Introduction
Introduction to Programming Appreciation: VBSystemApp (VBSA)

CHAPTER 1
SPECIAL EFFECTS!
Lab1Practice1 (MovingCar)
Lab1Practice2 (FigureEights)
Lab1Practice3 (RandomFlipSpinRoll)
Lab1Credit (RandomSpinRollTravel)

CHAPTER 2
BOXES AND CALCULATIONS!
Lab2Practice1 (TriplePlay)
Lab2Practice2 (Prizes)
Lab2Practice3 (Race)
Lab2Credit (CableTV)

CHAPTER 3
LISTS AND NESTING
Lab3Practice1 (TimesTable1)
Lab3Practice2 (TimesTable2)
Lab3Practice3 (WinningStreak)
Lab3Credit (RandomMotion)
INTRODUCTION TO PROGRAMMING APPRECIATION

VBSystemApp (VBSA)

We use the VBSystemApp ©Dr. James D. Fabrey to appreciate how computer programmers instruct the computer to perform specified tasks. Perhaps this could be compared with art appreciation – how professional artists do their work! Anyway, we will use this app to present a simplified approach based on the Visual Basic Express programming language (©Microsoft Corporation). Henceforth, for brevity, we will refer to VBSystemApp as VBSA. It should be noted that VBSA is a significantly revised version of its predecessor VBDender. With VBDender, students designed their programs (or projects) and then used Visual Basic Express to write the instructions (or code) to go with the design, as well as to run these programs. Now, students can do all three of these phases of each VBSA project entirely with VBSA.

Students should set up a folder for the course on their flash drive or mobile hard drive (or even cloud drive), and then download VBSA from their D2L account. Here are the terms that we are using so far.

<table>
<thead>
<tr>
<th>TERMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>We will use the terms program and project interchangeably. Typically, the word project is used in naming files and folders that we will be using.</td>
</tr>
<tr>
<td>We will also use the terms window and form interchangeably. The term form is really short for business form. We will learn how to design the objects in the form.</td>
</tr>
<tr>
<td>We will use the terms instructions and code interchangeably. We will learn how to write the code that makes the project operate properly.</td>
</tr>
</tbody>
</table>

When you download VBSA and run it, here is the window that you will get.
It starts out in Design mode, as indicated in the Southwest panel of the window, which is called the ToolBox. The Code mode and Run mode can also be selected here.

The Project window to be designed is the initially blank Northwest panel.

The VBSA Guide is in the Southeast panel. You can right-click here to toggle back and forth between having the small version of the VBSA Guide here and the big version using the entire VBSA window. TRY IT NOW!

The Properties window for the objects that you select from the ToolBox is the initially blank Northeast panel.

Now try clicking on ALL of the objects in the ToolBox to see them moved to the Project window.

You can click on an object in the Project window and click on the Remove button in the Properties window to put it back into the ToolBox. TRY IT NOW – FOR A FEW OBJECTS!

You can get a completely fresh start by RIGHT-CLICKING on the NEW/OPEN/SAVE/EXIT area in the upper right-hand corner of the ToolBox and selecting NEW. TRY THIS!
You get the same screen that you had when you first ran VBSA. The next step is to put one object of each kind into your project window.

Now try changing the properties of these objects. First, for each object, click on it and then the Left box and type 200. This will line up all of the objects 200 pixels from the Left edge of the window. To space these objects apart better, change the Top value of some of the objects – this is the number of the pixels from the Top edge of the window. Leave room for the ListBox to expand by moving the line down a bit. Note that you can keep entering different values into the same property for the same object to adjust exactly the way that you want.
You can move these objects more quickly using the arrow buttons rather than typing numbers into the Properties window (this can be done to fine-tune the position to an exact pixel location). TRY THIS!

You can also change the Height and Width (in pixels) for these objects. This is frequently done if you enter text that requires more room. You cannot enter text for lines or pictures, but you can enter text for TextBoxes (sometimes called Boxes for short because they can contain text or numbers), Buttons, and ListBoxes. Try entering the word VALUE in place of Box1 and the words CLEAR VALUE in place of Button1. Text appears outside the box and on top of the button. You will have to make the Width of the button larger in order to have its entire text display. Clicking on the form itself will give the following screen:
NOTE: You are not allowed to enter duplicate names or keywords (such as END, WAIT, etc. (you will eventually learn about all of them)).

What about text for the ListBox? Unlike a regular Box, a ListBox is intended for a list of items, such as DOG,CAT,BIRD,TURTLE,HAMSTER. In fact, that is how you enter its text: type the items, separated by commas in the Text property for the ListBox. To make it official, press enter at the end of the list! Try it now! Notice that not all of the list displays at once, but that a vertical scroll bar magically appears and can be used to scroll through the items. If you make the Height of the ListBox large enough, all of the list displays and the scroll bar magically disappears.

We have skipped by the COLORS properties. After selecting an object, you can scroll through the BACK (background) and FORE (foreground) colors to make your selections. Obviously, this does not apply to Pictures, and it will not let you choose a FORE color for a Line. Also, for a Box, the FORE color applies to the labeled text next to it, while the BACK color applies to the interior of the Box. The default of Black on Silver for a Button is very good, but here is an illustration of a number of color changes.
WHAT ABOUT THE PROJECT WINDOW (OR FORM) ITSELF? This is actually considered an object in the project. You can select the form by clicking anywhere inside it – except for any of the objects that you have put inside it. There are two properties that you can change: the Text and the BACK color. The Text for the form is the wording on the Title Bar, which is the Top edge of the window. Here is an example of these changes.

Now let's try to Save our design with a Project folder name of our choosing, using the MAKE NEW FOLDER button if needed, clear our project Window with New, and then Open it back again using the Project folder name. Again, we will do this by RIGHT-CLICKING on the NEW/OPEN/SAVE/EXIT area for each step.

