

Learner Guide

Faculty of Information Technology

Programming 511

Year 1

Semester 1



RICHFIELD

richfield.ac.za

Registered with the Department of Higher Education as a Private Higher Education Institution under the Higher Education Act, 1997.
Registration Certificate No. 2000/HE07/008

FACULTY OF INFORMATION TECHNOLOGY

LEARNER GUIDE

**MODULE: PROGRAMMING 511
(VISUAL BASIC.NET)
(1ST SEMESTER)**

**PREPARED ON BEHALF OF
RICHFIELD GRADUATE INSTITUTE OF TECHNOLOGY (PTY) LTD**

AUTHOR: Mr. Oganne Shylock Pule

EDITOR: Mr. EMMANUEL MANY

FACULTY HEAD: ISAKA REDDY

Copyright © 2020

Richfield Graduate Institute of Technology (Pty) Ltd

Registration Number: 2000/000757/07

All rights reserved; no part of this publication may be reproduced in any form or by any means, including photocopying machines, without the written permission of the Institution.

CONTENT TABLE AND LESSON PLAN

INTERACTIVE ICONS USED IN THIS LEARNER GUIDE	4
Learning Outcomes	4
Study	4
Read	4
Writing Activity	4
Think Point	4
Research	4
Glossary	4
Key Point	4
Review Questions	4
Case Study	4
Bright Idea	4
Topic 1: PROBLEM SOLVING Lecturer 1-6	
1.1. Introduction	5
1.2. Performing a Task on the Computer?	5
1.3. Program Planning	6
1.4.2. Programming Tools	8
1.4.3 Flow Charts	8
1.4.4 Pseudo Code	11
1.4.5 Hierarchy Charts	12
KEY TERMS USED IN THIS SECTION	13
SECTION 1 REVIEW QUESTIONS	14
TOPIC 2: FUNDAMENTALS OF PROGRAMMING IN VB.NET Lecturer 6-13	
2.1. Introduction	15
2.2. Strong Programming Features VB.Net	16
2.3. Strong Programming Features VB.Net	16
2.4. Strong Programming Features VB.Net	16
2.5. Starting a New Visual Basic Program	17
2.6. How to Create Variables in VB .NET	18
KEY TERMS USED IN THIS SECTION	24
UNIT 2 REVIEW QUESTIONS	25
TOPIC 3: GENERAL PROCEDURES Lecturer 14-21	
3.1. Introduction	26
3.2. Sub Procedure	26
3.3. Declaring/Defining Function	29
3.4. Calling a Function	30
3.5. Arguments and Parameters	31
3.6. Passing Arguments to Procedures	32
3.7. Passing Parameters By Reference	33
3.8. Local Variables	33
3.9. Class-Level Variables	34
KEY TERMS USED IN THIS SECTION	37
UNIT3REVIEW QUESTIONS	38

TOPIC 4: DECISION MAKING **Lecturer 22-27**

4.1.	Introduction.....	39
4.2.	How to use IF ELSE in VB.NET	41
4.3.	Relational Operators	44
4.4.	Logical Operators.....	44
4.5.	Select Case.....	44
4.6.	Flow chart for Select Case	46
<i>UNIT 4 REVIEW QUESTIONS.....</i>		<i>49</i>

TOPIC 5: REPETITION STRUCTURE **Lecture 28-30**










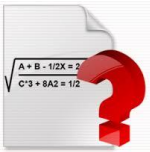



5.1.	Introduction.....	50
5.2.	Do Loops	50
5.3.	FOR...NEXT LOOPS	51
5.4.	While Loop.....	53
<i>SECTION 5 REVIEW QUESTIONS.....</i>		<i>57</i>

TOPIC 6: ARRAYS **Lecture 31- 37**


6.1.	Introduction.....	60
6.2.	Declaring an Array	60
6.3.	Declaring and Initializing a String Array	64
6.4.	How to resize an array.....	65
6.5.	How to Use ForEach with Arrays ?	66
6.6.	How to check if a value exists in an array?	66
6.7.	The GETUPPERBOUND Method	67
6.8.	Multi-Dimensional Arrays.....	68
6.9.	Sorting Arrays	72
<i>SECTION6 REVIEW QUESTIONS.....</i>		<i>75</i>
<i>SECTION6 REVIEW QUESTIONS.....</i>		<i>76</i>

Prescribed textbook:

An Introduction to Programming Visual Basic Using Visual Basic 2012 9th ed. 2013 David .I Schneider 9780133378504

INTERACTIVE ICONS USED IN THIS LEARNER GUIDE			
 <p>Learning Outcomes</p>	 <p>Study</p>	 <p>Read</p>	 <p>Writing Activity</p>
 <p>Think Point</p>	 <p>Research</p>	 <p>Glossary</p>	 <p>Key Point</p>
 <p>Review Questions</p>	 <p>Case Study</p>	 <p>Bright Idea</p>	 <p>Problem(s)</p>
 <p>Multimedia Resource</p>	 <p>References</p>		
			 <p>Web Resource</p>

ONE | PROBLEM SOLVING

	LEARNING OUTCOMES
	<ol style="list-style-type: none"> 1. Introduction 2. Understand the Program Development Cycle 3. Correlate data in flowcharts 4. Understand the use of Pseudo code and Hierarchy Charts

1.1. Introduction

Hardware refers to the machinery in a computer system (such as the monitor, keyboard, and CPU) and software refers to a collection of instructions, called a program that directs the hardware. Programs are written to solve problems or perform tasks on a computer. Programmers translate the solutions or tasks into a language the computer can understand. As we write programs, we must keep in mind that the computer will only do what we instruct it to do. Because of this, we must be very careful and thorough with our instructions. Note: A program is also known as a project, application, or solution.

1.2. Performing a Task on the Computer?

The first step in writing instructions to carry out a task is to determine what the output should be that is, exactly what the task should produce. The second step is to identify the data, or input, necessary to obtain the output. The last step is to determine how to process the input to obtain the desired output, that is, to determine what formulas or ways of doing things can be used to obtain the output.

This problem-solving approach is the same as that used to solve word problems in an algebra class. For example, consider the following algebra problem:

How fast is a car travelling if it goes 50 miles in 2 hours?

The first step is to determine the type of answer requested. The answer should be a number giving the speed in miles per hour (the output). (Speed is also called velocity.) The information needed to obtain the answer is the distance and time the car has travelled (the input).

The formula

$$\text{Speed} = \text{Distance} / \text{Time}$$

is used to process the distance travelled and the time elapsed in order to determine the speed. That is,

$$\begin{aligned} \text{Speed} &= 120 \text{ miles} / 3 \text{ hours} \\ &= 40 \text{ miles} / 1 \text{ hour} \end{aligned}$$

A pictorial representation of this problem-solving process is



We determine what we want as output, get the needed input, and process the input to produce the desired output.

In the following chapters we discuss how to write programs to carry out the preceding operations. But first we look at the general process of writing programs.

1.3. Program Planning

A baking recipe provides a good example of a plan. The ingredients and the amounts are determined by what is to be baked. That is, the output determines the input and the processing. The recipe, or plan, reduces the number of mistakes you might make if you tried to bake with no plan at all. Although it's difficult to imagine an architect building a bridge or a factory without a detailed plan, many programmers (particularly students in their first programming course) try to write programs without first making a careful plan. The more complicated the problem, the more complex the plan may be. You will spend much less time working on a program if you devise a carefully thought out step-by-step plan and test it before actually writing the program.

Many programmers plan their programs using a sequence of steps, referred to as the program development cycle. The following step-by-step process will enable you to use your time efficiently and help you design error-free programs that produce the desired output.

1. **Analyze:** Define the problem.

Be sure you understand what the program should do, that is, what the output should be. Have a clear idea of what data (or input) are given and the relationship between the input and the desired output.

2. **Design:** Plan the solution to the problem.

Find a logical sequence of precise steps that solve the problem. Such a sequence of steps is called an algorithm. Every detail, including obvious steps, should appear in the algorithm. In the next section, we discuss three popular methods used to develop the logic plan: flowcharts, pseudo code, and top-down charts. These tools help the programmer break a problem into a sequence of small tasks the computer can perform to solve the problem. Planning also involves using representative data to test the logic of the algorithm by hand to ensure that it is correct.

3. **Choose the interface:** Select the objects (text boxes, buttons, etc.).

Determine how the input will be obtained and how the output will be displayed. Then create objects to receive the input and display the output. Also, create appropriate buttons and menus to allow the user to control the program.

4. **Code:** Translate the algorithm into a programming language.

Coding is the technical word for writing the program. During this stage, the program is written in Visual Basic and entered into the computer. The programmer uses the algorithm devised in Step 2 along with knowledge of Visual Basic.

5. **Test and debug:** Locate and remove any errors in the program.

Testing is the process of finding errors in a program, and debugging is the process of correcting errors that are found. (An error in a program is called a bug.) As the program is typed, Visual Basic points out certain types of program errors. Other types of errors will be detected by Visual Basic when the program is executed; however, many errors due to typing mistakes, flaws in the algorithm, or incorrect use of the Visual Basic language rules can be uncovered and corrected only by careful detective work. An example of such an error would be using addition when multiplication was the proper operation.

6. **Complete the documentation:** Organize all the material that describes the program.

Documentation is intended to allow another person, or the programmer at a later date, to understand the program. Internal documentation (comments) consists of statements in the program that are not executed, but point out the purposes of various parts of the program. Documentation might also consist of a detailed

description of what the program does and how to use the program (for instance, what type of input is expected). For commercial programs, documentation includes an instruction manual and on-line help.

Other types of documentation are the flowchart, pseudo code, and top-down chart that were used to construct the program. Although documentation is listed as the last step in the program development cycle, it should take place as the program is being coded.

1.4.2. **Programming Tools**

This section discusses some specific algorithms and develops three tools used to convert algorithms into computer programs: flowcharts, pseudo code, and hierarchy charts.

1.4.3 **Flow Charts**

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. Each step in the sequence is noted within a diagram shape. Steps are linked by connecting lines and directional arrows.

In programming process a flow chart is a formal way of representing an **algorithm** of a problem by the diagram.

The shape of the symbol indicates the type of operation that is to occur. For instance, the parallelogram denotes input or output. The arrows connecting the symbols, called flow lines, show the progression in which the steps take place.

Algorithm: is a procedure or formula for solving a problem.

When to Use a Flowchart

- To develop understanding of how a process is done.
- To study a process for improvement.
- To communicate to others how a process is done.
- When better communication is needed between people involved with the same process.
- To document a process.
- When planning a project.






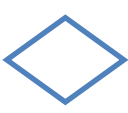




<http://sameeradilhan.com/what-is-system-development-life-cycle-sldc>

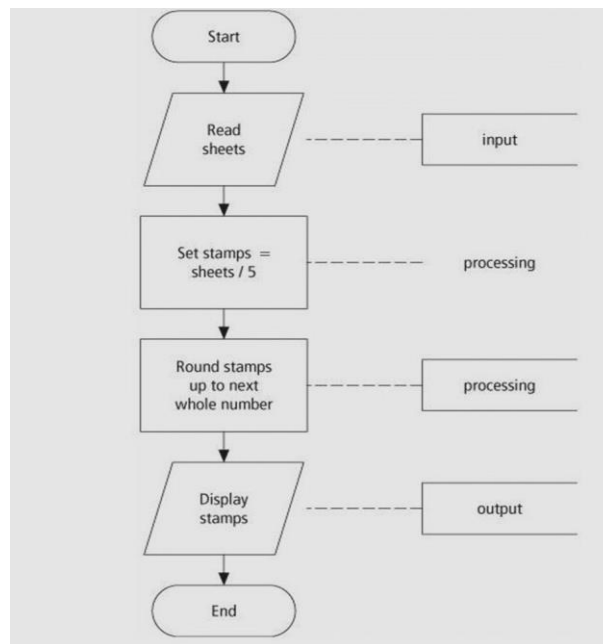
The link above will open a comprehensive online resource giving details about a Software development life cycle (SDLC) and Programming Tool used during SDLC. This resource makes for very interesting reading and it will be worth your while to visit this page and peruse rough its contents.



THINK POINT

Find out on the internet what are basic stages of computing, some of the tools used in programming and how to use each flowchart symbol during SDLC.

