

# SOFTWARE DEVELOPMENT USING VISUAL BASIC – TABLE OF CONTENTS

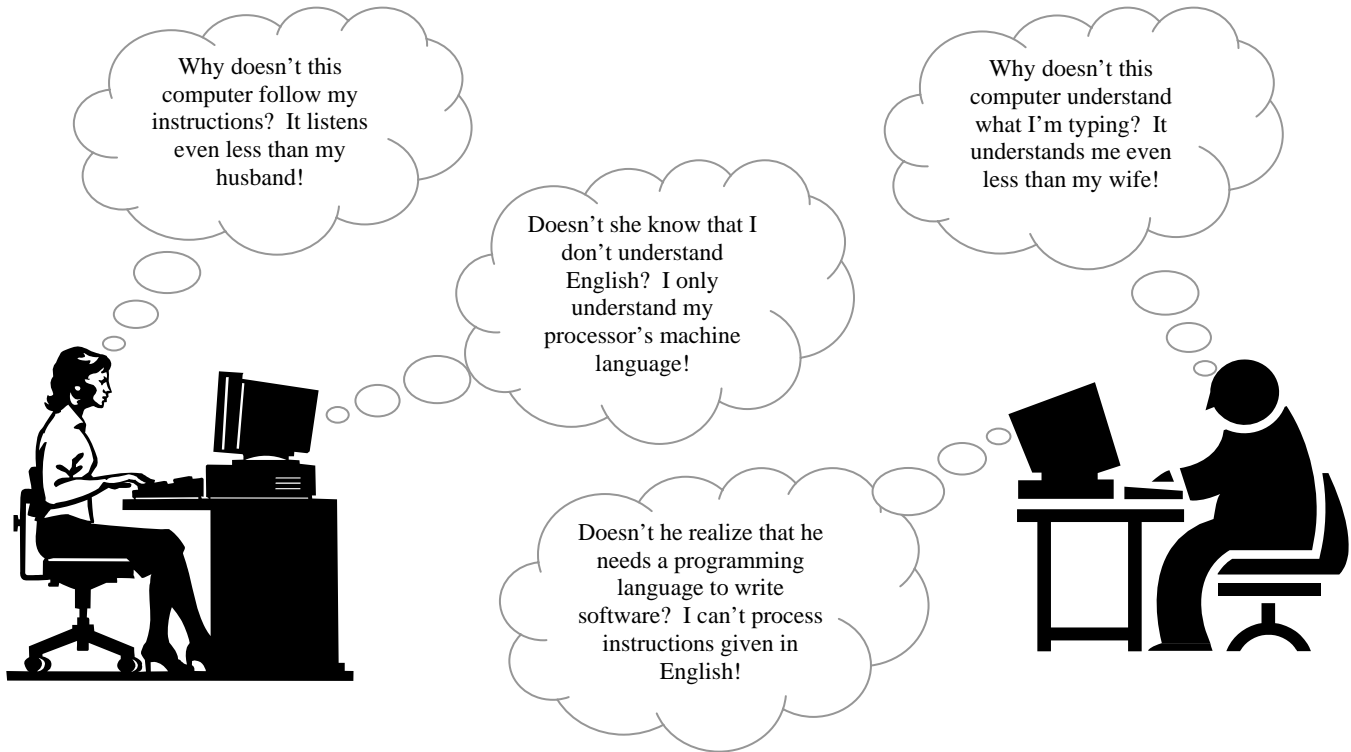
<b>SOFTWARE DEVELOPMENT USING VISUAL BASIC – TABLE OF CONTENTS</b> .....	<b>1</b>
<b>OBJECTS IN VISUAL BASIC, OBJECT-ORIENTED PROGRAMMING AND VISUAL PROGRAMMING</b> .....	<b>4</b>
WHAT IS A PROGRAM? WHAT IS A PROGRAMMING LANGUAGE? .....	4
WHAT IS OBJECTED-ORIENTED PROGRAMMING? (SIMPLIFIED DESCRIPTION FOR PROGRAMMING NOVICES).....	4
WHAT IS AN EVENT-DRIVEN PROGRAMMING LANGUAGE?.....	4
OBJECTS IN VISUAL BASIC.....	5
EVENTS COMMONLY USED IN VISUAL BASIC.....	5
QUESTIONS .....	6
<b>OBJECTS, PROPERTIES, EVENTS AND METHODS IN VB: CREATING YOUR FIRST VB PROGRAM THAT PROCESSES NUMERIC INFORMATION</b> .....	<b>8</b>
INSTRUCTIONS TO BE READ CAREFULLY AND FOLLOWED .....	8
QUESTIONS .....	11
<b>A PROGRAM THAT PROCESSES STRING (TEXT) INFORMATION</b> .....	<b>12</b>
EXTREMELY IMPORTANT QUESTIONS .....	12
<b>VARIABLES IN MICROSOFT VISUAL BASIC</b> .....	<b>14</b>
INTRODUCTION .....	14
SPECIFIC ASPECTS OF VARIABLES IN VISUAL BASIC.....	14
RULES FOR NAMING VARIABLES IN MICROSOFT VISUAL BASIC.....	14
SUMMARY.....	14
DATA TYPES USED TO STORE INTEGER VALUES (WHOLE NUMBER VALUES, POSSIBLY INCLUDING A NEGATIVE SIGN) .....	15
DATA TYPES USED TO STORE FLOATING-POINT AND FIXED-POINT VALUES (NUMBERS WITH A FRACTIONAL PART I.E., “DECIMALS”) .....	15
DATA TYPES USED TO STORE TEXT.....	15
MISCELLANEOUS DATA TYPES .....	15
QUESTIONS .....	16
<b>WHAT IS THE DIFFERENCE BETWEEN AN OBJECT AND A VARIABLE?</b> .....	<b>17</b>
VARIABLES .....	17
OBJECTS .....	17
QUESTIONS .....	17
<b>REVIEW OF ESSENTIAL CONCEPTS IN VISUAL BASIC</b> .....	<b>18</b>
DATA (INFORMATION) – A PARTIAL LIST OF VB DATA TYPES.....	18
A COMPUTER AS A DATA PROCESSING MACHINE.....	18
SOME USEFUL INTRINSIC (BUILT-IN) FUNCTIONS.....	18
<b>A COMPLETE LIST OF VISUAL BASIC DATA TYPES</b> .....	<b>19</b>
DATA (INFORMATION).....	19
<b>UNDERSTANDING VISUAL BASIC PROGRAMMING</b> .....	<b>20</b>
REVIEW: CONCEPTS THAT YOU NEED TO UNDERSTAND BEFORE YOU CAN CREATE GOOD VB PROGRAMS.....	20
<b>WRITING YOUR OWN CODE: CURRENCY CONVERTER PROGRAM</b> .....	<b>21</b>
THE MOST IMPORTANT LESSON OF THE ENTIRE UNIT .....	21
PLANNING AND DEVELOPING SOLUTIONS TO SOFTWARE DEVELOPMENT PROBLEMS.....	21
<i>Wrong!!!! (Most Students)</i> .....	21
<i>Right!!!!!! (George Polya)</i> .....	21
<i>How these Steps Apply to Software Development</i> .....	21
EXAMPLE OF A GENERAL PROBLEM.....	22
<i>Step 1 – Analysis (Understand the Problem)</i> .....	22
<i>Step 2 – Design (Choose a Strategy)</i> .....	22
<i>Step 3 – Implementation (Carry out the Strategy)</i> .....	22
<i>Step 4 – Validation (Check the Solution)</i> .....	23

<b><u>VISUAL BASIC PRACTICE PROBLEMS: INPUT, PROCESSING, OUTPUT .....</u></b>	<b><u>23</u></b>
<b><u>VB REVIEW 1- IMPORTANT PROGRAMMING TERMINOLOGY .....</u></b>	<b><u>24</u></b>
<b><u>VB REVIEW 2- TRANSLATING MATH FORMULAS INTO VB .....</u></b>	<b><u>26</u></b>
<b><u>LEARNING ABOUT SELECTION STATEMENTS (IF STATEMENTS) BY STUDYING THE PYTHAGOREAN THEOREM PROGRAM.....</u></b>	<b><u>27</u></b>
GENERAL STRUCTURE OF AN IF STATEMENT .....	27
AN IMPROVED VERSION OF THE PYTHAGOREAN THEOREM PROGRAM .....	28
<i>Picturing “If” Statements.....</i>	28
EXERCISES .....	28
<b><u>THE AREA CALCULATOR PROGRAM - ANOTHER PROGRAM THAT REQUIRES “IF” STATEMENTS .....</u></b>	<b><u>29</u></b>
INTRODUCTION – WHAT YOU WILL LEARN BY STUDYING THE AREA CALCULATOR.....	29
WHAT YOU NEED TO DO.....	29
QUESTIONS .....	29
<b><u>PIZZA PROGRAM SOLUTIONS AND QUESTIONS .....</u></b>	<b><u>30</u></b>
THE PROBLEM.....	30
THE PLAN .....	30
<b><u>SEQUENCE, SELECTION AND REPETITION: THE UNDERPINNINGS OF PROGRAMMING .....</u></b>	<b><u>32</u></b>
SEQUENCE .....	32
SELECTION.....	32
REPETITION.....	32
QUESTIONS AND PROGRAMMING EXERCISES .....	32
<b><u>SELECTED SOLUTIONS TO ASSIGNED VB PROBLEMS .....</u></b>	<b><u>34</u></b>
SOLUTION 1 .....	34
QUESTIONS RELATED TO SOLUTION 1.....	34
SOLUTION 2 .....	35
QUESTIONS RELATED TO SOLUTION 2.....	35
<b><u>THE EVOLUTION OF SOFTWARE PART I: HOW TO KEEP IMPROVING YOUR SOFTWARE.....</u></b>	<b><u>36</u></b>
CASE STUDY – DEVELOPING A CRAPS GAME IN VB .....	36
A SOLUTION TO PHASE ONE .....	36
<i>Questions.....</i>	36
TWO DIFFERENT SOLUTIONS TO PHASE TWO.....	37
<i>Solution 1 – Craps 1.1a.....</i>	37
<i>Questions.....</i>	37
<i>Solution 2 – Craps 1.1b.....</i>	38
THINKING ABOUT PHASES THREE AND FOUR .....	38
<b><u>THE EVOLUTION OF SOFTWARE PART II: FURTHER IMPROVEMENTS TO THE CRAPS SOFTWARE.....</u></b>	<b><u>40</u></b>
QUESTIONS .....	40
RESEARCH ASSIGNMENT.....	41
QUESTIONS .....	41
<b><u>DO YOU UNDERSTAND THE CRAPS PROGRAM? .....</u></b>	<b><u>42</u></b>
QUESTIONS .....	42
<b><u>THE TEMPERATURE CONVERTER PROGRAM.....</u></b>	<b><u>45</u></b>
QUESTIONS TO CONSIDER BEFORE WRITING CODE.....	45
<b><u>OPERATORS IN VISUAL BASIC .....</u></b>	<b><u>47</u></b>
ARITHMETIC OPERATORS .....	47
MATHEMATICAL FUNCTIONS .....	47
COMPARISON OPERATORS .....	48
LOGICAL OPERATORS .....	48
TRUTH TABLES FOR THE LOGICAL OPERATORS .....	48

<u>OPERATOR PRECEDENCE (ORDER OF OPERATIONS).....</u>	<u>49</u>
<u>INFREQUENTLY USED OPERATORS.....</u>	<u>49</u>
<u>TYPE CONVERSION FUNCTIONS.....</u>	<u>49</u>
<b><u>IMPROVING YOUR VISUAL BASIC PROGRAMS.....</u></b>	<b><u>50</u></b>
<u>ROUNDING OFF VALUES.....</u>	<u>50</u>
<u>ROUND FUNCTION.....</u>	<u>50</u>
<u>USING THE KEYPRESS EVENT TO “TRAP” INVALID CHARACTERS.....</u>	<u>50</u>
<u><i>Argument</i>.....</u>	<u>50</u>
<u><i>Square Brackets</i>.....</u>	<u>50</u>
<u><i>Numeric Expression</i>.....</u>	<u>50</u>
<u>KEY CODE CONSTANTS IN VISUAL BASIC.....</u>	<u>51</u>
<b><u>USING MESSAGE BOXES IN VB PROGRAMS.....</u></b>	<b><u>53</u></b>
<u>SYNTAX.....</u>	<u>53</u>
<u>SETTINGS.....</u>	<u>53</u>
<u>RETURN VALUES.....</u>	<u>54</u>
<u>EXAMPLES.....</u>	<u>54</u>
<u>QUESTIONS.....</u>	<u>54</u>
<u>MORE QUESTIONS.....</u>	<u>54</u>
<u>ANOTHER EXAMPLE.....</u>	<u>55</u>
<u>EXERCISE.....</u>	<u>55</u>
<b><u>LEARNING NEW PROGRAMMING CONCEPTS BY STUDYING EXAMPLES.....</u></b>	<b><u>56</u></b>
<u>LEFT FUNCTION.....</u>	<u>58</u>
<u>RIGHT FUNCTION.....</u>	<u>58</u>
<b><u>OBJECT NAMING CONVENTIONS.....</u></b>	<b><u>59</u></b>
<u>SUGGESTED PREFIXES FOR CONTROLS.....</u>	<u>59</u>
<u>SUGGESTED PREFIXES FOR DATA ACCESS OBJECTS (DAO).....</u>	<u>60</u>
<u>SUGGESTED PREFIXES FOR MENUS.....</u>	<u>61</u>
<u>CHOOSING PREFIXES FOR OTHER CONTROLS.....</u>	<u>61</u>

# OBJECTS IN VISUAL BASIC, OBJECT-ORIENTED PROGRAMMING AND VISUAL PROGRAMMING

## What is a Program? What is a Programming Language?



A **computer program** is a set of instructions that can be **executed** (carried out) by a computer's CPU. A **programming language** is any precise and unambiguous language that can be used by a computer programmer to create a **computer program**. Examples of programming languages include C, C++, C#, J#, Perl, Java, Python, Cobra, Visual Basic, Pascal, Delphi, Turing, FORTRAN and COBOL.

## What is Objected-Oriented Programming? (Simplified Description for Programming Novices)

Object-oriented programming is in many ways similar to the process of manufacturing automobiles. Companies such as Ford, Chrysler and General Motors assemble automobiles, but many of the parts that are used are made by other companies. It is not commercially viable for auto manufacturers to produce all the parts needed to build cars. Instead, nuts, bolts, spark plugs, fan belts, upholstery and a host of other parts are purchased from companies that specialize in the production of such parts.

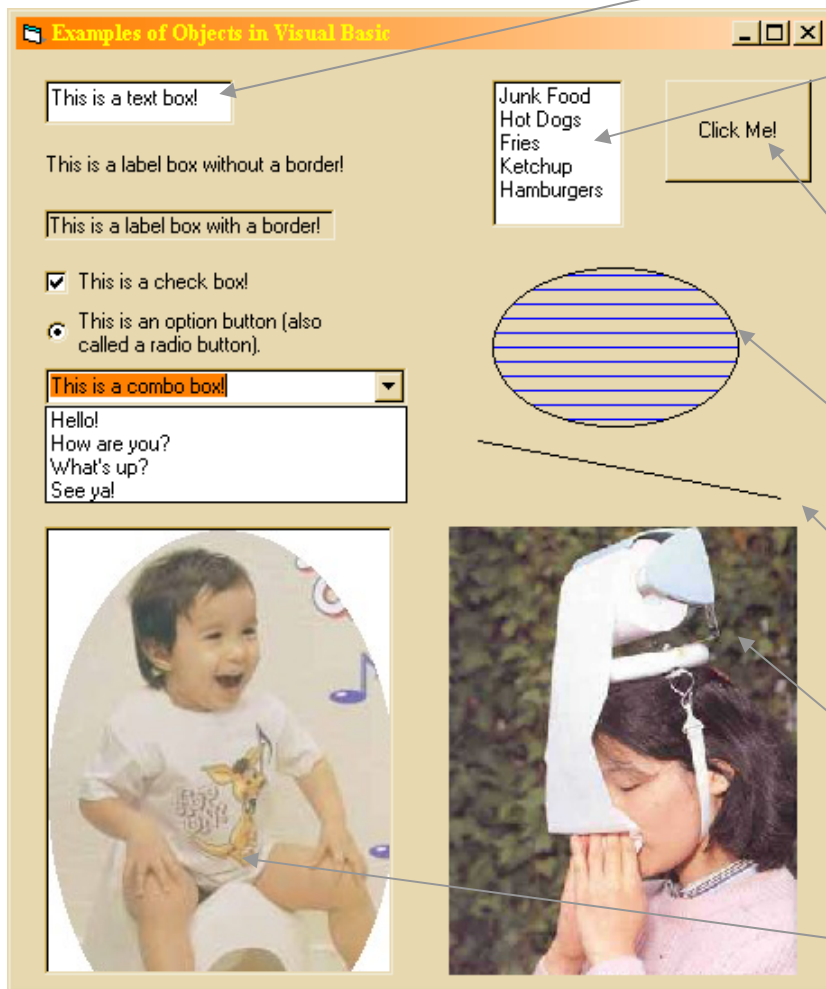
Similarly, object-oriented languages allow us to construct programs using reusable **objects** such as command buttons, text boxes, label boxes and picture boxes. Instead of writing new code (programming instructions) for commonly used objects every time a new program is written, the code is written only once (often by the developers of the language). Whenever we need to employ one of these commonly used objects, we simply use the code that is already provided for us.

In a **visual programming language** like Visual Basic, the process of creating a **user interface** is as simple as clicking and dragging. There is no need to learn how to write code for commonly used objects because it is automatically generated as the user interface is designed **visually**.

## What is an Event-Driven Programming Language?

Programs written in an event-driven programming language respond to **events** such as clicking a command button, changing the value of a text box or moving the mouse pointer.

## Objects in Visual Basic



This object is called a **text box**.

This object is called a **list box**.

This object is called a **form**. It is used to **contain** all the other objects.

This object is called a **command button**.

This object is called a **shape**.

This object is called a **line**.

This object is called an **image control**.

This object is called a **picture box**.

## Events Commonly used in Visual Basic





Event	Meaning
Click	The "Click" event occurs when the user presses and then releases a mouse button (left or right) over an object. It can also occur when the value of a control is changed. For a "Form" object, this event occurs when a blank area or a disabled control is clicked.
DbClick	The "DbClick" event occurs when the user presses and then releases a mouse button and then presses and releases it again over an object. For a "Form" object, this event occurs when a blank area or a disabled control is double clicked.
DragDrop	The "DragDrop" event occurs when a drag-and-drop operation is completed as a result of dragging a control over an object and releasing the mouse button or using the "Drag" method with its action argument set to 2 ("Drop").
DragOver	The "DragOver" event occurs when a drag-and-drop operation is in progress. You can use this event to monitor the mouse pointer as it enters, leaves or rests directly over a valid target. The mouse pointer position determines the target object that receives this event.
MouseDown	The "MouseDown" event occurs when a mouse button is pressed.
MouseUp	The "MouseUp" event occurs when a mouse button is released.
MouseMove	The "MouseMove" event occurs when the mouse pointer is moved across the screen.

## Questions

1. Complete the following table. You should *not* blindly copy what I have written. You must demonstrate your understanding by *paraphrasing*.

(a) Define the terms “paraphrase” and “unambiguous.”	
(b) What is a computer program?	
(c) What is a programming language?	
(d) What is the difference between a computer user and a computer programmer? Does a computer user need to understand programming? Is a computer programmer also a computer user?	
(e) What is an object? What is object-oriented programming? Explain the concept of visual programming.	
(f) List one use of each of the following objects in programs that you have used (such as a word processor or Internet browser): text box, label box, picture box, check box, option button, combo box, list box	
(g) Define the following terms:  user interface, event, visual programming language, event-driven programming language	

2. Complete the following table. The first row is done for you!

<i>Picture of Familiar Object</i>	<i>Name of Object</i>	<i>List at least five(5) PROPERTIES of the object</i>
	Office Chair	<ul style="list-style-type: none"> <li>(a) Colour of the upholstery</li> <li>(b) Material upholstery is made of</li> <li>(c) Colour of the frame</li> <li>(d) Material the frame made of</li> <li>(e) Does the chair swivel?</li> <li>(f) Is the height of the seat adjustable?</li> <li>(g) Does the chair have armrests?</li> <li>(h) Is the backrest adjustable?</li> </ul>
		
		
		

# OBJECTS, PROPERTIES, EVENTS AND METHODS IN VB: CREATING YOUR FIRST VB PROGRAM THAT PROCESSES NUMERIC INFORMATION

*Instructions to be Read carefully and followed*

## 1. LAUNCH VISUAL BASIC:

Click the **START** button, then move the mouse pointer to **Programming**, then to **Microsoft Visual Studio 6.0** and finally to **Visual Basic 6.0**. Click on **Visual Basic 6.0** and wait for the VB program to load.

2. Once the VB program loads, you will see a dialogue box as shown below:

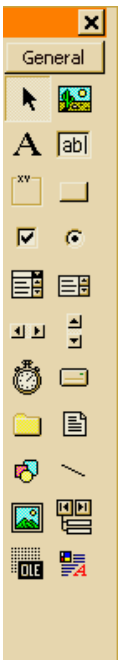


Choose the “Standard EXE” option and then click on “Open.”

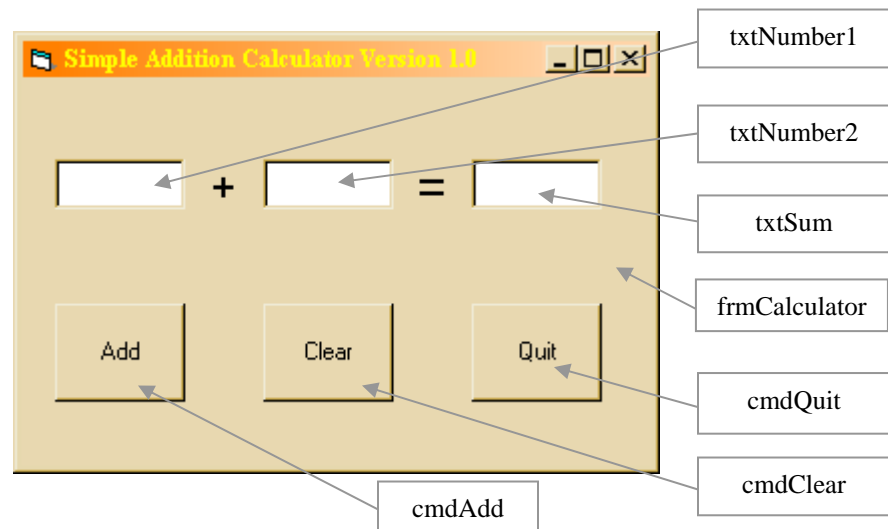
### Note

“Standard EXE” stands for “standard executable.” Choosing this option will allow you to create a standard Windows executable file. If you have ever explored the files stored on your computer’s hard drive, you probably will have noticed several files with a “.exe” extension. For example, if Microsoft Word is installed on your computer, you will find a file named “winword.exe.” Files with a “.exe” extension are called executable files and they contain the instructions that are executed (carried out) by a CPU.