This has been a sneak preview of the DESIGN mode, which is the first choice. If we try the third mode (RUN), nothing will happen! The reason is that we have not put any instructions into our project for the computer to execute! For example, if we click on a button, we normally expect something to happen but we have not instructed the computer what to do in this event. These instructions are called code, and we need to work with the second mode (CODE) before going to the RUN mode.

The CODE mode is the heart of computer programming, and we will give a wide variety of examples throughout the guide to help you to understand some of the basics. In each of three chapters, we will have at least two practice projects and one project to complete for credit.

NOTE: there is no EXIT button for VBSA. Instead, to exit VBSA, you need to click on the RED X button in the Northeast Corner of the VBSA window:
CHAPTER 1

SPECIAL EFFECTS!

(Practice Projects: MovingCar, FigureEights, RandomFlipSpinRoll)

(Credit Project: RandomSpinRollTravel)

or back to Contents

Lab1Practice1: MovingCar

Your instructor will use VBSA to open the MovingCar demo. Here are two screens to illustrate the Design Mode:

Button1 Properties (Note: we name the button VELOCITY instead of SPEED because we are not allowed to use keywords such as SPEED)

Button2 Properties (Note: we name the button EXIT instead of END because we are not allowed to use keywords such as END)
Your instructor will now go into Run mode and demonstrate this project. Note that the VBSA Guide now expands to use the Southeast and Northeast panels (half of the VBSA window) during Run mode. Clicking on the picture changes it to a car and makes it travel counter-clockwise.

Finally, your instructor will open the Code mode and explain the code or instructions that make the project do what it does! Here are three screens to illustrate the Code Mode:

VELOCITY CODE
EXIT CODE

PICTURE1 CODE
Notice that the Southwest panel, which was used for the ToolBox in the Design mode, is now used as the Code Build window in the Code mode.

Notice that the Northeast panel, which was used for Properties in the Design mode, is now used as the Code Display window in the Code mode.

Here are the complete code listings:

VELOCITY CODE

SPEED

Explanation:

The SPEED instruction causes an InputBox to appear:

![InputBox image]

This is used to change the speed of all special effects; for this project, it is the speed of the moving car. It requests that you enter a whole number (integer) between 1 and 999 to set the length in milliseconds of the interval between ticks of the hidden Special Effects clock. The smaller this number, the higher the speed. You can try 100 milliseconds (1/10 second) and then try higher and lower values to experiment a little bit!

EXIT CODE

END
Explanation:

This ends the running of the project and returns to the Design mode, still within VBSA. Reminder: to exit VBSA, you can click on the RED X button in the Northeast Corner of the VBSA window:

![Image of X button]

PICTURE1 CODE

```vbsa
' Lab1Practice1MovingCar
Picture1.Image = CarRight
Picture1.Width = 75
Picture1.Height = 50
travel East Picture1 250

Picture1.Image = CarUp
Picture1.Width = 50
Picture1.Height = 75
travel North Picture1 250

Picture1.Image = CarLeft
Picture1.Width = 75
Picture1.Height = 50
travel West Picture1 250

Picture1.Image = CarDown
Picture1.Width = 50
Picture1.Height = 75
travel South Picture1 250
```

Explanation:

The car travels around the block – East, North, West, South (counter-clockwise). For each of these four steps, the image of the car is changed to have it pointing in the right direction, and the width and height are changed to give it the right dimensions. The travel instruction specifies the direction, the picture containing the image, and the number of pixels to be traveled (250 in this example):

```
travel direction picture pixels
```

The first line is not executed by the computer at all. It is called a Comment and is used to explain the project. In the Code Build window, select the Comment keyword on the left, and then an InputBox will appear for you to enter the Comment. When you click the ADD button, the comment will be placed in the Code Display, with a single quote ‘ inserted at the beginning. The blank lines are used for readability only.
and are not executed either. They are inserted by left-clicking a row in the Code Display to select it and then right-clicking to insert a blank row above it.

This project is just for practice, not for credit, so it is your turn to design, code, and run the project yourself! Your instructor will demonstrate each of these steps and you should follow along on your computer. Try different speeds!

- In the Design mode, set up your button and picture – with their properties
- Go to Code mode, then select an object that needs code, and then use the Code window. You DO NOT type the instruction!!! Instead, you use between 1 and 6 lists to put together the components of an instruction and then click the ADD button to add the instruction. As needed, you can edit the instructions by selecting them with a left click and editing them with a right click.
- Try the Run mode, just to make sure you have done everything right!

When everything is OK, you should SAVE your project. Then, in the future, you can OPEN it whenever you want to do so. These commands are located in the same area as the NEW command described previously:

Notice that you can select your flash drive and perhaps a folder that you have set up for this course – and then make a new folder with a name of your choosing for this practice project. In the future, all projects will be saved in this way.

Finally, notice that you can also exit VBSA by using the EXIT command located here.

Lab1Practice2: FigureEights

or back to Contents

Your instructor will use VBSA to open the FigureEights demo. Here is a screen to illustrate the Design Mode:
This time, there are many different objects in your project window, so this should take some time. It includes:

- a Northwest `ListBox` to keep track of the direction traveled
- a Northeast `ListBox` to use in selecting an animal face
- two `Pictures`: one for the die to be rolled and one for the animal face (it begins as a ThumbsUp)
- two `TextBoxes`: they are used for what we call `variables` (numerical quantities that can vary during the course of the project run). Frequently we put these objects near the bottom, away from the action. *Also, for brevity, we frequently call TextBoxes simply Boxes.*
  - the variable `n` is used for a Code concept called loop, which causes repetition – more later on this!
  - the variable outcome is used for the result of rolling the die
- three buttons to be used before clicking the picture to make it travel

However, it is good practice! Also, notice that you can change the text in the title bar and the color in the project window as follows.

You need to click in the project window, away from all other objects, to select the window or form itself (which is considered an object). The text property that you type goes into the Title Bar and the BackColor is used for the entire form. We could have done a similar thing for our Practice1 project, but we had enough other new ideas already!