Symbols	Names	Functions
	Action / Process	A box can represent a single step ("add two cups of flour"), or and entire sub-process ("make bread") within a larger process.
	Connector	Indicates that the flow continues where a matching symbol (containing the same letter) has been placed. Link to another page.
	Flow of control / Arrow	Direction of flow from one step/ direction to another.
	Decision	Decision based on a question (Written in a diamond) A decision or branching point. Lines representing different decisions emerge from different points of the diamond.
	Input or Output	Represents material or information entering or leaving the system, such as customer order (input) or a product (output).
	Module /Subroutine	Indicates a sequence of actions that perform a specific task embedded within a larger process. This sequence of actions could be described in more detail on a separate flowchart.
	Delay or wait	Indicates a delay in the process.
	Start /End	The terminator symbol marks the starting or ending point of the system. It usually contains the word "Start" or "End."



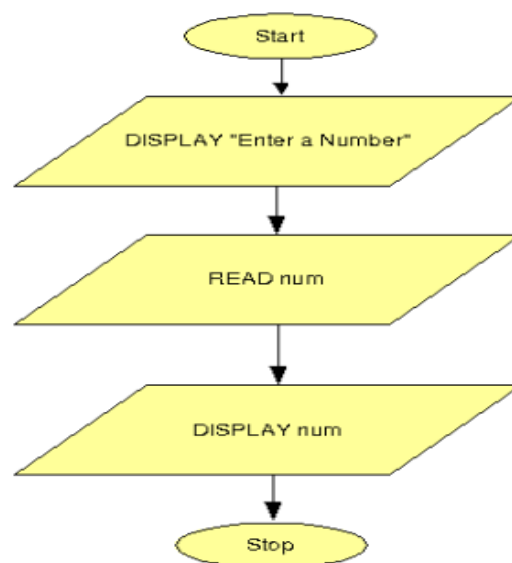
Flowcharts should "flow" from the top of the page to the bottom. Although the symbols used in flowcharts are standardized, no standards exist for the amount of detail required within each symbol.

Example:

Create a flowchart that prompts a user to enter a number. After the number has been entered, it is displayed.

Desk Check Table		
Input	Process	Output
number	Read number	number
	Display number	

Flow-chart



1.4.4 Pseudo Code

Pseudo code is an abbreviated plain English version of actual computer code (hence, pseudo code). The geometric symbols used in flowcharts are replaced by English-like statements that outline the process. As a result, pseudo code looks more like computer code than does a flowchart.

Pseudo code allows the programmer to focus on the steps required to solve a problem rather than on how to use the computer language. The programmer can describe the algorithm in Visual Basic-like form without being restricted by the rules of Visual Basic. When the pseudo code is completed, it can be easily translated into the Visual Basic language.

The rules of Pseudocode are reasonably straightforward. All statements showing "dependency" are to be indented. These include while, do, for, if, switch. Examples below will illustrate this notion.

Examples 1:

```
If student's grade is greater than or equal to 60
    Print "passed"
Else
    Print "failed"
End if
```

Examples2:

```
Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
    Input the next grade
    Add the grade into the total
End While
Set the class average to the total divided by ten

Print the class average.
```

Examples 2:

```
Initialize total to zero
Initialize counter to zero
Input the first grade

while the user has not as yet entered the sentinel
    add this grade into the running total
    add one to the grade counter

input the next grade (possibly the sentinel)
Wend
if the counter is not equal to zero
    set the average to the total divided by the counter
    print the average
else
    print 'no grades were entered'
End if
```

1.4.5 Hierarchy Charts

The last programming tool we'll discuss is the hierarchy chart, which shows the overall program structure. Hierarchy charts are also called structure charts, HIPO (Hierarchy plus Input-Process-Output) charts, top-down charts, or VTOC (Visual Table of Contents) charts. All these names refer to planning diagrams that are similar to a company's organization chart.

Hierarchy charts depict the organization of a program but omit the specific processing logic. They describe what each part, or module, of the program does and they show how the modules relate to each other. The details on how the modules work, however, are omitted. The chart is read from top to bottom and from left to right. Each module may be subdivided into a succession of sub modules that branch out under it. Typically, after the activities in the succession of sub modules are carried out, the module to the right of the original module is considered. A quick glance at the hierarchy chart reveals each task performed in the program and where it is performed.



SECTION SUMMARY

- Hardware refers to the machinery in a computer system (such as the monitor, keyboard, and CPU) and software refers to a collection of instructions, called a program that directs the hardware.
- Programs are written to solve problems or perform tasks on a computer. Programmers translate the solutions or tasks into a language the computer can understand.
- Many programmers plan their programs using a sequence of steps, referred to as the program development cycle or System Development Life Cycle (SDLC).
- Three types of programming tools: Pseudo-code, hierarchy chart and flowchart
- Pseudocode (pronounced SOO-doh-kohd) is a detailed yet readable description of what a computer program or algorithm must do, expressed in a formally-styled natural language rather than in a programming language.



KEY TERMS USED IN THIS SECTION

Flowcharts: A number that uniquely identifies each workstation and device on a network. Without unique addresses, computers on the network could not reliably communicate.

Pseudo code: Uses English-like phrases with some Visual Basic terms to outline the task.

Hierarchy charts: Show how the different parts of a program relate to each other.

SDLC(System Development Life Cycle).A Software Development Life Cycle is essentially a series of steps, or phases, that provide a model for the development and lifecycle management of an application or piece of software.

Algorithm: is a procedure or formula for solving a problem.



SECTION 1 REVIEW QUESTIONS

1. What does the acronym SDLC mean, and what does an SDLC portray?
2. List five steps of Program Development Cycle or System Development Life Cycle (SDLC).
3. What are three tools that are used to in programming?
4. Define the following:
 - a. Flow chart
 - b. Pseudo code
 - c. Hierarchy chart
5. Suppose you have been asked to create an information system for a manufacturing plant that produces nuts and bolts of many shapes, sizes, and functions. What will you envision the SDLC to be?
6. Briefly summarize the purpose of the implementation phase in SDLC. Explain why it exists and what it contributes to the completion of the System.
7. Write a pseudocode program to find celsius degrees into fahrenheit.
8. Draw flow-chart that will count all the even numbers up to a user defined stopping point. For example, say we want to see the first 5 even numbers starting from 0. well, we know that evens numbers are 0, 2, 4, etc.
The first 5 even numbers are 0, 2, 4, 6, 8.
The first 8 even numbers are 0, 2, 4, 6, 8, 10, 12, 16

9. Covert below code into flow-chart.

```

initialize passes,failures to zero
initialize student to one


while student counter is less than or equal to ten
  input the next exam result

  if the student passed
    add one to passes
  else

    add one to failures
    add one to student counter
  print the number of passes
  print the number of failures
  if eight or more students passed
    print "raise tuition"
  
```

TWO | FUNDAMENTALS OF PROGRAMMING IN VISUAL BASIC.NET

TWO | FUNDAMENTALS OF PROGRAMMING IN VISUAL BASIC.NET

	LEARNING OUTCOMES
	<ol style="list-style-type: none"> 1. VB.NET Background 2. Integrated Development Environment (IDE) For VB.Net 3. Manipulate controls and change their properties 4. Define VB.Net events 5. Declare number variables, increment variable values. 6. Utilise Built-In Functions : Math.Sqrt, Int, Math.Round 7. Work with string variables 8. Format output with format functions 9. Format Output with Zones 10. Get input form an input dialog box, use a message dialog box for output 11. Use masked text box for input

2.1. Introduction

Visual Basic .NET (VB.NET) is an object-oriented computer programming language implemented on the .NET Framework. Although it is an evolution of classic Visual Basic language, it is not backwards-compatible with VB6, and any code written in the old version does not compile under VB.NET.

Like all other .NET languages, VB.NET has complete support for object-oriented concepts. Everything in VB.NET is an object, including all of the primitive types (Short, Integer, Long, String, Boolean, etc.) and user defined types, events, and even assemblies. All objects inherits from the base class Object.

VB.NET is implemented of Microsoft's .NET framework. Therefore, it has full access to all the libraries in the .Net Framework. It's also possible to run VB.NET programs on Mono, the open-source alternative to .NET, not only under Windows, but even Linux or Mac OS.

The following reasons make VB.Net a widely used professional language:

- Modern, general purpose.
- Object oriented.
- Component oriented.
- Easy to learn.
- Structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- Part of .Net Framework.

2.2. Strong Programming Features VB.Net

VB.Net has numerous strong programming features that make it endearing to multitude of programmers worldwide. Let us mention some of these features:

- Boolean Conditions
- Automatic Garbage Collection
- Standard Library
- Assembly Versioning
- Properties and Events
- Delegates and Events Management
- Easy-to-use Generics
- Indexers
- Conditional Compilation
- Simple Multithreading



You can download visual studio from this

link:<http://www.microsoft.com/visualstudio/eng/downloads>

It gets automatically installed in your machine. Please note that you need an active internet connection for installing the express edition.

2.3. Strong Programming Features VB.Net

- Microsoft provides the following development tools for VB.Net programming:
- Visual Studio 2010 (VS)
- Visual Basic 2010 Express (VBE)
- Visual Web Developer

2.4. Strong Programming Features VB.Net

Visual Basic is a programming language that is designed especially for windows programming. This tutorial will step through and demonstrate some of the features of Visual Basic. It will explain most of the tools available for implementing GUI based programs. After introducing the basic facilities and tools provided by Visual Basic, we apply our knowledge to implementing a small VB program. Our program will implement a visual interface for a commonly know “stack” abstract data type.

Visual Basic programs display a Windows-style screen (called a form) with boxes into which users type (and in which users edit) information and buttons that they click to initiate actions. The boxes and buttons are referred to as controls. In this section, we examine forms and four of the most useful Visual Basic controls.

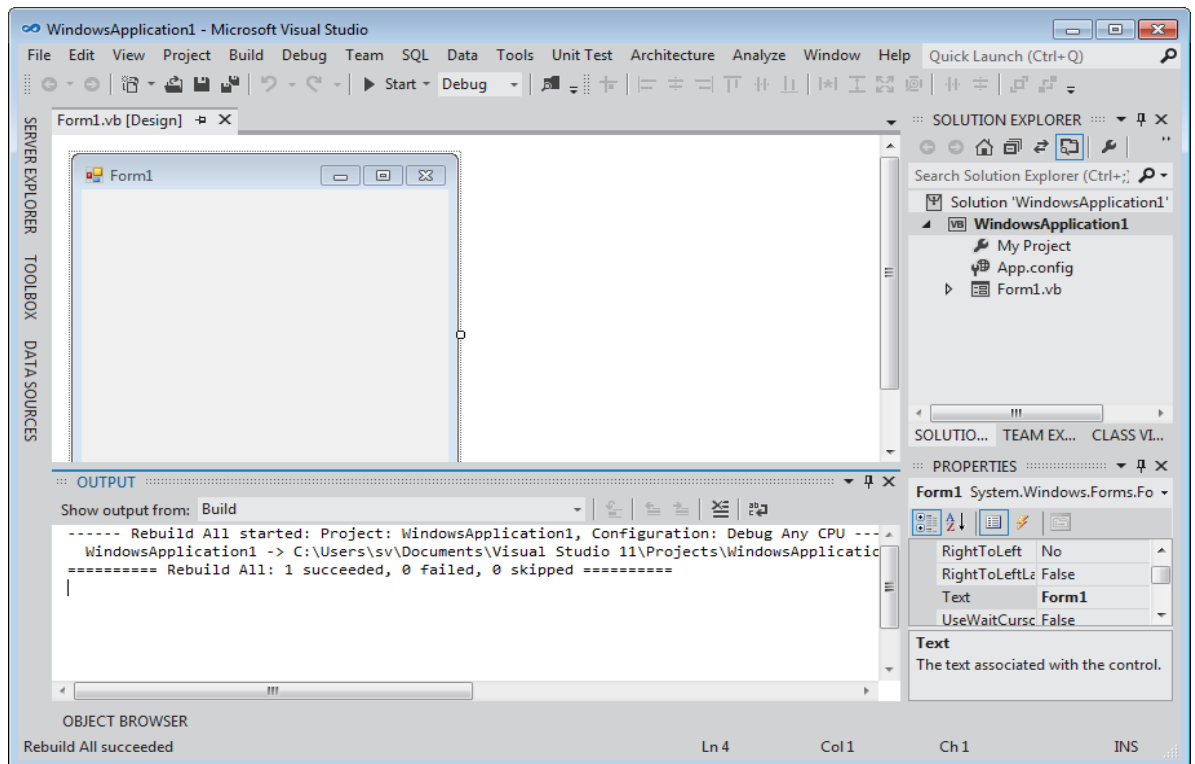


Figure 2-1 VB.NET IDE

2.5. Starting a New Visual Basic Program

For our purposes, Visual Basic programs are also known as applications, solutions, or projects. Each program is saved (as several files and subfolders) in its own folder. Before starting a new program, you should use Windows Explorer to create a folder to hold the folders for your programs.