3.

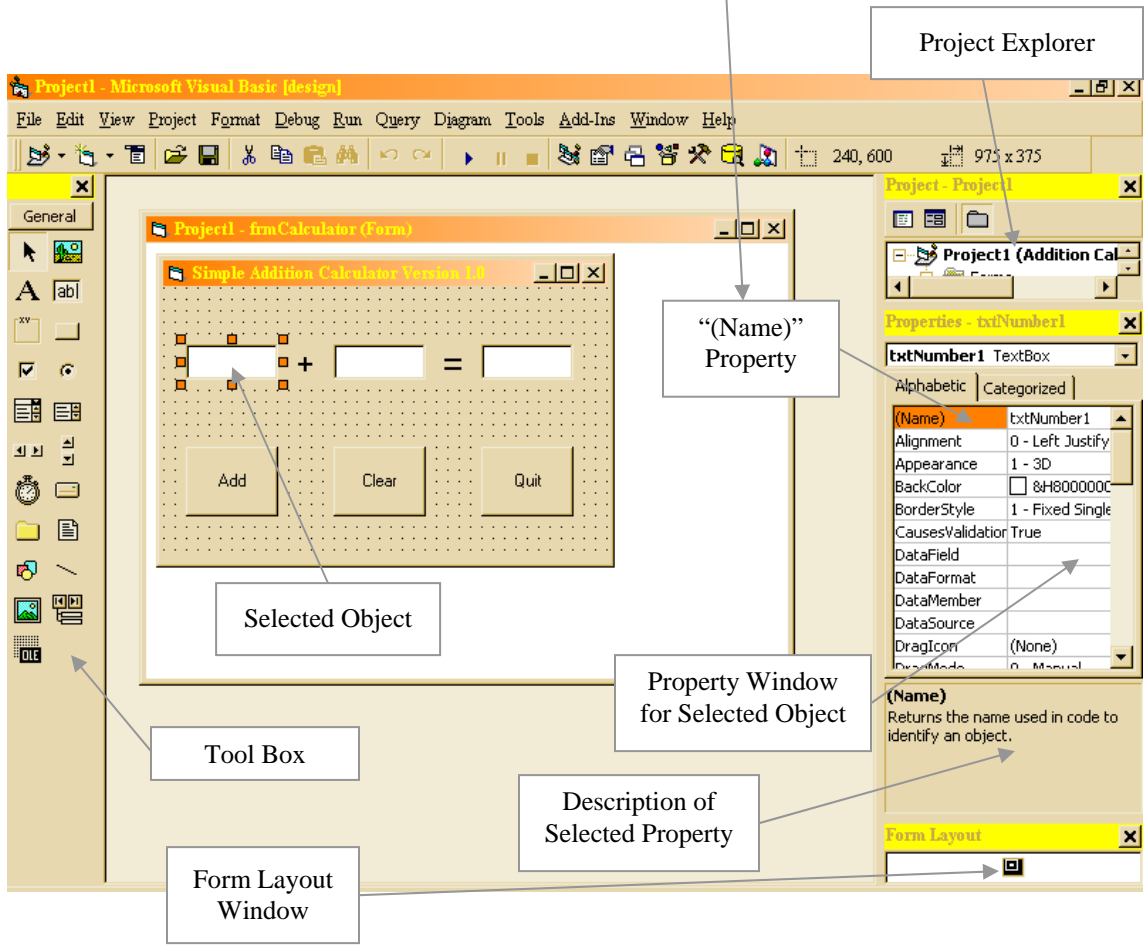


Use the Visual Basic Tool Box (shown at left) to create a form exactly like the one shown below. Use the given names to name the objects on the form (see step 4 on the next page). **NOTE:** To change the “message” that appears on a command button or a title bar of a form, change the value of the “**Caption**” property (see step 4 again).

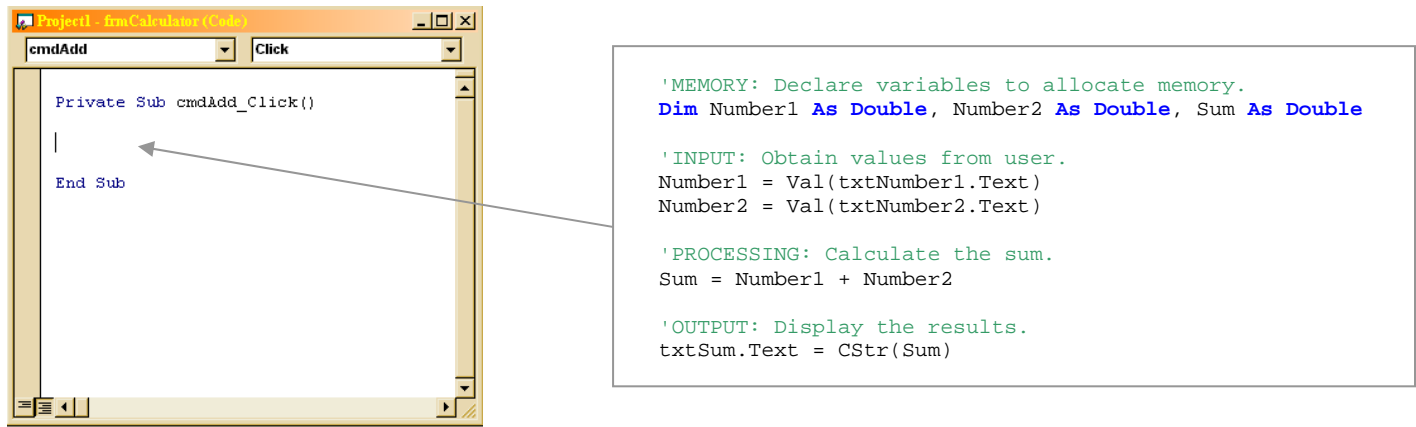




4. The default object names assigned by Visual Basic, names like “Text1” or “Command2,” are not very helpful because they do not describe the functions of the objects bearing these names. *A much better practice is to use a descriptive name for each object.* This helps us understand the purpose of each object and makes our code (programming instructions) much easier to read. To improve the readability of your first VB program, change the “(Name)” property of each object on your form to the name given in step three.



5. Now change the “Locked” property of the “txtSum” text box to “True.” This prevents the user from editing the answer that is displayed in the text box. In addition, change the **Text** property of all three text boxes so that nothing is displayed in each text box.
6. Double click on the “Add” command button. Then type the following code where indicated in the diagram.



7. Double click on the “Clear” command button. Then type the given code where indicated in the diagram.

```
cmdClear Click
Private Sub cmdAdd_Click()
    'MEMORY: Declare variables to allocate memory.
    Dim Number1, Number2, Sum As Double

    'INPUT: Obtain values from user and store in variables.
    Number1 = Val(txtNumber1.Text)
    Number2 = Val(txtNumber2.Text)

    'PROCESSING: Calculate the sum and assign to "Sum"
    Sum = Number1 + Number2

    'OUTPUT: Display results.
    txtSum.Text = Trim(Str(Sum))
End Sub

Private Sub cmdClear_Click()

End Sub
```

Notice that the code that appears between “Private Sub cmdAdd\_Click ( )” and “End Sub” is indented one TAB space to the right. This is done to make the code easier to read. Please ensure that *all your code is indented in the same way!*

'Clear all the text boxes by assigning 'the NULL STRING to the "Text" property 'of each text box.  
 txtNumber1.Text = ""  
 txtNumber2.Text = ""  
 txtSum.Text = ""

8. Double click on the “Quit” command button. Then type the given code where indicated in the diagram.

```
cmdQuit Click
'INPUT: Obtain values from user and store in variables.
Number1 = Val(txtNumber1.Text)
Number2 = Val(txtNumber2.Text)

'PROCESSING: Calculate the sum and assign to "Sum"
Sum = Number1 + Number2

'OUTPUT: Display results.
txtSum.Text = Trim(Str(Sum))
End Sub

Private Sub cmdClear_Click()
    'Clear all the text boxes by assigning 'the NULL STRING to the "Text" property 'of each text box.
    txtNumber1.Text = ""
    txtNumber2.Text = ""
    txtSum.Text = ""
End Sub

Private Sub cmdQuit_Click()
|
End Sub
```

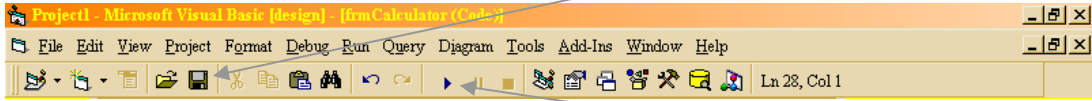
'End the program  
**End**

9. Type the command “Option Explicit” (without the quotation marks) at the very top of the program. By the time you are done, your code should look as follows:

```
Option Explicit
Private Sub cmdAdd_Click()
    'MEMORY: Declare variables to allocate memory.
    Dim Number1 As Double, Number2 As Double, Sum As Double
    'INPUT: Obtain values from user and store in variables.
    Number1 = Val(txtNumber1.Text)
    Number2 = Val(txtNumber2.Text)
    'PROCESSING: Calculate the sum and assign to "Sum"
    Sum = Number1 + Number2
    'OUTPUT: Display results.
    txtSum.Text = CStr(Sum)
End Sub
Private Sub cmdClear_Click()
    'Clear all the text boxes by assigning the NULL STRING to the "Text" property
    'of each text box.
    txtNumber1.Text = ""
    txtNumber2.Text = ""
    txtSum.Text = ""
End Sub
Private Sub cmdQuit_Click()
    'End the program
    End
End Sub
```

**10. SAVE YOUR WORK! IT IS CRITICAL THAT YOU UNDERSTAND THIS STEP!** Click on the “save” button. If you have not already saved your work, you will be asked to provide *two filenames*. One filename is used for the *project file* and it will have a “.vbp” filename extension (“vbp” stands for “Visual Basic Project”). The other filename is used to store the *form file* and it will have a “.frm” filename extension (“frm” stands for “form”). Of these two files, the *form file* is *by far the more important!* The form file stores all the Visual Basic code that you create as well as information regarding the appearance of your form. The project file becomes more important when you create programs that have two or more forms. Think of the project file as an “umbrella” file that contains information on each of the forms in your program. *To avoid serious difficulties, store the project file in the same folder as all your forms!*

Click the “save” button to save your program.



Click the “play” button to run your program.

**11.** Carefully check your code. When you are confident that it is correct, *execute* (run) your program by clicking the “play” button on the Visual Basic standard tool bar. Test your program carefully to ensure that it works properly.

**Questions**

**1.** Define the terms “object” and “property.”

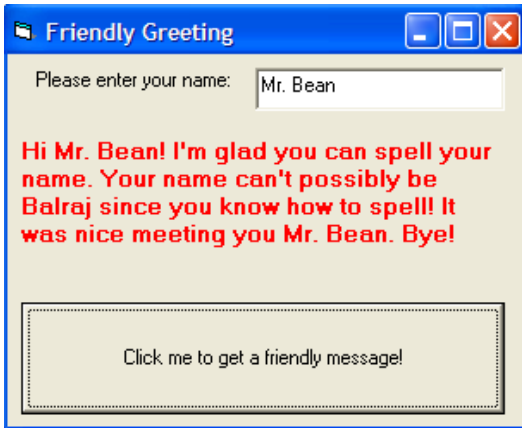
**2.** List names of the types of objects that you have used so far. In addition, list the corresponding properties that you have learned so far. The first one is done for you.

Object Type	Important Properties and Their Purpose
Command Button	<p><b>(Name)</b> Stores the name used in code to identify an object.</p> <p><b>Caption</b> Stores the text displayed on the command button.</p>

**3. Properties** store *characteristics* of objects. **Methods** are *procedures* (operations) that can be performed on objects. For example, the **SetFocus** method can be used to set the “focus” to any control that can accept input. If used on a text box, the **SetFocus** method causes the cursor to appear in the text box, making it unnecessary for the user to click in the text box. If used on a command button, the **SetFocus** method causes a “dotted” rectangle to appear on the button. When such a rectangle appears, the user does not need to click on the button. It is enough to press the space bar or ENTER.

Add the code “txtNumber1.SetFocus” (without the quotation marks) just before the “**End Sub**” in the “cmdAdd\_Click( )” and “cmdClear\_Click( )” sub procedures. **Describe the effect of this change on your program and explain how this change improves your program.**

## A PROGRAM THAT PROCESSES STRING (TEXT) INFORMATION



```
Option Explicit
Private Sub cmdMessage_Click()
'MEMORY: Variable declaration
Dim UserName As String
'INPUT
UserName = Trim(txtName.Text)
'OUTPUT
lblMessage.Caption = "Hi " & UserName _
    & "! I'm glad you can spell your name. " _
    & "Your name can't possibly be Balraj " _
    & "since you know how to spell! " _
    & "It was nice meeting you " & UserName _
    & ". Bye!"
End Sub
```

### Extremely Important Questions

1. The first statement in every VB program should be “**Option Explicit**.” What is its purpose? How does it help you to *debug* your programs? What can go wrong if you forget to include it?
2. An apostrophe (single quotation mark) is used to begin certain statements in VB. (The word “**Rem**” can also be used to begin this type of statement.) What are such statements called? What is their purpose? How does the computer process such statements? How can these statements be used to remove a statement from a program without deleting it?
3. A “**Sub**” is a program *subroutine* or *subprogram*, that is, a *portion of a program that is named* so that it can be accessed whenever needed. The “**Sub**” shown above is automatically named “cmdMessage\_Click” when you double click the “cmdMessage” command button. Explain how VB determines this name.
4. The statement “**Dim** UserName **As** **String**” is used to *declare the variable* “UserName.” The *name* of the variable being declared is \_\_\_\_\_. Its *type* is \_\_\_\_\_, which means that it is used to store \_\_\_\_\_ information. Declaring variables helps programmers to \_\_\_\_\_ their programs, allows an operating system to determine how much \_\_\_\_\_ is needed to store the values of the variables and it helps to determine which \_\_\_\_\_ can be used to process information of a given \_\_\_\_\_.

5. The statement `UserName = Trim(txtName.Text)` is called an *assignment statement* because it is used to *assign* (give) a *value* to a *variable*. Complete the following:
- Name of the variable being assigned a value: \_\_\_\_\_
- Name of the object from which a property is being used in the assignment statement \_\_\_\_\_
- Name of the property whose value is being assigned to the variable: \_\_\_\_\_
- Purpose of the “Trim” intrinsic (built-in) function: \_\_\_\_\_
6. Explain the difference between the *name of a variable* and the *value of a variable*. Give an example to illustrate your answer.
7. Explain the difference between the *name of an object* and the *name of a variable*. Give an example to illustrate your answer.
8. What is the purpose of the “&” operator? What is it called? To what *type of data* does it apply?
9. What is the purpose of *quotation marks* in VB programs? What will happen if you forget to use quotation marks when they are needed? What will happen if you use quotation marks when they are *not* needed?
10. What is the purpose of using a space followed by an underscore? Why is this useful?

# VARIABLES IN MICROSOFT VISUAL BASIC

## Introduction

Most of us are familiar with variables in a mathematical *context*. As you probably already know, a *variable* is a quantity that is capable of *assuming* any of a set of values. In other words, variables are used to *represent* (stand for) unspecified or unknown values. We typically use symbols such as letters to represent such quantities. For example, when we write “ $x \in \mathbb{N}$ ,” we mean that the letter  $x$  represents a quantity that can assume any *natural number* (i.e. 1, 2, 3, ...) as its value.

A variable that is used in a computer program is not very different from one that would be used in mathematics. There is only one *major* difference. Whereas in mathematics we use single letters (such as  $x$ ,  $y$  and  $z$ ) as variable names, programming languages allow us to use *longer and more descriptive names*. This is done for two very simple reasons.

First, in mathematics, we rarely need to use more than three or four different variables to solve any given problem; there is never any danger of running out of variable names. In programming, however, it is quite possible that a program could require the use of far more variables than can be *accommodated* by the English alphabet.

Second, and more importantly, it is simply not possible to restrict variable names to single characters and at the same time write programs that are easy to read and understand. It is important to realize that software developers spend approximately *eighty percent of their time modifying and debugging existing software* (most of which is written by other people). If programmers fail to communicate effectively their thoughts and methods, the task of completing projects in a timely fashion becomes difficult, if not impossible.

## Specific Aspects of Variables in Visual Basic

*Variables* are *placeholders* that are used to store values in memory (RAM); they have *names* and *data types*. The values of variables must be stored in such a way that the CPU of a computer can access them quickly whenever necessary. Therefore, *primary storage* (main memory, “RAM”) is the logical choice. Since it is always wise to conserve memory, and since some types of data are incompatible with other types, programming languages provide the programmer with various *data types*, each having different memory requirements and different applications.

The *data type* of a variable determines how the bits representing those values are stored in a computer’s memory. When you *declare* a variable, you can also supply a data type for it. All variables have a data type that determines what kind of data they can store. By default, if you do not supply a data type, the variable is given the **Variant** data type. The **Variant** data type is like a *chameleon* — it can represent many different data types in different situations. You do not have to convert between these types of data when assigning them to a **Variant** variable. Visual Basic automatically performs any necessary conversion.

If you know that a variable will always store data of a particular type, however, Visual Basic can handle those data more efficiently if you declare a variable of that type. For example, a variable to store a person’s name is best represented as a **String** data type, because a name is always composed of characters. Data types apply to other things besides variables. When you assign a value to a *property*, that value has a data type. In fact, just about anything in Visual Basic that involves data also involves data types.

### Rules for Naming Variables in Microsoft Visual Basic

- The first character of a variable name must be a letter.
- Variable names can contain the letters “a” to “z” or “A” to “Z,” the underscore and the digits 0 to 9.
- Variable names are case sensitive (e.g. ‘sum’ is different from ‘Sum’).
- The maximum length of a variable name is 255 characters.

## Summary

- Variables are used to *store values that need to be accessed by a computer’s CPU*. It is especially important to store such values whenever the CPU will need to access the values more than once. *The values of variables are stored in RAM.*
- *Variables have names and data types*. The names that can be used are governed by the rules listed above. In the interests of making programs easier to read, understand, modify and debug, *variable names should be descriptive*. This means that the name of a variable should be closely connected with its purpose.
- Although the **Variant** data type can be used to store any kind of data, it is generally not wise to use it unless the type of data is not known beforehand. **Variant** variables use a great deal of memory and can significantly reduce the execution speed of a program.
- If the type of data is known beforehand, a specific data type should be used. This helps to conserve memory and it avoids the problems that can arise when incompatible data types are used together.



## Questions

1. In what way does a variable in a computer program differ from a variable used in math class? Why is this difference important?
2. List the rules for VB variable names. Why do you suppose that variable names are not allowed to begin with a digit?
3. Why do programming languages offer software developers so many different data types?
4. Define the terms *debugging*, *context*, *variable*, *primary storage*, *technicality* and *contrast*.
5. Why would it be extremely confusing in mathematics to allow variable names to be longer than a single character?
6. When a computer program is running, where are the values of the variables stored?



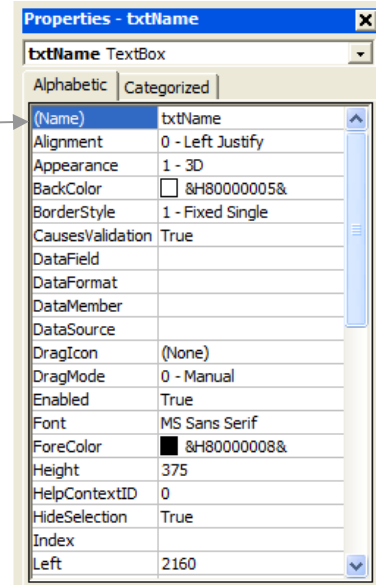
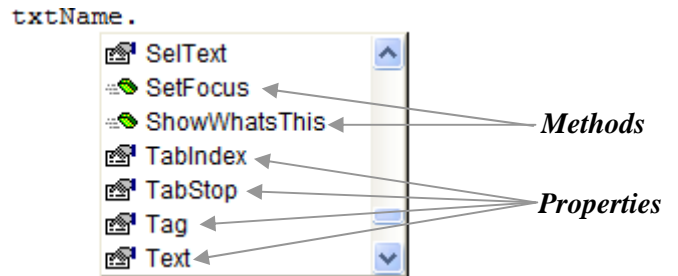
# WHAT IS THE DIFFERENCE BETWEEN AN OBJECT AND A VARIABLE?

## Variables

- A *variable* has a *very simple structure* compared to an object.
- A *variable* is used *only* to store a *single value* in RAM.
- When a *new value* is assigned to a variable, its *old value* is overwritten (i.e. erased, deleted).
- Variable names *should never begin* with the special prefixes that are reserved for object names (e.g. “txt,” “cmd,” “lbl,” etc). This causes a great deal of confusion. If you do this, it means that you do not understand the difference between an object and a variable.
- If “**Option Explicit**” is used, variables must be *declared explicitly*. (Keep in mind that it is a *very bad idea* to omit “**Option Explicit**.” Doing so will make it impossible for VB to detect misspelled variable names.)

## Objects

- As shown in the diagram at the right, an *object* has a *very complex structure* compared to a variable.
- An *object* is a collection of *properties* and *methods*. As such, an *object can store many different values* (properties) and *can perform a variety of different actions* (methods).
- Values can be assigned to the *properties* of objects *but not to the objects themselves*.
- Object names in VB *should always begin* with the *special prefixes that are reserved for object names* (e.g. “txt,” “cmd,” “lbl,” etc). A list of suggested prefixes for objects in VB can be obtained by searching for “Object Naming Conventions” in the MSDN help collection or by clicking here.
- Normally, *objects are not declared* using the “**Dim**” keyword. Objects are *usually created visually* using the VB form editor and *named* by using the “Name” property in the *properties window*.



