UNIT II VISUAL BASIC

TOPIC OUTLINE

USE VISUAL BASIC TO WRITE AND RUN SIMPLE PROGRAMS

1. **EXPLAIN HOW TO WRITE SIMPLE PROGRAMS IN VISUAL BASIC**
2. **WRITE SIMPLE PROGRAMS IN VISUAL BASIC ACCORDING TO GUIDELINES**
3. **ESSENTIAL CONCEPTS FOR WRITING SIMPLE PROGRAMS**

ESSENTIAL CONCEPTS FOR WRITING SIMPLE VISUAL BASIC PROGRAMS

a) Class
b) Modules
c) Variables
d) Object
e) Methods
f) Functions
g) Decision making (control structures)
h) Operators

CLASS

Class is defined as a blueprint or a template used to create objects. It serves as a foundational structure that defines the specific data (variables) and actions (methods or subroutines) that an object created from that class will possess.

For example;

Class: A plan for a person that specifies they have a name and can speak.

Object: A specific person, like "John," who is 21 years old and is an actual instance of that "Person" plan.

Syntax of a class;

```
Public Class ClassName
' Variables (Fields)
Public variableName As DataType
' Method (Procedure)
Public Sub MethodName()
' Method code
End Sub
End Class
```

Example code;

```
Public Class Student
    ' Fields (variables)
    Public Name As String
    Public Age As Integer
    ' Method
    Public Sub DisplayInfo()
        Console.WriteLine("Name: " & Name)
        Console.WriteLine("Age: " & Age)
    End Sub
End Class
```

OBJECT

An **object** in Visual Basic is a real instance of a class. It represents a real-world entity and allows you to access the variables, properties, and methods defined in a class. An object is an instance of a class that occupies memory and is used to access the class members.

In simple words:
A **class** is a *blueprint*, and an object is the *actual thing created from that blueprint*.

Example

Class

```
Public Class Student
    Public Name As String
    Public Age As Integer

    Public Sub Display()
        Console.WriteLine("Name: " & Name)
        Console.WriteLine("Age: " & Age)
    End Sub
End Class
```

Object Creation

Dim s1 As New Student()

Here:

- Student → Class
- s1 → Object
- New → Allocates memory and creates the object

Using the Object

s1.Name = "Salum"

s1.Age = 20

s1.Display() …… here the object tries to call the method of the class student which is method display.

The object s1 is used to:

- Store data
- Call methods of the class

From the above explanations to use the class and object together the full code on how to write it will be as follows; when using class and object use both class and module as follows

```
' Define a class
Public Class Student

    ' Variables (Fields)
    Public Name As String
    Public Age As Integer

    ' Method
    Public Sub Display()
        Console.WriteLine("Student Name: " & Name)
        Console.WriteLine("Student Age: " & Age)
    End Sub
End Class
```

```
' Main module
Module Module1

    Sub Main()

        ' Creating an object of the class Student
        Dim s1 As New Student()

        ' Assigning values using the object
        s1.Name = "Salum"
        s1.Age = 20

        ' Calling the method using the object
        s1.Display()
        Console.ReadLine()

    End Sub
End Module
```

Key Characteristics of an Object

- Created using the New keyword
- Uses memory
- Accesses class members using dot (.) operator
- Multiple objects can be created from one class

VARIABLES

A **variable** is **a** named storage location in **memory** that holds a **value**. In simple words: A variable is like a box where you can store data and use it later.

Why Variables Are Important

a) Store **user input**
b) Store **calculated results**
c) Make programs **dynamic** (data can change while the program runs)
d) Give **names to values** for easy reference

Variable Declaration

In VB, you **declare a variable** using Dim keyword:

Dim variableName As DataType

Example:

Dim age As Integer
Dim name As String
Dim salary As Double
Dim isStudent As Boolean

- age → Integer (whole number)
- name → String (text)
- salary → Double (decimal number)
- isStudent → Boolean (True/False)

Variable Initialization

You can **assign a value** while declaring:

Dim age As Integer = 20

Dim name As String = "Salum"

Dim salary As Double = 1500.50

Dim isStudent As Boolean = True

Or assign later**:**

Dim age As Integer
age = 25

Rules for Naming Variables

a) Must **start with a letter**
b) Can contain letters, numbers, or underscore _
c) Cannot use **spaces**
d) Cannot use **reserved keywords** (like If, For, Module)

e) Should be **meaningful**

Examples: Accepted

```
Dim studentName As String
Dim totalMarks As Integer
Dim isRegistered As Boolean
```