The process for invoking Visual Basic varies slightly with the edition of Visual Basic installed on the computer. To invoke Visual Basic from a computer that has Visual Basic Express installed, click the Windows Start button, hover over All Programs, and then click on Microsoft Visual Basic Express Edition. With the other editions of Visual Basic, hover over All Programs, hover over Microsoft Visual Studio, and then click on Microsoft Visual Studio in the short list that is revealed.

Launch your Visual Basic .NET or Visual Studio software. When the software first loads, you'll see a screen something like this one, if you have the 2010 version:

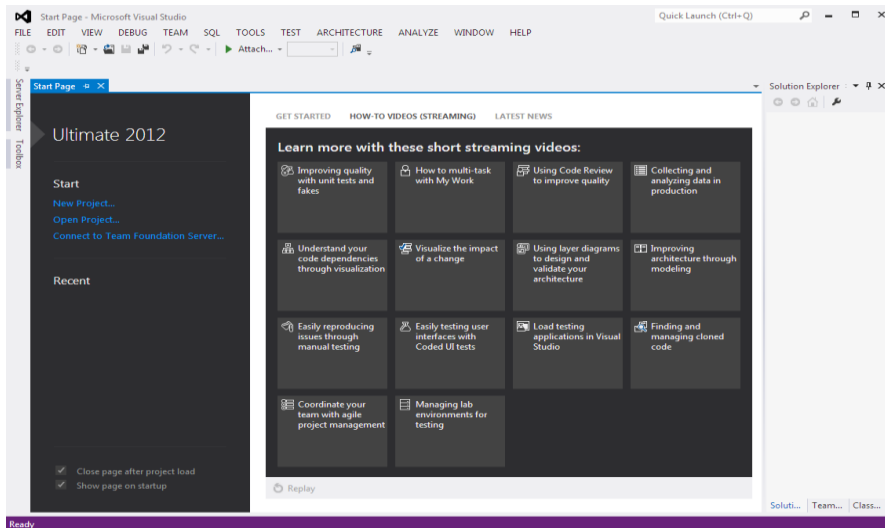
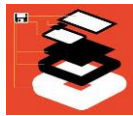


Figure 2-2



Reference

Please go refer to: [How To Get Started with VB.NET](#).
The attachment will show you step-by-step on how to create your first vb.net project.

Common Properties and Methods of Controls

To create a control, the primary piece of information you must provide is its name. This allows you and the compiler to know what control you are referring to when the program is running. Specifying the name of a control may depend on the technique you decide to use to create the control.

After adding a control to a form, it automatically receives a name. In the Properties window, the control's name displays in the (Name) field. The newly added control

reflects the name of its button on the Properties window and adds an incremental number. For example, if you click the TextBox button on the Toolbox and click the form, the control would be named TextBox1. If you add a second TextBox control, it would be named TextBox2.

This causes the default names to be incremental. Since a program usually consists of many controls, it is usually a good idea to rename your controls and give them meaningful and friendlier names. The name should help you identify what the button is used for. To change the name of a control, on the Properties window, click the (Name) field, type the desired name and press Enter.

2.6. How to Create Variables in VB .NET

With Visual Basic, and most programming languages, what you are doing is storing things in the computer's memory, and manipulating this store. If you want to add two numbers together, you put the numbers into storage areas and "tell" Visual Basic to add them up. But you can't do this without variables.

So a variable is a storage area of the computer's memory. Think of it like this: a variable is an empty cardboard box. Now, imagine you have a very large room, and in this room you have a whole lot of empty cardboard boxes. Each empty cardboard box is a single variable. To add two numbers together, write the first number on a piece of paper and put the piece of paper into an empty box. Write the second number on a piece of paper and put this second piece of paper in a different cardboard box.

Now, out of all your thousands of empty cardboard boxes two of them contain pieces of paper with numbers on them. To help you remember which of the thousands of boxes hold your numbers, put a sticky label on each of the two boxes. Write "number1" on the first sticky label, and "number2" on the second label.

What have we just done? Well, we've created a large memory area (the room and the cardboard boxes), and we've set up two of the boxes to hold our numbers (two variables). We've also given each of these variables a name (the sticky labels) so that we can remember where they are.

Now examine this:

```
Dim number1 As Integer
Dim number2 As Integer
number1 = 3
number2 = 5
```

That's code from Visual Basic Net. It's VB's way of setting up (or declaring) variables. Here's a breakdown of the variable Declaration

Dim:

Short for Dimension. It's a type of variable. You declare (or "tell" Visual Basic) that you are setting up a variable with this word. We'll meet other types of variables later, but for now just remember to start your variable declarations with Dim.

number1 and number2

Are the Identifiers: A valid identifier is a sequence of one or more letters, digits or underscore characters (_). Neither spaces nor punctuation marks or symbols can be part of an identifier. Only letters, digits and single underscore characters are valid. In addition, variable identifiers always have to begin with a letter. They can also begin with an underline character (_), but in some cases these may be reserved for compiler specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere.

As Integer

We're telling Visual Basic that the variable is going to be a number (integer). We'll meet alternatives to Integer later.

Number1 = 3

The equals sign is not actually an equals sign. The = sign means assign a value of. In other words, here is where you put something in your variable. We're telling Visual Basic to assign a value of 3 to the variable called number1. Think back to the piece of paper going into the cardboard box. Well, this is the programming equivalent of writing a value on a piece of paper.

Utilise Built-In Functions : Math.Sqrt, Int, Math.Round

Visual Studio .NET provides an easy way of performing mathematical functions, such as addition, subtraction, multiplication, division, exponentiation, integer division, and finding a remainder. For all other tasks, you can utilize the System.Math class. In this tip, I will look at a simple way of working with math-related functions in VB.NET.

.NET offers common operators to facilitate the basic mathematical functions, such as:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Exponentiation (^)
- Integer Division (\)
- Finding the remainder (Mod)

However, for other operations, you can use the methods available in the System.Math class.

Math.Sqrt takes the square root of a number. With it, you pass one Double value as an argument, and receive the square root of that as another Double. This method requires more time to compute fractional square roots

Example

```

Dim result1 As Double = Math.Sqrt(1)
Dim result2 As Double = Math.Sqrt(2)
Dim result3 As Double = Math.Sqrt(3)
Dim result4 As Double = Math.Sqrt(4)

With lstDisplay.Items
    .Add (result1)
    .Add (result2)
    .Add (result3)
    .Add (result4)
End with

```

Output

1
1.4142135623731
1.73205080756888
2

Math.Round

Numbers can be rounded in many ways. In the .NET Framework, Math.Round provides built-in logic. With it we round with special options—away from zero, or to even. It acts on many numeric types, including Double and Decimal.

Example

Math written on chalkboard

This program calls Math.Round on the Double 123.45. With no options, it rounds this number to 123. With a second option of 1, we round to one decimal place. This yields the values 123.5 (for AwayFromZero) and 123.4 (for ToEven).

AwayFromZero:

With a positive number, this option will round up—so 123.45 becomes 123.5.

ToEven:

This will round to an even number—so 123.45 becomes 123.4 because 4 is an even number and 5 is not.

Example

' Call Math.Round on this Double.

```
Dim before As Double = 123.45
Dim after1 As Double = Math.Round(before, 1, _
MidpointRounding.AwayFromZero)
```

```
Dim after2 As Double = Math.Round(before, 1, _
MidpointRounding.ToEven)
Dim after3 As Double = Math.Round(before)
```

```
With lstDisplay.Items
    .Add (before)
    .Add (after1)
    .Add (after2)
    .Add (after3)
```

End with

' Use on this Decimal.

```
Dim before2 As Decimal = 125.101
Dim after4 As Decimal = Math.Round(before2)
Dim after5 As Decimal = Math.Round(before2, 1)
```

```
With lstDisplay.Items
    .Add (before)
    .Add (after4)
    .Add (after5)
```

End with

Output

```
123.45
123.5
123.4
123
```

```
125.101
125
125.1
```



SECTION SUMMARY

- Visual Basic .NET (VB.NET) is an object-oriented computer programming language implemented on the .NET Framework.
- The following reasons make VB.Net a widely used professional language:
- Modern, general purpose.
 - Object oriented.
 - Component oriented.
 - Easy to learn.
 - Structured language.
 - It produces efficient programs.
 - It can be compiled on a variety of computer platforms.
 - Part of .Net Framework.
- To invoke Visual Basic from a computer that has Visual Basic Express installed, click the Windows Start button, hover over All Programs, and then click on Microsoft Visual Basic Express Edition.
- After adding a control to a form, it automatically receives a name.
- The newly added control reflects the name of its button on the Properties window and adds an incremental number.
- To change the name of a control, on the Properties window, click the (Name) field, type the desired name and press Enter.
- variable is a storage area of the computer's memory.
- You declare or create a variable with the use of keyword, Dim.
- Dim: Short for Dimension
- Visual Studio .NET provides an easy way of performing mathematical functions, such as addition, subtraction, multiplication, division, exponentiation, integer division, and finding a remainder.
- However, for other operations, you can use the methods available in the System.Math class.
- Math.Sqrt takes the square root of a number.
- Math.Round provides built-in logic. With it we round with special options—away from zero, or to even.



KEY TERMS USED IN THIS SECTION

Boolean Conditions: Condition that yield two condition, either True or False – (Yes or No, 1 or 0)

GUI: Graphical User Interface

Variable: is a storage area of the computer's memory.

Dim: Short for Dimension: is a key word of visual basic, that is use to declare a variable.

Built-in-function: A function is similar to a normal procedure but the main purpose of the function is to accept a certain input from the user and return a value which is passed on to the main program to finish the execution.




UNIT 2 REVIEW QUESTIONS

1. Which controls you can use to accept entry from the user?
2. List any five controls commonly used in VB.NET?
3. Declare two variables of different data type.
4. Declare an integer variable and assign 1900 to it.
5. Determine the output of the code below:

```
Dim dbData As Double  
Dim intData As Integer  
  
dbData = 3.14159  
intData = dbData  
lblDisplay.Text = dbData  
lblDis.Text = intData
```

THREE | GENARAL PROCEDURES

	Learning Outcomes
<ol style="list-style-type: none"> 1. Make Use of Sub Procedures 2. Assign variables by value and by reference 3. Understand the use of Local and Class Level variables • Use functions to return values for processing data. 	

3.1. Introduction

Visual Basic has two devices, Sub procedures and Function procedures that are used to break complex problems into small problems to be solved one at a time. To distinguish them from event procedures, Sub and Function procedures are referred to as general procedures. General procedures also eliminate repetitive code and can be reused in other programs.

Procedures are an essential part of almost every VB program. When you define a procedure, whether it's a Function or a Sub procedure, you need to decide whether the procedure arguments are passed by reference or by value.

3.2. Sub Procedure

A procedure is a group of statements that together perform a task when called. After the procedure is executed, the control returns to the statement calling the procedure. VB.Net has two types of procedures:

- Functions
- Sub procedures or Subs

A Sub procedure returns no value. We implement and call this kind of method. Subs optionally have argument lists and formal parameters. These parameters can be separated with line continuations for readability.

A sub procedure is an assignment that is carried but doesn't give back a result. To create a sub procedure, start by typing the Sub keyword followed by a name (like everything else, a procedure must have a name). The name of a procedure is always followed by parentheses. At the end of the sub procedure, you must type End Sub.

The syntax of a sub procedure is:

```
Sub ProcedureName()
```

```
End Sub
```

The name of a procedure should follow the same rules we learned to name the variables, omitting the prefix. In addition:

If the procedure performs an action that can be represented with a verb, you can use that verb to name it. Here are examples: show, display

To make the name of a procedure stand, you should start it in uppercase. Examples are Show, Play, Dispose, Close

You should use explicit names that identify the purpose of the procedure. If a procedure would be used as a result of another procedure or a control's event, reflect it on the name of the sub procedure. Examples would be: afterupdate, longbefore.