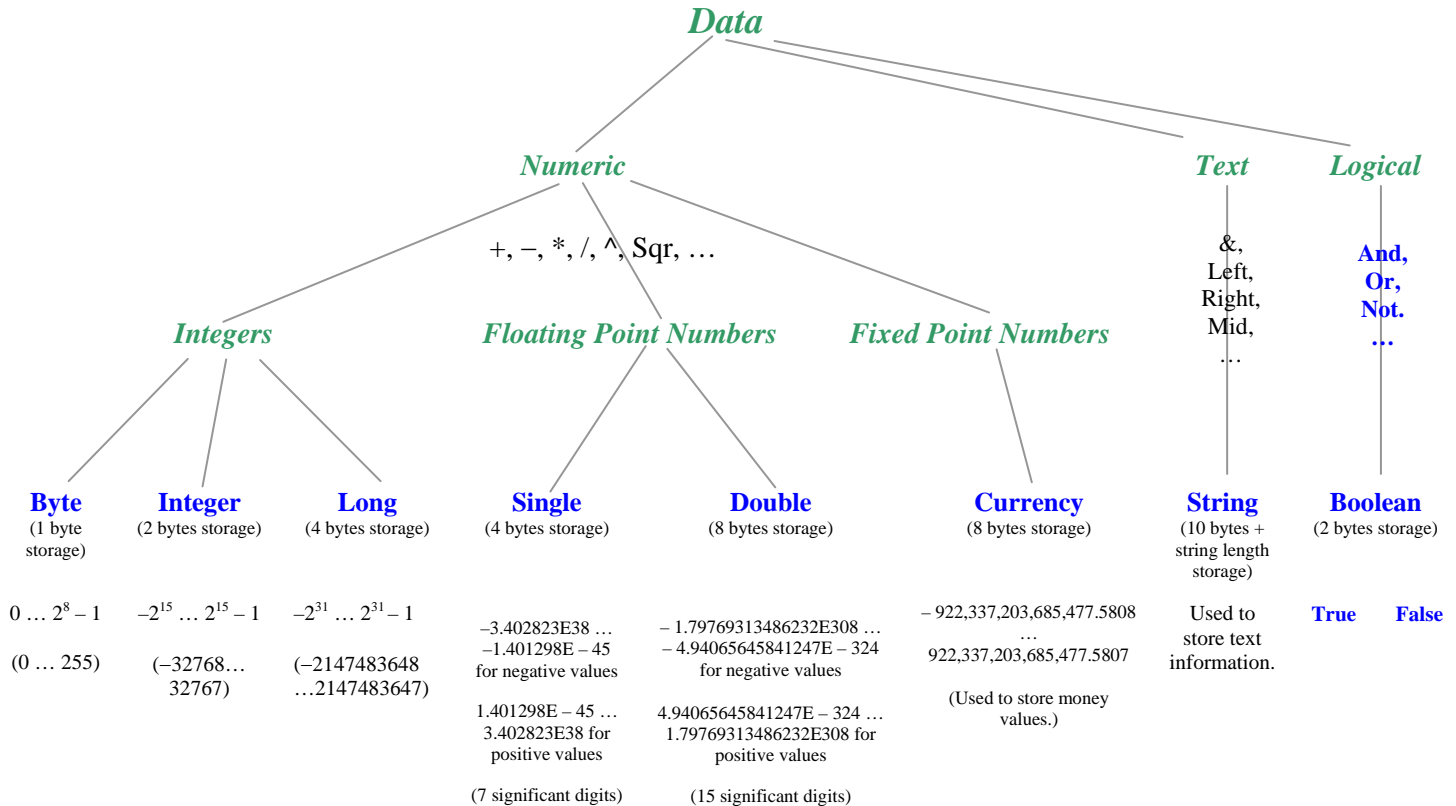
## Questions

1. An example of a *rule* in VB is that variable names *must* begin with a letter. An example of a *convention* in VB is that object names *should* begin with a suggested prefix that helps to identify the type of object. Explain the difference between a *rule* and a *convention*.
2. Explain the differences between a *property* and a *method*. Which of the two is similar to a variable? Which is similar to a **Sub**?

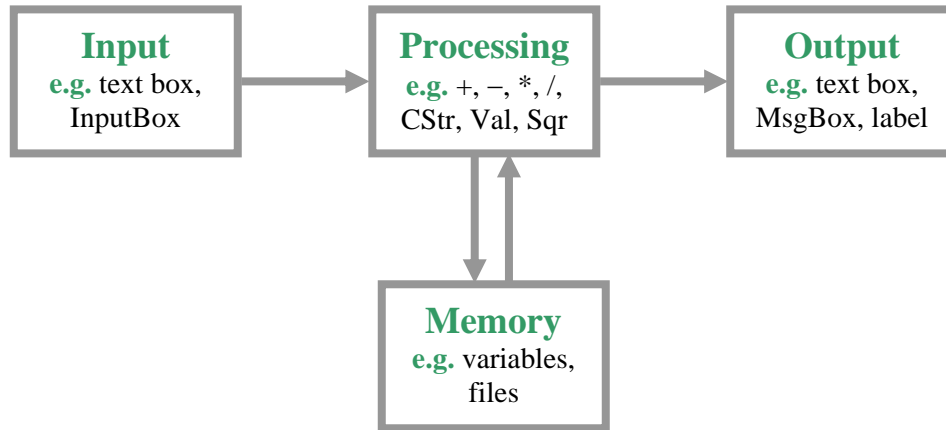
# REVIEW OF ESSENTIAL CONCEPTS IN VISUAL BASIC

## Data (Information) – A Partial List of VB Data Types

A computer can be viewed as a *data processing machine*. Since data can be categorized into various forms that require *differing amounts of memory* and *different types of operations*, programming languages offer diverse *data types*. A summary of the *most commonly used types of data* studied in this course is given in the following diagram.



## A Computer as a Data Processing Machine



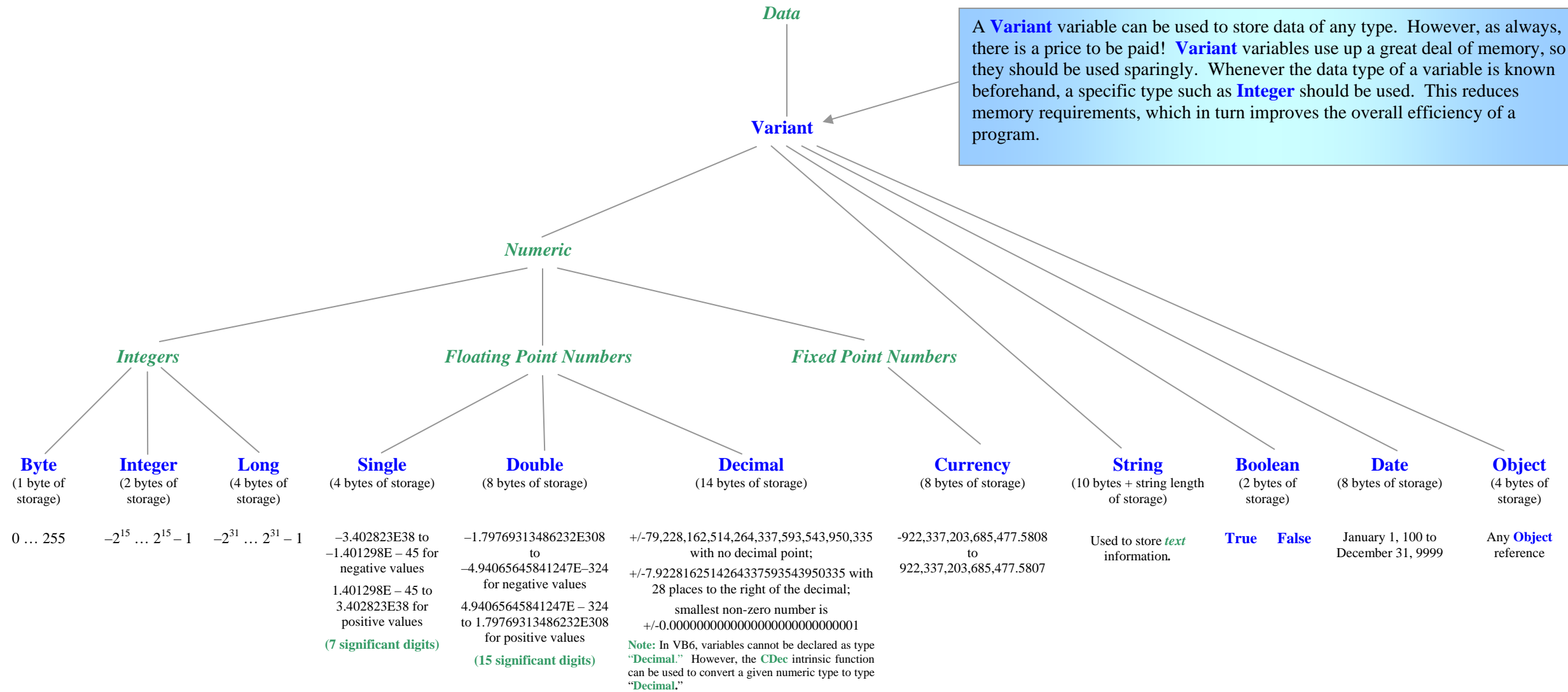
## Some Useful Intrinsic (Built-In) Functions

- **Val** Converts a string value to a numeric value **e.g.** Val ("23.47") → 23.47
- **CStr** Converts any value to a string value **e.g.** CStr (23.47) → "23.47"
- **Sqr** Returns the square root of any non-negative numeric value **e.g.** Sqr (100) → 10
- **Chr** Converts an ASCII (ANSI) value to its corresponding character **e.g.** Chr (122) → "z"
- **Asc** Returns the ASCII (ANSI) value of a character **e.g.** Asc ("z") → 122
- **Trim** Remove all leading and trailing blank spaces from a string **e.g.** Trim(" Ashley Walsh ") → "Ashley Walsh"

# A COMPLETE LIST OF VISUAL BASIC DATA TYPES

## Data (Information)

A computer can be viewed as a *data processing machine*. Since data can be categorized into various forms that require differing amounts of memory, designers of programming languages separate data into diverse *types*. A summary of all the types of data studied in this course is given in the following diagram.



A **Variant** variable can be used to store data of any type. However, as always, there is a price to be paid! **Variant** variables use up a great deal of memory, so they should be used sparingly. Whenever the data type of a variable is known beforehand, a specific type such as **Integer** should be used. This reduces memory requirements, which in turn improves the overall efficiency of a program.

## UNDERSTANDING VISUAL BASIC PROGRAMMING

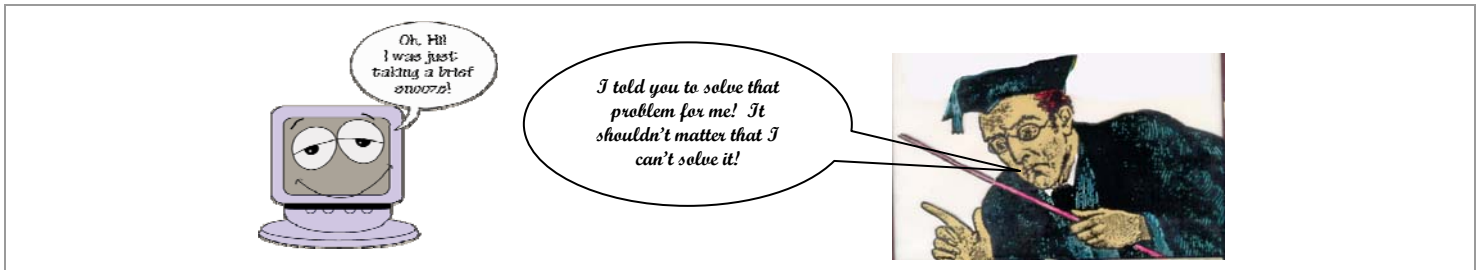
### Review: Concepts that you need to Understand before you can Create Good VB Programs

<i>Concept</i>	<i>Explanation</i>
1. Variables and Variable Types	1. Variables in programming languages are similar to mathematical variables. They are used to <i>store any value (in RAM) from a set of values</i> (i.e. to stand for a certain value from a set of values). For example, the VB statement “ <b>Dim Age As Byte</b> ” means that the name “ <b>Age</b> ” is used to store any whole number from 0 to 255 inclusive.
2. Why is it necessary to declare variables?	2. The VB statement shown above (“ <b>Dim Age As Byte</b> ”) is called a <i>variable declaration</i> . It “declares” both the <i>name</i> and the <i>type</i> of the variable. Declaring the name helps in debugging programs because misspelled variable names are detected immediately. Declaring the type is also helpful because it allows the OS to know exactly how much memory your program needs to store the values of variables and it determines which operations can be applied to the data.
3. What are the rules for naming variables?	3. Variable names can be 1 to 255 characters long. The only allowed characters are the letters from “a” to “z” (uppercase or lowercase), the digits from 0 to 9 and the underscore. To avoid confusing variable names with numbers, variable names <i>are not allowed to begin with a digit. They must begin with an alphabetic character (i.e. a letter)</i> .
4. What are the most commonly used variable types?	4. In this course, we shall be using the integer variable types <b>Byte</b> , <b>Integer</b> and <b>Long</b> , the floating-point types <b>Single</b> and <b>Double</b> , the text type <b>String</b> and the logical type <b>Boolean</b> . Programming languages offer many variable types to allow programmers to use memory efficiently and to allow for logical distinctions to be made among different types of data.
5. What on Earth is a <b>String</b> ?	5. A <i>string</i> is simply a sequence or a “string” of characters, that is, a <i>piece of text</i> . Examples of strings include “Central Peel” and “32 Kennedy Road North.” <i>String values</i> are always enclosed in quotation marks to distinguish them from variable names, object names, procedure names (e.g. “ <b>Sub</b> ” names and “ <b>Function</b> ” names) and VB keywords.
6. What is a good practice for naming variables?	6. “UpperCamelCase” should be used for naming variables in VB. Since spaces are not allowed in variable names, a good practice is to <i>capitalize the first letter of each word</i> . For example, it is much easier to read the variable name <b>FirstName</b> than it is to read the variable name <b>firstname</b> .
7. Does the user know anything about variable names?	7. Everything about a program, except for the interface, is invisible to users. Variable names, object names and all other aspects of <b>code</b> are seen only by programmers. Remember that <i>variable names allow programmers to store, recall and modify values in memory</i> . It is useful to think of memory (RAM) as a workspace in which data and instructions are stored.
8. How should objects be named?	8. “lowerCamelCase” should be used to name objects in VB. In addition, to minimize confusion, most VB programmers use prefixes to identify the various types of objects. Text box names begin with <b>txt</b> , form names begin with <b>frm</b> , command button names begin with <b>cmd</b> and label box names begin with <b>lbl</b> . (e.g.
9. What is an <b>assignment statement</b> ?	9. Assignment statements are used to give values to variables. e.g. <code>FirstName = "Jordan"</code> 'String value assigned to string variable <code>Age=16</code> 'Numeric value assigned to numeric variable
10. How are math operations written in VB?	10. +, -, *, /, ^ are used for “add,” “subtract,” “multiply,” “divided by” and “to the exponent of” respectively. <i>Scientific notation</i> is written with an “E” (e.g. $2 \times 10^{30}$ is written 2E30).
11. What is a <i>property</i> ?	11. A property is an attribute or characteristic of an object. Objects have many different properties. Some of the properties can be modified at <b>design-time</b> (when the program is being written) or <b>run-time</b> (when the program is running), while others can only be modified at <b>run-time</b> . In VB code, the object name and property name are separated by a dot (e.g. <code>txtDisplay.Text</code> ).
12. What is <i>debugging</i> ?	12. Debugging is the process of systematically eliminating errors from software. Two types of programming errors can occur. <i>Syntax errors</i> are caused by breaking the rules of the programming language. Such errors are detected by the programming environment. <i>Logic errors</i> exist in code when the software does not perfectly accomplish what it is intended to. Such errors can only be detected by extremely thorough testing.
13. Why is it important to <i>indent</i> programs correctly?	13. Indenting programs correctly <i>makes them much easier to read, understand, debug and modify</i> . Each statement within a <b>Sub</b> should be indented one TAB space to the right. See pages 10, 27 and 30 for more information on proper indentation. In addition, you should study the programming examples stored on the “I:” drive to see examples of proper indentation.

# WRITING YOUR OWN CODE: CURRENCY CONVERTER PROGRAM



## The Most Important Lesson of the Entire Unit

The process of writing a program can be viewed as a form of “teaching.” Whenever you write any computer program, you are, in a sense, “teaching” a computer how to solve a particular problem. **KEEP IN MIND THAT YOU CANNOT “TEACH” A COMPUTER TO SOLVE A PROBLEM THAT YOU DO NOT KNOW HOW TO SOLVE!**



**BEFORE YOU EVEN ATTEMPT TO WRITE CODE (PROGRAMMING INSTRUCTIONS), FIRST YOU MUST DEVISE A STRATEGY! BEFORE YOU CAN DEVISE A STRATEGY, YOU MUST ENSURE THAT YOU UNDERSTAND THE PROBLEM! THE FOLLOWING TABLE DESCRIBES A SOUND APPROACH TO SOFTWARE DEVELOPMENT. IF YOU HOPE TO BE SUCCESSFUL, FOLLOW THE GUIDELINES IN THE SECOND AND THIRD COLUMNS. DO NOT FOLLOW THE STEPS IN THE FIRST COLUMN!**

### Planning and Developing Solutions to Software Development Problems

<b>Wrong!!!! (Most Students)</b>	<b>Right!!!! (George Polya)</b>	<b>How these Steps Apply to Software Development</b>
<del>                     1. Read problem                      2. Type code                      3. Click on the  button                 </del>	<b>1. Analysis</b> Understand the problem	<ul style="list-style-type: none"> <li>Do you understand what is required? If you do not, you run the risk of creating an incomplete solution or solving the wrong problem altogether!</li> <li>What information must be input?</li> <li>What information must be output?</li> </ul> <div style="text-align: center;">  </div> <ul style="list-style-type: none"> <li>Have you tried solving a few <i>specific examples</i> of the given <i>general programming problem</i> using pencil, paper and calculator?</li> </ul>
	<b>2. Design</b> Choose a strategy	<ul style="list-style-type: none"> <li>Do not write any code yet!</li> <li>On paper, design a few different possible interfaces (i.e. forms) for your program.</li> <li>Develop or look up an <i>algorithm</i> for solving the problem. (An algorithm is a step-by-step recipe for solving a given problem.)</li> <li>Consider alternative algorithms.</li> </ul>
	<b>3. Implementation</b> Carry out the strategy	<ul style="list-style-type: none"> <li>Write the code but not all in one fell swoop.</li> <li>Break up the large problem into several smaller sub-problems.</li> <li>Solve each sub-problem separately.</li> <li>Do not integrate a solution to a sub-problem into the larger solution until you are confident that it is correct.</li> <li>It is also wise to save each version of your program. In case of a catastrophe, you can always go back to an earlier version.</li> </ul>
	<b>4. Validation</b> Check the solution	<ul style="list-style-type: none"> <li>Extensive testing should take place to find bugs that were not noticed in the implementation phase.</li> <li>It is best to allow the testing to be done by average computer users who are not programmers. Because of their computer expertise, programmers subconsciously tend to avoid actions that cause computer programs to fail.</li> <li>Once the software is released, additional bug fixes will usually be necessary as users report previously undiscovered bugs. This is known as the <i>maintenance phase</i>.</li> </ul>



### Step 4 – Validation (Check the Solution)

Test your program thoroughly before you decide that you are finished.

- Try entering very large numbers or extremely small numbers.
- Try entering characters other than digits.
- Try leaving certain text boxes blank.
- Pretend that you are two years old and that buttons fascinate you. Press a variety of different keys, including combinations of keys, to see what happens. In addition, try clicking the mouse buttons in the most ridiculous ways imaginable.

If your program survives these tests, it is *likely* that it is correct. If not, track down the bugs and correct them. Then repeat the testing process until all the pesky bugs have been exterminated.

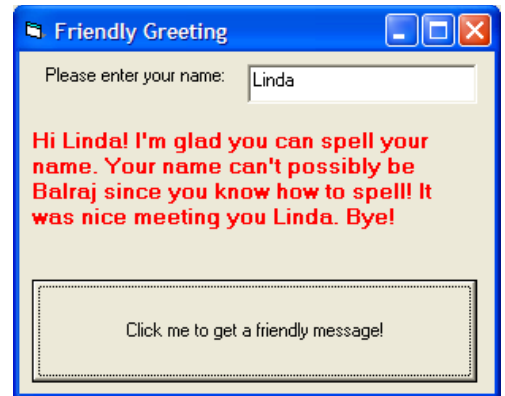
## VISUAL BASIC PRACTICE PROBLEMS: INPUT, PROCESSING, OUTPUT

1. Write a program that asks the user to type her name in a text box. The program responds by displaying a friendly message in a label box. For example, if Linda were to type in her name, the computer would display the message “Hi Linda! I’m glad you can spell your name. Your name can’t possibly be Balraj since you know how to spell! It was nice meeting you Linda. Bye!”

**Hint:** To solve this problem, you need to know how to use the string concatenation operator (“&”).

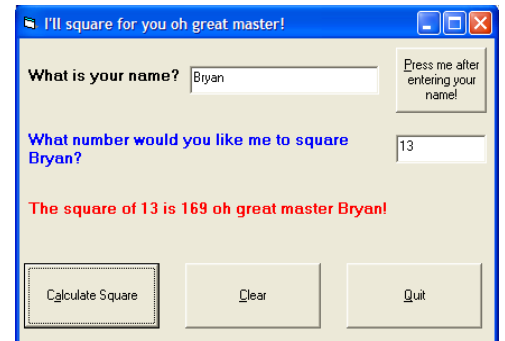
#### Solution

I:\Out\Nolfi\Ics3m0\Simple VB Examples\Friendly Message\FriendlyMessage.vbp



2. Write a program that asks the user for his name and a number. The program then responds by displaying, in a label box, a message along with the square of the number. For instance, if Bryan were to enter his name and the number 13, the program would display the message, “The square of 13 is 169, oh great master Bryan!”

**Hint:** To solve this problem, you need to know understand the difference between *local variables* and *global variables*.

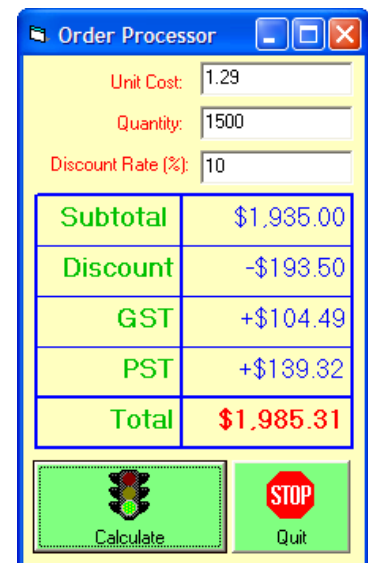


3. Write a program that allows a user to enter unit cost, quantity and a rate of discount. The program then displays the discount, the GST (6%), the PST (8%) and the total cost.