Bad examples: Not accepted

```
Dim 1name As String   ' starts with a number
Dim for As Integer    ' reserved word
Dim my age As Integer ' contains space
```

Variable Scope

The **scope** of a variable tells where it can be used:

1. **Local Variables** – Declared inside a Sub or Function. Can be used **only inside that procedure**.

```
Sub Main()
   Dim age As Integer = 20  ' Local variable
   Console.WriteLine(age)
End Sub
```

2. **Module-Level Variables** – Declared inside **a** module **but outside procedures. Can be used** anywhere inside the module**.**

```
Module Module1
   Dim count As Integer = 0   ' Module-level variable
End Module
```

3. **Global Variables** – Declared with Public in a module. Can be used **anywhere in the project**.

```
Module Module1
   Public totalStudents As Integer = 50
End Module
```

MODULES

A **Module** is a container of code that holds:

- Variables
- Procedures (Sub)
- Functions (Function)

A module is used to organize code and run a program **in a** console application**.** In simple words:

A module is a place where your VB program starts and where shared code is written.

Modules are used to:

a) Start program execution (Sub Main)
b) Group related code together
c) Share variables and methods across the program
d) Avoid creating objects (no instantiation needed)

Structure of the module;

```
Module Module1

    Sub Main()

        Console.WriteLine("Hello VB")

        Console.ReadKey()

    End Sub

End Module
```

Example programs;

1. Displaying user name and age

```
Module Module1

    Sub Main()

        Dim name As String
```

```
        Dim age As Integer
        name = "Salum"
        age = 20
        Console.WriteLine("Name: " & name)
        Console.WriteLine("Age: " & age)
        Console.ReadKey()
    End Sub
End Module
```

2. User to input data both age and name

```
Module Module1
    Sub Main()
        Dim name As String
        Dim age As Integer
        ' Ask user to enter their name
        Console.Write("Enter your name: ")
        name = Console.ReadLine()
        ' Ask user to enter their age
        Console.Write("Enter your age: ")
        age = Integer.Parse(Console.ReadLine())  ' Convert input text to number
        ' Display the entered data
        Console.WriteLine("Name: " & name)
        Console.WriteLine("Age: " & age)
        Console.ReadKey()
```

```
    End Sub

End Module
```

OPERATORS

An **operator** is a **symbol or keyword** that tells VB to perform an **operation** on **one or more values (operands)**.

Types of operators

a) Arithmetic Operators
b) Comparison Operators
c) Logical Operators
d) Assignment Operators
e) String (Concatenation) Operator

1. ARITHMETIC OPERATORS

Used to do **math calculations**:

- + → Adds numbers. Example: 5 + 3 gives 8.
- - → Subtracts numbers. Example: 10 - 4 gives 6.
- * → Multiplies numbers. Example: 2 * 3 gives 6.
- / → Divides numbers and returns a decimal. Example: 10 / 4 gives 2.5.
- \ → Integer division (returns whole number). Example: 10 \ 4 gives 2.
- Mod → Remainder of division. Example: 10 Mod 3 gives 1.
- ^ → Exponent. Example: 2 ^ 3 gives 8.

Code example:

```
Dim a As Integer = 10
Dim b As Integer = 3

Console.WriteLine(a + b)  ' 13
Console.WriteLine(a / b)  ' 3.3333
Console.WriteLine(a Mod b) ' 1
```

2. COMPARISON OPERATORS

Used to **compare values**. They return **True or False**.