If the name of a procedure is a combination of words, you should start each word in uppercase. Examples are: AfterUpdate, SayItLoud

The section between the Sub and the End Sub lines is referred to as the body of the procedure. Here is an example:

```
Sub Assignment()
```

```
End Sub
```

The body of the procedure is used to define what, and how the, assignment should be carried. For example, if you need to use a variable, you can declare it and specify the kind of variable you need. There is no restriction on the type of variables that can be declared in a procedure. Here is an example in which a string variable is declared in the body of a sub routine:

```
Sub Assignment()  
    Dim strFullName As String  
End Sub
```

In the same way, you can declare as many variables as you need inside of a procedure. The actions you perform inside of a procedure depend on what you are trying to

accomplish. For example, a procedure can simply be used to create a string. The above procedure can be changed as follows:

```
Sub Assignment()  
    Dim strFullName As String  
    strFullName="Paul Bertrand  
    Yamaguchi"  
End Sub
```

User-defined procedures are procedures that you design and write. These are in contrast to the methods contained in the FCL classes, which are supplies for you. You create user-defined procedures to give your code structure and organization. In addition, you can create reusable code by designing procedures that others can invoke. There are two categories of user-defined procedures: Public or Private.

The declaration of a Sub procedure begins with a Sub statement and the Name of the procedure followed by a set of (). It ends with the End Sub statement. The code that appears between these two statements is the body of the procedure. When the sub procedure executes, the code within will execute.

Example (Calling or invoking a sub procedure)

```
Private Sub Display_Message( )  
    'name of procedure is Display_Message  
    Code statements  
End Sub
```

This type of statement is known as a procedure call. It causes the procedure to execute. You can also use the Call keyword, which is an optional command.

Call DisplayMessage()

When a procedure call executes, the application branches to the procedure and executes the body of the sub procedure. When it has finished, it returns to the location of the procedural call and resumes executing at the next statement.

Example 1:

The following example demonstrates how to call and define a Sub procedure.

```
Private Sub btnAdd_Click(...)
Handles btnAdd.Click
    Dim intValue1, intValue2 as Integer
    lstResult.Items.Add("Hello")
    DISPLAY ()
    lstResult.Items.Add("Back To Click Event")
End Sub
```

Example

```
Private Sub DISPLAY()
lstResult.Items.Add("Sub procedure ")
End Sub
```

Output:

Hello

Sub procedure

Back To Click Event

3.3. Declaring/Defining Function

Like a sub procedure, a function is used to perform an assignment. The main difference between a sub procedure and a function is that, after carrying its assignment, a function gives back (return) a result. We also say that a function "returns a value". To distinguish both, there is a different syntax you use for a function.

**IMPORTANT TIP:**

To promote software reusability, every sub procedure or function should be limited to performing a single, well defined task, and the name of the procedure should express that task effectively.

A function procedure returns a value to the part of the program that called the function procedure. Like a Sub procedure, a function procedure is a set of statements that perform a task when the function is called. Like the Sub procedure, it is written in the code window and is not attached to any control. In addition, a function returns a value that can be used in an expression.

Declaring a Function – the general formation of a function declaration is:

```
[AccessSpecifier] Function FunctionName ([Parameterlist]) As Datatype
    Statements
    Return expression

End Function
```

Similar to a Sub procedure declaration the AccessSpecifier is optional and specifies the accessibility of the function. If you do not specify an access specifier, the default is Public.

Next is the keyword Function, followed by the function name. Inside the parenthesis is an optional list of parameters. Following the parenthesis is As Datatype. The datatype listed is the part of the declaration that determines the data type of the value returned by the function.



GOOD PROGRAMMING PRACTICE:
Capitalize the first letter of procedure name. Alternatively use only uppercase letters.

An example of a function is:

```
Private Function Sum(ByVal sngNum1 As Single, ByVal sngNum2 As Single) As Single
    Dim sngResult As Single
    sngResult = sngNum1 + sngNum2
    Return sngResult
```

Or

```
Private Function CALCULATE_SUM (ByVal sngNum1 As Single, ByVal sngNum2 As Single) As Single
    Dim sngResult As Single
    sngResult = sngNum1 + sngNum2

    CALCULATE_SUM = sngResult
End Function
```

3.4. Calling a Function

When calling a function, the normal procedure is to use an assignment statement so that the value returned through the function will be assigned to a variable or other form of output.

```
sngTotal = Sum(sngValue1, sngValue2)
```

This statement passes the variables `sngValue1` and `sngValue2` as arguments in the function called `Sum`. It assigns the values to the function variables `sngNum1` and `sngNum2`. After the function executes, the return statement sends a value back to the statement and it is assigned to `sngTotal`.

3.5. Arguments and Parameters

So far, to use a value in a procedure, we had to declare it. In some cases, a procedure may need an external value in order to carry its assignment. A value that is supplied to a procedure is called an argument.

When creating a procedure that will use an external value, declare the argument that represents that value between the parentheses of the procedure. For a sub routine, the syntax you use would be:

```
Sub ProcedureName(Argument)
```

```
End Sub
```

If you are creating a function, the syntax would be:

```
Function CalculatePayroll(strName As String) As Double
```

```
Function Sub
```

```
Function ProcedureName(Argument) As DataType
```

```
Function Sub
```

The argument must be declared as a normal variable, omitting only the `Dim` keyword. Here is an example that creates a function that takes a string as argument:

```
Private Sub EvaluateInvoice(EmplName As String, HourlySalary As Currency)
```

```
End Sub
```

A certain procedure can take more than one argument. In this case, in the parentheses of the procedure, separate the arguments with a comma. Here is an example of a sub routine that takes two arguments:

In the body of a procedure that takes one or more arguments, use the argument(s) as you see fit as if they were locally declared variables. For example, you can involve them with values inside of the procedure. You can also exclusively use the values of the arguments to perform the assignment.

3.6. Passing Arguments to Procedures



IMPORTANT TIP:

To learn more about passing and receiving variable in a sub procedure or function follow this link.

<http://www.functionx.com/visualbasic/functions/Lesson2.htm>

There are two ways to pass an argument to a procedure; by value or by reference.

- **Pass-by-Value** – is when only a copy of the argument is passed to the procedure. Because the procedure only has a copy, it cannot make changes to the original argument. The code is ByVal.

- **Pass-by-Reference** – here the procedure has access to the original argument and can make changes to the

original argument. The code is ByRef.

For the receiving procedure to accept an argument, it must also be equipped with a parameter located in the (). A parameter is a special variable that receives an argument being passed into the procedure. An example of a Sub procedure that uses a parameter is:

```
Sub DisplayValue(ByVal intNumber As Integer)
    MessageBox.Show("The value passed is " & intNumber.ToString)
End Sub
```

The *ByVal* intNumber as Integer statement declares the variable intNumber as an integer parameter. The *ByVal* keyword indicates that the arguments passed into the variable are passed by value. The parameter allows the DisplayValue proceducr to accept an integer value argument but it does NOT change the original value passed. Any changes made to intNumber will not affect the variable in the calling statement.

For example, the calling statement could be:

```
'code within a control on the form
Dim IntNumAs Integer = 5
Call DisplayValue(intNum)      'calling statement for Sub procedure
```

' 'code within the code window itself; not attached to any control

```
Private Sub DisplayValue(ByVal intNum As Integer)
    MessageBox.Show("The value passed in was " & intNum.ToString)
    intNum = intNum * 5

    MessageBox.Show("The new value of intNum in the subroutine is " & _
        &intNum.ToString)

End Sub
```

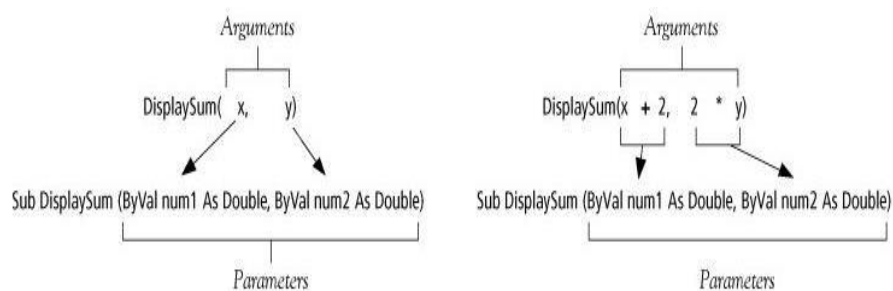
MessageBox.Show("The value of intNum in the program is " & intNum.ToString)

Output result:

The value passed in was 5

The new value of intNum in the subroutine is 25

The value of intNum in the program is 5



3.7. Passing Parameters By Reference

Unlike an argument passed by value, with an argument passed by reference, the procedure has access to the original argument. Any changes made to the parameter variable are actually performed on the original argument. To do this, you use the `ByRef` keyword in the declaration of the parameter variable.

3.8. Local Variables

When a variable is declared in an event or Sub procedure with a `Dim` statement, a portion of memory is set aside to hold the value of the variable. As soon as the `End Sub` statement for the procedure executes, the memory location is freed up; that is, the variable ceases to exist. The variable is said to be local to the procedure.

When variables of the same name are declared with Dim statements in two different procedures (either event or Sub), Visual Basic gives the variables separate identities and treats them as two different variables. A value assigned to a variable in one part of the program will not affect the value of the likenamed variable in the other part of the program.

3.9. Class-Level Variables

Visual Basic provides a way to make a variable visible to every procedure in a form's code without being passed. Such a variable is called a class-level variable. The Dim statement for a class-level variable can be placed anywhere between the statements Public Class formName and End Class, provided that the Dim statement is not inside a procedure.

Normally, we place the Dim statement just after the Public Class formName statement (We refer to this region as the Declarations section of the Code window.) A classlevel variable is visible to every procedure. When a class-level variable has its value changed by a procedure, the value persists even after the procedure has finished executing. We say that such a variable has class-level scope. Variables declared inside a procedure are said to have local scope.

In general, the scope of a variable is the portion of the program that can refer to it. Class-level scope also is referred to as module-level scope, and local scope also is referred to as procedure-level scope. If a procedure declares a local variable with the same name as a class-level variable, then the name refers to the local variable for code inside the procedure.



UNIT SUMMARY

- A general procedure is a portion of a program that is accessed by event procedures or other general procedures. The two types of general procedures are Sub procedures and Function procedures.
- Sub procedures are defined in blocks beginning with Sub statements and ending with End Sub statements. A Sub procedure is accessed (called) by a statement consisting of the name of the procedure.
- Function procedures are defined in blocks beginning with Function statements and ending with End Function statements. A function is invoked by a reference in an expression and returns a value.
- In any procedure, the arguments appearing in the calling statement must match the parameters of the Sub or Function statement in number, type, and order. They need not match in name.
- A variable declared in the Declarations section of the Code window is class-level. Such a variable is available to every procedure in the form's code and retains its value from one procedure invocation to the next.
- Variables declared with a Dim statement inside a procedure are local to the procedure. The values of these variables are reinitialized each time the procedure is called. A variable with the same name appearing in another part of the program is treated as a different variable.
- Structured programming uses modular design to refine large problems into smaller subproblems. Programs are coded using the three logical structures of sequences, decisions, and loops.
- The best practice for installing cable is to follow the TIA/EIA 568 specifications and the manufacturer's recommendations. Be careful not to exceed a cable's bend radius, untwist wire pairs more than one-half inch, or remove more than one inch of insulation from copper wire. Install plenum-rated cable in ceilings and floors, and run cabling away from where it might suffer physical damage. Maintain clear, comprehensive documentation on your cable plant.
- Visual Basic.NET (VB.NET) provides two types of procedures: Sub procedures and Function procedures. The difference between the two is that a Function procedure returns a value but a Sub procedure does not.
- A Sub procedure definition begins with a procedure header containing the keyword **Sub** followed by one or more statement and ends with the keywords **End Sub**.
- Arguments are passed into parameters. The parameters list consists of variable declarations that receive arguments being sent from client objects.
- A function procedure begins with a procedure header containing the keyword **Function** followed by one or more statement, and ends with the keywords **End Function**. Function procedure returns a single variable. You write its data type at the end of the procedure header.