Note that the discount should be applied **before** taxes are calculated! In the example shown at the right, the amount before taxes is \$1935.00. Then a 10% discount is applied, resulting in a cost of \$1741.50 before taxes. The GST and PST are then calculated using the \$1741.50 figure.

#### Solution

I:\Out\Nolfi\Ics3m0\Simple VB Examples\GST PST\GSTPSTExample.vbp



## VB REVIEW 1- IMPORTANT PROGRAMMING TERMINOLOGY

1. State the meaning of each of the operators given in the table below. In addition, provide an example of how each operator can be used in Visual Basic.

<i>Operator</i>	<i>Meaning</i>	<i>Example</i>
+		
-		
*		
/		
^		
<		
>		
<=		
>=		
<>		
&		

2. Explain the difference between an object and a variable. Give an example to illustrate your answer.
3. Victor has used the following variable declarations. Explain *what is wrong with the variable names* that Victor has chosen. Has he made any *errors in choosing the data type* of any of the variables?

```
Dim txtName As String, lblNumber As Integer, cmdAddress As String
Dim numberofstudents As String, familyname As Integer, x As String, y As String
```

4. Complete the following.
- ```
Private Sub cmdPressMe_Click(*)
    Dim Name As String
    Name = txtName.Text
    lblGreeting.Visible = True
    lblGreeting.Caption = "Have a nice day " & Name & "."
End Sub
```
- Name of \_\_\_\_\_
- Name of \_\_\_\_\_
- Name of \_\_\_\_\_



5. The following VB sub will work correctly but it is somewhat unclear; that is, the statements within the sub may be difficult for some people to understand. Explain why.

```
Private Sub cmdClear_Click()
    txtName = ""
    txtAddress = ""
    lblSalePrice = ""
    lblSalesTax = ""
End Sub
```

What is this string called? What is its purpose?

6. Classify each of the following VB concepts.

| VB Concepts                                                                                                      | Circle the term that best describes the VB concepts in the left column |                  |                  |            |                 |                     |           |                      |          |              |        |                     |        |            |
|------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------|------------------|------------------|------------|-----------------|---------------------|-----------|----------------------|----------|--------------|--------|---------------------|--------|------------|
| <b>Private, Sub, End, Dim, String, Byte, Integer, Long, Single, Double, Currency, As, If, ElseIf, Else, True</b> | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| +, -, *, /, ^, <, >, <=, >=, <>, &                                                                               | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| "Central Peel Secondary"                                                                                         | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| <b>String, Byte, Integer, Long, Single, Double, Currency</b>                                                     | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| Val, Int, Round, Trim, Format, CStr, Str                                                                         | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| 23.7395624584                                                                                                    | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| Age (declared as <b>Byte</b> )                                                                                   | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| Address (declared as <b>String</b> )                                                                             | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| SubTotal * 0.07                                                                                                  | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| GST = SubTotal * 0.07                                                                                            | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| PizzaBasePrice = <u>18.9</u><br>(underlined part)                                                                | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| txtName. <u>Text</u><br>(underlined part)                                                                        | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| <u>txtName</u> .Text<br>(underlined part)                                                                        | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |
| txtName. <u>SetFocus</u><br>(underlined part)                                                                    | String Constant                                                        | Numeric Constant | Numeric Variable | Data Types | String Variable | Value of a variable | Operators | Assignment Statement | Property | VB key-words | Object | Intrinsic Functions | Method | Expression |

## VB REVIEW 2- TRANSLATING MATH FORMULAS INTO VB

1. Translate the following mathematical formulas into VB. Remember to use *meaningful, descriptive variable names!*

|                                                                                                                                                                                                                                       |  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <p><b>Hints:</b> += +, -= -, × = *, ÷ = /, integer division = \, exponent = ^, <math>\sqrt{\quad}</math> = Sqr,<br/>                 (See “Math Functions” in the MSDN collection for a complete list of mathematical functions.)</p> |  |
| $A = \pi r^2$                                                                                                                                                                                                                         |  |
| $V = \frac{4}{3} \pi r^3$                                                                                                                                                                                                             |  |
| $A = \frac{bh}{2}$                                                                                                                                                                                                                    |  |
| $A = \frac{h(a+b)}{2}$                                                                                                                                                                                                                |  |
| $c = \sqrt{a^2 + b^2}$                                                                                                                                                                                                                |  |
| $b = \sqrt{c^2 - a^2}$                                                                                                                                                                                                                |  |

2. Write a VB program that can calculate the area of a circle, triangle, rectangle, parallelogram or trapezoid. Your program should have some way of allowing the user to *select* the desired shape and to *enter* the required dimensions. Remember to use the following format to plan your solution.

|                                                               |                                                                      |                                                                                 |
|---------------------------------------------------------------|----------------------------------------------------------------------|---------------------------------------------------------------------------------|
| <b>ALGORITHM</b>                                              |                                                                      |                                                                                 |
| <p><b>INPUT</b><br/>What information does the user enter?</p> | <p><b>PROCESSING</b><br/>What must be done with the information?</p> | <p><b>OUTPUT</b><br/>What should be displayed after processing is complete?</p> |
| <b>VARIABLES (MEMORY)</b>                                     |                                                                      |                                                                                 |
| <b>LOCAL VARIABLES</b>                                        | <b>GLOBAL VARIABLES</b>                                              |                                                                                 |
| <b>CODE</b>                                                   |                                                                      |                                                                                 |
| <b>Code for Input</b>                                         | <b>Code for Processing</b>                                           | <b>Code for Output</b>                                                          |
|                                                               |                                                                      |                                                                                 |

# LEARNING ABOUT SELECTION STATEMENTS (IF STATEMENTS) BY STUDYING THE PYTHAGOREAN THEOREM PROGRAM

```
'Code for Solve button
Private Sub cmdSolve_Click()

    'MEMORY
    Dim Base As Double, Height As Double
    Dim Hypotenuse As Double

    'INPUT
    Height = Val(txtHeight.Text)
    Base = Val(txtBase.Text)
    Hypotenuse = Val(txtHypotenuse.Text)

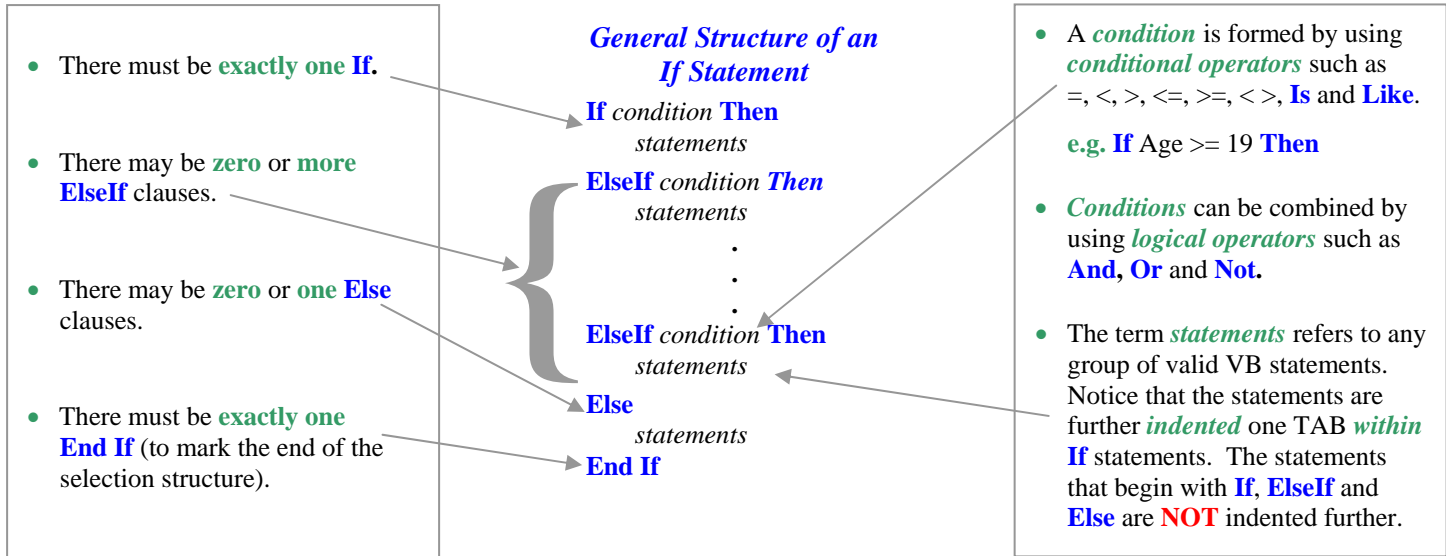
    'PROCESSING and OUTPUT
    If Hypotenuse = 0 Then
        Hypotenuse = Sqr(Base ^ 2 + Height ^ 2)
        txtHypotenuse.Text = CStr(Hypotenuse)
    ElseIf Base = 0 Then
        Base = Sqr(Hypotenuse ^ 2 - Height ^ 2)
        txtBase.Text = CStr(Base)
    ElseIf Height = 0 Then
        Height = Sqr(Hypotenuse ^ 2 - Base ^ 2)
        txtHeight.Text = CStr(Height)
    Else
        ' There are errors in data entered by user.
        MsgBox "You have entered invalid data!", _
            vbExclamation, "There is a problem!"
    End If
End Sub
```

**Review 1**  
 The words displayed in **boldface** are called VB **keywords**. Keywords have special meanings in VB and cannot be used as variable or object names. In other words, keyword names are *reserved* names.  
 When you are in the process of designing a VB program, in what colour are the keywords displayed?

**Review 2**  
 These are VB comments. What is their purpose? In what colour are they displayed while you are designing a VB program?

If statements are used in programs to make **decisions** or **selections**. The rules for **If** statements are as follows:

- **If** statements begin with the word **If** and end with the words **End If**
- There must be **exactly one If** and **one End If**
- There may be **zero** or **more ElseIf** clauses. **ElseIf** clauses must follow **If** and precede **Else**.
- Both **If** and **ElseIf** clauses must have a **condition** and must have the keyword **Then**.
- There may be **zero Else** clauses or **one Else** clause. **Else** must follow **If** and **ElseIf**, and **Else must not** have a **condition** or the keyword **Then**. **Else** means “if all else fails.”



## An Improved Version of the Pythagorean Theorem Program

If you have thoroughly tested the preliminary version 1.0.0 of the “Triangle Solver,” you should have noticed several bugs. The following version uses **conditional operators** and extra conditions to correct the bugs and make the program foolproof.

```
'Code for Solve button
Private Sub cmdSolve_Click()

    'MEMORY
    Dim Base As Double, Height As Double, Hypotenuse As Double

    'INPUT
    Height = Val(txtHeight.Text)
    Base = Val(txtBase.Text)
    Hypotenuse = Val(txtHypotenuse.Text)

    'PROCESSING and OUTPUT
    If Hypotenuse = 0 And Base > 0 And Height > 0 Then
        Hypotenuse = Sqr(Base ^ 2 + Height ^ 2)
        txtHypotenuse.Text = CStr(Hypotenuse)
    ElseIf Base = 0 And Height > 0 And Hypotenuse > Height Then
        Base = Sqr(Hypotenuse ^ 2 - Height ^ 2)
        txtBase.Text = CStr(Base)
    ElseIf Height = 0 And Base > 0 And Hypotenuse > Base Then
        Height = Sqr(Hypotenuse ^ 2 - Base ^ 2)
        txtHeight.Text = CStr(Height)
    Else
        ' There are errors in the data entered by the user.
        MsgBox "You have entered invalid data!", _
            vbExclamation, "There is a problem!"
    End If

End Sub
```

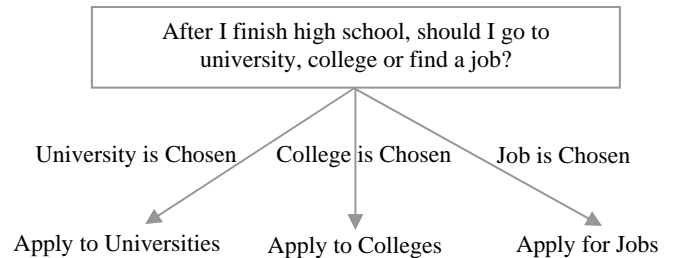
The logical operator **And** is being used to create a **compound condition**. For the compound condition to be **True**, **ALL** the conditions connected by **And** must be true.

For example, for the hypotenuse to be calculated correctly, the user must enter a positive value for the height, a positive value for the base and leave the text box for the hypotenuse blank. The first compound condition checks that all these requirements are met.

In the other two compound conditions, why do we have `Hypotenuse > Base` and `Hypotenuse > Height`? Why don't we use the condition `Hypotenuse > 0`?

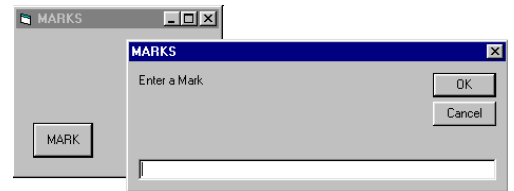
## Picturing “If” Statements

The following diagram can be useful in understanding the flow of information during the execution of an “**If**” statement is executed. “**If**” statements are a lot like travelling along a path and suddenly reaching a “fork.” When this happens, a **decision** needs to be made.



## Exercises

- Write a program that allows a user to enter a mark in an input box. The program then displays “Congratulations you have PASSED,” or “Sorry, you have FAILED” in a **message box** depending on whether the mark is greater than or equal to 50 or less than 50.
- Most universities in North America use a grading system known as the GPA (grade point average) system. It is summarized in the table given below.



| Percentage Grade | Grade Point Score |
|------------------|-------------------|
| 85% – 100%       | 4.0               |
| 80% – 84%        | 3.7               |
| 77% – 79%        | 3.3               |
| 74% – 76%        | 3.0               |
| 70% – 73%        | 2.7               |
| 67% – 69%        | 2.3               |
| 64% – 66%        | 2.0               |
| 60% – 63%        | 1.7               |
| 57% – 59%        | 1.3               |
| 54% – 56%        | 1.0               |
| 50% – 53%        | 0.7               |
| 0% – 49%         | 0.0               |

## THE AREA CALCULATOR PROGRAM – ANOTHER PROGRAM THAT REQUIRES “IF” STATEMENTS

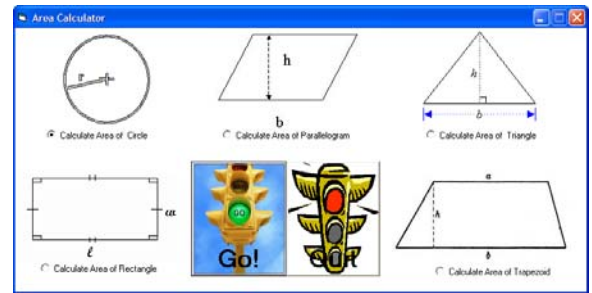
### Introduction – What you will learn by studying the Area Calculator

- How to use option buttons
- How to create a VB project that uses two or more forms
- How a form can access objects on a different form
- The most basic way of using a message box (For a more detailed explanation of message boxes, see pages 53-55.)

### What you need to do

The “Area Calculator” program can be found in

**I:\Out\Nolfi\Tik2o0\Area Calculator** . Load this program and study the code carefully. Notice that an “**IF**” statement is used to determine the shape selected by the user.



```
Private Sub cmdGo_Click()
```

```
    If optRectangle.Value = True Then
        frmChosenShape.imgShape.Picture=imgRectangle.Picture
        frmChosenShape.Caption = "Area of Rectangle"
        frmChosenShape.lblDimension1.Visible = True
        frmChosenShape.lblDimension2.Visible = True
        frmChosenShape.lblDimension3.Visible = False
        frmChosenShape.lblDimension1.Caption = "l="
        frmChosenShape.lblDimension2.Caption = "w="
        frmChosenShape.lblDimension3.Caption = ""
        frmChosenShape.txtDimension1.Visible = True
        frmChosenShape.txtDimension2.Visible = True
        frmChosenShape.txtDimension3.Visible = False
        frmChosenShape.Show
```

```
    ElseIf optParallelogram.Value = True Then
        frmChosenShape.imgShape.Picture=imgParallelogram.Picture
        frmChosenShape.Caption = "Area of Parallelogram"
        frmChosenShape.lblDimension1.Visible = True
        frmChosenShape.lblDimension2.Visible = True
        frmChosenShape.lblDimension3.Visible = False
        frmChosenShape.lblDimension1.Caption = "b="
        frmChosenShape.lblDimension2.Caption = "h="
        frmChosenShape.lblDimension3.Caption = ""
        frmChosenShape.txtDimension1.Visible = True
        frmChosenShape.txtDimension2.Visible = True
        frmChosenShape.txtDimension3.Visible = False
        frmChosenShape.Show
```

```
    ElseIf optTriangle.Value = True Then
        frmChosenShape.imgShape.Picture=imgTriangle.Picture
        frmChosenShape.Caption = "Area of Triangle"
        frmChosenShape.lblDimension1.Visible = True
        frmChosenShape.lblDimension2.Visible = True
        frmChosenShape.lblDimension3.Visible = False
        frmChosenShape.lblDimension1.Caption = "b="
        frmChosenShape.lblDimension2.Caption = "h="
        frmChosenShape.lblDimension3.Caption = ""
        frmChosenShape.txtDimension1.Visible = True
        frmChosenShape.txtDimension2.Visible = True
        frmChosenShape.txtDimension3.Visible = False
        frmChosenShape.Show
```

```
    ElseIf optCircle.Value = True Then
```

```
        frmChosenShape.imgShape.Picture=imgCircle.Picture
        frmChosenShape.Caption = "Area of Circle"
        frmChosenShape.lblDimension1.Visible = False
        frmChosenShape.lblDimension2.Visible = True
        frmChosenShape.lblDimension3.Visible = False
        frmChosenShape.lblDimension1.Caption = ""
        frmChosenShape.lblDimension2.Caption = "r="
        frmChosenShape.lblDimension3.Caption = ""
        frmChosenShape.txtDimension1.Visible = False
        frmChosenShape.txtDimension2.Visible = True
        frmChosenShape.txtDimension3.Visible = False
        frmChosenShape.Show
```

```
    ElseIf optTrapezoid.Value = True Then
```

```
        frmChosenShape.imgShape.Picture=imgTrapezoid.Picture
        frmChosenShape.Caption = "Area of Trapezoid"
        frmChosenShape.lblDimension1.Visible = True
        frmChosenShape.lblDimension2.Visible = True
        frmChosenShape.lblDimension3.Visible = True
        frmChosenShape.lblDimension1.Caption = "a="
        frmChosenShape.lblDimension2.Caption = "b="
        frmChosenShape.lblDimension3.Caption = "h="
        frmChosenShape.txtDimension1.Visible = True
        frmChosenShape.txtDimension2.Visible = True
        frmChosenShape.txtDimension3.Visible = True
        frmChosenShape.Show
```

```
    Else
```

```
        MsgBox "Please select one of the shapes before clicking 'Go!'", _
            vbExclamation
```

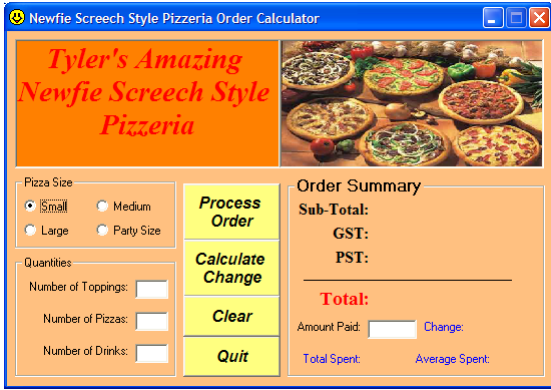
```
    End If
```

```
End Sub
```

### Questions

1. What prefix should be used for naming option buttons?
2. What property of an option button can your program check to see if the option button is selected?
3. Explain how a form can access objects from a different form.
4. The area calculator program uses two forms, one that is used to select the shape and another that is used to allow the user to enter the dimensions of the shape. How is this accomplished?
5. Once the user chooses a shape and clicks “Go,” another form is displayed to allow the user to enter the dimensions of the shape. How would you prevent the user from returning to the original form (the *parent form*) unless the new form (the *child form*) is first closed?