- = → Checks if two values are equal. Example: x = 5
- <> → Checks if two values are not equal. Example: x <> 5
- > → Checks if a value is greater. Example: x > 3
- < → Checks if a value is less. Example: x < 10
- >= → Greater than or equal to
- <= → Less than or equal to

Code example:

```
Dim x As Integer = 7
Console.WriteLine(x > 5)   ' True
Console.WriteLine(x = 10)  ' False
```

3. LOGICAL OPERATORS

Used to **combine or reverse True/False values**:

- And → True if both conditions are True. Example: (x > 5) And (x < 10)
- Or → True if at least one condition is True. Example: (x > 5) Or (x < 3)
- Not → Reverses True/False. Example: Not(True) gives False
- Xor → True if only one condition is True (not both)

Code example:

```
Dim x As Boolean = True
Dim y As Boolean = False

Console.WriteLine(x And y)  ' False
Console.WriteLine(x Or y)   ' True
Console.WriteLine(Not x)    ' False
```

4. ASSIGNMENT OPERATORS

Used to **store values in variables** or update them:

- = → Assigns a value to a variable. Example: x = 5
- += → Adds and assigns. Example: x += 3 is same as x = x + 3
- -= → Subtracts and assigns
- *= → Multiplies and assigns
- /= → Divides and assigns
- \= → Integer division and assigns
- Mod= → Finds remainder and assigns

Code example:

```
Dim x As Integer = 10
x += 5   ' x becomes 15
x *= 2   ' x becomes 30
```

5. STRING CONCATENATION

Used to **combine text**:

- & → Joins two strings. Example: "Hello" & " World" gives "Hello World"

Code example:

```
Dim firstName As String = "Salum"
Dim lastName As String = "Juma"
Console.WriteLine(firstName & " " & lastName)  ' Salum Juma
```

DECISION MAKING

Decision making is the process where a program **chooses a path to follow** based on a **condition**.

- VB uses **Boolean expressions** (True or False) to decide which block of code to execute.
- It allows programs to **respond differently under different conditions**.

Why Decision Making is Important

a) Programs can behave differently depending on user input.
b) Helps in performing tasks **only if certain conditions are met**.
c) Makes programs **dynamic and intelligent** instead of always doing the same thing.

Types of Decision Making Statements in VB

a) If...Then
b) If...Then...Else
c) If...Then...ElseIf...Else
d) Select Case

1. **If...Then Statement**

Executes a block of code **only if the condition is True**.

Syntax:

```
If condition Then
   ' Code to execute if condition is True
End If
```

Example:

```
Dim age As Integer = 20

If age >= 18 Then
   Console.WriteLine("You are an adult")
End If
```

Here, the message is displayed only if age >= 18.

2. **If...Then...Else Statement**

Executes one block if the condition is True, **another block if False**.

Syntax:

```
If condition Then
   ' Code if True
Else
   ' Code if False
End If
```

Example:

```
Dim age As Integer = 16

If age >= 18 Then
   Console.WriteLine("You are an adult")
Else
   Console.WriteLine("You are a minor")
End If
```

Output: "You are a minor" because age is less than 18.

3. **If...Then...ElseIf...Else Statement**

Used when there are **multiple conditions**.

Syntax:

```
If condition1 Then
   ' Code if condition1 is True
ElseIf condition2 Then
   ' Code if condition2 is True
Else
   ' Code if none of the above are True
End If
```

Example:

```
Dim marks As Integer = 75

If marks >= 90 Then
   Console.WriteLine("Grade A")
ElseIf marks >= 75 Then
   Console.WriteLine("Grade B")
ElseIf marks >= 50 Then
   Console.WriteLine("Grade C")
Else
   Console.WriteLine("Fail")
End If
```

Output: "Grade B" because marks are 75.

4. **Select Case Statement**

Used when you have **one variable and many possible values**. Cleaner than multiple ElseIf statements.

Syntax:

```
Select Case variable
   Case value1
      ' Code if variable = value1
   Case value2
      ' Code if variable = value2
   Case Else
      ' Code if variable does not match any value
End Select
```

Example:

```
Dim day As Integer = 3

Select Case day
   Case 1
      Console.WriteLine("Monday")
   Case 2
      Console.WriteLine("Tuesday")
   Case 3
      Console.WriteLine("Wednesday")
   Case Else
      Console.WriteLine("Other day")
End Select
```

Output: "Wednesday" because day = 3.

5. Logical Operators in Decision Making

a) And → True if both conditions are True
b) Or → True if at least one condition is True
c) Not → Reverses True/False

Example:

```
Dim age As Integer = 20
```

```
Dim hasID As Boolean = True

If age >= 18 And hasID Then
   Console.WriteLine("Allowed to enter")
Else
   Console.WriteLine("Not allowed")
End If
```

Output: "Allowed to enter" because both conditions are True.

6.Nested If Statements

You can put an If inside another If for complex conditions.