- User-defined procedures are procedures that you design and write. These are in contrast to the methods contained in the Framework Class Library (FCL) classes.
- There are two categories of user-defined procedures: public and private. Private procedures are you write with in the module to provide services so that objects; private procedures cannot be invoked by code in other objects. In contrast, public procedures are procedures you write specifically designed to be invoked by other objects.
- The scope of a variable represents its visibility or accessibility and is determined by where you declare it. A variable declared within a procedure has procedure scope. However, a variable declared outside the procedures within a module is accessible to code within all of the procedures in the module-it has module scope.
- Variables can also have what is called block scope. When you write code that is grouped using an **If**, **Do** or **For**, and terminated with **End**, **Next**, and **Loop**, the code is called a block. If you declare a variable within such a code block, its scope is limited to that block.
- You create an optional arguments by adding the keyword **Optional** to a parameter variable declaration in the procedure header and then indicating a default value.
- Overloading a procedures means you write multiples procedures with the *same name* but with *different signatures*. Overloading is a convenient technique to avoid using unique names for procedures.
- You can overload a function procedure with a Sub procedure and vice versa. The return data type is not part of a procedure's signature.
- When passing an argument to a procedure, be aware of several different distinctions that interact with each other:
 - Whether the underlying programming element is modifiable or non modifiable
 - Whether the argument itself is modifiable or non modifiable
 - Whether the argument is being passed by value or by reference
 - Whether the argument data type is a value type or a reference type



KEY TERMS USED IN THIS SECTION

Procedure: A group of statements that together perform a task, does not return a value.

Function procedure: A group of statements that together perform a task, return a value to the part of the program that called the function procedure

Pass-by-Value – is when only a copy of the argument is passed to the procedure. Because the procedure only has a copy, it cannot make changes to the original argument. The code is ByVal):

Pass-by-Reference: the procedure has access to the original argument and can make changes to the original argument. The code is ByRef.

Local Variables: When a variable is declared in an event or Sub procedure with a Dim statement, a portion of memory is set aside to hold the value of the variable. As soon as the End Sub statement for the procedure executes, the memory location is freed up.

Class-Level Variables: A classlevel variable is visible to every procedure



UNIT3REVIEW QUESTIONS

1. Distinguish between Private and Public accessibility.
2. Explain the difference between an argument and a parameter.
3. What is an Optional parameter? What are the rules for writing one?
4. What is a user-defined procedure?
5. What is scope? Distinguish between module and procedure scope.
6. What is block scope?
7. What is a procedures signature?
8. What is an overloading procedure?
9. Write a program that uses a function COMPARE to compare two values input by the user. The program should state whether the first value is less than, equal to or greater than the second value.
10. Write a program that uses a sub-module called COMPARE to compare two values input by the user. The program should state whether the first value is less than, equal to or greater than the second value.
11. Write a program that input a series of integer and passes them one at the time to a function isEven, which uses the modulus operator to determine whether an integer is even. The function should take an integer argument and return true if the integer is even and false otherwise.
12. Write a sub procedure that displays as the solid square of asterisks whose side is specified in integer parameter size. For example, if side is 5, the sub procedure display the following:


```
*****
*****
*****
*****
*****
```
13. An integer is said to be prime if it's divided by only 1 and itself. For example, 2,3,5,7 are prime, but 4,6,8 and 9 are not.
 - a. Write a function that determine whether a number is prime.
 - b. Use this function in a program that determine and print all the prime numbers being 2 and 10, 000. How many of these numbers do you really have to test before being sure that you've found all the primes?

FOUR | DECISION MAKING



LEARNING OUTCOMES

- Use Relational and Logical Operators
- Use and Apply the If Block Statement
- Use Select Case Blocks



Visual Studio will automatically insert the "Then" and "End If" parts of an If-statement

4.1. Introduction

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:

Statement	Description
If ... Then statement	An If...Then statement consists of a boolean expression followed by one or more statements.
If...Then...Else statement	An If...Then statement can be followed by an optional Else statement , which executes when the boolean expression is false.
nested If statements	You can use one If or Else if statement inside another If or Else if statement(s).
Select Case statement	A Select Case statement allows a variable to be tested for equality against a list of values.
nested Select Case statements	You can use one select case statement inside another select case statement(s).

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.

```
Private SubADD()
'local variable definition
Dim a AsInteger=10

' check the boolean condition using if statement
If(a <20)Then
' if condition is true then print the following
    lblAnswer.Text = "A is less than 20")
EndIf

End Sub
```

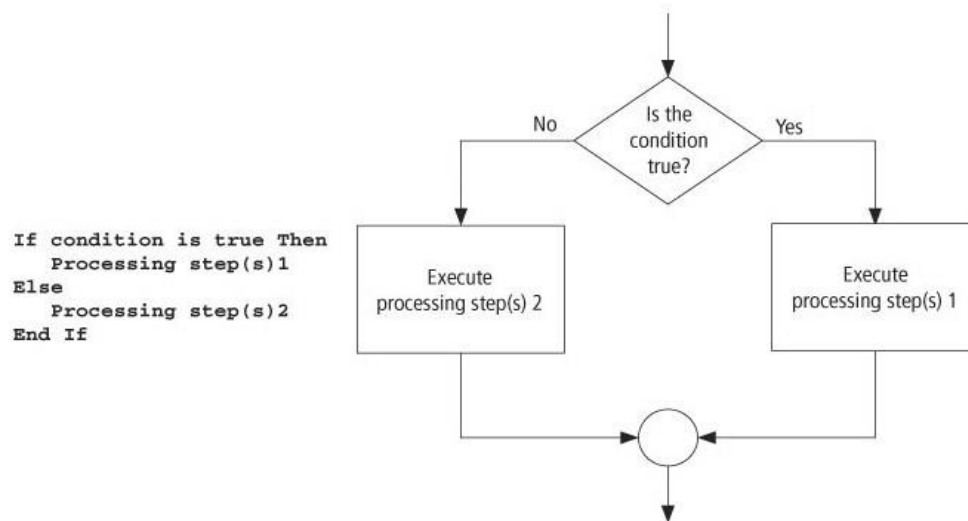
A condition is an expression involving relational operators (such as <and=) that is either true or false. Conditions also may incorporate logical operators (such as And, Or, and Not). ANSI values determine the order used to compare strings with the relational operators. Boolean variables and literals can assume the values True or False.

4.2. How to use IF ELSE in VB.NET

The conditional statement IFELSE, is use for examining the conditions that we provided, and making decision based on that condition. The conditional statement examining the data using comparison operators as well as logical operators.

```

If [your condition here]
    Your code here
Else
    Your code Here
End If
  
```



If the condition is TRUE then the control goes to between IF and Else block, that is the program will execute the code between IF and ELSE statements.

If the condition is FALSE then the control goes to between ELSE and END IF block, that is the program will execute the code between ELSE and END IF statements.

If you want to check more than one condition at the same time, you can use Else If.

Visual Basic programs display a Windows-style screen (called a form) with boxes into which users type (and in which users edit) information and buttons that they click to initiate actions. The boxes and buttons are referred to as controls. In this section, we examine forms and four of the most useful Visual Basic controls.

```

if [your condition here] Then
    Your code here
Else
    If [your condition here] Then
        Your code here
    Else
        If [your condition here] Then
            Your code here
        Else
            Your code here
        Endif
    Endif
Endif

```

Example 1:

- 1) If the marks is greater than 80 then the student get higher first class
- 2) If the marks less than 80 and greater than 60 then the student get first class
- 3) If the marks less than 60 and greater than 40 then the student get second class
- 4) The last condition is, if the marks less than 40 then the student fail.

Now here implementing these conditions

```

1.    If totalMarks >= 80 Then
2.        MsgBox("Got Higher First Class ")
3.    ElseIf totalMarks >= 60 Then
4.        MsgBox("Got First Class ")
5.    ElseIf totalMarks >= 40 Then
6.        MsgBox("Just pass only")
7.    Else
8.        MsgBox("Failed")
9.    End If

```

Line 1: Checking the total marks greater than or equal to 80

Line 2: If total marks greater than 80 show message - "Got Higher First Class"

Line 3: Checking the total marks greater than or equal to 60

Line 4: If total marks greater than 60 show message - "Got First Class"

Line 5: Checking the total marks greater than or equal to 40

Line 6: If total marks greater than 40 show message - "Just pass only"

Line 7: If those three conditions failed program go to the next coding block

Line 8: If all fail shows message "Failed"

Line 9: Ending the condition block

Example

```
Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim totalMarks As Integer
    totalMarks = 59

    If totalMarks >= 80 Then
        MsgBox("Gor Higher First Class ")
    ElseIf totalMarks >= 60 Then
        MsgBox("Gor First Class ")
    ElseIf totalMarks >= 40 Then
        MsgBox("Just pass only")
    Else
        MsgBox("Failed")
    End If
End Sub
End Class
```

4.3. Relational Operators

Mathematical Notation	Visual Basic Notation	Numeric Meaning	String Meaning
=	=	equal to	identical to
≠	<>	not equal to	different from
<	<	less than	precedes alphabetically
>	>	greater than	follows alphabetically
≤	<=	less than or equal to	precedes alphabetically or is identical to
≥	>=	greater than or equal to	follows alphabetically or is identical to

4.4. Logical Operators

Programming situations often require more complex conditions than those considered so far. For instance, suppose we would like to state that the value of a numeric variable, *n*, is strictly between 2 and

The proper Visual Basic condition is

$(2 < n) \text{ And } (n < 5)$

The condition $(2 < n) \text{ And } (n < 5)$ is a combination of the two conditions $2 < n$ and $n < 5$ with the logical operator And.



MORE INFO:

Follow the

link: <http://www.homeandlearn.co.uk/NET/nets1p20.html>

<http://www.homeandlearn.co.uk/NET/nets1p21.html>

The three main logical operators are And, Or, and Not. If cond1 and cond2 are conditions, then the condition.

cond1 And cond2

is true if both cond1 and cond2 are true. Otherwise, it is false. The condition

cond1 Or cond2

is true either cond1 or cond2 (or both) is true
Otherwise, it is false.

4.5. Select Case

In the previous lesson, we have learned how to control the program flow using the **If...Else If** control structure. In this chapter, you will learn another way to control the program flow, that is, the **Select Case** control structure.

However, the Select Case control structure is slightly different from the If... Else

The difference is that the Select Case control structure basically only make decision on one expression or dimension (for example the examination grade) while the If ...Else If

statement control structure may evaluate only one expression, each If....Else If statement may also compute entirely different dimensions. Select Case is preferred when there exist many different conditions because using If...Then..else If statements might become too messy.

The format of the Select Case control structure is show below:

A typical form of the Select Case block is:

```
Select Case Selector
    Case ValueList1
        Action1
    Case ValueList2
        Action2
    Case Else
        Action3
End Select
```



GOOD TIP:

Use Select Case over If Statement when you have several options to choose from.

where Case Else (and its action) is optional, and each value list contains one or more of the following types of items:

1. a literal;
2. a variable;
3. an expression;
4. an inequality sign preceded by Is and followed by a literal, variable, or expression;
5. a range expressed in the form a To b, where a and b are literals, variables, or expressions.

Different items appearing in the same list must be separated by commas. Each action consists of one or more statements. After the selector is evaluated, Visual Basic looks for the first value-list item including the value of the selector and carries out its associated action. (If the value of the selector appears in two different value lists, only the action associated with the first value list will be carried out.) If the value of the selector does not appear in any of the value lists and there is no Case Else clause, execution of the program will continue with the statement following the Select Case block.