# PIZZA PROGRAM SOLUTIONS AND QUESTIONS



| SIZE       | BASE PRICE | EACH TOPPING |
|------------|------------|--------------|
| Small      | \$9.95     | \$1.00       |
| Medium     | \$12.95    | \$1.25       |
| Large      | \$15.95    | \$1.50       |
| Party Size | \$18.95    | \$2.00       |
| Drinks     |            | \$1.25       |

## The Problem

“Newfoundland Style Pizzeria Problem”

## The Plan

| INPUT                                                                                                                                                                                                                           | PROCESSING                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | OUTPUT                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>What information must the user enter?</p> <p><b>Process Order Button</b><br/>Pizza Size, Number of Pizzas, Number of Toppings, Number of Drinks</p> <p><b>Calculate Change Button</b><br/>Amount of money customer pays.</p> | <p>What must be done with the information?</p> <p><b>Process Order Button</b></p> <ol style="list-style-type: none"> <li>Determine base price for pizza size chosen</li> <li>Determine price per topping for chosen size</li> <li>Calculate cost before taxes</li> <li>Calculate GST and PST</li> <li>Calculate total for order</li> <li>Add total to total for all customers</li> <li>Increase the number of orders by 1</li> <li>Calculate the average cost of each order</li> </ol> <p><b>Calculate Change Button</b><br/>Calculate change.</p> | <p>What should be displayed after processing is complete?</p> <p><b>Process Order Button</b></p> <ol style="list-style-type: none"> <li>Display subtotal</li> <li>Display GST</li> <li>Display PST</li> <li>Display total</li> <li>Display total spent by all customers</li> <li>Display average amount spent by each customer</li> </ol> <p><b>Calculate Change Button</b><br/>Display change.</p> |

1. Explain why *most* of the variables are declared as *local variables* while a few are declared as *global variables*.

3. Explain the purpose of the “NumOrders” variable.

| VARIABLES (MEMORY)                                                                                                                                             |                                                                                                                                                                                                                                                            |                                                                                                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| LOCAL VARIABLES                                                                                                                                                |                                                                                                                                                                                                                                                            | GLOBAL VARIABLES                                                                                          |
| <p><b>Integer Variables</b></p> <p>NumPizzas</p> <p>NumToppings</p> <p>NumDrinks</p> <p><i>These variables store values that involve a number of items</i></p> | <p><b>Currency Variables</b></p> <p>PizzaBasePrice</p> <p>PricePerTopping</p> <p>SubTotal</p> <p>GST</p> <p>PST</p> <p>Change</p> <p>CashTendered</p> <p>AverageAmountSpent</p> <p><i>These variables store values that involve an amount of money</i></p> | <p><b>Currency Variables</b></p> <p>TotalCostOfOrder</p> <p>TotalSpentByAllCustomers</p> <p>NumOrders</p> |

**Option Explicit** 'Used to force variable declarations.

```
' The following variables are called GLOBAL VARIABLES because they are declared OUTSIDE the subs, which means  
' 1. the values of these variables remain stored in RAM as long as the  
' form is loaded in RAM (i.e. the computer will "remember" the values  
' of these variables for as long as the form remains loaded)  
' 2. these variables are VISIBLE to all the subs. Each sub can access each  
' global variable, allowing two or more subs to SHARE their values.  
' A variable should be declared GLOBALLY whenever two or more subs need to access it (i.e. use or change its  
' value) and/or whenever its value needs to be "remembered" after a sub has finished executing.
```

```
Dim TotalSpentByAllCustomers As Currency, TotalCostOfOrder As Currency, NumOrders As Integer
```

```
' A "Form_Load" sub is executed automatically as soon as a form is loaded "Form_Load" subs should be used.  
' whenever INITIALIZATION code needs to be executed BEFORE the user is allowed to interact with the program.  
' This "Form_Load" sub is used to set to zero the initial values of the number of orders and the total spent  
' by all customers. Each time an order is processed, the values of these variables are updated.
```

```
Private Sub Form_Load()  
    NumOrders = 0  
    TotalSpentByAllCustomers = 0  
End Sub
```

```
Private Sub cmdProcessOrder_Click()
```

```
' The variables declared inside a sub are called LOCAL VARIABLES. Local variables are  
' 1. VISIBLE only within the sub in which they are declared.  
' 2. CREATED when the sub is invoked (i.e. called or executed).  
' 3. DESTROYED when the sub returns (has finished executing).  
' Local variables should be used whenever possible. They help to reduce the time needed to debug a program  
' because they keep information PRIVATE. If information is needed only by a particular sub, it is best  
' to HIDE it from other subs. Local variables also help to conserve memory because they are discarded  
' as soon as the sub returns.
```

```
Dim PizzaBasePrice As Currency, PricePerTopping As Currency, SubTotal As Currency  
Dim GST As Currency, PST As Currency, AverageAmountSpent As Currency  
Dim NumPizzas As Integer, NumDrinks As Integer, NumToppings As Integer
```

```
'INPUT: Obtain information from user.  
NumPizzas = Val(txtPizzas.Text)  
NumToppings = Val(txtToppings.Text)  
NumDrinks = Val(txtDrinks.Text)
```

```
'PROCESSING  
'Decide what the base price and price per topping should be.
```

```
If optSmall.Value = True Then  
    PizzaBasePrice = 9.95  
    PricePerTopping = 1  
ElseIf optMedium.Value = True Then  
    PizzaBasePrice = 12.95  
    PricePerTopping = 1.25  
ElseIf optLarge.Value = True Then  
    PizzaBasePrice = 15.95  
    PricePerTopping = 1.5  
Else  
    PizzaBasePrice = 18.95  
    PricePerTopping = 2  
End If
```

```
'Now perform all calculations  
SubTotal = (PizzaBasePrice + PricePerTopping * NumToppings) * NumPizzas + NumDrinks * 1.25  
GST = Round(SubTotal * 0.07, 2)  
PST = Round(SubTotal * 0.08, 2)  
TotalCostOfOrder = SubTotal + GST + PST  
TotalSpentByAllCustomers = TotalSpentByAllCustomers + TotalCostOfOrder  
NumOrders = NumOrders + 1  
AverageAmountSpent = Round(TotalSpentByAllCustomers / NumOrders, 2)
```

```
'OUTPUT: Display results.  
lblSubTotal.Caption = Format(SubTotal, "Currency")  
lblGST.Caption = Format(GST, "Currency")  
lblPST.Caption = Format(PST, "Currency")  
lblTotal.Caption = Format(TotalCostOfOrder, "Currency")  
lblTotalSpent.Caption = Format(TotalSpentByAllCustomers, "Currency")  
lblAverageSpent.Caption = Format(AverageAmountSpent, "Currency")
```

```
End Sub
```

```
Private Sub cmdCalculateChange_Click()
```

```
Dim Change As Currency, CashTendered As Currency  
'INPUT  
CashTendered = Val(txtAmountPaid.Text)  
'PROCESSING  
Change = Round(CashTendered - TotalCostOfOrder, 2)  
'OUTPUT  
lblChange.Caption = Format(Change, "Currency")
```

```
End Sub
```

```
Private Sub cmdClear_Click()
```

```
optSmall.Value = True  
txtToppings.Text = ""  
txtPizzas.Text = ""  
txtDrinks.Text = ""  
txtAmountPaid.Text = ""  
lblSubTotal.Caption = ""  
lblGST.Caption = ""  
lblPST.Caption = ""  
lblTotal.Caption = ""  
lblChange.Caption = ""
```

```
End Sub
```

1. Explain why “Else” is used instead of “ElseIf” for the final clause of this “If” statement.

2. Explain why the change calculations are done in the “cmdCalculateChange\_Click” sub instead of the “cmdProcessOrder\_Click” sub.

# SEQUENCE, SELECTION AND REPETITION: THE UNDERPINNINGS OF PROGRAMMING

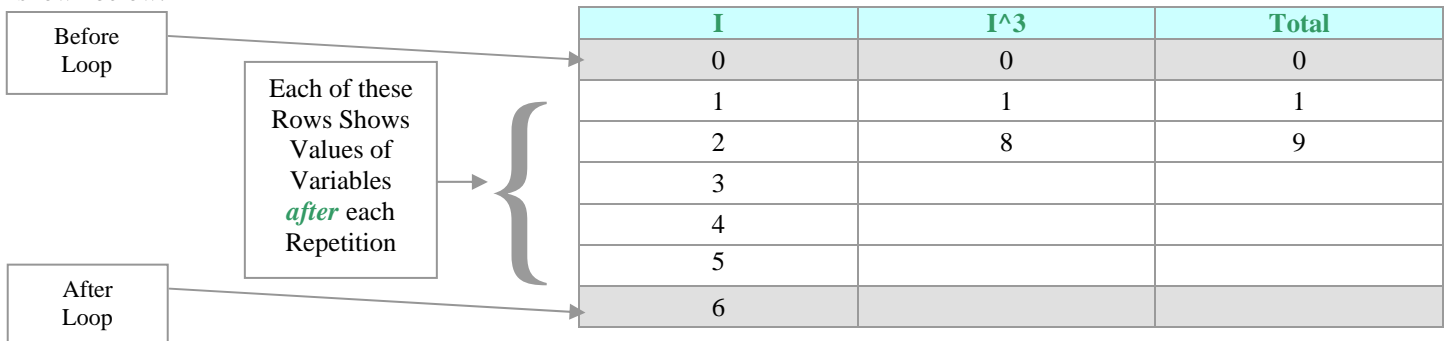
| <i>Sequence</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | <i>Selection</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <i>Repetition</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Instructions are executed (carried out) in <b>sequence</b> (in order, one after the other). All statements are executed exactly once; none of the statements is omitted.</p> <p><b>Example</b></p> <pre>' Friendly greeting program Option Explicit Private Sub cmdPressMe_Click()     'Memory     Dim FirstName As String     'Input     FirstName = Trim(txtName.Text)     'Processing and Output     lblGreeting.Visible = True     lblGreeting.Caption = "Have a " &amp; _         "nice day " &amp; FirstName &amp; "!" End Sub Private Sub cmdQuit_Click()     Unload frmGreeting End End Sub</pre> | <p>Based on a condition or a set of conditions, some statements are <b>selected</b> and some are rejected. The idea of <b>selection</b> should be used whenever your program needs to make a <b>decision</b>.</p> <p><b>Example</b></p> <pre>' Which number is larger? Option Explicit Private Sub cmdLarger_Click()     'Memory     Dim Num1 As Double, _         Num2 As Double, _         Larger As Double     'Input     Num1 = Val(txtNum1.Text)     Num2 = Val(txtNum2.Text)     'Processing     If Num1 &gt; Num2 Then         Larger = Num1     Else         Larger = Num2     End If     'Output     lblLarger.Caption = CStr(Larger) End Sub</pre> | <p>Whenever your program needs to <b>repeat</b> certain instructions two or more times, the concept of <b>repetition</b> (looping) is used. Many different types of <b>loops</b> can be constructed, depending on the particular situation.</p> <p><b>Example</b></p> <pre>' Program to add the cubes of the ' numbers from 1 to 5 Option Explicit Private Sub cmdSumOfCubes_Click()     'Memory     Dim I As Byte     Dim Total As Double     'Processing     Total = 0     'I is called a loop counter     'variable     For I = 1 To 5         Total = Total + I ^ 3     Next I     'Output     Print "The sum is "; Total End Sub</pre> |

If you are typing a long logical line of code, it will be easier to read if you break it up into two or more physical lines. To do this, use the **statement continuation character** “\_” (a space followed by an underscore).

Notice the **indentation** used in these programs. Although your programs will work without proper indentation, they will be extremely difficult to read, understand and debug. The rules of indentation are simple and **must be observed by all students**. Failing to indent properly will result in a significant loss of marks. **RULES OF INDENTATION:** Indent one tab space within **subs, if statements** and **loops** (more details will be given in subsequent examples).

## Questions and Programming Exercises

1. What is the purpose of the statement continuation character?
2. Why is it important to indent programs properly?
3. Explain the terms *sequence*, *selection* and *repetition*.
4. Define the term *underpinning*.
5. To understand the example of repetition given above, it is very helpful to trace the execution of the program by using something called a **memory map**. A memory map is simply a table that displays the changing values of variables. Complete the memory map shown below.







# SELECTED SOLUTIONS TO ASSIGNED VB PROBLEMS

## Solution 1

```
' This program will calculate the sum of consecutive
' integers each raised to an exponent chosen by the user.
' For example, if the user chooses to start at 5, end at
' 50 and an exponent of 3, the program will calculate the
' sum of the cubes of the integers from 5 to 50.
```

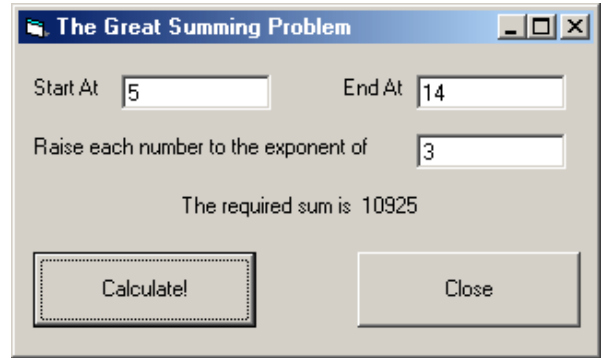
Option Explicit

```
Private Sub cmdCalculate_Click()
    ' Memory
    Dim I As Integer, Start As Integer, Finish As Integer
    Dim Total As Double, Exponent As Byte
    ' Input
    Start = Val(txtStart.Text)
    Finish = Val(txtEnd.Text)
    Exponent = Val(txtExponent.Text)
    ' Processing
    Total = 0
    For I = Start To Finish
        Total = Total + I ^ Exponent
    Next I
    ' Output
    lblAnswer.Visible = True
    lblAnswer.Caption = "The sum is " & Str(Total) & "."
End Sub
Private Sub txtEnd_Change()
    lblAnswer.Visible = False
End Sub
Private Sub txtExponent_Change()
    lblAnswer.Visible = False
End Sub
Private Sub txtStart_Change()
    lblAnswer.Visible = False
End Sub
Private Sub cmdClose_Click()
    Dim Response As VbMsgBoxResult
    Response = MsgBox("Are you sure you want to" & _
        "quit?", vbYesNo, "Quitting...")
    If Response = vbYes Then
        Unload Me
    End If
End Sub
```

## Questions Related to Solution 1

1. Explain the purpose of the “Val” function and the “CStr” function. In your answer, do not forget to mention the difference between string and numeric data types.
2. What is the function of the “&” operator?
3. Explain why the words “End” and “Stop” cannot be used for Visual Basic variable or object names.
4. Explain how the “Change” event differs from the “Click” event.
5. Explain why this program is superior to the programs that only work for specific numbers. How does the use of variables make it possible for this program to work for any numbers chosen by the user?
6. Notice in the solution above that all the variables are declared **within** a sub procedure. Such variables are called **local variables**. How do they differ from variables that are declared **outside** of any procedure (called **global variables**)?
7. Complete the following memory map for the program shown above.

| Total | I | I ^ Exponent | Start | Finish | Exponent |
|-------|---|--------------|-------|--------|----------|
| 0     | 0 | 0            | 3     | 8      | 2        |
|       |   |              | 3     | 8      | 2        |
|       |   |              | 3     | 8      | 2        |
|       |   |              | 3     | 8      | 2        |
|       |   |              | 3     | 8      | 2        |
|       |   |              | 3     | 8      | 2        |
|       |   |              | 3     | 8      | 2        |
|       |   |              | 3     | 8      | 2        |



## Note

1. For the given program to work, you must use the same object names that I have used. Otherwise, you will need to modify the given program so that the object names agree with the ones that you have chosen.
2. Also, note that the variable called **Finish** may seem like a strange choice. Names like **End** and **Stop** would have been simpler choices, however, both of these names **cannot be used as variable or object names because both are Visual Basic keywords (reserved words)**. Therefore, I was forced to choose a different name.
3. I have added code for the three text boxes. The sub procedures for the text boxes simply cause the label box “**lblAnswer**” to disappear whenever the value in any of the text box changes. Notice the use of the “**Change**” event in each sub procedure.

## Solution 2

### Option Explicit

```
Dim SecretNumber As Byte
```

```
'This sub procedure is executed as soon as the form loads.
```

```
Private Sub Form_Load()
```

```
    Randomize 'This is used to make the game unpredictable  
    SecretNumber = Int(Rnd * 100 + 1)  
    frmGuess.Show 'Make form visible  
    txtGuess.SetFocus 'Give focus to the txtGuess text box
```

```
End Sub
```

```
Private Sub cmdClose_Click()
```

```
    Unload Me  
End
```

```
End Sub
```

```
Private Sub cmdEnterGuess_Click()
```

```
    Dim Guess As Byte  
    Guess = Val(txtGuess.Text)  
    If Guess > SecretNumber Then  
        lblMessage.Caption = "Sorry," & Str(Guess) & " is too high."  
    ElseIf Guess < SecretNumber Then  
        lblMessage.Caption = "Sorry," & Str(Guess) & " is too low."  
    Else  
        lblMessage.Caption = "Great work," & Str(Guess) & " is correct!"  
    End If  
    txtGuess.Text = "" 'Clear the txtGuess text box.  
    txtGuess.SetFocus 'Give focus to the txtGuess text box
```

```
End Sub
```

```
Private Sub cmdGenerateRandomNum_Click()
```

```
    SecretNumber = Int(Rnd * 100 + 1)  
    lblMessage.Caption = "" 'Clear message label box  
    txtGuess.SetFocus 'Give focus to the txtGuess text box
```

```
End Sub
```

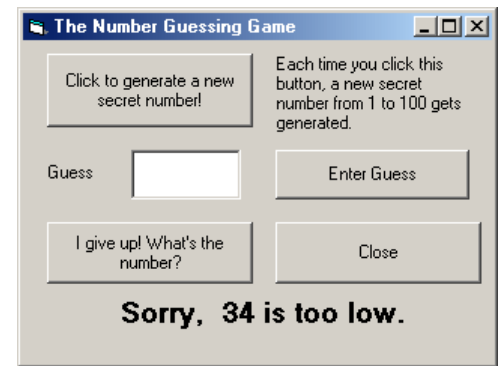
```
Private Sub cmdIGiveUp_Click()
```

```
    lblMessage.Caption="The secret number is " & CStr(SecretNumber)&"."  
    txtGuess.SetFocus 'Give focus to the txtGuess text box
```

```
End Sub
```

## Questions Related to Solution 2

1. Why is **SecretNumber** declared as a **global** (module or external) variable while **Guess** is declared as a **local** (procedure level or automatic) variable?
2. Explain the purpose of the **SetFocus** method.
3. Explain the purpose of a **Form\_Load** sub procedure. How would you get Visual Basic to automatically generate the first line and the last line of a **Form\_Load** sub procedure.
4. Modify the program shown above so that
  - a) it can generate a secret number between any two integers chosen by the user
  - b) it gives the user a limited number of guesses (allow the user to set this number according to the desired level of difficulty)
  - c) the messages in the label box are displayed in different colours (e.g. red for incorrect guess, green for correct guess)
  - d) the total number of guesses is displayed (and updated every time that the "Enter Guess" button is clicked)
5. Suggest other modifications to the above program that will improve its functionality and make it more interesting to use.



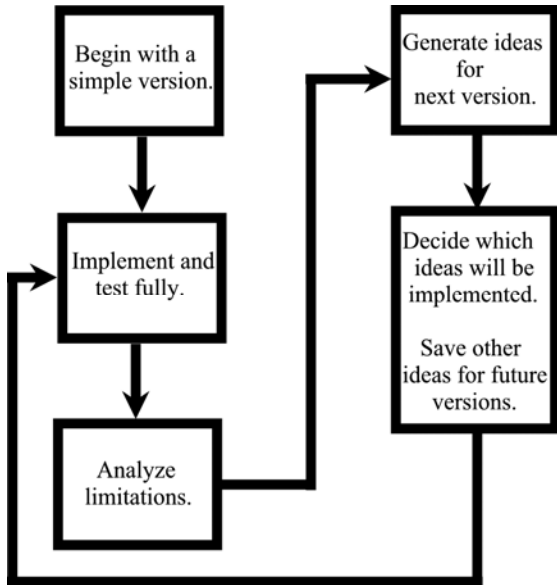
### Note

1. This program uses two variables, **SecretNumber** and **Guess**. One is a **local** variable and the other is **global**.
2. The **SetFocus** method is used in several places to give the focus to the **txtGuess** text box. This is done so that the user does not need to keep clicking in the text box to enter a guess. Regardless of which button is clicked, the focus always immediately returns to the text box.
3. The **null string** has been used to clear the contents of the **txtGuess** text box and the **lblMessage** label box. The **null string** is simply a string consisting of **zero** characters. This is why it is written as two consecutive quotation marks ("").
4. **Randomize** is used to prevent your program from generating the same random numbers every time it is run. This makes your game unpredictable.

# THE EVOLUTION OF SOFTWARE PART I: HOW TO KEEP IMPROVING YOUR SOFTWARE

## Case Study – Developing a CRAPS Game in VB

- **Phase 1:** Develop a program that simulates the rolling of a pair of dice.
- **Phase 2:** Use the Internet to find pictures of dice and then incorporate them into your program. After every roll, your software should display pictures of dice that correspond to the roll. Write code that is as general and efficient as possible.
- **Phase 3:** Use the Internet to find the rules for the game of CRAPS.
- **Phase 4:** Incorporate the rules into your program.
- **Phase 5:** Enhance your program by including new features and by improving the code. (The following diagram describes the process of continually improving software.)



The flowchart shown at the left is a simplified visual representation of the software development process. Notice that programmers use a simple version as a foundation upon which future versions can be built. Also, note that once the initial simple version has been implemented, an essentially infinite loop is entered. Since software development involves open-ended tasks, there is virtually no limit to the improvements that can be made.

When engaged in this process, try to keep in mind the following points:

- Break up large, complex problems into several smaller problems.
- Solve one small problem at a time. Ensure that each solution is perfect before integrating it into the overall system.
- Be realistic! It is far better to produce simple software that works well than it is to produce sophisticated software that does not work at all.
- Do not limit yourself during the idea generation phase. Write down all your ideas (including those that seem over-ambitious or downright crazy).

## A Solution to Phase One

```
.....  
' CRAPS Version 1.0 (N. Nolfi)  
' This version simply gets the dice rolling properly.  
.....
```

### Option Explicit

```
Private Sub Form_Load()  
    Randomize  
End Sub
```

```
'Sub procedure to roll dice and display results
```

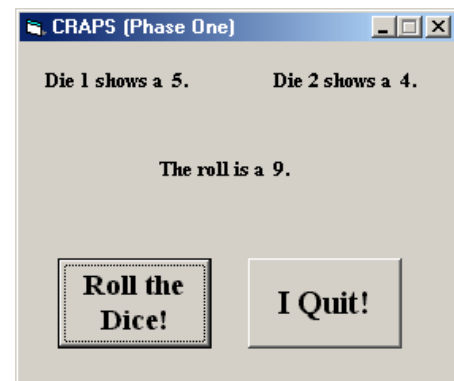
```
Private Sub cmdRoll_Click()  
  
    Dim Die1 As Byte, Die2 As Byte, Roll As Byte  
  
    'Generate random integers and find sum  
    Die1 = Int(Rnd * 6 + 1)  
    Die2 = Int(Rnd * 6 + 1)  
    Roll = Die1 + Die2
```

```
    'Display the results in label boxes  
    lblDie1.Caption = "Die 1 shows a" & Str(Die1) & "."  
    lblDie2.Caption = "Die 2 shows a" & Str(Die2) & "."  
    lblRoll.Caption = "The roll is a" & Str(Roll) & "."
```

```
End Sub
```

```
'Sub to close the program
```

```
Private Sub cmdClose_Click()  
    Unload frmCraps  
End  
End Sub
```



## Questions

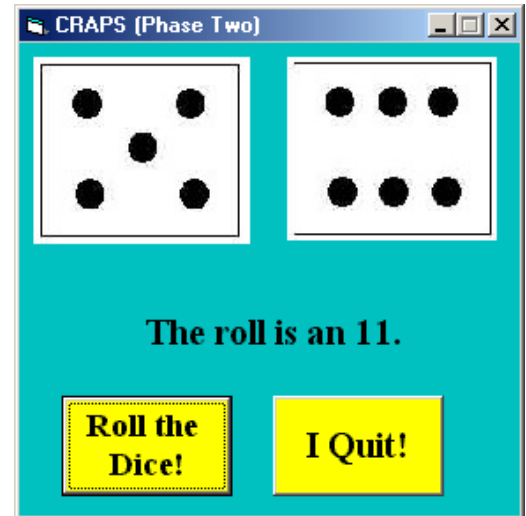
1. By carefully reading the code at the left, determine the name of each object on the above form.
2. When you run this program, you will notice that the maximize button is disabled and that the form cannot be “resized.” How was this accomplished?
3. Why does the command “Randomize” appear in the “Form\_Load” sub procedure? What would happen if this command were omitted?
4. Why are the variables “Die1,” “Die2” and “Roll” declared as “Byte” variables?