Your instructor will now go into Run mode and demonstrate this project:

- Click the `Roll Die` button. The result goes into the `outcome` box.
- Click the `Animal Face` button. The `outcome` is used to select a row in the Northeast listbox and then an animal face specified by the code. Note that rows in listboxes are started at 0, not 1, so we have to subtract a 1 from the outcome, which could be any number from 1 to 6.
- Click the `VELOCITY` button to decide how fast you want the animal face to move. The lower the number, the less time between ticks of the Special Effects clock and the faster the movement. A good choice for your first run might be 100.
- Click the picture to start the figure-eight pattern for the picture!
- After the Special Effects are completed, you may click the `EXIT` button to stop running your project.
As the travel takes place, the directions will be indicated in the *Northwest* listbox.

Finally, your instructor will open the Code mode and explain the code or instructions that make the project do what it does! We will present the code listings for three buttons and one picture.

The first and second code listings were already used in the previous practice project:

**VELOCITY CODE**

```
SPEED
```

**EXIT CODE**

```
END
```

The third code is new for us but is brief and easy to explain:

**ROLL DIE CODE**

```
rollDie outcome Picture2
```

Explanation:

rollDie is the action, picture2’s image is the face of the die, and `outcome` is the final result.

The next two codes, Animal Face and Picture1, illustrate the two most important concepts in this Guide:

- Animal Face: `selection`
- Picture1: `repetition`

These are examples of *control structures*, which are statements in the code which do not directly appear in the project design but are used to manipulate the objects. The `selection` in the Animal Face code is accomplished by the `Select Case` control structure, and the `repetition` in the Picture1 code is accomplished by the `For/Next loop` structure.

**ANIMAL FACE CODE**

```
List2.Index = outcome
SELECT CASE List2
CASE 1
    Picture1.Image = Bear
CASE 2
    Picture1.Image = Cat
CASE 3
    Picture1.Image = Dog
CASE 4
    Picture1.Image = Horse
CASE 5
```
CASE 6
Picture1.Image = Tiger
CASE ELSE
‘Not used – in the future, will remove this line if not used!
END SELECT

Explanation:

The statement \textit{List2.Index} = \textit{outcome} selects the index (or row) of List2 to be the outcome of rolling the die.

The Select Case control structure uses a List object to specify what should be done, depending on which index (or row) is selected. In this example, row 0 is not used because there is not a 0 on the die (the top Header row is numbered 0). The CASE ELSE refers to any other possibilities, such as forgetting to select an index at all. Here it is unused.

To write this code, choose \texttt{SELECT} as the first component, the correct \texttt{list} as the second component, and click the \texttt{Add} button. What results is an outline of the code, with you to insert the instructions needed for each case:

\begin{verbatim}
SELECT CASE List2
CASE 1
CASE 2
CASE 3
CASE 4
CASE 5
CASE 6
CASE ELSE
END SELECT
\end{verbatim}

It automatically inserts the word CASE as appropriate - using a row for each item in the list and CASE ELSE if no index is specified. It puts END SELECT at the end of this control structure.

\textit{As noted in the code, the most common situation has nothing to be done in the CASE ELSE situation – we will simply remove the line between CASE ELSE and END SELECT for brevity.}
Additional instructions (if any) that always are to be executed, regardless of the case, can be put below END SELECT. NOTE: when you click on a blank row in each case, you can then Add an instruction. What if you have more than one instruction for each case?

Every time you Add an instruction, it puts a blank line below it in the Code window, making room for another instruction. If you wind up with unused blank rows, it will not change the running of the project, but you can select such a row and right-click remove unused rows. Note also that you can right-click to remove and paste rows of code that were accidentally put in the wrong place! You can even right-click to copy and paste rows of code that need to be in more than one place!

NOTE: if you decide to change a list – either the items or the number of items, VBSA will insist that you delete any SELECT CASE code making use of it!

NOTE: you can remove the entire SELECT CASE control structure, including the code within it, by simply removing the top SELECT CASE statement.

PICTURE1 CODE

```vbs
'Lab1Practice2FigureEights

Picture1.Height = 75
Picture1.Width = 75

'Counter-clockwise

FOR n = 1 TO 4
    List1.Index = n
    travel List1 Picture1 120
NEXT

'Clockwise

List1.Index = 3
travel List1 Picture1 120
List1.Index = 4
travel List1 Picture1 120
List1.Index = 1
travel List1 Picture1 120
List1.Index = 2
travel List1 Picture1 120
```

Explanation:
The bottom half of the code is easy to explain. The indices (rows) of the directions from the list are taken in non-numerical order to make the picture go clockwise: 3 (West), 4 (South), 1 (East), and 2 (North).

The nice thing about the counter-clockwise motion is that the list of directions was set up so this would happen by going through the list in numerical order: 0 (East), 1 (North), 2 (West), and 3 (South). We have set up a variable n for the index or row value, and we want n to take on the values 0, 1, 2, 3. That is exactly what the code does, using a LOOP or FOR/NEXT control structure; it starts with the keyword FOR and ends with the keyword NEXT. Here is the format that we are using:

```
FOR variable-name = starting-value TO ending-value
    Statement or statements to be repeated
    (the BODY of the loop)
NEXT
```

The way that this works is to let the variable go through all of the whole number (integer) values in the range specified and execute ALL of the statements in the body of the loop for each such value. The way to picture this in your mind is that when the computer gets to the word NEXT, it increases the value of the variable by 1 (the next value), and then goes back (loops) to the FOR statement to make sure that it has not gone too far!

The way to set up such a loop is to specify the variable-name, the starting-value and the ending value. VBScript will put in the keywords FOR, TO, and NEXT automatically and will put a blank line in the body of the loop. To put together this loop control structure, we choose the following components and press the Add button:

Component 1: variable (a textbox name)
Component 2: skip
Component 3: =
Component 4: constant or another variable (which can be a starting value)
Component 5: TO
Component 6: constant or another variable (which can be an ending value)

NOTE: each time that you choose Constant, an InputBox appears to prompt you to enter an integer between -9999 and 9999.