Example:

```
Dim age As Integer = 20
Dim isStudent As Boolean = True

If age >= 18 Then
   If isStudent Then
      Console.WriteLine("Adult student")
   Else
      Console.WriteLine("Adult non-student")
   End If
Else
   Console.WriteLine("Minor")
End If
```

Output: "Adult student"

METHODS

A **method** is a block of code that performs a specific task**. Methods help you** organize code, reuse it, and make your program modular**.**

In VB, methods are mainly of two types**:**

1. **Sub (Subroutine)**
2. **Function**

Why Use Methods?

a) Avoid writing the same code multiple times (**code reuse**)
b) Make programs **organized and readable**
c) Break complex tasks into **smaller, manageable pieces**
d) Can **accept inputs (parameters)** and sometimes **return results (functions)**

1. Sub (Subroutine)

A Sub performs a task but does not return a value**.**

Syntax:

```
Sub SubName([parameters])
   ' Code to execute
End Sub
```

Example: Subroutine without parameters/Values.

```
Sub Greet()
   Console.WriteLine("Hello! Welcome to VB.NET")
End Sub

Sub Main()
   Greet()  ' Calling the Sub
   Console.ReadKey()
End Sub
```

Example: Subroutine with parameters

```
Sub GreetUser(name As String)
   Console.WriteLine("Hello, " & name & "!")
End Sub


Sub Main()
   GreetUser("Salum")  ' Pass value to Sub
   Console.ReadKey()
End Sub
```

- name is a **parameter** passed to the Sub
- Sub **does not return any value**

Other methods (Inputs/Output methods)

1. Console.WriteLine()

Purpose: Displays output on the console **and moves to the next line**. **Usage:** Print messages, variables, or results to the screen.

Example:

```
Console.WriteLine("Hello, World!")
Console.WriteLine(10 + 5)
```

Output:

```
Hello, World!
15
```

2. Console.Write()

Purpose: Displays output on the console **but does NOT move to the next line**. **Usage:** Useful when you want to continue writing on the **same line**.

Example:

```
Console.Write("Hello ")
Console.Write("World!")
```

Output:

```
Hello World!
```

3. Console.ReadLine()

Purpose: Reads **input from the user** as text (string). **Usage:** Collect data typed by the user in the console.

Example:

```
Console.Write("Enter your name: ")
```

```
Dim name As String
name = Console.ReadLine()
Console.WriteLine("Hello, " & name & "!")
```

Sample Run:

```
Enter your name: Salum
Hello, Salum!
```

ReadLine() waits for the user to type something and press **Enter**. The input is always a **string**, so if you want a number, you need to **convert it**:

```
Console.Write("Enter your age: ")
Dim age As Integer
age = Integer.Parse(Console.ReadLine())
```

FUNCTIONS

A **function** is a block of code that performs a specific task and returns a value. Unlike a Sub, a **Function always returns a value**. You can pass **inputs (parameters)** to it and use the **output (return value)** in your program. In simple words: **A function takes input, does some work, and gives back a result.**

Why Functions Are Important

a) **Code Reusability** – Write once, use many times.
b) **Modularity** – Break complex programs into smaller tasks.
c) **Readability** – Makes programs easier to understand.
d) **Return Results** – Unlike Sub, functions give you a value to use.

Function Syntax

```
Function FunctionName([parameters]) As DataType
   ' Code to perform the task
   Return value
End Function
```

- FunctionName → Name of the function
- [parameters] → Optional input values
- As DataType → The type of value the function returns (Integer, String, Double, etc.)
- Return → Sends the result back to the caller

1.**Simple Function Without Parameters**

```
Function GetPi() As Double
   Return 3.14159
End Function

Sub Main()
   Dim piValue As Double
   piValue = GetPi()
   Console.WriteLine("Value of Pi: " & piValue)
   Console.ReadKey()
End Sub
```

- GetPi() always returns 3.14159.
- You **store the returned value** in a variable (piValue) and use it.

2.Function With Parameters

```
Function AddNumbers(a As Integer, b As Integer) As Integer
   Return a + b
End Function

Sub Main()
   Dim result As Integer
   result = AddNumbers(5, 10)
   Console.WriteLine("Sum = " & result)
   Console.ReadKey()
End Sub
```

a and b are **parameters** passed to the function. Function returns their sum, which is stored in result.