## Two Different Solutions to Phase Two

### Solution 1 – Craps 1.1a

```
.....
' CRAPS Version 1.1a (N. Nolfi)
' This version incorporates pictures of dice. Notice that
' this version uses a selection structure (i.e. "If" statement)
' to decide which dice pictures to display.
.....

Option Explicit
Private Sub Form_Load()
    Randomize
End Sub
Private Sub cmdClose_Click()
    Unload Me
End Sub
End Sub
'Sub procedure to roll dice and display results
Private Sub cmdRoll_Click()
    Dim Die1 As Byte, Die2 As Byte, Roll As Byte
    Dim Die1PathName As String, Die2PathName As String
    'Generate random integers and find sum
    Die1 = Int(Rnd * 6 + 1)
    Die2 = Int(Rnd * 6 + 1)
    Roll = Die1 + Die2
    'Decide which dice pictures to display
    If Die1 = 1 Then
        Die1PathName = "g:\dice-1.jpg"
    ElseIf Die1 = 2 Then
        Die1PathName = "g:\dice-2.jpg"
    ElseIf Die1 = 3 Then
        Die1PathName = "g:\dice-3.jpg"
    ElseIf Die1 = 4 Then
        Die1PathName = "g:\dice-4.jpg"
    ElseIf Die1 = 5 Then
        Die1PathName = "g:\dice-5.jpg"
    Else
        Die1PathName = "g:\dice-6.jpg"
    End If
    If Die2 = 1 Then
        Die2PathName = "g:\dice-1.jpg"
    ElseIf Die2 = 2 Then
        Die2PathName = "g:\dice-2.jpg"
    ElseIf Die2 = 3 Then
        Die2PathName = "g:\dice-3.jpg"
    ElseIf Die2 = 4 Then
        Die2PathName = "g:\dice-4.jpg"
    ElseIf Die2 = 5 Then
        Die2PathName = "g:\dice-5.jpg"
    Else
        Die2PathName = "g:\dice-6.jpg"
    End If
    'Display the results
    imgDie1.Picture = LoadPicture(Die1PathName)
    imgDie2.Picture = LoadPicture(Die2PathName)
    If Roll <> 8 And Roll <> 11 Then
        lblRoll.Caption = "The roll is a "& CStr(Roll) & "."
    Else
        lblRoll.Caption = "The roll is an " & CStr(Roll) & "."
    End If
End Sub
```



### Questions

1. When you run this version of the program, you will notice that the form has a light blue background and that the buttons have a yellow background. How was this accomplished?
2. State at least three ways in which version 1.1b is superior to version 1.1a.
3. In version 1.1b, why have I used “App.Path” instead of a drive letter and folder path name?

## Solution 2 – Craps 1.1b

```
.....  
' CRAPS Version 1.1b (N. Nolfi)  
' This version incorporates pictures of dice. Notice that  
' this version DOES NOT use an "if" statement to decide which  
' dice pictures to display.  
.....  
Option Explicit  
Private Sub Form_Load()  
    Randomize  
End Sub  
Private Sub cmdClose_Click()  
    Unload frmCraps  
End  
End Sub  
'Sub procedure to roll dice and display results  
Private Sub cmdRoll_Click()  
    Dim Die1 As Byte, Die2 As Byte, Roll As Byte  
    Dim Die1PathName As String, Die2PathName As String  
    'Generate random integers and find sum  
    Die1 = Int(Rnd * 6 + 1)  
    Die2 = Int(Rnd * 6 + 1)  
    Roll = Die1 + Die2  
    'Build path names for die1 and die 2 pictures  
    Die1PathName = App.Path & "\dice-" & CStr(Die1) & ".jpg"  
    Die2PathName = App.Path & "\dice-" & CStr(Die2) & ".jpg"  
    'Display the results  
    imgDie1.Picture = LoadPicture(Die1PathName)  
    imgDie2.Picture = LoadPicture(Die2PathName)  
    If Roll <> 8 And Roll <> 11 Then  
        lblRoll.Caption = "The roll is a "& CStr(Roll) & "."  
    Else  
        lblRoll.Caption = "The roll is an " & CStr(Roll) & "."  
    End If  
End Sub
```

4. What is the purpose of the “If” statement in the “cmdRoll\_Click” sub procedure?

5. What is the purpose of the “CStr” *intrinsic (built-in) function*?

## Thinking about Phases Three and Four

Presented below are the rules for the game of *Craps*. As you can see, the rules are quite complicated and it would be far too ambitious to try to implement them all at once. A better approach is to implement them one at a time. For now, we shall focus only on the “Pass Line” rules. Very skilful programmers may decide to do more.

*Craps is a game that allows you to bet with or against a “shooter” (the person who rolls the dice). To shoot the dice, you must make a bet on either the “Pass Line” or the “Don’t Pass Line.”*

### Pass Line

When you put your money on the “Pass Line” and the shooter rolls a 7 or 11 on the first roll of the dice, you win. If the shooter rolls a 2, 3 or 12 (*Craps*), you lose. If any other number occurs (4, 5, 6, 8, 9 or 10), it becomes the shooter’s “point.” The shooter must roll that point again, before a 7, in order for you to win. If the point is rolled, you will be paid the amount of your “Pass Line” bet.

### Don’t Pass Line

When you bet the “Don’t Pass Line,” the reverse occurs. You lose if a 7 or 11 is rolled on the first roll and you win if 3 or 12 is rolled. However, if a 2 shows up on the first roll, it is a “push.” If a 4, 5, 6, 8, 9 or 10 is rolled, it becomes the “front line point.” In order for you to win, the 7 must be rolled before the point is rolled again. Should the 7 roll first, you will win your “Don’t Pass” bet.

### Come

After a “point” has been established, you may make a “Come” bet. This area is also clearly defined on the Craps layout. You may bet this any time after the first roll. When you bet the “Come Line,” you win if the next roll is a 7 or 11. You lose if the next roll is a 2, 3 or 12. If any other number is rolled (4, 5, 6, 8, 9 or 10), it becomes your “point.” Should a 7 roll before your point, you lose. However, if your point rolls before a 7 is rolled, you win the amount you bet on the “Come Line.”

### Don’t Come

After a point has been established, you may make a “Don’t Come” bet. You may bet this area any time after the first roll. When you bet the “Don’t Come Line,” you win if the next roll is a 3 or a 12. If that next roll is a 2, it is a stand-off. Should that next roll be a 7 or 11, you lose. If any other number is rolled (4, 5, 6, 8, 9 or 10), it becomes your point. If that point rolls before a 7, your bet loses. However, if a 7 rolls first, you win your “Don’t Come” bet.

### Field

This is a one-roll bet and may be bet at any time. If you bet the "Field" and a 3, 4, 9, 10 or 11 rolls, you win the amount you have bet. However, if a 2 rolls, you win twice the amount bet and if a 12 rolls you win three times your bet. Should any other number roll (5, 6, 7 or 8), you lose your bet.

### Big Six or Eight

On the outside corners of each side of the Craps layout, you will see a big red 6 and 8. On this bet, you are wagering that a 6 or 8 will roll before a 7 rolls. If it does, you win.

### Any Seven

On this, you are wagering that a 7 will roll on the next throw of the dice. If it does, you win four times the amount of your bet. This is a one-roll bet.

### Bet the Horn

On this bet, you are wagering that a 2, 3, 11 or 12 will roll on the next throw of the dice. "Betting the Horn" is a fun bet that pays high odds if you win. If a 2 or 12 rolls, you win \$6.75 for every dollar bet. If a 3 or 11 rolls, you win \$3 for every dollar bet. This is a one-roll bet.

### Hard Ways

You are wagering that the next time a 4, 6, 8 or 10 is rolled, it will be rolled as a pair of 2s, 3s, 4s or 5s. This area on the Craps layout shows the odds paid. Ask the dealer to explain this particular bet.

### Any Craps

This is also a one-roll bet. If a 2, 3 or 12 is rolled, you win 7 times your bet.

### Proposition Bets

You bet on 2, 3, 11 or 12 before any roll. If a 2 or 12 occurs, you win 30 to 1; a 3 or 11 pays 15 to 1.

### Place Bets

You **can** bet on 4, 5, 6, 8, 9 or 10. If a 6 or 8 is rolled, you collect 7 to 6; if 5 or 9 come up, you get 7 to 5. A roll of 4 or 10 pays 9 to 5. "Place Bets" can be removed any time before a roll.



# THE EVOLUTION OF SOFTWARE PART II: FURTHER IMPROVEMENTS TO THE CRAPS SOFTWARE

## Questions

1. If you test this program carefully, you will notice an annoying time lag every time you press the “Roll” button (i.e the program takes too long to display the pictures of the dice). What causes this? Suggest a method for improving the performance of this program.
  
2. Explain the bug in version 1.1a. What sometimes causes the “File not found” error to be displayed? (**Hint:** g:\)
  
3. Study the description of the string manipulation keyword “Right” given below. Then study the examples on the next page. After doing so, write a brief explanation of how it is used in version 1.1b of CRAPS.

```

.....
' CRAPS Version 1.1c (N. Nolfi)
' NOTE: This version is a correction of version 1.1b. In
'       version 1.1a there is a bug that sometimes causes the
'       path name to be incorrect.
.....

Option Explicit
Private Sub cmdClose_Click()
    End
End Sub

'Sub procedure to roll dice and display results
Private Sub cmdRoll_Click()
    Dim Die1 As Byte, Die2 As Byte
    Dim Die1PathName As String, Die2PathName As String, ApplPath As String

    'Generate random integers and find sum
    Die1 = Int(Rnd * 6 + 1)
    Die2 = Int(Rnd * 6 + 1)
    Roll = Die1 + Die2

    'Build path names for die1 and die 2 pictures. If the path name
    'already has a backslash at the end, then remove it. This is
    'done to correct the bug in version 1.1a.
    ApplPath = App.Path
    If Right(ApplPath, 1) = "\" Then
        ApplPath = Right(ApplPath, Len(ApplPath) - 1)
    End If

    Die1PathName = ApplPath & "\dice-" & CStr(Die1) & ".jpg"
    Die2PathName = ApplPath & "\dice-" & CStr(Die2) & ".jpg"

    'Display the results
    imgDie1.Picture = LoadPicture(Die1PathName)
    imgDie2.Picture = LoadPicture(Die2PathName)
    If Roll <> 8 And Roll <> 11 Then
        lblRoll.Caption = "The roll is a " & CStr(Roll) & "."
    Else
        lblRoll.Caption = "The roll is an " & CStr(Roll) & "."
    End If
End Sub

Private Sub Form_Load()
    Randomize
End Sub

```

## Syntax

### Right(string, length)

Returns a **Variant (String)** containing a specified number of characters from the right side of a string.

The **Right** function syntax has these named arguments:

| Part          | Description                                                                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>string</i> | Required. String expression from which the rightmost characters are returned. If <i>string</i> contains Null, <b>Null</b> is returned.                                                                                                             |
| <i>length</i> | Required; <b>Variant (Long)</b> . Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in <i>string</i> , the entire string is returned. |



## Right Function Example

' This example uses the Right function to return a specified number of characters from the right side of a string.

```
Dim AnyString As String, MyString As String
AnyString = "Hello World" ' Define string
MyString = Right (AnyString, 1) ' Returns "d"
MyString = Right (AnyString, 6) ' Returns " World"
MyString = Right (AnyString, 20) ' Returns "Hello World"
```

## Research Assignment

Use <http://msdn.microsoft.com> or Visual Basic MSDN files to find information on each of the following string manipulation functions:

- Left, Right, Mid, Len, Trim, RTrim, Ltrim, InStr

Write a brief explanation of each.

## Questions

1. Explain why version 1.2 displays the dice pictures much more quickly than version 1.1
2. In what other ways is version 1.2 better than version 1.1?
3. Hard disk drive access times are measured in milliseconds (ms). RAM access times are measured in nanoseconds (ns). Approximately how many times faster than a hard drive is RAM?

```
.....
' CRAPS Version 1.2 (N. Nolfi)
'
' This version incorporates pictures of dice, but is faster than
' version 1.1. The improved speed is due to the fact that all
' the pictures are stored on a separate form which is not
' visible to the user. Since the form is stored in RAM, the
' pictures can be copied very quickly. In version 1.1, the
' pictures were loaded from a file (stored on a disk drive).
' Since disk drives are so slow compared to RAM, version 1.1 was
' quite slow.
.....

Option Explicit
Private Sub Form_Load()
    Randomize
    Load frmPictures ' Load the pictures form, but keep it invisible.
End Sub

Private Sub cmdClose_Click()
    Unload Me
End Sub

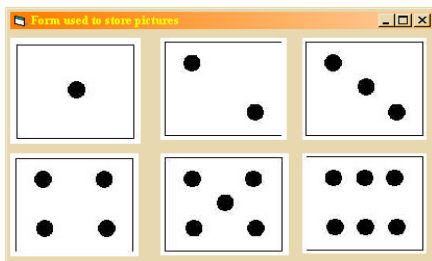
End Sub

'Sub procedure to roll dice and display results
Private Sub cmdRoll_Click()

    Dim Die1, Die2 As Byte, Roll As Byte
    'Generate random integers and find sum
    Die1 = Int(Rnd * 6 + 1)
    Die2 = Int(Rnd * 6 + 1)
    Roll = Die1 + Die2

    'Display the results
    imgDie1.Picture = frmPictures.imgDie(Die1 - 1).Picture
    imgDie2.Picture = frmPictures.imgDie(Die2 - 1).Picture
    If Roll <> 8 And Roll <> 11 Then
        lblRoll.Caption = "The roll is a " & CStr(Roll) & "."
    Else
        lblRoll.Caption = "The roll is an " & CStr(Roll) & "."
    End If
End Sub

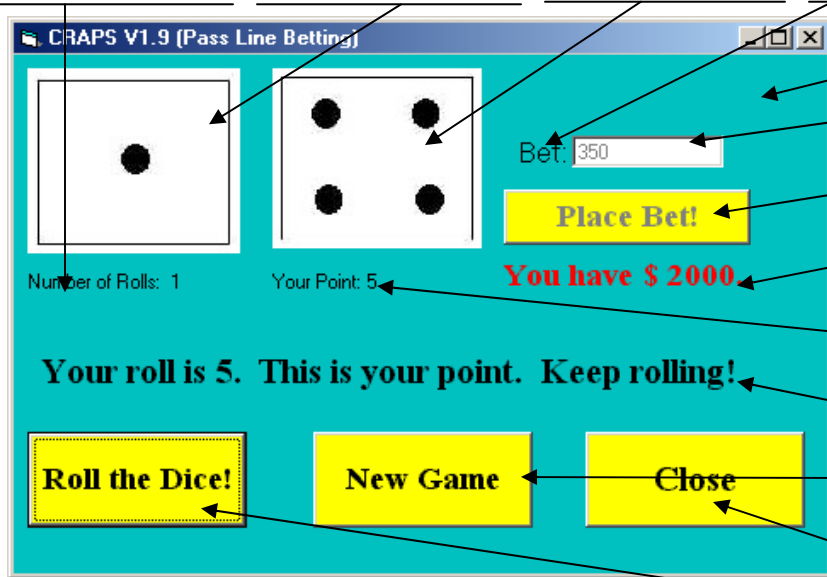
End Sub
```



This form is called **frmPictures**. It is never visible, but it is stored in RAM. This allows much faster access to the dice pictures. Loading the pictures from a file stored on a disk drive is extremely slow.

## DO YOU UNDERSTAND THE CRAPS PROGRAM?

Try the following practice quiz to find out how well you understand version 1.91 of the “Craps” program as well as VB in general. When you are finished, obtain an answer sheet from me and compare your answers to mine. If you do well, you are ready to move on. Otherwise, some remedial work is necessary!



```

.....
' CRAPS Version 1.91 - Corrections of Minor Bugs in V1.9
' (N. Nolfi) pass line betting only
.....
Option Explicit
Dim Money As Currency, Bet As Currency, NumOfRolls As Integer, YourPoint As Byte
'Code for "New Game" button
Private Sub cmdNewGame_Click()
    Dim Response As String
    Response = InputBox("How much money are you willing to risk?", _
        "Are You a Gambler?")

    'Start a new game only if a value is entered and "OK" is clicked.
    'Do not start a new game if "Cancel" is clicked or no value is entered.
    If Response <> "" Then
        Money = Val(Response)
        If Money > 0 And Money <= 1000000 Then
            lblMoney.Caption = "You have " & Format(Money, "Currency") & "."
            cmdPlaceBet.Visible = True
            lblBet.Visible = True
            lblNumberOfRolls.Visible = True
            lblNumberOfRolls.Caption = "Number of Rolls: 0"
            txtBet.Visible = True
            txtBet.SetFocus
            Call ResetGame
        Else
            MsgBox "Please enter a value between $0.01 and $1000000", _
                vbExclamation, "Try Again"
        End If
    End If
End Sub
Private Sub Form_Load()
    Randomize
    frmCraps.Show
    Load frmPictures ' Load the pictures form, but keep it invisible.
End Sub
*****The code continues on the next page.*****

```

### Questions

1. Carefully study the provided code. Then state the name of each object on the form at the left.
2. Explain why the variables *Money*, *Bet*, *NumofRolls* and *YourPoint* are all declared as global variables.
3. Why are *Money* and *Bet* declared as **Currency** variables?
4. Why is *NumOfRolls* declared as an **Integer** variable while *YourPoint* is declared as a **Byte** variable?

```

'Sub procedure to roll dice and display results
Private Sub cmdRoll_Click()
    Dim Die1 As Byte, Die2 As Byte, Roll As Byte
    'Generate random integers and find sum
    Die1 = Int(Rnd * 6 + 1)
    Die2 = Int(Rnd * 6 + 1)
    Roll = Die1 + Die2
    'Display the results of the roll
    imgDie1.Visible = True
    imgDie2.Visible = True
    imgDie1.Picture = frmPictures.imgDie(Die1 - 1).Picture
    imgDie2.Picture = frmPictures.imgDie(Die2 - 1).Picture
    lblRoll.Visible = True
    lblRoll.Caption = "Your roll is" & Str(Roll) & "."
    lblNumberOfRolls.Caption = "Number of Rolls:" & Str(NumOfRolls)
    'Decide if the shooter wins, loses or continues to roll. (Pass line rules)
    If NumOfRolls > 1 Then 'After the first roll

        If Roll = 7 Then 'LOSE!
            lblRoll.Caption = lblRoll.Caption & " You lose" & _
                Format(Bet, "Currency") & "!"
            Money = Money - Bet
            lblMoney.Caption = "You have " & Format(Money, "Currency") & "."
            MsgBox lblRoll.Caption, vbInformation, "Craps Pass Line Betting"
            Call ResetGame
        ElseIf Roll = YourPoint Then 'WIN!
            Money = Money + Bet
            lblRoll.Caption = lblRoll.Caption & " You win $" & _
                Format(Bet, "Currency") & "!"
            lblMoney.Caption = "You have " & Format(Money, "Currency") & "."
            MsgBox lblRoll.Caption, vbInformation, "Craps Pass Line Betting"
            Call ResetGame
        Else 'Keep rolling!
            lblRoll.Caption = lblRoll.Caption & " Keep rolling!"
            NumOfRolls = NumOfRolls + 1
        End If
    Else 'First roll of the dice
        If Roll = 7 Or Roll = 11 Then 'WIN!
            Money = Money + Bet
            lblRoll.Caption = lblRoll.Caption & " You win $" & _
                Format(Bet, "Currency") & "!"
            lblMoney.Caption = "You have " & Format(Money, "Currency") & "."
            MsgBox lblRoll.Caption, vbInformation, "Craps Pass Line Betting"
            Call ResetGame
        ElseIf Roll = 2 Or Roll = 3 Or Roll = 12 Then 'LOSE!
            Money = Money - Bet
            lblRoll.Caption = lblRoll.Caption & " You lose $" & _
                Format(Money, "Currency") & "!"
            lblMoney.Caption = "You have " & Format(Money, "Currency") & "."
            MsgBox lblRoll.Caption, vbInformation, "Craps Pass Line Betting"
            Call ResetGame
        Else 'Establish point.
            YourPoint = Roll
            lblRoll.Caption = lblRoll.Caption & _
                " This is your point. Keep rolling!"
            NumOfRolls = NumOfRolls + 1
            lblYourPoint.Visible = True
            lblYourPoint.Caption = "Your Point:" & Str(YourPoint)
        End If
    End If
End Sub

```

' \*\*\*\*\*The code continues on the next page.\*\*\*\*\*

5. In what way does the *ResetGame* sub procedure differ from all the other sub procedures in this program?

6. How is the *ResetGame* procedure invoked (“called into action”)?

7. Explain the difference between the **Visible** property and the **Enabled** property. Use an example from the given code to illustrate your answer.

```

Private Sub cmdPlaceBet_Click()
    Bet = Val(txtBet.Text)
    If Bet > 0 And Bet <= Money Then
        cmdRoll.Enabled = True
        cmdPlaceBet.Enabled = False 'Disallow change of bet once
        txtBet.Enabled = False      'bet is placed.
    Else
        txtBet.Text = ""
        txtBet.SetFocus
    End If
End Sub

'A "General" procedure that is used to reset several game parameters
'once the shooter either wins or loses.
Private Sub ResetGame()
    NumOfRolls = 1
    lblYourPoint.Visible = False
    lblRoll.Visible = False
    lblNumberOfRolls.Caption = "Number of Rolls: 0"
    cmdRoll.Enabled = False
    txtBet.Text = ""
    txtBet.Enabled = True
    txtBet.SetFocus
    cmdPlaceBet.Enabled = True
    imgDie1.Visible = False
    imgDie2.Visible = False
    If Money <= 0 Then
        MsgBox "Sorry, you have run out of money.", vbInformation, "Game Over!"
        cmdPlaceBet.Enabled = False
        txtBet.Enabled = False
    End If
End Sub

'Code for "Close" button
Private Sub cmdClose_Click()
    Unload frmCraps
End Sub
End Sub

```

8. In various places in the given code, you will find the assignment statements  $Money = Money - Bet$  and  $Money = Money + Bet$ . Explain the purpose of each line.

9. In this program, you will notice the use of the VB keyword “Or”. In other programs that we have studied, you will have noticed the use of the VB Keyword “And.”

Explain the meaning of each keyword.

Explain the uses of each keyword.

# THE TEMPERATURE CONVERTER PROGRAM

Write a program that will convert a temperature expressed in a certain unit to three other units (as shown in the form given below).