Finally, when you click the Add button, the loop control structure is set up, with the keywords FOR, TO, and NEXT.

```NOTE: if you want to remove an entire loop control structure, including the body of the loop, you have to remove each statement individually.```

```NOTE: if you accidentally remove the NEXT statement, you can add it back separately!```
NOTE: if you want to change the top of the loop (different variable, different starting-value, and/or different ending-value), select the FOR instruction in the code window and then add the new FOR instruction to take its place. BE CAREFUL: you will get an extra NEXT statement automatically, so you need to remove it!

FORM LOAD CODE

List1.Header DiRECTION
List2.Visible = False
n.Visible = False
outcome.Visible = False

Explanation:

When you enter the Run mode, it changes the Visible property of List2, n, and outcome to False, i.e. they become invisible. Here is the screen after Form Load:

Why would we do this? List2, n, and outcome play important roles in the code, but the code is hidden from the user during the Run mode, and these 3 objects are confusing to the user of the project and they clutter the screen. Generally, in the future, we will want to hide the following types of objects by means of the Form Load code:

- Lists that are used in a Select Case control structure: for example, List2 (List1 really is useful for the user of the project to see)
- Loop variables: for example, n
  - During the Special Effects, the loop executes entirely and n gets its final value before the Special Effects take place
    - The reason: the instructions themselves run MUCH more quickly than the Special Effects invoked by some of them
- Some other variables: for example, outcome
  - It all depends on whether the user needs to see the values of these variables. In this case, the user can see the face of the die to see a picture of the outcome!
This practice illustrates 3 more Special Effects that are available to you (in addition to the *travel* and *rollDie* previously demonstrated):

- **random**: generates a random integer between 1 and a specified maximum integer
- **flipCoin**: flips a coin (a quarter goes back and forth between heads and tails) - stopping randomly
- **spin**: spins a list by going through the rows, going back to the top after reaching the bottom – stopping randomly

So – design the following project:

Notice that the text in the Title Bar has been changed to “Lab1Practice3RandomFlipSpinRoll”, and the color of the project window has been changed to LightBlue.

Now put in one line of code for each of the following objects:

- **RANDOM NUMBER**: random Number Max
- **FLIP**: flipCoin Number Picture1
- **SPIN LISTBOX**: spin Number List1
- **ROLL DIE**: rollDie Number Picture2
- **EXIT**: END
Now try running the project and checking out these Special Effects:

- Enter a value into Max, then click **RANDOM NUMBER** many times – each time it gives a value of Number chosen at random between 1 and the value of Max.
- Click **FLIP** and the coin in Picture1 will flip a random number of times and the value of Number is set to be 1 for Heads or 2 for Tails.
- Click **SPIN LISTBOX** and the selected row in List1 will advance by 1, going back to the top, skipping the Header row, after reaching the bottom, a random number of times. The value of the final row is placed into Number.
- Click **ROLL DIE** to roll the die in Picture2, with the die face changing a random number of times, with the final face value placed into Number.

Here is a sample screen in Run mode:

![Sample Screen](image)

**Lab1Credit: RandomSpinRollTravel**

or back to **Contents**

Your instructor will display a sample run, using **Lab1CreditRandomSpinRollTravel**. This hides the code from you, and you will have to figure out the code on your own!

Design your project as follows:
Picture1 is ThumbsUp and Picture2 is Dice1. List1 has the directions, and List2 has the numbers from 1 to 6. Notice the text “Lab1CreditRandomSpinRollTravel” in the Title Bar and the form’s Aqua color.

*Also include your first name and last name in the Title Bar.*

Then code your project as follows.

The VELOCITY and EXIT codes are the same as before.

The Form Load code should make visibility property False for List2, direction, and result, and it should replace the Header row of List1 with DIRECTION:

- Put a random number from 1 to 200 into the pixels box.
- Spin List1 and put the final row into the direction box
Roll the die and put the number of the final face into result.

Use the instruction WAIT, which causes the following MessageBox to appear.

(This gives the special effects time to get “caught up” before the rest of your instructions execute.)

- Set the index of List2 to be result.
- Use SELECT CASE, based on List2, to change the Picture1 (ThumbsUp). For example, for CASE 1, use
  ```vbnet
  Picture1.Image = Bear
  ```
  For cases 2, 3, 4, 5, 6, use similar instructions for Cat, Dog, Horse, Lion, Tiger.
- Finally, travel – with Picture1 moving in the direction given by List1 a distance given by pixels

NOTE: you can stay in Run mode and click Picture1 and see what happens. This can be repeated as much as you want! If Picture1 gets to a boundary, it is “stuck” until a direction takes it away from the boundary.

**LAB ASSIGNMENT 1: COMMENTS**

- There is NO code for you to copy this time, or the lab credits would be given to you without having to think about the solution!
- You do have the already running executable program Lab1CreditDemo to serve as your guide – it will show you WHAT it should do, but not HOW it is done.
- Refer to this Guide to see how similar actions are programmed
- For this lab, your instructor CAN help you, but refer to this Guide first!
- **NOTE**: for a quiz, your instructor CANNOT help you – that is why the best way to prepare for a quiz is to understand fully the practice and credit lab projects!

**TO SUBMIT THE PROJECT**

- Zip your lab folder (MAKE SURE that it is the correct one!)
- Upload it to the location in Desire2Learn announced by your instructor
- Your grade will be posted in Desire2Learn

**NOTE TO INSTRUCTORS**

To grade student projects (for this and the other labs):
Go to the gradebook in Desire2Learn
Click on the project link, download, and unzip
After grading the project, post the grade in Desire2Learn

CHAPTER 2

BOXES AND CALCULATIONS!

*(Projects: TriplePlay, Prizes)*

*(Credit Project: CableTV)*

*or back to Contents*

In Chapter 1, we used TextBoxes (*Boxes*, for short) to store numbers involved in special effects. However, in this chapter we will learn how Boxes can indeed also be used for text. We will learn how to use text for:

- Messages to the user via the MessageBox (*MsgBox*, for short)
- Input from the user via the *InputBox*
- *Strings* of characters assigned to a Box

Then, returning to numbers, we will see how numbers contained in Boxes can be combined using elementary arithmetic to do *calculations*.