3.**Function Returning a String**

```
Function GreetUser(name As String) As String
   Return "Hello, " & name & "!"
End Function

Sub Main()
   Dim message As String
   message = GreetUser("Salum")
   Console.WriteLine(message)
```

```
    Console.ReadKey()
End Sub
```

Function can return **text** (String) instead of numbers.

4.**Function With Multiple Parameters and Calculation**

```
Function CalculateArea(length As Double, width As Double) As Double
    Return length * width
End Function

Sub Main()
    Dim area As Double
    area = CalculateArea(5, 3)
    Console.WriteLine("Area = " & area)
    Console.ReadKey()
End Sub
```

Input: length and width

Output: Area calculated and returned

Calling a Function

Calling a function means **executing or using a function that you have defined** in your program.

When you **call a function**, the program **jumps to the function code**, executes it, and **returns a value (if it's a Function)**. The returned value can be **stored in a variable, used in calculations, or displayed**. In simple words: **Calling a function is like asking the function to "do its job" and give you the result**.

Why We Call Functions

a) To **re-use code** without writing it again.
b) To **perform a task and get the result**.
c) To **make programs organized and modular**.

How to Call a Function

1. **Call a function and store its return value in a variable**

```
Function Add(a As Integer, b As Integer) As Integer
   Return a + b
End Function

Sub Main()
   Dim sum As Integer
   sum = Add(5, 10)   ' Calling the function
   Console.WriteLine("Sum = " & sum)
   Console.ReadKey()
End Sub
```

Add(5, 10) → This is the **function call**. The returned value 15 is stored in sum

2. **Use it directly inside another statement:**

```
Console.WriteLine("Sum = " & Add(3, 7))
```

Here, Add(3,7) is called **directly inside WriteLine**

ARRAY

An **array** is a **collection of variables of the same data type** stored under a **single name**. Instead of creating many separate variables (num1, num2, num3…), you can store all values in an **array**. Each value in the array is called an **element**. Elements are **accessed using an index** (position), starting from **0**. **In simple words**: An array is like a row of boxes, each holding a value, and you can pick any box by its number.

Why Use Arrays

a) Store **multiple values** in one variable.
b) Simplify **loops** and processing of multiple items.
c) Improve **readability** and **efficiency**.
d) Essential for tasks like **storing lists, sequences, or user inputs**.

Declaring Arrays in VB

1. **Fixed-size array**

Dim numbers(4) As Integer

Creates an array with **5 elements** (index 0 to 4). All elements are initialized to **0** (for numbers).

2. **Array with values (Initialization)**

Dim numbers() As Integer = {2, 4, 6, 8, 10}

Automatically creates an array of **5 elements** with given values.

3. **String array**

Dim names() As String = {"Salum", "Juma", "Amina"}

Stores **text values** instead of numbers.

Accessing Array Elements

Use the **index** (position) to read or update a value.

```
Dim numbers() As Integer = {10, 20, 30, 40, 50}
Console.WriteLine(numbers(2))  ' Output: 30
numbers(2) = 100               ' Change value at index 2
Console.WriteLine(numbers(2))  ' Output: 100
```

Note: Index always starts from **0**, so numbers(0) is the first element.

Array Length

To get the number of elements in an array:

Console.WriteLine(numbers.Length) ' Output: 5

Useful in **loops** to process all elements.

Looping Through an Array

1. **Using For Loop**

```
Dim numbers() As Integer = {10, 20, 30, 40, 50}
For i As Integer = 0 To numbers.Length - 1
   Console.WriteLine(numbers(i))
Next
```

2. **Using For Each Loop**

```
For Each num As Integer In numbers
   Console.WriteLine(num)
Next
```

For Each is simpler when you **don't need the index**.

Multi-Dimensional Arrays

Two-dimensional arrays (like a table or matrix):

```
Dim matrix(1, 2) As Integer  ' 2 rows, 3 columns
matrix(0, 0) = 1
matrix(0, 1) = 2
matrix(0, 2) = 3
matrix(1, 0) = 4
matrix(1, 1) = 5
matrix(1, 2) = 6
```

Access elements using **two indices**:

```
Console.WriteLine(matrix(1, 2))  ' Output: 6
```

Can also create **string or double multi-dimensional arrays**.

