Table of Contents – Understanding Programming through Graphics and Animation

A Detailed Description of George Polya's Four Steps of Problem Solving

1. Understand the Problem (Define the Problem)

- ***Carefully read*** the problem ***several times***.
- ***Identify*** what you are being asked to ***find***.
- ***Ensure*** that you ***understand all terminology***.
- ***Highlight*** all ***given information***.
- ***Identify*** all the ***information*** that ***is required*** to solve the problem.
- ***Identify*** the ***given information*** that ***is required*** to solve the problem.
- ***Identify*** any ***extraneous information*** (information that is not needed).
- ***Identify*** any ***missing information***.
- ***Do research*** to ***find*** or ***estimate*** any ***missing information***.
- ***Keep*** an ***open mind***.
- ***Do not make*** any ***unnecessary*** or ***incorrect assumptions***.
- ***Think logically*** and ***creatively!***
- ***Consult colleagues, peers, experts,*** etc.
- ***Do not worry*** about ***possible strategies yet***.
- ***Predict*** what a ***reasonable answer*** or ***range of answers*** would be.

2. Choose a Strategy

- ***Unleash your creative powers! Be imaginative!***
- ***Do not be afraid to take risks!***
- ***Do not dismiss any ideas*** at this stage. Feel free to be ***whacky!***
- ***Avoid*** feelings of ***frustration*** or ***inadequacy***.
- ***Do not give up quickly!***
- If you have the desire to quit, ***take a break*** and ***try solving the problem later***.
- ***Do not be afraid*** to be ***unconventional***. Perhaps you are correct and everyone else is wrong!
- ***Draw*** a ***diagram*** or ***visualize***.
- ***Compare*** the problem to an ***equivalent*** or ***similar problem*** that you have already solved.
- ***Compare*** the problem to a ***simpler*** but ***related problem***.
- ***Solve*** a ***specific example*** of the problem.
- ***Look for patterns***.
- ***Write*** a list of ***as many possible strategies*** as you can.
- ***Do research*** to discover if ***anyone else*** has solved the problem.

3. Carry out the Strategy

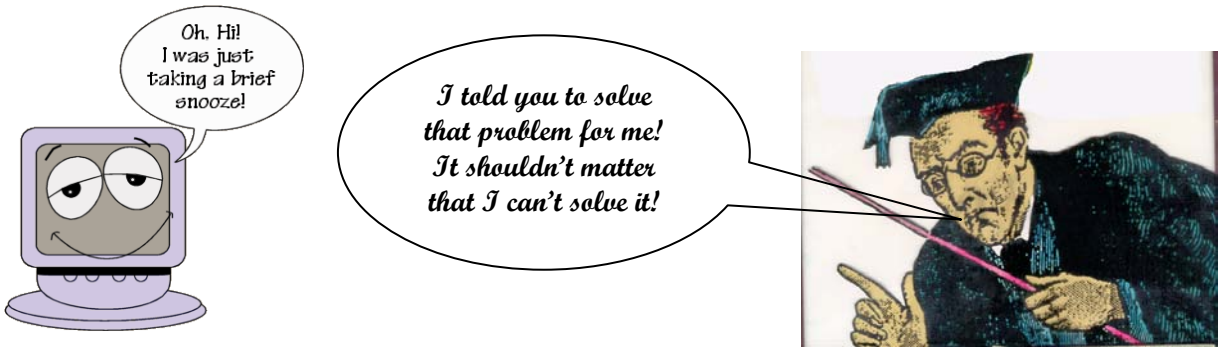
- ***Check*** your list of strategies and ***select one*** that you think is likely to work.
- ***Carry out*** your strategy ***logically*** and ***carefully***, paying close attention to ***detail***.
- If your strategy ***fails***, return to ***steps 1*** and ***2***.

4. Check the Solution

- Is your answer ***reasonable?***
- Does your ***answer agree*** with the ***prediction*** you made in ***step 1***?
- Does your ***answer agree*** with the ***answers obtained by others***?
- Is there a ***better way*** to solve the problem?
- Ask ***peers***, ***colleagues***, etc to check your solution.