## Note

1. Your program should behave intelligently if the user enters invalid information. Please ensure that your program can deal with errors such as
  - no temperatures entered by the user
  - impossible temperatures entered by the user
2. You should be aware of the fact that you will need to use the intrinsic functions **CStr** and **Val**. (If nothing is entered in a text box, then its Text property is assigned a value of the null string (i.e. ""). In this case, using Val will return a value of zero.)
3. The following formulas are used to convert from Celsius degrees to each of the others. The variables *C*, *K*, *F* and *R* represent Celsius degrees, Kelvin degrees, Fahrenheit degrees and Rankine degrees respectively.
$$F = 1.8C + 32$$
$$K = C + 273.15$$
$$R = 1.8C + 491.67$$
Notice that all these equations represent **straight lines**.
4. The image shown on the form at the left is stored in a file called tempGraph.bmp in the folder **I:\Out\Nolfi\Tik2o0**. Use an image control to display this picture on your form.

## Questions to Consider before Writing Code

1. *Absolute zero*, the lowest temperature possible, is the temperature at which atoms stop emitting heat energy. Therefore, the *Kelvin* and *Rankine* temperature scales can never be negative because for both, zero degrees is defined as *absolute zero*. The only difference between the two is that the Kelvin scale has units of exactly the same size as the Celsius scale and the Rankine scale has units of the same size as the Fahrenheit scale.

Use this information to express absolute zero in degrees Celsius and degrees Fahrenheit. What temperatures should your program reject?



## OPERATORS IN VISUAL BASIC

### Arithmetic Operators

| <i>Operator</i> | <i>Meaning</i>                                                                                                                                                            |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| +               | Add two numbers and return a <i>floating-point</i> result.                                                                                                                |
| -               | Subtract two numbers and return a <i>floating-point</i> result.                                                                                                           |
| *               | Multiply two numbers and return a <i>floating-point</i> result.                                                                                                           |
| /               | Divide two numbers and return a <i>floating-point</i> result.                                                                                                             |
| \               | Divide two numbers and return an <i>integer</i> result. Any fractional portion is truncated. (e.g. $7 \setminus 5$ returns 1 because the "0.4" portion is "chopped off.") |
| ^               | Used to raise a number to an <i>exponent</i> and return a <i>floating-point</i> result.                                                                                   |
| Mod             | Used to divide two numbers and return only the <i>remainder</i> .                                                                                                         |

### Mathematical Functions

| <i>Function</i>      | <i>Meaning</i>                                                                                                                                                                                                                                                       |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Int                  | Returns a number <i>rounded down</i> to the nearest integer.                                                                                                                                                                                                         |
| Fix                  | Returns the <i>integer portion</i> of a number (any fractional part is truncated).                                                                                                                                                                                   |
| Sqr                  | Returns the <i>square root</i> of a <i>non-negative</i> number.                                                                                                                                                                                                      |
| Abs                  | Returns the <i>absolute value</i> of a number.                                                                                                                                                                                                                       |
| Sin                  | Returns the <i>sine</i> of a number.                                                                                                                                                                                                                                 |
| Cos                  | Returns the <i>cosine</i> of a number.                                                                                                                                                                                                                               |
| Tan                  | Returns the <i>tangent</i> of a number.                                                                                                                                                                                                                              |
| Atn                  | Returns the <i>arctangent</i> (inverse tangent) of a number.                                                                                                                                                                                                         |
| Rnd                  | Returns a <b>Single</b> value containing a <i>random number</i> that is greater than or equal to zero and less than one.                                                                                                                                             |
| Exp                  | Returns a <b>Double</b> value specifying $e$ (the base of natural logarithms) raised to an exponent.                                                                                                                                                                 |
| Log                  | Returns a <b>Double</b> specifying the natural logarithm of a positive number.                                                                                                                                                                                       |
| Other math functions | Other math functions can be derived from the basic functions listed above. For more information, refer to "Derived Math Functions" in the Microsoft Visual Studio help files or "MSDN Online" at <a href="http://msdn.microsoft.com">http://msdn.microsoft.com</a> . |

## Comparison Operators

| Operator | Meaning                     |
|----------|-----------------------------|
| =        | Is equal to                 |
| <        | Is less than                |
| <=       | Is less than or equal to    |
| >        | Is greater than             |
| >=       | Is greater than or equal to |
| <>       | Is not equal to             |

## Logical Operators

| Operator                                               | Meaning                                                                                                                                                                                                      |
|--------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>And</b><br>(logical conjunction)                    | expression1 <b>And</b> expression2 is <b>True</b> if and only if expression1 is <b>True</b> and expression2 is <b>True</b> ; otherwise, it is <b>False</b>                                                   |
| <b>Or</b><br>(logical disjunction)                     | expression1 <b>Or</b> expression2 is <b>True</b> if expression1 is <b>True</b> or expression2 is <b>True</b> ; it is <b>False</b> if and only if expression1 is <b>False</b> and expression2 is <b>False</b> |
| <b>Not</b><br>(logical negation)                       | <b>Not</b> expression is <b>True</b> if expression is <b>False</b> and <b>False</b> if expression is <b>True</b>                                                                                             |
| <b>Xor</b> *<br>(logical exclusion or exclusive OR)    | expression1 <b>Xor</b> expression2 is <b>True</b> if expression1 is <b>True</b> or expression2 is <b>True</b> <i>but not both</i>                                                                            |
| <b>Eqv</b> *<br>(logical equivalence or exclusive NOR) | expression1 <b>Eqv</b> expression2 is <b>True</b> if and only if expression1 <b>Xor</b> expression2 is <b>False</b>                                                                                          |
| <b>Imp</b> *<br>(logical implication)                  | expression1 <b>Imp</b> expression2 is <b>False</b> if and only if expression1 is <b>True</b> and expression2 is <b>False</b> ; otherwise, it is <b>True</b>                                                  |

\* These logical operators are not used very frequently in the kinds of applications that we have written. **And**, **Or** and **Not** are used much more frequently (and are much easier to understand).

## Truth Tables for the Logical Operators

| expression1 | expression2 | expression1<br><b>And</b><br>expression2 | expression1<br><b>Or</b><br>expression2 | expression1<br><b>Xor</b><br>expression2 | expression1<br><b>Eqv</b><br>expression2 | expression1<br><b>Imp</b><br>expression2 |
|-------------|-------------|------------------------------------------|-----------------------------------------|------------------------------------------|------------------------------------------|------------------------------------------|
| F           | F           | F                                        | F                                       | F                                        | T                                        | T                                        |
| F           | T           | F                                        | T                                       | T                                        | F                                        | T                                        |
| T           | F           | F                                        | T                                       | T                                        | F                                        | F                                        |
| T           | T           | T                                        | T                                       | F                                        | T                                        | T                                        |

| expression | <b>Not</b> expression |
|------------|-----------------------|
| F          | T                     |
| T          | F                     |



## Operator Precedence (Order of Operations)

- When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next and logical operators are evaluated last.
- Comparison operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear.
- Arithmetic and logical operators are evaluated in the following order of precedence:

| Arithmetic                             | Comparison                    | Logical    |
|----------------------------------------|-------------------------------|------------|
| Exponentiation (^)                     | Equality (=)                  | <b>Not</b> |
| Negation ( <i>not</i> subtraction) (-) | Inequality (< >)              | <b>And</b> |
| Multiplication and division (*, /)     | Less than (<)                 | <b>Or</b>  |
| Integer division (\)                   | Greater than (>)              | <b>Xor</b> |
| Modulus arithmetic ( <b>Mod</b> )      | Less than or equal to (<=)    | <b>Eqv</b> |
| Addition and subtraction (+, -)        | Greater than or equal to (>=) | <b>Imp</b> |
| String concatenation (&)               | <b>Like, Is</b>               |            |

### Note

- When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right.
- When addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right.
- Parentheses can be used to override the order of precedence and force some parts of an expression to be evaluated before others. Operations within parentheses are always performed before those outside. Within parentheses, however, operator precedence is maintained.
- The string concatenation operator (&) is not an arithmetic operator, but in precedence, it does *follow* all arithmetic operators and *precede* all comparison operators.
- The **Like** operator is equal in precedence to all comparison operators, but is actually a pattern-matching operator.
- The **Is** operator is an object reference comparison operator. It does not compare objects or their values; it checks only to determine if two object references refer to the same object.

## Infrequently Used Operators

| Operator    | Description                          | For More Information        |
|-------------|--------------------------------------|-----------------------------|
| <b>Is</b>   | object reference comparison operator | See “Is Operator” in MSDN   |
| <b>Like</b> | a pattern-matching operator          | See “Like Operator” in MSDN |

## Type Conversion Functions

Type conversion functions are used to *coerce* (force) a conversion from one data type to another. The two most commonly used type conversion functions are **Val** and **CStr**. For more information, consult the MSDN collection.

# IMPROVING YOUR VISUAL BASIC PROGRAMS

## Rounding Off Values

In our first attempt to write the “Temperature Converter” program, we noticed several flaws. Two of these problems, hardware round off errors and the displaying of answers with too many decimal places, can be solved easily with the help of the “Round” function.

The following is a description (taken directly from the MSDN collection) of the “Round” intrinsic (built-in) function. It is used to round off any numeric expression to a desired number of decimal places. (The square brackets shown in the description indicate that the numDecimalPlaces argument is optional.)

### Round Function

#### Description

Returns a number rounded to a specified number of decimal places.

#### Syntax

**Round**(*expression* [,*numDecimalPlaces*])

The **Round** function syntax has these parts:

#### Argument

An argument is a value on which a function or procedure operates.

#### Square Brackets

Square brackets are used to indicate that an argument is optional.

#### Numeric Expression

A numeric expression is any expression that can be evaluated as a number. Elements of an expression can include any *combination* of keywords, variables, constants and operators that result in a number.

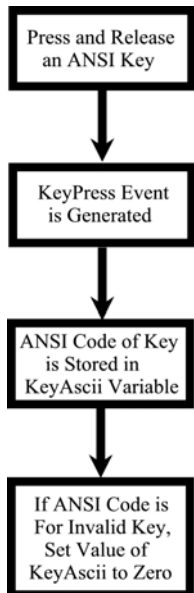
| Part                    | Description                                                                                                                                                           |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>expression</i>       | Required. <u>Numeric expression</u> being rounded.                                                                                                                    |
| <i>numDecimalPlaces</i> | Optional. Number indicating how many places to the right of the decimal are included in the rounding. If omitted, integers are returned by the <b>Round</b> function. |

#### Example

```
' Round off the Fahrenheit temperature to 2 decimal places.
Fahrenheit = Round(1.8 * Celsius + 32, 2)
```

## Using the KeyPress Event to “Trap” Invalid Characters

In the “Temperature Converter” program version 1.0, non-numeric characters are not rejected. This produces strange results that might confuse users. Luckily, Visual Basic provides a simple method for rejecting invalid characters. *The KeyPress event* consists of *pressing and releasing* an ANSI key (see table of characters below). When the KeyPress event is generated by the pressing and releasing of an ANSI key, the numeric code (ASCII code) representing the character is stored in a variable called *KeyAscii*. Your program can check the value of the *KeyAscii* variable to determine if the stored ANSI code is in the set of acceptable characters.



Your “Temperature Converter” program needs to have a sub procedure, like the one shown below, for each text box. To create a sub procedure for a particular text box, double-click on the text box. Note that this will not produce a sub procedure for the *KeyPress* event. Instead, the default event for text boxes (*Change*) will appear. To create a sub procedure for *KeyPress*, simply choose “KeyPress” from the drop-down list. Then erase the *Change* event sub procedure (e.g. txtCelsius\_Change).

```
' Reject any characters typed in the "txtCelsius" text box that do
' not lie between 0 and 9, except for the backspace key,
' the decimal point (".") and the negative sign ("-").
```

```
Private Sub txtCelsius_KeyPress(KeyAscii As Integer)
```

```
    If (KeyAscii < vbKey0 Or KeyAscii > vbKey9) And KeyAscii <> vbKeyBack _
        And Chr(KeyAscii) <> "." And Chr(KeyAscii) <> "-" Then
        KeyAscii = 0
```

```
    End If
```

```
End Sub
```

## Key Code Constants in Visual Basic

| Constant      | Value | Description | Constant             | Value | Description  | Constant             | Value | Description |
|---------------|-------|-------------|----------------------|-------|--------------|----------------------|-------|-------------|
| <b>vbKeyA</b> | 65    | A key       | <b>vbKey0</b>        | 48    | 0 key        | <b>vbKeyF1</b>       | 0x70  | F1 key      |
| <b>vbKeyB</b> | 66    | B key       | <b>vbKey1</b>        | 49    | 1 key        | <b>vbKeyF2</b>       | 0x71  | F2 key      |
| <b>vbKeyC</b> | 67    | C key       | <b>vbKey2</b>        | 50    | 2 key        | <b>vbKeyF3</b>       | 0x72  | F3 key      |
| <b>vbKeyD</b> | 68    | D key       | <b>vbKey3</b>        | 51    | 3 key        | <b>vbKeyF4</b>       | 0x73  | F4 key      |
| <b>vbKeyE</b> | 69    | E key       | <b>vbKey4</b>        | 52    | 4 key        | <b>vbKeyF5</b>       | 0x74  | F5 key      |
| <b>vbKeyF</b> | 70    | F key       | <b>vbKey5</b>        | 53    | 5 key        | <b>vbKeyF6</b>       | 0x75  | F6 key      |
| <b>vbKeyG</b> | 71    | G key       | <b>vbKey6</b>        | 54    | 6 key        | <b>vbKeyF7</b>       | 0x76  | F7 key      |
| <b>vbKeyH</b> | 72    | H key       | <b>vbKey7</b>        | 55    | 7 key        | <b>vbKeyF8</b>       | 0x77  | F8 key      |
| <b>vbKeyI</b> | 73    | I key       | <b>vbKey8</b>        | 56    | 8 key        | <b>vbKeyF9</b>       | 0x78  | F9 key      |
| <b>vbKeyJ</b> | 74    | J key       | <b>vbKey9</b>        | 57    | 9 key        | <b>vbKeyF10</b>      | 0x79  | F10 key     |
| <b>vbKeyK</b> | 75    | K key       | <b>vbKeyLButton</b>  | 0x1   | Left mouse   | <b>vbKeyF11</b>      | 0x7A  | F11 key     |
| <b>vbKeyL</b> | 76    | L key       | <b>vbKeyRButton</b>  | 0x2   | Right mouse  | <b>vbKeyF12</b>      | 0x7B  | F12 key     |
| <b>vbKeyM</b> | 77    | M key       | <b>vbKeyCancel</b>   | 0x3   | CANCEL key   | <b>vbKeyF13</b>      | 0x7C  | F13 key     |
| <b>vbKeyN</b> | 78    | N key       | <b>vbKeyMButton</b>  | 0x4   | Middle mouse | <b>vbKeyF14</b>      | 0x7D  | F14 key     |
| <b>vbKeyO</b> | 79    | O key       | <b>vbKeyBack</b>     | 0x8   | BACKSPACE    | <b>vbKeyF15</b>      | 0x7E  | F15 key     |
| <b>vbKeyP</b> | 80    | P key       | <b>vbKeyTab</b>      | 0x9   | TAB key      | <b>vbKeyF16</b>      | 0x7F  | F16 key     |
| <b>vbKeyQ</b> | 81    | Q key       | <b>vbKeyClear</b>    | 0xC   | CLEAR key    | <b>vbKeyEnd</b>      | 0x23  | END key     |
| <b>vbKeyR</b> | 82    | R key       | <b>vbKeyReturn</b>   | 0xD   | ENTER key    | <b>vbKeyHome</b>     | 0x24  | HOME key    |
| <b>vbKeyS</b> | 83    | S key       | <b>vbKeyShift</b>    | 0x10  | SHIFT key    | <b>vbKeyLeft</b>     | 0x25  | LEFT        |
| <b>vbKeyT</b> | 84    | T key       | <b>vbKeyControl</b>  | 0x11  | CTRL key     | <b>vbKeyUp</b>       | 0x26  | UP          |
| <b>vbKeyU</b> | 85    | U key       | <b>vbKeyMenu</b>     | 0x12  | MENU key     | <b>vbKeyRight</b>    | 0x27  | RIGHT       |
| <b>vbKeyV</b> | 86    | V key       | <b>vbKeyPause</b>    | 0x13  | PAUSE key    | <b>vbKeyDown</b>     | 0x28  | DOWN        |
| <b>vbKeyW</b> | 87    | W key       | <b>vbKeyCapital</b>  | 0x14  | CAPS LOCK    | <b>vbKeySelect</b>   | 0x29  | SELECT      |
| <b>vbKeyX</b> | 88    | X key       | <b>vbKeyEscape</b>   | 0x1B  | ESC key      | <b>vbKeyPrint</b>    | 0x2A  | PRT SCR     |
| <b>vbKeyY</b> | 89    | Y key       | <b>vbKeySpace</b>    | 0x20  | SPACEBAR     | <b>vbKeyExecute</b>  | 0x2B  | EXECUTE     |
| <b>vbKeyZ</b> | 90    | Z key       | <b>vbKeyPageUp</b>   | 0x21  | PAGE Up      | <b>vbKeySnapshot</b> | 0x2C  | SNAPSHOT    |
|               |       |             | <b>vbKeyPageDown</b> | 0x22  | PAGE Down    | <b>vbKeyInsert</b>   | 0x2D  | INSERT      |
|               |       |             |                      |       |              | <b>vbKeyDelete</b>   | 0x2E  | DELETE      |
|               |       |             |                      |       |              | <b>vbKeyHelp</b>     | 0x2F  | HELP key    |
|               |       |             |                      |       |              | <b>vbKeyNumlock</b>  | 0x90  | Num Lock    |

| Constant              | Value | Description       |
|-----------------------|-------|-------------------|
| <b>vbKeyNumpad0</b>   | 0x60  | 0 key             |
| <b>vbKeyNumpad1</b>   | 0x61  | 1 key             |
| <b>vbKeyNumpad2</b>   | 0x62  | 2 key             |
| <b>vbKeyNumpad3</b>   | 0x63  | 3 key             |
| <b>vbKeyNumpad4</b>   | 0x64  | 4 key             |
| <b>vbKeyNumpad5</b>   | 0x65  | 5 key             |
| <b>vbKeyNumpad6</b>   | 0x66  | 6 key             |
| <b>vbKeyNumpad7</b>   | 0x67  | 7 key             |
| <b>vbKeyNumpad8</b>   | 0x68  | 8 key             |
| <b>vbKeyNumpad9</b>   | 0x69  | 9 key             |
| <b>vbKeyMultiply</b>  | 0x6A  | MULT. SIGN (*)    |
| <b>vbKeyAdd</b>       | 0x6B  | PLUS SIGN (+)     |
| <b>vbKeySeparator</b> | 0x6C  | ENTER key         |
| <b>vbKeySubtract</b>  | 0x6D  | MINUS SIGN        |
| <b>vbKeyDecimal</b>   | 0x6E  | DECIMAL POINT     |
| <b>vbKeyDivide</b>    | 0x6F  | DIVISION SIGN (/) |

This table lists the VB key code constants for the numeric keypad. Notice that the constant names are different from the names of corresponding keys elsewhere on the keyboard.

**e.g. vbKeyReturn** → ENTER key near SHIFT key

**vbKeySeparator** → ENTER key on numeric keypad

Notice that some of the key code values are written as ordinary *decimal* (base 10) values while others are written as *hexadecimal* (base 16) values. The hexadecimal values are preceded by the “0x” prefix. Details on the hexadecimal system will be given in class.

# USING MESSAGE BOXES IN VB PROGRAMS

The **MsgBox** function displays a message in a dialogue box, waits for the user to click a button and returns an **Integer** indicating which button the user clicked.

The **square brackets** mean that the argument is *optional*.

### Syntax

**MsgBox** (*prompt* [, *buttons*] [, *title*] [, *helpfile*, *context*])

The **MsgBox** function syntax has these named arguments:

| Part            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>prompt</i>   | <b>Required.</b> String expression displayed as the message in the dialogue box. The maximum length of <i>prompt</i> is approximately 1024 characters, depending on the width of the characters used. If <i>prompt</i> consists of more than one line, you can separate the lines using a carriage return character ( <b>Chr</b> (13)), a linefeed character ( <b>Chr</b> (10)) or carriage return – linefeed character combination ( <b>Chr</b> (13) & <b>Chr</b> (10)) between each line. |
| <i>buttons</i>  | <b>Optional.</b> Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If omitted, the default value for <i>buttons</i> is 0.                                                                                                                                                                                                                  |
| <i>title</i>    | <b>Optional.</b> String expression displayed in the title bar of the dialogue box. If you omit <i>title</i> , the application name is placed in the title bar.                                                                                                                                                                                                                                                                                                                              |
| <i>helpfile</i> | <b>Optional.</b> String expression that identifies the Help file to use to provide context-sensitive Help for the dialogue box. If <i>helpfile</i> is provided, <i>context</i> must also be provided.                                                                                                                                                                                                                                                                                       |
| <i>context</i>  | <b>Optional.</b> Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If <i>context</i> is provided, <i>helpfile</i> must also be provided.                                                                                                                                                                                                                                                                                        |

### Settings

The *buttons* argument settings are:

| Constant                  | Value | Description                                                    | Constant                     | Value   | Description                                                                                                    |
|---------------------------|-------|----------------------------------------------------------------|------------------------------|---------|----------------------------------------------------------------------------------------------------------------|
| <b>vbOKOnly</b>           | 0     | Display <b>OK</b> button only.                                 | <b>vbDefaultButton1</b>      | 0       | First button is default.                                                                                       |
| <b>vbOKCancel</b>         | 1     | Display <b>OK</b> and <b>Cancel</b> buttons.                   | <b>vbDefaultButton2</b>      | 256     | Second button is default.                                                                                      |
| <b>vbAbortRetryIgnore</b> | 2     | Display <b>Abort</b> , <b>Retry</b> and <b>Ignore</b> buttons. | <b>vbDefaultButton3</b>      | 512     | Third button is default.                                                                                       |
| <b>vbYesNoCancel</b>      | 3     | Display <b>Yes</b> , <b>No</b> , and <b>Cancel</b> buttons.    | <b>vbDefaultButton4</b>      | 768     | Fourth button is default.                                                                                      |
| <b>vbYesNo</b>            | 4     | Display <b>Yes</b> and <b>No</b> buttons.                      | <b>vbApplicationModal</b>    | 0       | Application modal; the user must respond to the message box before continuing work in the current application. |
| <b>vbRetryCancel</b>      | 5     | Display <b>Retry</b> and <b>Cancel</b> buttons.                | <b>vbSystemModal</b>         | 4096    | System modal; all applications are suspended until the user responds to the message box.                       |
| <b>vbCritical</b>         | 16    | Display <b>Critical Message</b> icon.                          | <b>vbMsgBoxHelpButton</b>    | 16384   | Adds Help button to the message box                                                                            |
| <b>vbQuestion</b>         | 32    | Display <b>Warning Query</b> icon.                             | <b>VbMsgBoxSetForeground</b> | 65536   | Specifies the message box window as the foreground window                                                      |
| <b>vbExclamation</b>      | 48    | Display <b>Warning Message</b> icon.                           | <b>vbMsgBoxRight</b>         | 524288  | Text is right aligned                                                                                          |
| <b>vbInformation</b>      | 64    | Display <b>Information Message</b> icon.                       | <b>vbMsgBoxRtlReading</b>    | 1048576 | Specifies text should appear as right-to-left reading on Hebrew and Arabic systems                             |

The **first group of values (0–5)** describes the *number* and *type* of buttons displayed in the dialogue box. The **second group (16, 32, 48, 64)** describes the *icon style*; the **third group (0, 256, 512, 768)** determines *which button is the default*; and the **fourth group (0, 4096)** determines the *modality of the message box*. When **adding numbers to create a final value** for the *buttons* argument, use **only one number from each group**.

