ' check the boolean condition using if statement

 If (a < 20) Then

 ' if condition is true then print the following

 Console.WriteLine("a is less than 20")

 End If

 Console.WriteLine("value of a is : {0}", a)

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

a is less than 20

value of a is : 10

# VB.Net - If...Then...Else Statement

An **If** statement can be followed by an optional **Else** statement, which executes when the Boolean expression is false.

## Syntax:

The syntax of an If...Then... Else statement in VB.Net is as follows:

If(boolean\_expression)Then

 'statement(s) will execute if the Boolean expression is true

Else

 'statement(s) will execute if the Boolean expression is false

End If

If the Boolean expression evaluates to **true**, then the if block of code will be executed, otherwise else block of code will be executed.

## Flow Diagram:



## Example:

Module decisions

 Sub Main()

 'local variable definition '

 Dim a As Integer = 100

 ' check the boolean condition using if statement

 If (a < 20) Then

 ' if condition is true then print the following

 Console.WriteLine("a is less than 20")

 Else

 ' if condition is false then print the following

 Console.WriteLine("a is not less than 20")

 End If

 Console.WriteLine("value of a is : {0}", a)

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

a is not less than 20

value of a is : 100

## The If...Else If...Else Statement

An **If** statement can be followed by an optional **Else if...Else** statement, which is very useful to test various conditions using single If...Else If statement.

When using If... Else If... Else statements, there are few points to keep in mind.

* An If can have zero or one Else's and it must come after an Else If's.
* An If can have zero to many Else If's and they must come before the Else.
* Once an Else if succeeds, none of the remaining Else If's or Else's will be tested.

## Syntax:

The syntax of an if...else if...else statement in VB.Net is as follows:

If(boolean\_expression 1)Then

 ' Executes when the boolean expression 1 is true

ElseIf( boolean\_expression 2)Then

 ' Executes when the boolean expression 2 is true

ElseIf( boolean\_expression 3)Then

 ' Executes when the boolean expression 3 is true

Else

 ' executes when the none of the above condition is true

End If

## Example:

Module decisions

 Sub Main()

 Dim a As Integer = 100

 If (a = 10) Then

 Console.WriteLine("Value of a is 10")

 ElseIf (a = 20) Then

 Console.WriteLine("Value of a is 20")

 ElseIf (a = 30) Then

 Console.WriteLine("Value of a is 30")

 Else

 Console.WriteLine("None of the values is matching")

 End If

 Console.WriteLine("Exact value of a is: {0}", a)

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

None of the values is matching

Exact value of a is: 100

# VB.Net - Select Case Statement

A **Select Case** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each select case.

## Syntax:

The syntax for a Select Case statement in VB.Net is as follows:

Select [ Case ] expression

 [ Case expressionlist

 [ statements ] ]

 [ Case Else

 [ elsestatements ] ]

End Select

Where,

* ***expression***: is an expression that must evaluate to any of the elementary data type in VB.Net, i.e., Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong, and UShort.
* ***expressionlist***: List of expression clauses representing match values for *expression*. Multiple expression clauses are separated by commas.
* ***statements***: statements following Case that run if the select expression matches any clause in *expressionlist*.
* ***elsestatements***: statements following Case Else that run if the select expression does not match any clause in the *expressionlist* of any of the Case statements.

## Flow Diagram:



## Example:

Module decisions

 Sub Main()

 'local variable definition

 Dim grade As Char

 grade = "B"

 Select grade

 Case "A"

 Console.WriteLine("Excellent!")

 Case "B", "C"

 Console.WriteLine("Well done")

 Case "D"

 Console.WriteLine("You passed")

 Case "F"

 Console.WriteLine("Better try again")

 Case Else

 Console.WriteLine("Invalid grade")

 End Select

 Console.WriteLine("Your grade is {0}", grade)

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

Well done

Your grade is B

# VB.Net - Do Loop

It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement.

The syntax for this loop construct is:

Do { While | Until } condition

 [ statements ]

 [ Continue Do ]

 [ statements ]

 [ Exit Do ]

 [ statements ]

Loop

-or-

Do

 [ statements ]

 [ Continue Do ]

 [ statements ]

 [ Exit Do ]

 [ statements ]

Loop { While | Until } condition

## Flow Diagram:



## Example:

Module loops

 Sub Main()

 ' local variable definition

 Dim a As Integer = 10

 'do loop execution

 Do

 Console.WriteLine("value of a: {0}", a)

 a = a + 1

 Loop While (a < 20)

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

The program would behave in same way, if you use an Until statement, instead of While:

Module loops

 Sub Main()

 Dim a As Integer = 10

 'do loop execution

 Do

 Console.WriteLine("value of a: {0}", a)

 a = a + 1

 Loop Until (a = 20)

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

# VB.Net - For...Next Loop

It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.