The Most Important Lesson of the Entire Course

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

Wrong!!!!! (Most Students)	***Right!!!!!! (George Polya)***	***How these Steps Apply to Software Development***
~~1. Read problem~~ ~~2. Type code~~ ~~3. Click on the~~  ~~button~~	1. Understand the problem *(Analysis)*	1. Before you begin constructing a solution to a problem, you must know exactly what is required. Otherwise, you run the risk of solving the wrong problem or providing an incomplete solution to a given problem. In particular, you need to know what should be the output of the program given every possible input and you need to understand what features are needed (e.g. sound, graphics, networking, etc). 
	2. Choose a strategy *(Design)*	2. On paper, design a few different possible interfaces for your program. Do not write any code yet! In addition, it is important to consider a wide variety of algorithms. Choose the algorithms that best balance user ease, execution speed, programming complexity (ease of implementation) and storage requirements.
	3. Carry out the strategy *(Implementation)*	3. Write the code but not all in one fell swoop. Break up the large problem into several smaller sub-problems. Solve each sub-problem separately. In addition, consider different algorithms that can be used to solve a given sub-problem. Choose the algorithm that best suits our application. Do not integrate a solution to a sub-problem into the larger solution until you are confident that it is correct. 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.
	4. Check the solution *(Validation)*	4. Extensive testing should take place to find bugs that were not noticed in the implementation phase. 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. Once the software is released, additional bug fixes will usually be necessary as users report previously undiscovered bugs. This is known as the maintenance phase.

Important Terminology

Program

- A ***program*** is a sequence of ***instructions*** that a computer can ***interpret*** and ***execute***. ("Execute" means "carry out" in this context.)

Programming Language

- A ***programming language*** is a very ***precise*** and ***unambiguous*** language that is designed to allow ***instructions*** to be given to a computer.

Code

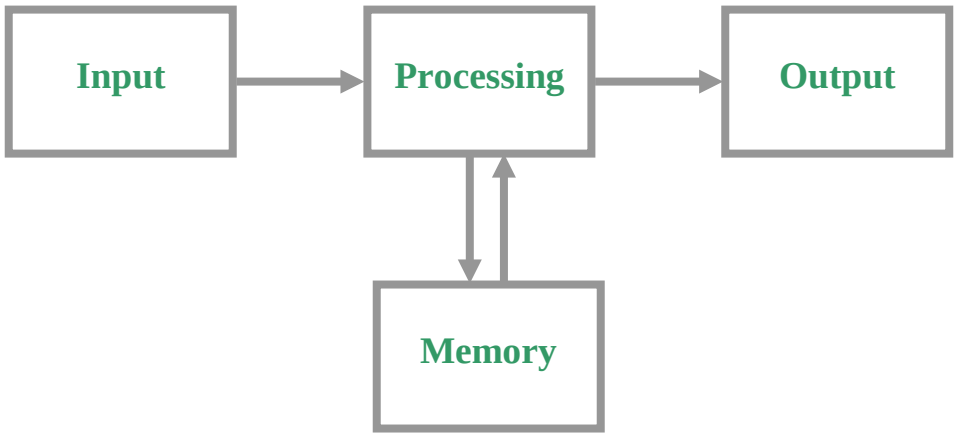
Programming instructions are often called "code." Programmers say that they are "writing code" when they write programs.