Some Useful Array Methods

1. **Array.Reverse()** → Reverse the array:

```
Dim numbers() As Integer = {1, 2, 3, 4}
Array.Reverse(numbers)
```

2. **Array.Sort()** → Sort the array in ascending order:

```
Array.Sort(numbers)
```

3. **Array.IndexOf()** → Find the position of an element:

```
Dim pos As Integer = Array.IndexOf(numbers, 3)
Console.WriteLine(pos)  ' Output: 2
```

4. **Array.Length** → Get the number of elements:

```
Console.WriteLine(numbers.Length)
```

Dynamic Arrays

Sometimes you don't know the size in advance:

```
Dim numbers() As Integer
ReDim numbers(4)   ' Resize array to 5 elements
numbers(0) = 10
```

Can resize dynamically using ReDim Preserve to **keep existing data**:

```
ReDim Preserve numbers(6)  ' Resize to 7 elements without losing old values
```

Example: PROGRAM

```
Module Module1
    Sub Main()
        Dim names() As String = {"Salum", "Juma", "Amina"}
        Console.WriteLine("All names:")
        For Each name As String In names
            Console.WriteLine(name)
        Next
```

```
        ' Change one element
        names(1) = "Hassan"
        Console.WriteLine("Updated names:")
        For i As Integer = 0 To names.Length - 1
            Console.WriteLine(names(i))
        Next

        Console.ReadKey()
    End Sub
End Module
```

Output:

```
All names:
Salum
Juma
Amina
Updated names:
Salum
Hassan
Amina
```

Another Simple array program

```
Module Module1

    Sub Main()

        ' Declare and initialize an array of numbers

        Dim numbers() As Integer = {10, 20, 30, 40, 50}


        ' Display all numbers using a For loop
```

```
        Console.WriteLine("Numbers in the array:")
        For i As Integer = 0 To numbers.Length - 1
            Console.WriteLine(numbers(i))
        Next

        Console.ReadKey() ' Wait for a key press before closing
    End Sub
End Module
```

Explanation:

Dim numbers() As Integer = {10, 20, 30, 40, 50}

Creates an array named numbers with 5 elements.

For i As Integer = 0 To numbers.Length - 1

Loops through each element in the array.

Console.WriteLine(numbers(i))

Displays the current number.

Console.ReadKey()

Keeps the console window open until a key is pressed.

Output:

Numbers in the array:

10

20

30

40

50

LOOPS IN VISUAL BASIC

A **loop** is a programming structure that repeats a block of code multiple times until a condition is met. Loops are used to **perform repetitive tasks efficiently** without writing the same code again.

Every loop has:

1. **Initialization** – starting point
2. **Condition** – decides when to stop
3. **Increment/Decrement** – moves toward the stopping condition

In simple words: **A loop is like telling the computer: "Do this again and again until I say stop."**

Why Loops Are Important

a) Avoids **repeating code**.
b) Makes programs **shorter and easier to read**.
c) Essential for **processing arrays, lists, or user inputs**.
d) Useful for **automating repetitive calculations**.

Types of Loops in VB

a) For...Next
b) For Each...Next
c) Do While...Loop
d) Do Until...Loop
e) While...End While

1□.**For...Next Loop**

Used when you **know in advance how many times** you want to repeat a task.

Syntax:

```
For counter As Integer = start To end
   ' Code to repeat
Next
```

Example: Print numbers from 1 to 5

```
For i As Integer = 1 To 5
   Console.WriteLine(i)
Next
```

Output:

```
1
2
3
4
5
```

You can also **step by a value**:

```
For i As Integer = 1 To 10 Step 2
   Console.WriteLine(i)   ' Prints 1, 3, 5, 7, 9
Next
```

2□.**For Each...Next Loop**

Used to **loop through all elements in an array or collection**.

Syntax:

```
For Each element As DataType In collection
   ' Code to repeat
Next
```

Example: Display array elements

```
Dim numbers() As Integer = {10, 20, 30, 40}
For Each num As Integer In numbers
   Console.WriteLine(num)
Next
```

Output:

10
20
30
40

Simple when **you don't need the index**.

3.**Do While...Loop**

Repeats a block of code **as long as a condition is True**. The condition is checked **before each iteration**.