We will reinforce the idea of a *decision control structure* by using *SELECT CASE* again. We will go beyond that to a second, very widely used *decision control structure IF/ELSE/END IF*, which is particularly useful when there are only two cases and we do not wish to use a list of cases.

*Lab2Practice1: TriplePlay*

*or back to Contents*

Your instructor will use VBSA to open the *TriplePlay* demo. Here is a screen to illustrate the Design Mode:
Your instructor will now go into Run mode and demonstrate this project. What happens first is caused by the Form Load code.

**FORM LOAD CODE**

```vbs
List1.Header PLAN
Price = 0
Rebate = 0

Msg1.Visible = False
Msg2.Visible = False
Rate.Visible = False
Dollar.Visible = False

Name = InputBox What is your name?

Msg1 = , here is your total!
Dollar = $
```

This puts a 0 into the Price and Rebate boxes. These are called an *assignment statements*, and, due to their location in the Form Load code, the 0 is called an *initial value*. The way to build this is to select the Price and Rebate boxes (one at a time), the assignment =, and then *Constant*. VBSA will then ask you to input the constant that you want, and it will be built into the assignment statement.

In an assignment statement, the *variable goes on the left of the = sign and the value on the right.*

After rolling the die, a random rebate will be calculated – replacing the 0. If you click on the Compute button, it will check to make sure that the 0 has been replaced (the die has been rolled) before calculating the subtotal.
Then the three boxes have their Visible property set to False – making them invisible to the user. They are needed for the code, but the user need not see them, and they clutter the window! This results in the window – after the user inputs a name via an InputBox:

This **InputBox** displays a message to the user, who reads it, enters whatever is called for by the message, and presses OK to continue. Here the InputBox displays the message “What is your name?” Whatever is entered is assigned to the Box on the left of the assignment =.

The way to build this is to select the box as the first component of the instruction and **InputBox** as the fourth component. VBSA will then ask for you to enter the message to be used by the InputBox.

The last two instructions in the Form Load code each demonstrate assigning a **String**, not a constant, to boxes (Msg1 and Dollar).

The way to build this is to select the box as the first component of the instruction and **String** as the fourth component. VBSA will then ask for you to enter the message to be assigned directly to the box without any input from the user through an InputBox.

**Which should you choose: an InputBox or a String instruction?** If you want the user to provide input, use **InputBox**, but if you want to hide what is happening from the user, make the box invisible and assign a **String** to it.
The goal is to be able somehow to combine the name that the user inputs into the Name box with the string stored in the Msg1 box. This will happen in the COMPUTE code. Thus, if the name is Kevin, our hope ultimately is to display the message “Kevin, here is your total!” just before displaying the total.

After that the user selects a plan from the communications company. The plan either combines two of phone, internet, and cable TV or even all three (called Triple Play). In the design phase, we put all of these into List1; now we use this List1 to SELECT a CASE and assign a price. The result is assigned to the Price box.

**LIST1 CODE**

```plaintext
SELECT CASE List1
  CASE Phone/Internet
    Price = 50
  CASE Phone/Cable
    Price = 75
  CASE Internet/Cable
    Price = 100
  CASE TriplePlay
    Price = 125
  CASE Else
END SELECT
```

Now click on the die to roll it. Put the result into the Rebate box, and use the WAIT statement to give the Special Effects time to finish before multiplying the result by 5 and putting it back into the Rebate box. Here is the code.

**PICTURE1 CODE**

```plaintext
rollDie Rebate Picture1
WAIT
  Rebate = 5 * Rebate
```

Yes, it is legal to have the Rebate box on both sides of the assignment = symbol! It does not mean mathematical equality. Instead, think of Rebate on the right of the = as standing for its current value and Rebate on the left of the = as its new value.

*Order does matter! The new value is always on the left, and the current value on the right – of an assignment statement.*

*NOTE: it is very important have the WAIT statement after any Special Effects if other statements involving these effects follow.*

In this case, without the WAIT, it would multiply the 0 in Rebate by 5, giving 0, and then putting numbers into Rebate as the die is rolling!

While the die is rolling we get the Special Effects message box as we did in Chapter 1:
Suppose we roll a 5. Then the Rebate is 5 * 5:

The EXIT code is always the same.

Finally, let us look at the COMPUTE CODE, which takes care of the calculations. After typing the Name and selecting a plan and clicking the die to roll it, we click on COMPUTE:

```vbnet
COMPUTE CODE

IF Price = 0 THEN
    MsgBox You must select a plan!
END

ELSE
END IF

IF Rebate = 0 THEN
    MsgBox Click die to get random rebate = $5 x face value
END

ELSE
This code first checks to make sure that the user has selected a plan and rolled the die. If the user has failed to do either one, the initial value of 0 will still be in effect. In these cases, a MessageBox is used to explain what happened and then the project run is ended. The user must run the project again.

*These MessageBox (MsgBox, for short) instructions are built by selecting MsgBox as the first component and then entering the desired message when prompted by VBSA.*

The IF/THEN/ELSE ENDIF control structure is used here to do the checking. If Price = 0 (or Rebate = 0, then all instructions ABOVE the ELSE are executed. If not, then all instructions between ELSE and END IF are executed. (Here this means that the Price > 0 (or Rebate > 0), so there are no special instructions between ELSE and END IF -- the calculation simply goes below the END IF. The general format for this control structure is