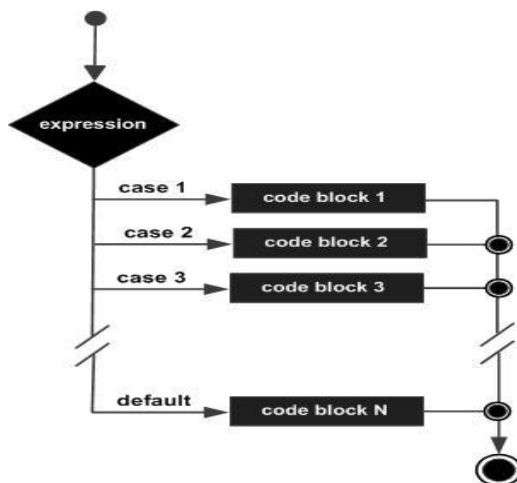
Example 2

```

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim age, price As Double
    age = CDbI(InputBox("What is your age?"))

    Select Case age
        Case Is < 6
            Price = 0
        Case 6 To 17
            Price = 3.75
        Case Is >= 17
            Price = 5
    End Select
End Sub

```

4.6. Flow chart for Select Case**MULTIMEDIA REFERENCE**

For deeper discussion on If Statement and Select Case follow the link:

www.youtube.com/watch?v=h8uR3zOiwUM
<http://www.vbtutor.net/lesson8.html>

```

Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim age, price As Double
    age = CDbI(InputBox("What is your age?"))

    Select Case age
        Case Is < 6
            Price = 0
        Case 6 To 17
            Price = 3.75
        Case Is >= 17
            Price = 5
    End Select
End Sub

```

Example

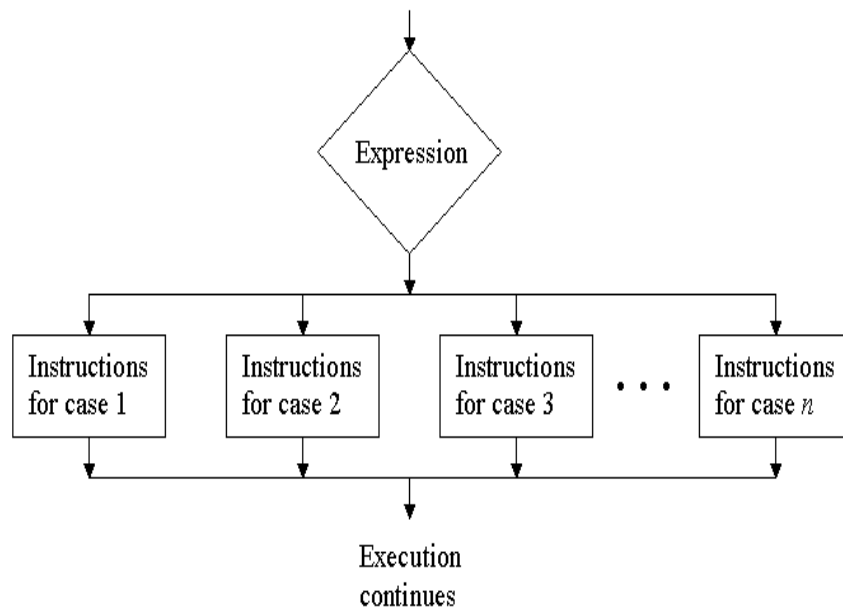
```

Dim age As Integer

age = CInt(TextBoxAge.Text)

Select Case age
    Case Is < 3
        MessageBox.Show("You are younger than Chris' niece")
    Case Is > 3
        MessageBox.Show("You are older than Chris' niece")
    Case Is = 3
        MessageBox.Show("You are 3, the same age as Chris' niece")
End Select

```



FOUR | DECISION MAKING

**GOOD PRACTICE:**

There are performance advantages in utilising the SELECT CASE STATEMENT but I wouldn't use that as the primary means of using it. Expressions like the IF Statement and SELECT CASE statement are typically executed in millionths of a second these days. Not even your most critical of users will notice the difference. However it is simply much easier to read. You should always, without exception, try to keep your code as clean and as elegant as possible. Your co-workers will thank you for it. You will thank yourself for it when you go to look at some old code. Take pride in your work. Be a good craftsman.



UNIT SUMMARY


- You make decision in a selection statement by writing a logical expression, which evaluates to either **true** or **false**.
- Visual Basic .NET (VB.NET) uses relational operators (**=, >, >=, <, <=, <>**) and logical operators (**Not, And, Or, Xor, AndAlso, OrElse**) in writing logical expressions.
- The logical operators join two logical expressions to form a compound expression.
- The logical operators **AndAlso** and **OrElse** correspond to the **And** and **Or** operators, respectively, except that they employ a short-circuit evaluation technique, which does not evaluate the second logical expression of a compound expression if its evaluation will not alter the results of the compound expression.
- A one-way selection statement evaluates a logical expression and then executes one or more statements only if the expression is true. The two-way selection statement also evaluates a logical expression and executes one or more statements if it is true, but executes one or more different statements if it is false.
- A flowchart is a graphical representation of logic. Flowcharts use symbols to represent the logical components of an algorithm.
- One-way selection is implemented in VB .NET by writing either a single-line or a multi-line **If** statements.
- You generally use a multi-line **If** when more than one statement is to be executed when the logical expression is true. However, you can substitute a multi-line **If** for a single-line **If**.
- Generally, a single-line **If** is easier to write when you want to execute a single statement if an expression is true. However, when you want to execute more than one statement, you use the multi-line **If**.
- You implement the multi-way selection structure by writing an **If** statement together with the keyword **Else**.
- VB .NET implements the multi-way selection structure, sometimes called the case structure, with the keywords **Select Case**. This statement acts like a multi-way **If** statement by transferring control to one. Use a **Select Case** statement to make a decision where there are more than two values of a variable you want to evaluate.



UNIT 4 REVIEW QUESTIONS

1. Correct the following code so that it compiles and executes correctly. Assume the variable have been correctly declared and populated.
`If hoursWorked Not> 40 Then otPay = 0`
2. Correct the following code so that it compiles and executes correctly. Assume the variable have been correctly declared and populated.
`If hoursWkd Not> 40 Then
otPay = 0`
3. What is the output of the following code?
`If 1 > 2 Or 5 < 6 Then
lblDisplay.Text = ("True")
Else
lblDisplay.Text = ("False")
End If`
4. What is the output of the following code?
`If 1 > 2 Xor 5 < 6 Then
lblDisplay.Text = ("True")
Else
lblDisplay.Text = ("False")
End If`
5. What is the output of the following code?
`If 1 > 2 OrElse 5 < 6 Then
lblDisplay.Text = ("True")
Else
lblDisplay.Text = ("False")
End If`
6. Correct the following code so that it compiles and executes correctly.
`Dim gpa As Double = 2.95
Dim cscMajor As Boolean = True
Select Case gpa
Case> 3
Console.WriteLine("Fantastic")
Case< 1
If cscMajor Then Console.WriteLine("Change Majors")
Case Else
Console.WriteLine("Meet With Advisor")
End Select`
7. What is the output of the following code?
`If 1 < 2 AndAlso 5 < 6 Then
Console.WriteLine("True")
Else
Console.WriteLine("False")
End If`

FIVE | REPETITION

	LEARNING OUTCOMES
	<ul style="list-style-type: none"> • Understand and utilise Do Loops • Process Lists of Data with Do Loops • Understand the For...Next Loop

5.1. Introduction

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.

A loop, one of the most important structures in Visual Basic, is used to repeat a sequence of statements a number of times. At each repetition, or pass, the statements act upon variables whose values are changing.

VB.Net provides following types of loops:

- Do Loop.
- For ... Next.
- For Each ... Next
- While..... End While.



Follow the link to watch video on loops:

www.youtube.com/watch?v=LPMRGKXAe-8

www.youtube.com/watch?v=V7WT5a8slic

www.youtube.com/watch?v=uY01sEPvUll

5.2. Do Loops

The Do loop repeats a sequence of statements either as long as or until a certain condition is true. A Do statement precedes the sequence of statements, and a Loop statement follows the sequence of statements. The condition preceded by either the word "While" or the word "Until", follows the word "Do" or the word "Loop". When Visual Basic executes a Do loop of the form

```
Do While condition
    statement(s)
Loop
```

```

Dim intMultiplied, intCounter As Integer

Do While intCounter < 100
    intMultiplied = intCounter * 2
    lblDisplay.Text = lblDisplay.Text & intMultiplied
    intCounter = intMultiplied + 10
Loop

```

*The above example will keep on until intCounter > 100



Thinking Point?
What will happen if initial counter is not initialized?

5.3. FOR...NEXT LOOPS

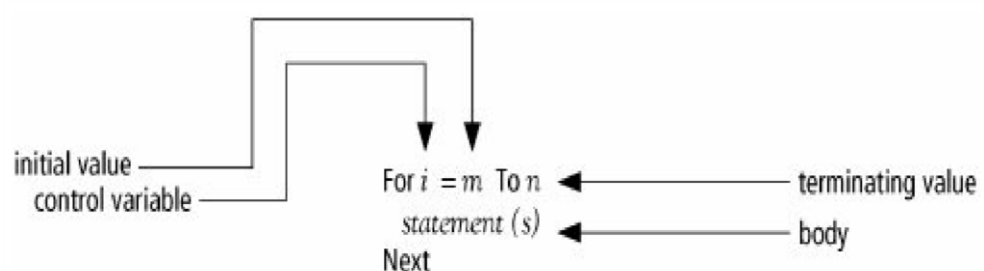
You use a **For...Next** structure when you want to repeat a set of statements a set number of times.

```

For counter [As datatype] = start To end [Step step]
    [ statements ]
    [ Continue For ]
    [ statements ]
    [ExitFor]
    [ statements ]
Next [ counter ]

```

When we know exactly how many times a loop should be executed, a special type of loop, called a For... Next loop, can be used. For...Next loops are easy to read and write, and have features that make them ideal for certain common tasks. The following code uses a For... Next loop to display a table:



```

Private Sub btnDisplayTable_Click(...) Handles
    btnDisplayTable.Click
    Dim intSub As Integer
        For IntSub = 1 To 5
            lstDisplay.Items.Add( intSub & " " & intSub ^ 2)
        Next
    End Sub

```

Output

```

1 1
2 4
3 9
4 16
5 25

```

A portion of a program of the form constitutes a For...Next loop. The pair of statements For and Next cause the statements between them to be repeated a specified number of times. The For statement designates a numeric variable, called the control variable, that is initialized and then automatically changes after each execution of the loop. Also, the For statement gives the range of values this variable will assume. The Next statement increments the control variable. If $m \leq n$, then i is assigned the values $m, m + 1, \dots, n$ in order, and the body is executed once for each of these values. If $m > n$, then the body is skipped and execution continues with the statement after the For... Next loop.

When program execution reaches a For... Next loop, such as the one shown previously, the For statement assigns to the control variable i the initial value m and checks to see whether i is greater than the terminating value n . If so, then execution jumps to the line following the Next statement. If $i \leq n$, the statements inside the loop are executed. Then, the Next statement increases the value of i by 1 and checks this new value to see if it exceeds n . If not, the entire process is repeated until the value of i exceeds n . Nested For... Next Loops The body of a For... Next loop can contain any sequence of Visual Basic statements. In particular, it can contain another For...Next loop. However, the second loop must be completely contained inside the first loop and must have a different control variable. Such a configuration is called nested For...Next loops.



Reference

For the link to watch video
(For....Next Loop.

www.youtube.com/watch?v=uvWusTkHO_S
http://www.homeandlearn.co.uk/NET/net_s3p1.html
<http://howtostartprogramming.com/vb-net/vb-net-tutorial-16-do-while/>

5.4. While Loop

Whenever you face a situation in programming to repeat a task for several times (more than one times) or you have to repeat a task till you reach a condition, in these situations you can use loop statements to achieve your desired results.