Syntax:

```
Do While condition
    ' Code to repeat
Loop
```

Example: Print numbers 1 to 5

```
Dim i As Integer = 1
Do While i <= 5
    Console.WriteLine(i)
    i += 1
Loop
```

Output:

1
2
3
4
5

4.**Do Until...Loop**

Repeats a block of code **until a condition becomes True**. Essentially the **opposite of Do While**.

Syntax:

```
Do Until condition
    ' Code to repeat
Loop
```

Example: Print numbers 1 to 5

```
Dim i As Integer = 1
Do Until i > 5
   Console.WriteLine(i)
   i += 1
Loop
```

Output is the same as Do While.

5□.**While...End While Loop**

Similar to Do While, repeats **while a condition is True**.

Syntax:

```
While condition
   ' Code to repeat
End While
```

Example:

```
Dim i As Integer = 1
While i <= 5
   Console.WriteLine(i)
   i += 1
End While
```

Output:

```
1
2
3
4
5
```

Exiting a Loop Early

Use Exit For or Exit Do to **stop a loop before the condition is false**.

Example:

```
For i As Integer = 1 To 10
   If i = 5 Then
      Exit For
```

```
    End If
    Console.WriteLine(i)
Next
```

Output:

```
1
2
3
4
```

Nested Loops

Loops can be **inside other loops** for more complex tasks.

Example: Multiplication table

```
For i As Integer = 1 To 3
    For j As Integer = 1 To 3
        Console.Write(i * j & vbTab)
    Next
    Console.WriteLine()
Next
```

Output:

```
1   2   3
2   4   6
3   6   9
```

SIMPLE PROGRAMS IN VISUAL BASIC

1. Display Name and Age

```
Module Module1
    Sub Main()
        Console.Write("Enter your name: ")
        Dim name As String = Console.ReadLine()
        Console.Write("Enter your age: ")
        Dim age As Integer = Integer.Parse(Console.ReadLine())
        Console.WriteLine("Name: " & name)
        Console.WriteLine("Age: " & age)
        Console.ReadKey()
    End Sub
End Module
```

2.Add Two Numbers

```
Module Module1
    Sub Main()
        Console.Write("Enter first number: ")
        Dim num1 As Integer = Integer.Parse(Console.ReadLine())
        Console.Write("Enter second number: ")
        Dim num2 As Integer = Integer.Parse(Console.ReadLine())

        Dim sum As Integer = num1 + num2
        Console.WriteLine("Sum = " & sum)
        Console.ReadKey()
    End Sub
End Module
```

3.Check Even or Odd

```
Module Module1
    Sub Main()
        Console.Write("Enter a number: ")
        Dim num As Integer = Integer.Parse(Console.ReadLine())

        If num Mod 2 = 0 Then
            Console.WriteLine(num & " is Even")
        Else
            Console.WriteLine(num & " is Odd")
        End If
        Console.ReadKey()
    End Sub
End Module
```

4.Find Largest of Two Numbers

```
Module Module1
    Sub Main()
        Console.Write("Enter first number: ")
        Dim a As Integer = Integer.Parse(Console.ReadLine())
```

```
        Console.Write("Enter second number: ")
        Dim b As Integer = Integer.Parse(Console.ReadLine())

        If a > b Then
            Console.WriteLine(a & " is larger")
        ElseIf b > a Then
            Console.WriteLine(b & " is larger")
        Else
            Console.WriteLine("Both numbers are equal")
        End If

        Console.ReadKey()
    End Sub
End Module
```

5.Simple Calculator Using Function

```
Module Module1
    Function Add(x As Integer, y As Integer) As Integer
        Return x + y
    End Function


    Sub Main()
        Console.Write("Enter first number: ")
        Dim num1 As Integer = Integer.Parse(Console.ReadLine())
        Console.Write("Enter second number: ")
        Dim num2 As Integer = Integer.Parse(Console.ReadLine())


        Dim result As Integer = Add(num1, num2)
        Console.WriteLine("Sum = " & result)
        Console.ReadKey()
    End Sub
End Module
```

6. Print Numbers from 1 to 10

```
Module Module1
```

```
    Sub Main()
        For i As Integer = 1 To 10
            Console.WriteLine(i)
        Next
        Console.ReadKey()
    End Sub
End Module
```