```
IF comparison THEN
  Statements to be executed if TRUE
ELSE
  Statements to be executed if FALSE
END IF
```

To build this control structure, select IF as the first component, then select a box, select a comparison >, =, or <, and then select a box or a Constant. VBSA automatically supplies the THEN, ELSE, and END IF, and it leaves blank lines between them.

*NOTE: if you want to remove an entire IF decision control structure, you have to remove each statement individually.*

*NOTE: if you accidentally remove the ELSE or the END IF statements, these MUST be added back separately!*

*NOTE: you must have the ELSE statement, even if there is nothing to do when the comparison is FALSE (so that there are no statements to go between it and END IF)!*

*NOTE: if you want to change the top of the IF decision control structure (different variable, different comparison), select the IF instruction in the code window and then add the new IF instruction to take its
BE CAREFUL: you will get an extra ELSE statement and an extra END IF statement automatically, so you need to remove them!

Now comes the arithmetic. In building an assignment statement, you can make use of the operators +, -, *, and / (for division). So – the Subtotal = Price – Rebate.

Then an InputBox appears – with the following message:

![InputBox Image]

This is accomplished by building the instruction Rate = InputBox, followed by the desired message.

**NOTE:** the user can enter a decimal, such as 5.5, to represent 5.5%. It is entered as a String but then converted to a decimal for purposes of arithmetic.

Now we need to multiply: Tax = Rate * Subtotal.

**NOTE:** The order of the calculations is very important! You cannot calculate the Tax until both the Rate and the Subtotal are known!

Now we divide by 100 because the rate that we have input is in percent (per hundred): Tax = Tax/100.

The computer replies with a MessageBox:

![MessageBox Image]

**NOTE:** to build this MsgBox instruction, when prompted for the message, we are allowed to enter the name of the box containing the message.

This message comes from Msg2, which in turn comes from Msg2 = Name & Msg1. This is called the concatenation of Name and Msg1, and & is called the concatenation operator. The way it works is that
If \textit{String1} and \textit{String2} both are strings of characters, \textit{String1 & String2} puts the two strings literally \textit{side by side (adjacent to one another)}. In this example, \textit{Msg1} starts with a comma and a blank to make this more effective.

We are now at the final calculation \textit{Total = Subtotal + Tax}, after which we put a dollar sign in front by means of another concatenation

\textit{Total = Dollar & Total},

resulting in the final screen:

![Final Screen Image]

\textbf{Debugging:} sometimes your program might “crash”, or have a “run-time error”, such as trying to divide by a variable whose value is 0. Sometimes your program might run but simply give incorrect results. For example, suppose you accidentally used the $\geq$ symbol (greater than) when you meant to use the $\leq$ symbol (less than) – in an IF/THEN comparison. To help find mistakes (“\textit{bugs}”) in your program, the \textit{BREAK} instruction may be used. Like the \textit{MsgBox} instruction, the \textit{BREAK} instruction stops execution of the program until the user clicks the OK button. To give helpful information, the \textit{BREAK} instruction also \textit{temporarily} makes \textit{visible} all variables (boxes) that were made invisible by the Form Load code for appearance purposes. This might help the programmer find out what instruction/instructions was/were at fault.

For example, suppose the Form Load code and the Compare (button) code for a short program are as follows

\textbf{FORM LOAD CODE}

\texttt{firstNumber.\text{Visible} = False}
\texttt{secondNumber.\text{Visible} = False}

\textbf{COMPARE CODE}

\texttt{firstNumber = InputBox}
secondNumber = InputBox
IF firstNumber > secondNumber THEN
    MsgBox First number greater than second number!
ELSE
    BREAK
END IF

If we input the number 3, then the number 2, we get:

If we input the number 2, then the number 3, we get:

Lab2Practice2: Prizes
Your instructor will use VBSA to open the Prizes demo. Here is a screen to illustrate the Design Mode:

Your instructor will now go into Run mode and demonstrate this project. First, the following code executes

**FORM LOAD CODE**

```
List1.Header PRIZE
Dollar.Visible = False
Dollar = $
```

Then the user clicks List1 to spin and get a prize at random – using the following code.

**LIST1 CODE**

```
spin PRIZE List1
WAIT
SELECT CASE List1
CASE Laptop
    PRIZE = 500
CASE Tablet
    PRIZE = 400
CASE SmartPhone
    PRIZE = 300
CASE HDTV
    PRIZE = 600
CASE ELSE
END SELECT
```
The user clicks the die to roll it and gets a bonus of 10 x the face value:

**ROLL DIE CODE**

```
rollDie BONUS Picture1
WAIT
BONUS = 10 * BONUS
```

Next, the user enters a $ sign in the CURRENCY box. Finally, the user clicks the Winnings button to add the prize and the bonus and put a $ sign next to the total winnings:

**WINNINGS CODE**

```
TOTAL = PRIZE + BONUS
TOTAL = Dollar & TOTAL
```

*Lab2Practice3: Race*

or back to *Contents*

Your instructor will use VBSA to open the *Race* demo. Here is a screen to illustrate the Design Mode:

The BlueCar image is placed into Picture1 and the RedCar image into Picture2. The two lines are changed to vertical by setting their height to 125 and their width to 10. The purpose is to see which car will win the race: the family weighted down with luggage on the roof of their BlueCar or the single person in the sporty RedCar?!
Your instructor will now go into Run mode and demonstrate this project. First, the following code executes:

**FORM LOAD CODE**