Algorithm

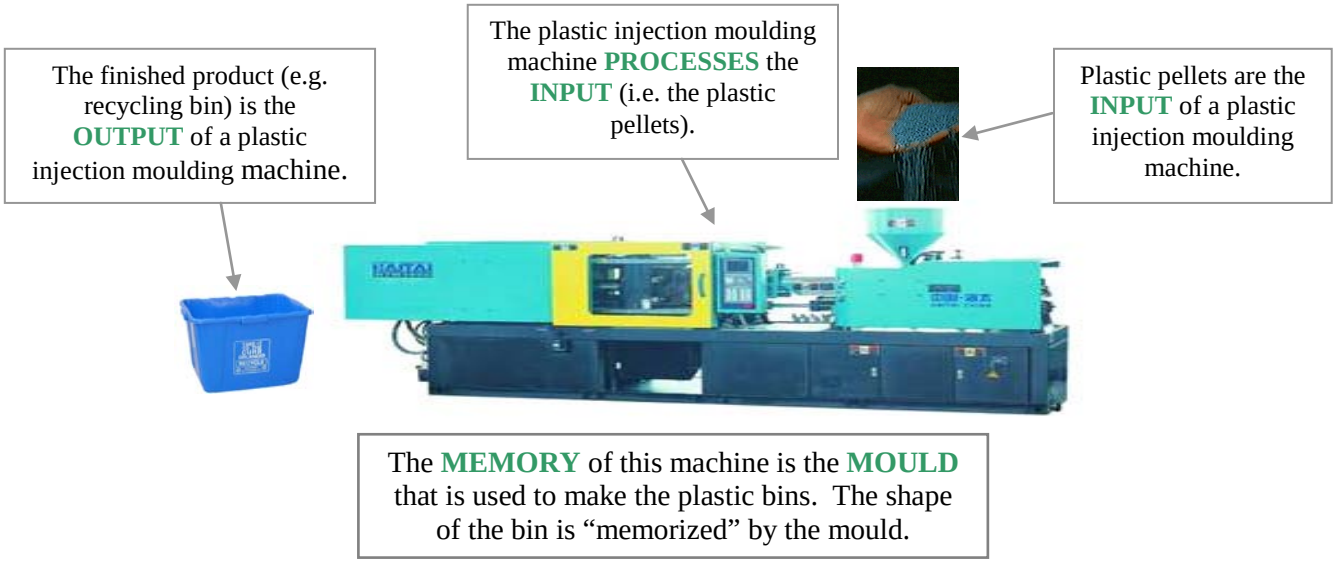
- An ***algorithm*** is a systematic procedure (finite series of steps) by which a problem is solved. Long division is an example.
- The steps of a particular algorithm remain the same whether you solve a problem by hand or by computer.
- In cooking, algorithms are called ***recipes***.
- Algorithms have been worked out for a wide range of problems.
- For many problems, there exist many different algorithms.
- For some problems, there are no known efficient algorithms (too slow and/or require too much memory). **e.g.** Is a number prime?
- Some problems cannot be solved by a computer (**i.e.** no algorithm exists that can be implemented on a computer).

A Computer as a Data Processing Machine

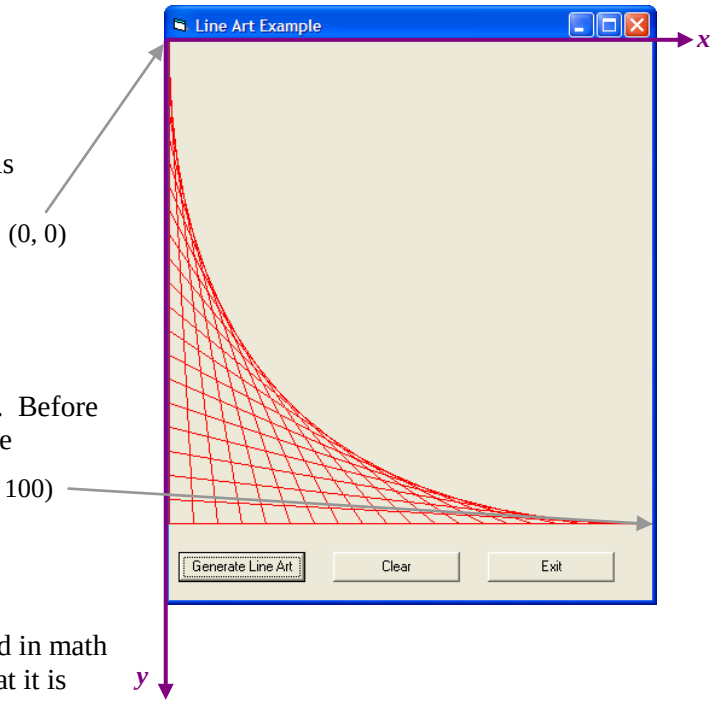
A simple but very useful model of a computer is shown below. A computer can be viewed, at a very simple level, as a machine that ***processes data*** (information). As the diagram suggests, information is given to a computer, the information is then processed by the computer and finally, the results are displayed.



This process is similar to industrial processes such as ***plastic injection moulding***. The main difference is that a computer requires ***memory*** to store the information that it processes. The diagram below shows the basic idea of how a plastic injection moulding machine produces its output.



Line Art



Introduction

Line art, as the name suggests, involves the creation of interesting designs using nothing but ***straight lines***. An interesting example is shown at the right. Although we perceive a curve, no curves were actually drawn. It turns out that the straight lines in this diagram are all ***tangent*** to a particular curve, which is why we "see" the curve.