While .. End While Loop execute the code body (the source code within While and End while statements) until it meets the specified condition. The expression is evaluated each time the loop is encountered. If the evaluation result is true, the loop body statements are executed.

```
While [condition]
[loop body]
End While
```

Condition : The condition set by the user

```
1. counter = 1
2. While (counter <= 10)
3.   Message
4.   counter = counter + 1
5. End While
```



RESEARCH:

For further read on the while....End while consult this book: Visual Basic.net Black Book. New Edition. page 87 -91

Line 1: Counter start from 1

Line 2: While loop checking the counter if it is less than or equal to 10

Line 3: Each time the Loop execute the message and show

Line 4: Counter increment the value of 1 each time the loop execute

Line 5: End of the While End While Loop body

```
Private Sub Button1_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim counter As Integer
    counter = 1
    While (counter <= 10)
        MsgBox("Counter Now is : " & counter)
        counter = counter + 1
    End While
End Sub

End Class
```



Topic Activities:

The following activities determines the output displayed when the button is clicked: **Private Sub btnDisplay_Click(...) Handles btnDisplay.Click**

```
Dim intSub As Integer
intSub = 3
Do While intSub < 3
    intSub = 2 * intSub - 1
Loop
lblAnswer.Text
```

```
Dim balance As Double = 1000
Dim interest As Double = 0.1
Dim n As Integer = 0    'Number of years
Do
    lstOutput.Items.Add(n & " " & balance)
    balance = (1 + interest) * balance
    n += 1
Loop Until (balance > 1200)

lstOutput.Items.Add(n)
```

```
'Display a message
Dim num As Double = 4
Dim message As String = ""
Do
    Select Case num
        Case 1
            message = "grammer!"
            num = -1
        Case 2
            message = "re a su"
            num = (5 - num) * (3 - num)
        Case 3
            message = "per
pro"
num            num = 4 -
        Case 4
            message = "You a"
            num = 2 * num - 6
        End Select

        txtOutput.Text &= message
    Loop Until (num = -1)
```



Topic Activities:

The following activities determines the output displayed when the button is clicked: **Private Sub btnDisplay_Click(...) Handles btnDisplay.Click**

'Display a table of the first 5 numbers and their squares

```
Dim i As Integer
For i = 1 To 5
    lstTable.Items.Add(i & " " & i ^ 2)
Next
```

[The following is displayed in the list box.]

```
1      1
2      4
3      9
4     16
5     25
```

```
Dim i As Integer
For i = 1 To 7step 2
    lstTable.Items.Add(i)
Next
```

[The following is displayed in the list box.]

```
1
3
5
7
```

```
'Display a message
Dim num As Double = 4
Dim message As String = ""
Do
    Select Case num
        Case 1
            message = "grammer!"

            num = -1

        Case 2
            message = "re a su"
            num = (5 - num) * (3 - num)

        Case 3
            message
= "per pro"
            num = 4 - num
        Case 4
            message = "You a"
            num = 2 *

num - 6
    End Select

    txtOutput.Text &= message
Loop Until (num = -1)
```




SECTION SUMMARY

- You use the iteration structure, also called a loop, to execute one or more statement repeatedly.
 - Basic iteration logic first tests to see if a loop should terminate. If the loop does not terminate, the statement in the loop body are executed. Control then returns to the beginning where once again the test determines whether the loop should terminate. The loop continues in this manner until the terminating condition occurs.
 - There are two kinds of loop logic: the pre-test loop and post-test loop. Using pre-test logic, the terminating condition is checked before the body of the loop is executed. A post-test loop checks for the terminating condition after the body is executed. When you use a post-test loop, the statement in the loop are executed at least once regardless of the terminating condition.
 - There are two general approaches to controlling the number of times a loop executes: counter control and sentinel control. Counter-control logic employs a variable called a counter to counter the number of times the loop has executed. Sentinel-control logic checks the user input for a specific values and terminates when this values, called a sentinel, is detected.
 - You use a counter-controlled loop when you want iterate a specific predetermined number of times. You employ a sentinel-controlled loop when you do not know how many value many value are to be input and you use a sentinel value to indicate there is no more input.
 - You implement the iteration structure in Visual Basic .NET (VB .NET) using either the Do statement or the For statement.
 - The Do statement uses two forms: Do While and Do Until. You indicate the end of the loop by writing the keyword Loop.
 - The Do While loop continues to execute as long as an expression is true.
 - The Do Until loop continues to execute until an expression is true.
 - You can write either a counter-controlled loop or a senile-controlled loop using the Do statement. You can also create either a pre-test or a post-test loop.
 - You use the For Next structure to create counter-controlled pre-test loops only. Although theoretically possible, you generally do not use For Next to write sentinel-controlled loops.



SECTION 5 REVIEW QUESTIONS



Give the output of programs below:

1. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 For j As Integer = 2 To 8 Step 2
 ListBox.Items.Add(j)
 Next
 ListBox.Items.Add("Who do we appreciate?")
 End Sub

2. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 For countdown As Integer = 10 To 1 Step -1
 ListBox.Items.Add(countdown)
 Next
 ListBox.Items.Add("blastoff")
 End Sub

3. Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
 For j As Integer = 2 To 8 Step 2
 ListBox.Items.Add(j)
 Next
 ListBox.Items.Add("Who do we appreciate?")
 End Sub

4. Dim counter, sum As Integer
 sum = 1000
 For counter = 100 To 5 Step -5
 sum -= counter
 ListBox1.Items.Add(sum)
 Next
5. Dim n as Integer
 For n=1 to 10
 If n>6 then
 Exit For
 End If

 Else
 ListBox1.Items.Add (n)
 Next
 End If
 Next



FIVE | REVIEW QUESTIONS

6. Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click

Dim sum, n As Integer

ListBox1.Items.Add("n" & vbTab & "Sum")

ListBox1.Items.Add("-----")

Do

n += 1

sum += n

ListBox1.Items.Add(n & vbTab & sum)

If n = 100 Then

Exit Do

End If

Loop

End Sub

7. What is the purpose of loop in programming?

- A. Repeat operation(s) many times
- B. make decision

8. The Exit command is used to exit a loop in VB.NET language.

- A. True
- B. False

9. Which of the followings is not a nested loop?

A. For i=0 to 9
For j=1 to i+3
Next
Next

B. For i=0 to 9
lblDisplay.Text = i
Next

For j=1 to i+3
lblDisplay.Text = j
Next

C. For i=0 to 9
While(j Mod 2<>0)
lblDisplay.Text = j

j+=1

End While

Next

10. Which of the following statements about the while loop is not true?
 - A. The while loop is a posttest loop.
 - B. Testing condition is made before each iteration.
 - C. The while loop statement must terminate with a semi-colon.
11. Write VB.NET code to display the asterisk pattern as shown below:


```
*****
*****
*****
*****
*****
```
12. Find error on the following code:


```
Dim intSub As Integer
While intSub <= 10
    lblDisplay.Text = intSub
Wend
```
13. Write a program that use both For.. Next and While statements to sum a sequence of

Integers.
14. Write a program that uses Do Until to calculate and print the product of the odd integer from 1 to 15.
15. Write VB.NET code to display the asterisk pattern as shown below:



```
*
**
***
****
*****
```
16. Rewrite the following loop using **Do While**.


```
Dim counter As Integer
For counter = 5 To 1 Step -1
    lstDisplay.items.Add(counter)
Next
```
17. Rewrite the following loop using **For Next**.


```
Dim counter As Integer = 1
Do While counter <= 9
    lstDisplay.items.Add (counter)
    Counter += 3
Loop
```
18. Rewrite the following loop using **Do Until**.


```
Dim counter As Integer = 1
Do While counter <=9
    lstDisplay.items.Add (counter)
    Counter += 3
Loop
```

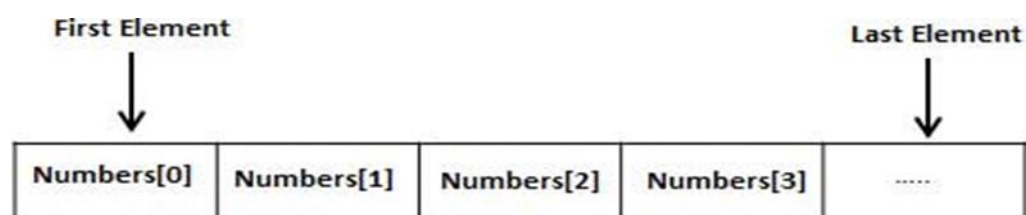
SIX | ARRAYS

	LEARNING OUTCOMES
	<ul style="list-style-type: none"> • Understand the basic concept of defining and declaring an array. • Use and apply arrays to programs • Define two dimensional array • Sort and search for elements in the array

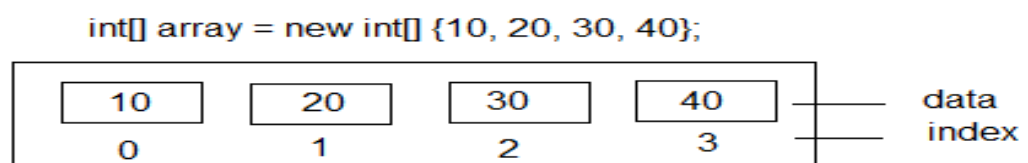
6.1. Introduction

This topic introduces the important topic of data structure – collections of related data items, 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 contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

**6.2. Declaring an Array**

Visual Basic provides a data structure called an array that lets us do what we tried to accomplish in the loop. The variable names, similar to those in the preceding program, will be



student(0), student(1), student(2), student(3), ..., student(29) and
score(0), score(1), score(2), score(3), ..., score(29)

We refer to these collections of variables as the array variables student() and score(). The numbers inside the parentheses of the individual variables are called subscripts, and

each individual variable is called a subscripted variable or element. For instance, `student(3)` is the fourth subscripted variable of the array `student()`, and `score(20)` is the 21st subscripted variable of the array `score()`. The elements of an array are assigned successive memory locations.

Array variables have the same kinds of names as simple variables. If `arrayName` is the name of an array variable and `n` is an Integer literal, variable, or expression, then the declaration statement.



To learn why arrays are important in programming, for this link.

<http://www.homeandlearn.co.uk/NET/nets6p1.html>

Dim arrayName(n) As varType

reserves space in memory to hold the values of the subscripted variables `arrayName(0)`, `arrayName(1)`, `arrayName(2)`, ..., `arrayName(n)`. The value of `n` is called the upper bound of the array. The number of elements in the array, `n+1`, is called the size of the array. The subscripted variables will all have the same data type; namely, the type specified by `varType`. For instance, they could be all String variables or all Integer variables. In particular, the statements.

Dim score(29) As Double

Dim score(29) As Double

declare the arrays needed for the preceding program.

Values can be assigned to individual subscripted variables with assignment statements and displayed in text boxes and list boxes just as values of ordinary variables. The default initial value of each subscripted variable is the same as with an ordinary variable; that is, the keyword `Nothing` for String types and `0` for numeric types. The statement

Dim score(29) As Double

sets aside a portion of memory for the Integer array `score()` and assigns the default value `0` to each element.

score(0) = 87

score(1) = 92

assign values to the zeroth and first elements.

The statements

For i As Integer = 0 To 2

`lstBox.Items.Add(score(i))`

Next

then produce the following output in the list box:

87
92
0

As with an ordinary variable, an array declared in the Declarations section of the Code window is classlevel. That is, it will be visible to all procedures in the form, and any value assigned to it in a procedure will persist when the procedure terminates. Array variables declared inside a procedure are local to that procedure and cease to exist when the procedure is exited.

```
Dim intData(30) ' 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:

```
Sub DISPLAY()
Dim arrNumber (10) As Integer ' n is an array of 11 integers '
Dim intSub, intIndex As Integer ' initialize elements of array n '

For intSub = 0 To 10
    arrNumber (intSub) = i + 100 ' set element at location i to i + 100
Next i

' output each array element's value '
For intIndex = 0 To 10
    LsrDisplay.Items.Add("Element ", intIndex, arrNumber (intIndex))
Next intIndex
End Sub
```

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
```

Example

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim array As Integer() = New Integer(3) {}
        array(0) = 10
        array(1) = 20
        array(2) = 30
        array(3) = 40

        For i As Integer = 0 To array.Length - 1
            MessageBox.Show(array(i))
        Next
    End Sub
End Class
```

We can declare and initialize an array in one statement.

```
Dim array As Integer() = New Integer() {10, 20, 30, 40}
```

Note that in the above code we did not specify the length of the array so the compiler will do it for us.

6.3. Declaring and Initializing a String Array

Dim week(6) As String

The above Vb.Net statements means that , an Array named as week declared as a String type and it can have the capability of seven String type values.

```
week(0) = "Sunday"
```

```
week(1) = "Monday"
```

In the above statement , we initialize the values to the String Array. week(0) = "Sunday" means , we initialize the first value of Array as "Sunday" ,

```
Dim weekName as String = week(1)
```

We can access the Arrays elements by providing its numerical index, the above statement we access the second value from the week Array.

In the following program, we declare an Array "week" capability of seven String values and assigns the seven values as days in a week. Next step is to retrieve the elements of the Array using a For loop. For finding the end of an Array we used the Length function of Array Object.