```vbscript
distBlue = 25
distRed = 25
over = 0
pixels.Visible = False
distBlue.Visible = False
distRed.Visible = False
over.Visible = False
```

After this code is executed, we now have:

All of the other code (except for that of the Exit button) is contained in an object that we have not used before this – the Clock! It has been convenient to use the clock to keep track of the actual time, but now, if you hover the mouse over it, it says

*Click Clock to turn Timer On/Off*

Thus, we can use the Clock as a timer, in milliseconds, for special effects, such as a race! When the user clicks the Clock, it asks for a millisecond delay, the time between ticks of this clock. If the user enters 100, the race is rather fast. If the user enters 300, the race is much slower. Try using different values! If you click the Timer during the race, it is like a Pause button and it turns back into a Clock. If you click again, you get to enter a new millisecond delay amount to continue the race. You can race as many times as you want – until you click the Exit button.

Here is the code – it is long, but we will explain each part of it.

**CLOCK CODE**
random pixels 20
distBlue += pixels
Picture1.Left = distBlue

IF distBlue >= 375 THEN
    TimeOut
    MsgBox The Blue Car wins!!!
    over = 1
ELSE
    ‘keep racing
END IF

‘Finish timer cycle even if the Blue car wins – could be a tie!
Random pixels 20
distRed += pixels
Picture2.Left = distRed

IF distRed >= 375 THEN
    TimeOut
    MsgBox The Red Car wins!!!
    over = 1
ELSE
    ‘keep racing
END IF

IF over = 1 THEN
    distBlue = 25
distRed = 25
    Picture1.Left = 25
    Picture2.Left = 25
    over = 0
ELSE
    ‘keep racing
END IF

For each car, there is similar code. For each tick of the timer, to be fair, a random number of pixels between 1 and 20 is generated for each car and added to the distance (distBlue, distRed) from the left edge of the window. The .Left property measures this distance.

If the distance traveled by a car is greater than or equal (>=) to 375, the distance to be traveled from the starting line to the finish line, then the instruction

*TimeOut, which turns the Timer back to a Clock, is executed*
and the winning message is displayed. The variable over is changed from 0 to 1, which means that the race is indeed over. However, TimeOut does not go completely into effect until ALL of the code for the tick of the Timer has been executed. Thus, it is possible that there will be a tie.

At the bottom of the code, if the race is indeed over (over = 1), then the initial distances and picture locations are restored and over is set back to 0. This permits the user to run more races without having to exit the project run.

Lab2Credit: CableTV

or back to Contents

Your instructor will display a sample run, using Lab2CreditCableTV. This hides the code from you, and you will have to figure out the code on your own!

Design your project as follows:

Notice the text “Lab2CreditCableTV” in the Title Bar and the form’s Yellow color.

Also include your first name and last name in the Title Bar.

Then code your project as follows.

The EXIT code is the same as before.

The Form Load code should replace the Header row of List1 by PLAN. It should make visibility property False for Dollar, Msg1, and Msg2, as well as put a 0 into Price of Plan and Discount:
It should put the string *(here is your net price!* into Msg1 and the character $ into Dollar. It should display an InputBox prompting for the user’s name:

The List1 code should set the Price of the Plan to 75, 125, 150, or 175, based on which CableTV plan is selected.

The Picture1 code should roll the die with the result going into Rate, WAIT, and then multiply the Rate by 6.

The CALCULATE code should do the following:

- If no plan is selected, tell the user “You must select a plan!” and end the program.
- If the die has not been rolled (so the discount is still 0), tell the user “Click die to get random discount = 6% x face value” and end the program
- Otherwise, multiply the Price of the Plan by the Rate to get the Discount, then divide the Discount by 100.
- Combine the Name and Msg1 to get Msg2 and then display to the user:
• Calculate the Net Price by subtracting the Discount from the Price of Plan.
• Put a $ sign next to Net Price.

For example:

CHAPTER 3
LISTS AND NESTING

(Practice Projects: TimesTable1, TimesTables2, WinningStreak)

(Credit Project: RandomMotion)

or back to Contents

So far, we have used Listboxes to store a small number of items, in a sense an expanded version of a TextBox) or Box, for short, which only can store one item at time.
In this chapter, we will learn how to use FOR loops (repetition control structure) to put lists with more than a small number of items, sometimes with a rather large number of items, into a ListBox. Then we will use an IF decision control structure NESTED inside the loop to sift through all of the items to find only those that satisfy some condition. This combination of decision within repetition is an extremely powerful strategy in computer science!

Next, we will see another way to add computing power to produce large lists of information by means of NESTED FOR loops, one inside the other! For example, if the outer loop repeats its body of instructions 100 times, and the inner loop repeats its body of instructions 100 times, then the innermost body of instructions executes $100 \times 100 = 10,000$ times!

In the credit project, we will use a third kind of nesting: a SELECT CASE decision control structure NESTED inside a loop!

Lab3Practice1: TimesTable1

or back to Contents

Your instructor will use VBSA to open the TimesTable1 demo. Here is a screen to illustrate the Design Mode:

Here is a screen to illustrate what the form looks like after the Form Load code is executed:
What happens first is caused by the Form Load code.

**FORM LOAD CODE**

```vbnet
num1.Visible = False
times.Visible = False
num2.Visible = False
equals.Visible = False
product.Visible = False
rowNum.Visible = False
row.Visible = False
List1.Clear
List1.Add PRODUCT
times = X
equals = =
```

Now click the TimesTable button to view what the purpose of your practice is.

Remember when you had to memorize the multiplication tables (times tables) in school? Wouldn’t you have loved to have a computer program to do the work for you? All you have to do is click on the Times Table Button! The times table for the integers 1 up to 9 has 81 rows in it, each row item consisting of the multiplication factors and the answer. This is obviously way more than the 4 items that you can put into a ListBox using VBDesigner. This is a perfect situation for using loops in Visual Basic! Also, notice that Visual Basic automatically puts in a Vertical Scroll Bar for the ListBox lstDisplayBox, which is not large enough to display all of its items.

Here then is the result. Note that the times table is longer than the ListBox, which automatically acquires a **vertical scroll bar** to aid in viewing the entire list.
The Times Table code is what puts all of the entries into the ListBox.

**TIMES TABLE CODE**

```vbnet
'For 9 x 9 rows, enter the times table

num1 = 1
num2 = 1
FOR rowNum = 1 TO 81
    product = num1 * num2
    'Prepare the row to go into the table
    row = num1 & times
    row = row & num2
    row = row & equals
    row = row & product
    List1.Add row
    num2 += 1
    IF num2 = 10 THEN
        num2 = 1
        num1 += 1
    ELSE
        'Keep going!
    END IF
NEXT
```

Here is the way that this works. We know that for a times table up to 9, there should be 9 x 9 = 81 rows, so we set up a loop in which rowNum goes from 1 to 81. We start both of the numbers at 1 and multiply to get the product. We prepare a row before putting it into the List1 by concatenating (putting adjacent) the first
number, the times symbol $\times$, the second number, the equals sign $=$, and the product. Now the row may be added to List1.

We are ready to advance in the times table by increasing the second number by 1 and repeating what we have just done. Where the IF decision control structure NESTED inside the loop comes in is the check to make sure the second number does not get too high. If it gets to 10, we put it back to 1 and increase the first number by 1. This check is actually done 81 times, but the computer is so fast that the user is not aware of any delay!

*Lab3Practice2: TimesTable2*

or back to *Contents*

Your instructor will use VBSA to open the TimesTable2 demo. Here is a screen to illustrate the Design Mode:

![Design Mode](image)

Here is a screen to illustrate what the form looks like after the Form Load code is executed:
The Form Load code is the same as in the previous Practice1.

FORM LOAD CODE