The syntax for this loop construct is:

For counter [ As datatype ] = start To end [ Step step ]

 [ statements ]

 [ Continue For ]

 [ statements ]

 [ Exit For ]

 [ statements ]

Next [ counter ]

## Flow Diagram:



## Example

Module loops

 Sub Main()

 Dim a As Byte

 ' for loop execution

 For a = 10 To 20

 Console.WriteLine("value of a: {0}", a)

 Next

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

value of a: 20

If you want to use a step size of 2, for example, you need to display only even numbers, between 10 and 20:

Module loops

 Sub Main()

 Dim a As Byte

 ' for loop execution

 For a = 10 To 20 Step 2

 Console.WriteLine("value of a: {0}", a)

 Next

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

value of a: 10

value of a: 12

value of a: 14

value of a: 16

value of a: 18

value of a: 20

# VB.Net - While... End While Loop

It executes a series of statements as long as a given condition is True.

The syntax for this loop construct is:

While condition

 [ statements ]

 [ Continue While ]

 [ statements ]

 [ Exit While ]

 [ statements ]

End While

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is logical true. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

## Flow Diagram:



Here, key point of the *While* loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

## Example 11

Module loops

 Sub Main()

 Dim a As Integer = 10

 While a < 20

 Console.WriteLine("value of a: {0}" & a)

 a = a + 1

 End While

 Console.ReadLine()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

Q8 Chapter 10

A small café sells five types of items:

Bun $0.50

Coffee $1.20

Cake $1.50

Sandwich $2.10

Dessert $4.00

Write a program, which

* Input every item sold during the day
* Uses an item called “end” to finish the day’s input
* Adds up the daily amount taken for each type of item
* Outputs the total takings ( for all items added together ) at the end of the day
* Output the item that had the highest takings at the end of the day

VB program

Module Module1

Sub Main( )

Dim Tbun, Tcoffee, Tcake, Tsandwich, Tdessert, quantity, TotalTakings, HighestTaking As Integer = 0

Dim Item As String

while ( Item < > “end” )

Console.writeline ( “Enter the item in lower case only”)

Item = console.readline( )

Console.writeline ( “Enter its quantity”)

quantity = Int(console.readline( ))

Select Item

 Case “bun”

 Tbun = Tbun + quantity

Case “coffee”

 Tcoffee = Tcoffee + quantity

 Case “cake”

 Tcake = Tcake + quantity

Case “sandwich”

 Tsandwich = Tsandwich + quantity

 Case “dessert”

 Tdessert = Tdessert + quantity

 Case Else

 Console.writeline(“ Enter relevant product ”)

End Select

End While

TotalTakings = Tbun + Tcoffee + Tcake + Tsandwich + Tdessert

Console.writeline(“The total takings of the whole day” & TotalTakings)

If (Tbun > HighestTaking) Then

HighestTaking = Tbun

 Item = “Bun”

ElseIf (Tcoffee > HighestTaking) Then

HighestTaking = Tcoffee

 Item = “Coffee”

ElseIf ( Tcake > HighestTaking) Then

HighestTaking = Tcake

 Item = “Cake”

ElseIf ( Tsandwich > HighestTaking) Then

HighestTaking = Tsandwich

 Item = “Sandwich”

ElseIf (Tdessert > HighestTaking) Then

HighestTaking = Tdessert

 Item = “Dessert”

End If

Console.writeline(“The item which has the highest sales today is : ” & Item)

Console.readkey( )

End Sub( )

End Module( )

Q7: A school is doing a check on the heights and weights of the students. The school has 1000 students. Write an Pseudocode and program in VB, which:

* Input height and weight of all 1000 students
* Output the average height and weight
* Include any necessary error traps for the input

Begin

TotalWeight =0

TotalHeight =0

For x= 1 to 1000

Repeat

Input height, weight

Until (height > 30) and (height < 80) and (weight > 30 ) and ( weight < 100)

TotalWeight = TotalWeight + weight

TotalHeight = TotalHeight + height

Next

AverageHeight = TotalHeight / 1000

AverageWeight = TotalWeight / 1000

Output “ Average height of the students is : ”, AverageHeight

Output “ Average weight of the students is : ”, AverageWeight

End

# VB.Net - Arrays

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

All arrays consist of continuous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



## Creating Arrays in VB.Net

To declare an array in VB.Net, you use the Dim statement. For example,

Dim intData(30) As Integer ' an array of 31 elements

Dim strData(20) As String ' an array of 21 strings

You can also initialize the array elements while declaring the array. For example,

Dim intData() As Integer = {12, 16, 20, 24, 28, 32}

Dim names() As String = {"Karthik", "Sandhya", \_

"Shivangi", "Ashwitha", "Somnath"}

The elements in an array can be stored and accessed by using the index of the array. The following program demonstrates this:

Module Module1

 Sub Main()

 Dim n(10) As Integer ' n is an array of 11 integers '

 Dim i, j As Integer

 For i = 0 To 10

 n(i) = i + 100

 Next i

 ' output each array element's value '

 For j = 0 To 10

 Console.WriteLine("Element({0}) = {1}", j, n(j))

 Next j

 Console.ReadKey()

 End Sub

End Module

When the above code is compiled and executed, it produces the following result:

Element(0) = 100

Element(1) = 101

Element(2) = 102

Element(3) = 103

Element(4) = 104

Element(5) = 105

Element(6) = 106

Element(7) = 107

Element(8) = 108

Element(9) = 109

Element(10) = 110