```
Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim i As Integer
    Dim week(6) As String
    week(0) = "Sunday"
week(1) = "Monday"
    week(2) = "Tuesday"
    week(3) = "Wednesday"
    week(4) = "Thursday"
    week(5) = "Friday"
    week(6) = "Saturday"
    For i = 0 To week.Length - 1
        MsgBox(week(i))
    Next
End Sub
End Class
```

6.4. How to resize an array

An array can be resized with `Array.Resize < T > Method`, that means We make an array bigger or smaller. `Array.Resize < T > Method` Changes the number of elements of a one-dimensional array to the specified new size.

`Array.Resize < T >` - T is the type of the elements of the array.

This method should be used with only one dimensional Array. This method allocates a new array with the specified size, copies elements from the old array to the new one, and then replaces the old array with the new one.

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim sArray As Char() = New Char(4) {}
        sArray(0) = "A"
        sArray(1) = "B"
        sArray(2) = "C"
        sArray(3) = "D"
        sArray(4) = "E"

        For i As Integer = 0 To sArray.Length - 1
            MessageBox.Show(sArray(i).ToString())
        Next

        Array.Resize(sArray, 3)

        For i As Integer = 0 To sArray.Length - 1
            MessageBox.Show(sArray(i).ToString())
        Next
    End Sub
End Class
```

`Array.Resize(sArray, 3)`

In the above code we resize the array to 3 elements.

6.5. How to Use ForEach with Arrays ?

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim array As Integer() = {10, 30, 50}
        'array declaration
        For Each element As Integer In array
            MsgBox(element)
        Next
    End Sub
End Class
```

6.6. How to check if a value exists in an array?

The following program shows how to find a specified value from an Array.

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click

        Dim stringToCheck As String = "GHI"
        Dim stringArray() As String = {"ABC", "DEF", "GHI", "JKL"}

        For Each x As String In stringArray

            If x.Equals(stringToCheck) Then
                MessageBox.Show("Find the string ..." + x)
            End If
        Next
    End Sub
End Class
```



MULTIMEDIA REFERENCE

Click the link to watch tutorial on
dynamic arrays.
<https://www.youtube.com/watch?v>

6.7. The GETUPPERBOUND Method

The value of `arrayName.GetUpperBound(0)`

Dynamic Arrays

After an array has been declared, its size (but not its type) can be changed with a statement of the form `ReDim arrayName(m)` where `arrayName` is the name of the already declared array and `m` is an Integer literal, variable, or expression. Note: Since the type cannot be changed, there is no need for an "As dataType" clause at the end of the `ReDim` statement.

Visual Basic allows you to declare an array without specifying an upper bound with a statement of the form `Dim arrayName() As varitype`

Later, the size of the array can be stipulated with a `ReDim` statement.

The `ReDim` statement has one shortcoming: It causes the array to lose its current contents. That is, it resets all String values to Nothing and resets all numeric values to 0. This situation can be remedied by following `ReDim` with the word `Preserve`. The general form of a `ReDim Preserve` statement is

`ReDim Preserve arrayName(m)`

Of course, if you make an array smaller than it was, data in the eliminated elements will be lost.

```
Private Sub MARKS()
    Dim arrMarks () As Integer
    ReDim arrMarks (2)
    arrMarks (0) = 85
    arrMarks (1) = 75
    arrMarks (2) = 90

    ReDim Preserve arrMarks (10)
    arrMarks (3) = 80
    arrMarks (4) = 76
    arrMarks (5) = 92
    arrMarks (6) = 99
    arrMarks (7) = 79
    arrMarks (8) = 75

    For i = 0 To 10
        lblResults.Items.Add(i & vbTab & arrMarks (i))
    Next i
End Sub
```

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

```
0      85
1      75
2      90
3      80
4      76
5      92
6      99
7      79
8      75
9      0
10     0
```

6.8. **Multi-Dimensional Arrays**

The arrays you have seen so far are only column arrays, 1 dimensional arrays. The arrays you have seen so far are only column arrays, 1 dimensional arrays as they are known.

A 1-D array holds one column of data:

```
array(0) = "0"
array(1) = "1"
array(2) = "2"
array(3) = "3"
```

Visually, it looks like this:

	Col 1
Row 1	0
Row 2	1
Row 3	2
Row 4	3

So we have four rows with one value in each column.

Another type of array is a 2 Dimensional array. This holds data in a grid pattern, rather like a spreadsheet. They look like this:

	Col 1	Col 2	Col 3	Col 4
Row 1	0, 0	0, 1	0, 2	0, 3
Row 2	1, 0	1, 1	1, 2	1, 3
Row 3	2, 0	2, 1	2, 2	2, 3
Row 4	3, 0	3, 1	3, 2	3, 3

We still have 4 rows, but now we have 4 columns. The values in the image above represent the array positions. Before, we only had to worry about positions 0 to 3. We could then place a value at that position. Now, though, we can have values at position such as 0, 0, and 2, 1.

VB.Net allows multidimensional arrays. Multidimensional arrays are also called rectangular arrays.

```
Dim twoDStringArray(10, 20) As String
```

or, a 3-dimensional array of Integer variables:

```
Dim threeDIntArray(10, 10, 10) As Integer
```

The following program demonstrates creating and using a 2-dimensional array:

```
Module arrayApl
  Sub Main()
    ' an array with 5 rows and 2 columns
    Dim a(,) As Integer = {{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4, 8}}
    Dim i, j As Integer
    ' output each array element's value '
    For i = 0 To 4
      For j = 0 To 1
        lblDisplay.Text = lblDisplay.Text ("a[{0},{1}] = {2}", i, j, a(i, j))
      Next j
    Next i

    End Sub
  End Module
```

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

Output

```
a[0,0]: 0
a[0,1]: 0
a[1,0]: 1
a[1,1]: 2
a[2,0]: 2
a[2,1]: 4
a[3,0]: 3
a[3,1]: 6
a[4,0]: 4
a[4,1]: 8
```

To set up a 2-D array in Visual Basic .NET you just add a second number between the round brackets:

```
Dim grid(3, 3) As Integer
```

Again, arrays start counting at 0, so the above declarations means set up an array with 4 rows and 4 columns.

To store values in the grid, you need both numbers between the round brackets:

```
grid(0, 0) = 1
```

This means fill row 0, column 0 with a value of 1. You could type out all your positions and values like this: (The left number between the round brackets is always the row number; the right number is always the columns.)

```
grid(0, 0) = 1
```

```
grid(1, 0) = 2
```

```
grid(2, 0) = 3
```

```
grid(3, 0) = 4
```

```
grid(0, 1) = 5
```

```
grid(1, 1) = 6
```

```
grid(2, 1) = 7
```

```
grid(3, 1) = 8
```

```
grid(0, 2) = 9
```

```
grid(1, 2) = 10
```

```
grid(2, 2) = 11
```

```
grid(2, 2) = 12
```

```
grid(0, 3) = 13
```

```
grid(1, 3) = 14
```

```
grid(2, 3) = 15
```

```
grid(3, 3) = 16
```

Typically, though, 2-D arrays are filled using a double for loop. The following code just places the numbers 1 to 16 in the same positions as in the example above:

```
Dim grid(3, 3) As Integer
Dim row As Integer
Dim col As Integer
Dim counter As Integer = 1

For row = 0 To 3

    For col = 0 To 3

        grid(row, col) = counter
        counter = counter + 1

    Next Col

Next
```

Notice how the outer loop goes from **row = 0 to 3** and the inner loop goes from **col = 0 to 3**. This allows you to fill all the rows and columns in the array.

To display all the above values in a listbox, you could have this code:

```
Dim temp as String = ""

For row = 0 To 3

    For col = 0 To 3

        temp = temp & "-" & grid(row, col).ToString()

    Next

    ListBox1.Items.Add (temp)
    temp = ""

Next
```


6.9. Sorting Arrays

Array sorting, are two words that would strike the fear on many young programmers. Modern programmers rarely have to worry about sorting algorithms, and nor should they; they've been done to death by folk far brighter than ourselves.

There are a wide variety of routines for sorting, with different characteristics, and today we look at the Selection Sort algorithm. This algorithm is relatively simple and performs reasonably well, especially with smaller datasets. To sort an array in ascending order the Selection Sort algorithm is as follows:

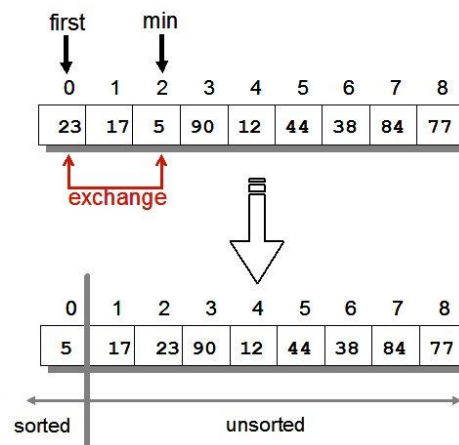


MULTIMEDIA REFERENCE

Click the link to watch tutorial on sorting array.

http://www.youtube.com/watch?v=Y4V8dvf_wDM&feature=player_embedded

1. Find the smallest element in the list.
2. Exchange the element in the first position and the smallest element. Now the smallest element is in the first position.
3. Repeat Step 1 and 2 with the list having one less element (i.e., the smallest element is discarded from further processing).



This is the result of one pass.

To implement this in Visual Basic we can use the following Subroutine

```
Private Sub SortArray(ByRef array() As Integer)

    Dim i As Integer
    Dim j As Integer
    Dim minimum As Integer
    Dim swapValue As Integer
    Dim upperBound As Integer
    Dim lowerBound As Integer

    lowerBound = LBound(array)
    upperBound = UBound(array)

    For i = lowerBound To upperBound

        minimum = i

        For j = i + 1 To upperBound
            'Search for the smallest remaining item in the array
            If array(j) < array(minimum) Then
                'A smaller value has been found, remember the
                'position in the array
                minimum = j
            End If
        Next j

        If minimum <> i Then
            'Swap array Values
            swapValue = array(minimum)
            array(minimum) = array(i)
            array(i) = swapValue
        End If

    Next i
End Sub
```



SECTION SUMMARY

- In VB.NET, arrays are instances of the Arrays class.
- To declare an array, you write the keyword DIM followed by the name of the reference variable you want to use. Next you write the index of the last element of the array in the parentheses.
- A series of variables with the same name is called arrays.
- The individual values referred to as elements, and each element is accessed by its subscript, which is a position number.
- VB.NET numbers the array elements beginning with zero(0). This means the index of the first element is 0; the second element is 1, and so forth.
- Arrays can be either one-dimensional or multi-dimensional.
- One-dimensional array consists of elements arranged in a single row. Conceptually, two-dimensional array has both columns and rows.
- You can access the individual elements of the one-dimensional array by writing the arrays reference variable followed by the index of the element enclosed in parenthesis.
- You can resize array size by using keyword [ReDim](#) statement.
- Arrays may be multidimensional. A two-dimensional table contains rows and column and is processed similarly to a one-dimensional array. Accessing a multidimensional array frequently requires the use of nested loop.
- Array subscripts or indexes are zero based. They must be integers in the range of the array elements.
- A special form of the For loop called For Each is available for working with arrays.



SECTION 6 REVIEW QUESTIONS

- What is the size of an array whose upper bound is 100?
- What is the upper bound of an array whose size is 100? In peer-to-peer networks, only one computer can send and receive transmissions on the network.
- Determine the output displayed when the button is clicked

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim num As Integer = 2
    Dim spoon(num) As String
    spoon(0) = "soup"
    spoon(1) = "dessert"
    spoon(2) = "coffee"

    txtOutput.Text = "Have a "& spoon(num - 1) & " spoon."
End Sub
```

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim a(20) As Integer
    a(5) = 1
    a(10) = 2
    a(15) = 7

    lstOutput.Items.Add(a(5) + a(10))
    lstOutput.Items.Add(a(5 + 10))
    lstOutput.Items.Add(a(20))
End Sub
```

- What would be printed from the following VB.NET program?

```
Dim arr() As Integer = New Integer({1,3,5,7,9}) 'initialize arr array
Dim i As Integer
For i = 4 to 0 Step -1
    lblDisplay.Text = lblDisplay.Text & arr(i) & vbCrLf
Next
```



SECTION 6 REVIEW QUESTIONS

- An arrays are related by the fact that they have the same _____ and _____
- A(n) _____ should be used to declare the size of an array
- An array that uses two subscripts is referred to as a(n) _____

State whether the following is true or false.

- An array can store many different types of values.
- An array subscript can be data type of double.

Write statement(s) that perform the following tasks.

- Declare the array to be integer array that have 5 elements.
- Declare the array to be integer array and to have 3 rows and 3 columns.
- elements