```vbnet
num1.Visible = False
times.Visible = False
num2.Visible = False
equals.Visible = False
product.Visible = False
rowNum.Visible = False
row.Visible = False

times = X
equals = =
```

To see how this project is different from the previous Practice1, enter a value into the high box before clicking on the Times Table button. Here are two sample results.

High = 3:
High = 100:

Note that the ListBox has 100 x 100 = 10,000 entries and actually takes a few seconds before displaying its results, even with the incredible speed of the computer!

In this example, we change our approach in writing the Times Table code.

TIMES TABLE CODE

'For 9 x 9 rows, enter the times table

List1.Clear
List1.Add PRODUCT

FOR num1 = 1 TO high
FOR num2 = 1 TO high
  product = num1 * num2
'Prepare the row to go into the table
  row = num1 & times
  row = row & num2
  row = row & equals
  row = row & product
  List1.Add row
NEXT
NEXT

This time we use NESTED FOR loops, one inside the other. If we enter 9 into the high box, we get the same result as in Practice1. Now, however, we can let high be smaller or bigger or even much bigger! We no longer need an IF decision control structure because here is the way the nested loops work:

- For num1 = 1, num2 varies from 1 to high
- Then for num1 = 2, num2 again varies from 1 to high
- This pattern keeps going, until num1 = high and num2 varies from 1 to high.

The only problems could be that
- high is so big that it takes too long to calculate or
- the product gets to be bigger than the largest permissible whole number (integer) in the computer system (in which case the project will “crash” — what is called a run-time error)!

Lab3Practice3: WinningStreak

or back to Contents

Have you ever tried flipping a coin many times in a row to get a streak going – say 3 heads in a row or 3 tails in a row? It is not so easy, is it? This project will let you do it electronically. All you do is click the Flip Coin Button, wait for the outcome, then click the Add To List Button to record this outcome in the Coin Tosses ListBox. Keep doing this until you get 3 in a row. The tricky part is that your project should recognize 3 in a row – easy for us visually but harder for the computer to track. We will figure how to do it...

Your instructor will use VBSA to open the WinningStreak demo. Here is a screen to illustrate the Design Mode:
Here is a screen to illustrate what the form looks like after the Form Load code is executed:

The Form Load code includes initializing the values to 0 for the boxes (variables) first, second, and third:

**FORM LOAD CODE**

```
List2.Visible = False
first.Visible = False
second.Visible = False
third.Visible = False
product.Visible = False
List1.Clear
List1.Add FLIPS
first = 0
```

List1.Clear
List1.Add FLIPS
first = 0
We use first, second, and third to represent the results of the 3 most recent flips of the coin, with a 1 for HEADS and a 2 for TAILS. The product will be the result of multiplying the 3 most recent results. Three HEADS in a row will give $1 \times 1 \times 1 = 1$, and three tails in a row will give $2 \times 2 \times 2 = 8$. All other combinations will give a 2 or a 4, so we will keep flipping!

Here is the result of a sample run:

The Flip Coin code takes care of flipping, selecting the numerical outcome in List2 (invisible to the user), putting the word outcome HEADS or TAILS in List1, and determining if the game is over and who won.

**FLIP COIN CODE**

```vbnet
second = 0
third = 0

wait
List2.index = third

'1 = HEADS, 2 = TAILS

select case List2
    case 1
        List1.Add = HEADS
    case 2
        List1.Add = TAILS
    case else
        end select
end select
```
`Multiply all three together
   product = first * second
   product = product * third

IF product = 1 THEN
   MsgBox 3 HEADS in a row – HEADS wins!!
   END
ELSE
ENDIF

IF product = 8 THEN
   MsgBox 3 TAILS in a row – TAILS wins!!
   END
ELSE
ENDIF

`Prepare for the next flip
   first = second
   second = third
`Try again!

The key is that at the top, the numerical result of the flip is put into third, and at the bottom the updated values of first and second are what were second and third, setting the stage for the next flip!

*Lab3Credit: (RandomMotion)*

or back to Contents

Your instructor will display a sample run, using *Lab3RandomMotion*. This hides the code from you, and you will have to figure out the code on your own!

Design your project as follows:
Notice the text “Lab2CreditRandomMotion” in the Title Bar and the form’s Red color.

Also include your first name and last name in the Title Bar.

The Form Load code should set the Visible property to FALSE for 4 variables and List2. It should replace the Header row of List1 by DIRECTION. It should add into List1 6 randomly generated directions by using a loop on the variable n. This should be done by using random to generate a number from 1 to 4, select this number in List2, and use this selection to place a direction into List1. At the bottom of your Form Load code, you should set n, the position in List1, back to 1.

NOTE: your Form Load code will use a third kind of nesting mentioned at the beginning of the chapter: a SELECT CASE decision control structure NESTED inside a loop. The loop will accomplish the repetition 6 times, and the SELECT CASE will be based on List2.

When you roll the die (Picture1), you WAIT for the outcome and then multiply it by 20 to get the number of pixels to travel in the direction currently selected in List1 by the value of n. You do NOT use a loop but
instead increase n by 1 (n += 1) each time you roll the die. After using the last direction on the list you should display a MsgBox and then END the program:

I hope you enjoyed using the guide...

Dr. Fabrey

*** END OF VBSytemApp (VBSA) Guide ***