## Return Values

| Constant | Value | Description |
|----------|-------|-------------|
| vbOK     | 1     | OK          |
| vbCancel | 2     | Cancel      |
| vbAbort  | 3     | Abort       |
| vbRetry  | 4     | Retry       |
| vbIgnore | 5     | Ignore      |
| vbYes    | 6     | Yes         |
| vbNo     | 7     | No          |

### Remarks

When both *helpfile* and *context* are provided, the user can press F1 to view the Help topic corresponding to the *context*. Some host applications, for example, Microsoft Excel, also automatically add a **Help** button to the dialogue box.

If the dialogue box displays a **Cancel** button, pressing the **ESC** key has the same effect as clicking **Cancel**. If the dialogue box contains a **Help** button, context-sensitive Help is provided for the dialogue box. However, no value is returned until one of the other buttons is clicked.

### Note

To specify more than the first named argument, you must use **MsgBox** in an expression. To omit some positional arguments, you must include the corresponding comma delimiter.

In this example, **MsgBox** is **not** being used to obtain a response from the user. It is simply used to make the user aware of something (usually a problem).

In this case, parentheses are **not** used and there is no assignment statement.

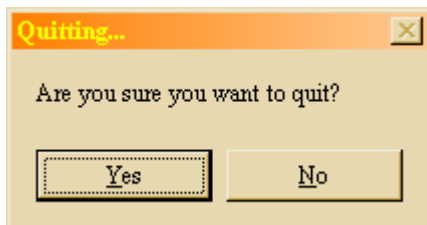
## Examples

'Code for Quit button

```
Private Sub cmdQuit_Click()
    'VbMsgBoxResult is a special data type used to
    'store responses obtained from message boxes.
    Dim Response As VbMsgBoxResult
    Response = MsgBox("Are you sure you want to _
        quit?", vbYesNo, "Quitting...")
    If Response = vbYes Then
        Unload Me
    End If
End Sub
```

In this example, **MsgBox** is being used as a **function**. **Parentheses** are used to contain the **arguments** (“inputs”) of the function and the value **returned** by the function is **assigned** to a variable of type **VbMsgBoxResult**.

A **MsgBox** is used this way whenever we want to obtain a response from the user.



### Questions

1. In the above example, what caused the “Yes” and “No” buttons to appear on the message box?
2. Although **MsgBox** returns an **Integer**, why is it helpful to declare the variable “Response” as type **vbMsgBoxResult**?

'Code for Solve button

```
Private Sub cmdSolve_Click()
    'MEMORY
    Dim Base As Double, Height As Double, _
        Hypotenuse As Double
    'INPUT
    Height = Val(txtHeight.Text)
    Base = Val(txtBase.Text)
    Hypotenuse = Val(txtHypotenuse.Text)
    'PROCESSING and OUTPUT
    If Hypotenuse = 0 Then
        Hypotenuse = Sqr(Base ^ 2 + Height ^ 2)
        txtHypotenuse.Text = Str(Hypotenuse)
    ElseIf Base = 0 Then
        Base = Sqr(Hypotenuse ^ 2 - Height ^ 2)
        txtBase.Text = Str(Base)
    ElseIf Height = 0 Then
        Height = Sqr(Hypotenuse ^ 2 - Base ^ 2)
        txtHeight.Text = Str(Height)
    Else
        ' Errors in the data entered by the user.
        MsgBox "You have entered invalid data!", _
            vbExclamation, "There is a problem!"
    End If
End Sub
```



### More Questions

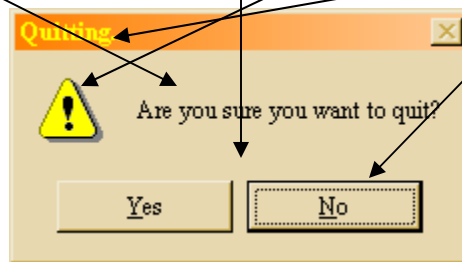
1. In the above example, what caused the exclamation mark icon to appear on the message box?
2. How would you obtain the “X” icon in the following message box?



## Another Example

The message box shown below can be created by using the following code:

```
Response = MsgBox("Are you sure you want to quit?", vbYesNo + vbExclamation + vbDefaultButton2, "Quitting...")
```



## Note

- **vbYesNo** causes the "Yes" and "No" buttons to appear
- **vbExclamation** causes the exclamation mark icon to appear
- **vbDefaultButton2** causes button #2 ("No") to be the default button (as soon as the message box is displayed, the "No" button has the focus → this is a good idea in quitting message boxes because it prevents the user from accidentally quitting by pressing ENTER)
- You may combine up to five different options for the second argument of **MsgBox** by picking **one** constant from each group in the table shown above and entering the **sum** of the constants as the second argument.

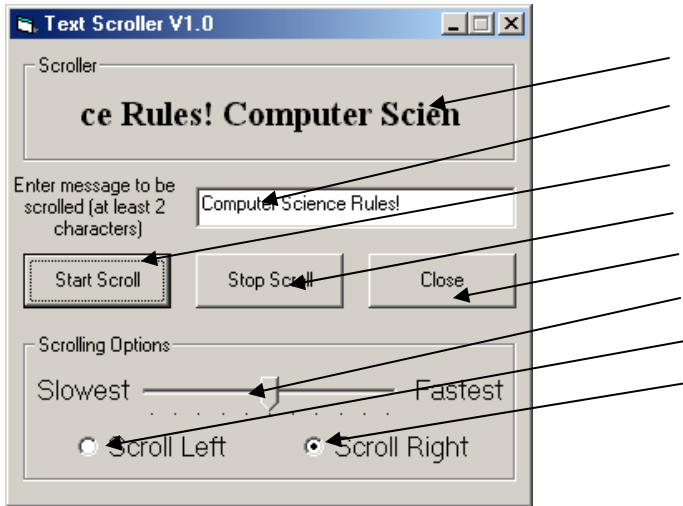
## Exercise

Write a VB statement that can be used to generate the message box shown below.



# LEARNING NEW PROGRAMMING CONCEPTS BY STUDYING EXAMPLES

1. Study the code for the "Text Scroller" program to determine the name of each object on the form shown below.



```

*****
'* PROGRAMMER'S NAME: Nick E. Nolfi
'* VERSION: Text Scroller V1.0
'*
'* DESCRIPTION: This program scrolls a message in a label box. The speed of the scroll can be controlled*
'* by a "Slider" control and the direction of the scroll is selected by using an option button.
'*
'* LIMITATIONS (BUGS): There are a few minor bugs.
'*
'* PURPOSE OF THIS PROGRAM: To introduce several new concepts
'* -> String manipulation using the "Left" and "Right" intrinsic functions
'* -> Slider control
'* -> Timer control
'* -> Using the "Const" keyword to declare constants (constant identifiers)
'* -> Changing the mouse pointer when "hovering" over an object
'* -> Frames
*****

```

## Option Explicit

```

Const ChangeInInterval = -50, MaxInterval = 700
' Initialize timer interval and slider setting
Private Sub Form_Load()
    sldScrollerSpeed.Value = 5
    tmrScrollTimer.Interval = ChangeInInterval * sldScrollerSpeed.Value + MaxInterval
End Sub
' When the START button is clicked, the message entered by the user is scrolled in the "lblMessage" label box.
' If the message is less than two characters in length, an error message is displayed.
Private Sub cmdStartScroll_Click()
    Dim Response As VbMsgBoxResult
    If Len(txtScrollMessage.Text) >= 2 Then
        lblScroller.Caption = txtScrollMessage.Text
        tmrScrollTimer.Enabled = True
    Else
        Response = MsgBox("Your message is too short. Do you understand?", vbYesNo, "There is a problem!")
        If Response = vbYes Then
            MsgBox "Great Stuff! You can count to two!", vbInformation, "Congratulations!"
        ElseIf Response = vbNo Then
            MsgBox "Do you know how to count to two?", vbCritical, "Oh brother!"
        End If
    End If
End Sub
' Disable timer to stop scroll
Private Sub cmdStopScroll_Click()
    tmrScrollTimer.Enabled = False
End Sub

```



```

' Close program
Private Sub cmdClose_Click()
    Unload frmScroller
End
End Sub
' Change scroller speed
Private Sub sldScrollerSpeed_Change()
    tmrScrollTimer.Interval = ChangeInInterval * sldScrollerSpeed.Value + MaxInterval
End Sub
' Scroll the message left or right depending on which option button is selected. This sub procedure is called
' every time the timer interval elapses (i.e. whenever the Timer event is generated).
Private Sub tmrScrollTimer_Timer()
    If optScrollRight.Value = True Then 'Scroll right
        lblScroller.Caption = Right(lblScroller.Caption, 1) & _
            Left(lblScroller.Caption, Len(lblScroller.Caption) - 1)
    Else 'Scroll left
        lblScroller.Caption = Right(lblScroller.Caption, Len(lblScroller.Caption) - 1) & _
            Left(lblScroller.Caption, 1)
    End If
End Sub

```

2. Before you answer this question, please ensure that you familiarize yourself with the “Text Scroller” program by loading it into Visual Basic from **I:\Out\Nolfi\Tik2o0**. Run the program and carefully observe what happens when the “Slider” control is used to change the scroll speed. In addition, study the objects on the form and their properties. Then complete the following table.

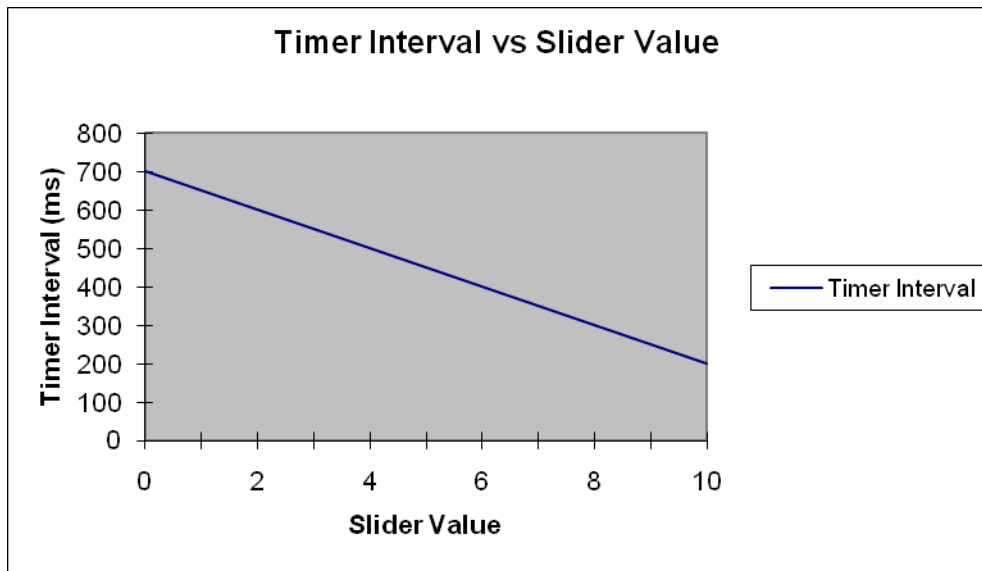
| tmrScrollTimer       |        | sldScrollerSpeed     |        | txtScrollMessage     |        | lblScroller          |        | frmScroller          |        |
|----------------------|--------|----------------------|--------|----------------------|--------|----------------------|--------|----------------------|--------|
| Important Properties | Values | Important Properties | Values | Important Properties | Values | Important Properties | Values | Important Properties | Values |
| Enabled              |        | LargeChange          |        | Alignment            |        | Alignment            |        | BorderStyle          |        |
| Interval             |        | Max                  |        | MouseIcon            |        | MouseIcon            |        | Caption              |        |
|                      |        | Min                  |        | MousePointer         |        | MousePointer         |        | MaxButton            |        |
|                      |        | MouseIcon            |        | ToolTipText          |        | ToolTipText          |        | MinButton            |        |
|                      |        | MousePointer         |        |                      |        | WordWrap             |        | WindowState          |        |
|                      |        | SmallChange          |        |                      |        |                      |        |                      |        |
|                      |        | ToolTipText          |        |                      |        |                      |        |                      |        |
|                      |        | Value                |        |                      |        |                      |        |                      |        |

3. Now that you have carefully studied the “Text Scroller” program, answer the following questions.

- (a) Why does the mouse pointer change to a “hand shape” when it is moved over the slider control?
- (b) Why does the text “Adjust Scroll Speed” appear when the mouse pointer is paused over the slider control?
- (c) Why does a “happy face” icon appear in the top left hand corner of the *frmScroller* form?
- (d) Why is it not possible to change the size of the *frmScroller* form? Why is the maximize button disabled?

4. The purpose of this question is to help you understand the formula that is used to change the speed of the “scroller.” The change in speed is based on a simple linear (“straight line”) equation. Find an equation of the line shown below and then compare your equation to the formula used in the program.

**Hint:** Let  $T$  represent the timer interval, let  $V$  represent the slider value and use the  $y = mx + b$  form of the equation of a straight line. Recall that “ $m$ ” represents the slope of the line and “ $b$ ” represents the  $y$ -intercept.



5. Study the technical descriptions of the “Left” and “Right” functions given below. Then explain how these two functions are used in the “scroller” program.

### Left Function

Returns a **Variant (String)** containing a specified number of characters from the left side of a string.

#### Syntax

**Left(string, length)**

The **Left** function syntax has these named arguments:

| Part          | Description                                                                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>string</i> | Required. String expression from which the leftmost characters are returned. If <i>string</i> contains Null, Null is returned.                                                                                                                     |
| <i>length</i> | Required. <b>Variant (Long)</b> . Numeric expression indicating how many characters to return. If 0, a zero-length string (“”) is returned. If greater than or equal to the number of characters in <i>string</i> , the entire string is returned. |

### Right Function

Returns a **Variant (String)** containing a specified number of characters from the right side of a string.

#### Syntax

**Right(string, length)**

The **Right** function syntax has these named arguments:

| Part          | Description                                                                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>string</i> | Required. String expression from which the rightmost characters are returned. If <i>string</i> contains Null, Null is returned.                                                                                                                    |
| <i>length</i> | Required. <b>Variant (Long)</b> . Numeric expression indicating how many characters to return. If 0, a zero-length string (“”) is returned. If greater than or equal to the number of characters in <i>string</i> , the entire string is returned. |

## OBJECT NAMING CONVENTIONS

Objects should be named with a consistent prefix that makes it easy to identify the type of object. Recommended conventions for some of the objects supported by Visual Basic are listed below.

### *Suggested Prefixes for Controls*

| <i>Control Type</i>                                                | <i>Prefix</i> | <i>Example</i>   | <i>Control Type</i>               | <i>Prefix</i> | <i>Example</i>   |
|--------------------------------------------------------------------|---------------|------------------|-----------------------------------|---------------|------------------|
| 3D Panel                                                           | pnl           | pnlGroup         | Gauge                             | gau           | gauStatus        |
| ADO Data                                                           | ado           | adoBiblio        | Graph                             | gra           | graRevenue       |
| Animated button                                                    | ani           | aniMailBox       | Grid                              | grd           | grdPrices        |
| <b>Check box</b>                                                   | chk           | chkReadOnly      | Hierarchical flexgrid             | flex          | flexOrders       |
| <b>Combo box</b><br>(drop-down list box)                           | cbo           | cboEnglish       | Horizontal scroll bar             | hsb           | hsbVolume        |
| <b>Command button</b>                                              | cmd           | cmdExit          | <b>Image</b>                      | img           | imgIcon          |
| Common dialog                                                      | dlg           | dlgFileOpen      | Image combo                       | imgcbo        | imgcboProduct    |
| Communications                                                     | com           | comFax           | ImageList                         | ils           | ilsAllIcons      |
| Control (used within procedures when the specific type is unknown) | ctr           | ctrCurrent       | <b>Label</b>                      | lbl           | lblHelpMessage   |
| Data                                                               | dat           | datBiblio        | Lightweight check box             | lwchk         | lwchkArchive     |
| Data-bound combo box                                               | dbcbo         | dbcboLanguage    | Lightweight combo box             | lwco          | lwcoGerman       |
| Data-bound grid                                                    | dbgrd         | dbgrdQueryResult | Lightweight command button        | lwcmb         | lwcmbRemove      |
| Data-bound list box                                                | dblst         | dblstJobType     | Lightweight frame                 | lwfra         | lwfraSaveOptions |
| Data combo                                                         | dbc           | dbcAuthor        | Lightweight horizontal scroll bar | lwhsb         | lwhsbVolume      |
| Data grid                                                          | dgd           | dgdTitles        | Lightweight list box              | lwlst         | lwlstCostCenters |
| Data list                                                          | dbl           | dblPublisher     | Lightweight option button         | lwopt         | lwoptIncomeLevel |
| Data repeater                                                      | drp           | drpLocation      | Lightweight text box              | lwtxt         | lwoptStreet      |
| Date picker                                                        | dtp           | dtpPublished     | Lightweight vertical scroll bar   | lwvsb         | lwvsbYear        |
| Directory list box                                                 | dir           | dirSource        | <b>Line</b>                       | lin           | linVertical      |
| Drive list box                                                     | drv           | drvTarget        | <b>List box</b>                   | lst           | lstPolicyCodes   |
| File list box                                                      | fil           | filSource        | List View                         | lvw           | lvwHeadings      |
| Flat scroll bar                                                    | fsb           | fsbMove          | MAPI message                      | mpm           | mpmSendMessage   |
| <b>Form</b>                                                        | frm           | frmEntry         | MAPI session                      | mps           | mpsSession       |
| <b>Frame</b>                                                       | fra           | fraLanguage      | MCI                               | mci           | mciVideo         |

*Suggested Prefixes for Controls (Continued from previous page)*

| <i>Control Type</i>  | <i>Prefix</i> | <i>Example</i>  | <i>Control Type</i> | <i>Prefix</i> | <i>Example</i>  |
|----------------------|---------------|-----------------|---------------------|---------------|-----------------|
| <b>Menu</b>          | mnu           | mnuFileOpen     | Slider              | sld           | sldScale        |
| Month view           | mvw           | mvwPeriod       | Spin                | spn           | spnPages        |
| MS Chart             | ch            | chSalesbyRegion | StatusBar           | sta           | staDateTime     |
| MS Flex grid         | msg           | msgClients      | SysInfo             | sys           | sysMonitor      |
| OLE container        | ole           | oleWorksheet    | TabStrip            | tab           | tabOptions      |
| <b>Option button</b> | opt           | optGender       | <b>Text box</b>     | txt           | txtLastName     |
| <b>Picture box</b>   | pic           | picVGA          | <b>Timer</b>        | tmr           | tmrAlarm        |
| Picture clip         | clp           | clpToolbar      | Toolbar             | tlb           | tlbActions      |
| ProgressBar          | prg           | prgLoadFile     | TreeView            | tre           | treOrganization |
| Remote Data          | rd            | rdTitles        | UpDown              | upd           | updDirection    |
| RichTextBox          | rtf           | rtfReport       | Vertical scroll bar | vsb           | vsbRate         |
| Shape                | shp           | shpCircle       |                     |               |                 |

*Suggested Prefixes for Data Access Objects (DAO)*

Use the following prefixes to indicate Data Access Objects.

| <i>Control Type</i> | <i>Prefix</i> | <i>Example</i> | <i>Control Type</i> | <i>Prefix</i> | <i>Example</i>   |
|---------------------|---------------|----------------|---------------------|---------------|------------------|
| Container           | con           | conReports     | Parameter           | prm           | prmJobCode       |
| Database            | db            | dbAccounts     | QueryDef            | qry           | qrySalesByRegion |
| DBEngine            | dbe           | dbeJet         | Recordset           | rec           | recForecast      |
| Document            | doc           | docSalesReport | Relation            | rel           | relEmployeeDept  |
| Field               | fld           | fldAddress     | TableDef            | tbd           | tbdCustomers     |
| Group               | grp           | grpFinance     | User                | usr           | usrNew           |
| Index               | ix            | idxAge         | Workspace           | wsp           | wspMine          |

*Examples*

**Dim** dbBiblio **As** Database

**Dim** recPubsInNY **As** Recordset, strSQLStmt **As** String

**Const** DB\_READONLY = 4

'Open database.

**Set** dbBiblio = OpenDatabase("BIBLIO.MDB")

' Set text for the SQL statement.

strSQLStmt = "SELECT \* FROM Publishers WHERE State = 'NY'"

' Create the new Recordset object.

**Set** recPubsInNY = db.OpenRecordset(strSQLStmt, dbReadOnly)

### *Suggested Prefixes for Menus*

Applications frequently use many menu controls, making it useful to have a unique set of naming conventions for these controls. Menu control prefixes should be extended beyond the initial "mnu" label by adding an additional prefix for each level of nesting, with the final menu caption at the end of the name string. The following table lists some examples.

| <i>Menu caption sequence</i> | <i>Menu handler name</i> |
|------------------------------|--------------------------|
| File Open                    | mnuFileOpen              |
| File Send Email              | mnuFileSendEmail         |
| File Send Fax                | mnuFileSendFax           |
| Format Character             | mnuFormatCharacter       |
| Help Contents                | mnuHelpContents          |

When this naming convention is used, all members of a particular menu group are listed next to each other in Visual Basic's Properties window. In addition, the menu control names clearly document the menu items to which they are attached.

### *Choosing Prefixes for Other Controls*

For controls not listed above, you should try to standardize on a unique two or three character prefix for consistency. Use more than three characters only if needed for clarity. For derived or modified controls, for example, extend the prefixes above so that there is no confusion over which control is really being used. For third-party controls, a lower-case abbreviation for the manufacturer could be added to the prefix. For example, a control instance created from the Visual Basic Professional 3D frame could use a prefix of fra3d to avoid confusion over which control is really being used.