7. Find Factorial of a Number

```
Module Module1
    Sub Main()
        Console.Write("Enter a number: ")
        Dim num As Integer = Integer.Parse(Console.ReadLine())
        Dim factorial As Integer = 1
        For i As Integer = 1 To num
            factorial *= i
        Next
        Console.WriteLine("Factorial of " & num & " is " & factorial)
        Console.ReadKey()
    End Sub
End Module
```

8.Simple Guessing Game

```
Module Module1
    Sub Main()
        Dim secretNumber As Integer = 7
        Console.WriteLine("Guess a number between 1 and 10: ")
        Dim guess As Integer = Integer.Parse(Console.ReadLine())
        If guess = secretNumber Then
            Console.WriteLine("Congratulations! You guessed it right.")
        Else
            Console.WriteLine("Sorry! The correct number was " & secretNumber)
        End If
```

```
        Console.ReadKey()
    End Sub
End Module
```

9.Sum of Elements in an Array

```
Module Module1
    Sub Main()
        Dim numbers() As Integer = {2, 4, 6, 8, 10}
        Dim sum As Integer = 0
        For i As Integer = 0 To numbers.Length - 1
            sum += numbers(i)
        Next
        Console.WriteLine("Sum of array elements = " & sum)
        Console.ReadKey()
    End Sub
End Module
```

Demonstrates **arrays** and **For loop**.

10. Find Largest Element in an Array

```
Module Module1
    Sub Main()
        Dim numbers() As Integer = {5, 12, 9, 20, 3}
        Dim largest As Integer = numbers(0)
        For i As Integer = 1 To numbers.Length - 1
            If numbers(i) > largest Then
                largest = numbers(i)
            End If
        Next
        Console.WriteLine("Largest number = " & largest)
        Console.ReadKey()
    End Sub
End Module
```

Demonstrates **array traversal**, **decision making**.

11. Reverse a String

```
Module Module1
    Sub Main()
        Console.Write("Enter a string: ")
        Dim input As String = Console.ReadLine()
        Dim reversed As String = ""
        For i As Integer = input.Length - 1 To 0 Step -1
            reversed &= input(i)
        Next
        Console.WriteLine("Reversed string: " & reversed)
        Console.ReadKey()
    End Sub
End Module
```

Demonstrates **loops** and **string manipulation**.

12. Check if Array Contains a Number

```
Module Module1
    Sub Main()
        Dim numbers() As Integer = {2, 4, 6, 8, 10}
        Console.Write("Enter a number to search: ")
        Dim num As Integer = Integer.Parse(Console.ReadLine())
        Dim found As Boolean = False
        For i As Integer = 0 To numbers.Length - 1
            If numbers(i) = num Then
                found = True
                Exit For
            End If
        Next
```

```
        If found Then
            Console.WriteLine(num & " is in the array")
        Else
            Console.WriteLine(num & " is not in the array")
        End If
        Console.ReadKey()
    End Sub
End Module
```

Demonstrates **loops, arrays, and decision making**.

13. Sum of Odd Numbers from 1 to N

```
Module Module1
    Sub Main()
        Console.Write("Enter a number N: ")
        Dim N As Integer = Integer.Parse(Console.ReadLine())
        Dim sum As Integer = 0
        For i As Integer = 1 To N
            If i Mod 2 <> 0 Then
                sum += i
            End If
        Next
        Console.WriteLine("Sum of odd numbers from 1 to " & N & " = " & sum)
        Console.ReadKey()
    End Sub
End Module
```

Demonstrates **loop with condition inside**.

14. Display Even Numbers in an Array

```
Module Module1
    Sub Main()
```

```
        Dim numbers() As Integer = {1, 2, 3, 4, 5, 6, 7, 8}
        Console.WriteLine("Even numbers in the array:")
        For Each num As Integer In numbers
            If num Mod 2 = 0 Then
                Console.WriteLine(num)
            End If
        Next
        Console.ReadKey()
    End Sub
End Module
```

Demonstrates **loops, arrays, and decision making**.

15. Reverse an Array

```
Module Module1
    Sub Main()
        Dim numbers() As Integer = {1, 2, 3, 4, 5}
        Array.Reverse(numbers)
        Console.WriteLine("Reversed array:")
        For Each num As Integer In numbers
            Console.WriteLine(num)
        Next
        Console.ReadKey()
    End Sub
End Module
```

Demonstrates **arrays, loops, and built-in methods**.