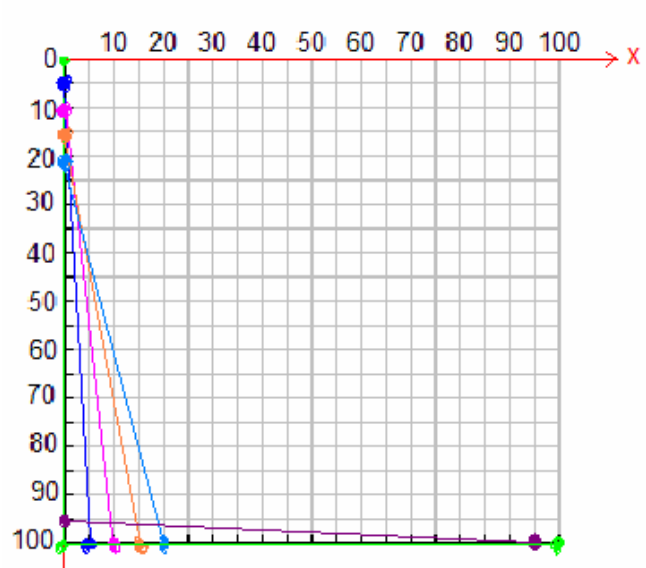
How can we COMMAND a Computer to create such a Picture?

As was mentioned on page 4, programming is much like teaching. Before we can "teach" a computer to solve a problem, we must first figure out how to solve it for ourselves!

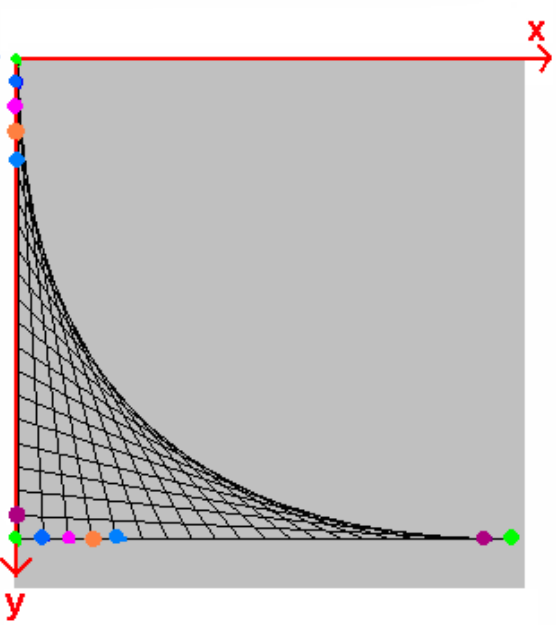
The diagram shown below is an example of a good method for understanding how such a picture is generated.

1. The drawing area is organized as a 100 by 100 Cartesian grid.
2. The horizontal axis (the *x*-axis) is just as it is usually presented in math class. The vertical axis (the *y*-axis) is also the same except that it is upside-down.
3. Each line is drawn from a certain point with co-ordinates (x_1, y_1) to another point with co-ordinates (x_2, y_2).
4. Usually, the lines are not drawn in a random fashion. Generally, they are drawn according to definite patterns.

As shown below, getting a computer to generate the picture in the above example is simply a matter of commanding the computer to draw a series of twenty-one line segments according to the pattern established by using the table of values shown below.



Start	End
(0,0)	(0,100)
(0,5)	(5,100)
(0,10)	(10,100)
(0,15)	(15,100)
(0,20)	(20,100)
(0, I)	(I, 100)
(0,95)	(95,100)
(0,100)	(100,100)

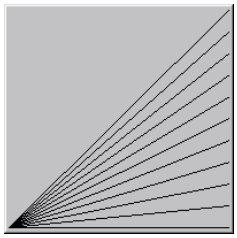


By completing a table of values and using a grid to sketch a few of the lines in the diagram, we are able to determine a pattern. For each line in the diagram, the y-co-ordinate of the "start" point is equal to the x-co-ordinate of the "end" point. In addition, we observe that the x-co-ordinate of the "start" point is always equal to 0 and the y-co-ordinate of the "end" point is always equal to 100.

Graph Paper and Pencil Exercises

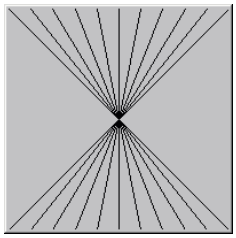
Using the example on the previous page as a model, determine the pattern(s) for generating each of the pictures shown below. (Don't be lazy! Use graph paper and tables of values.)

1.

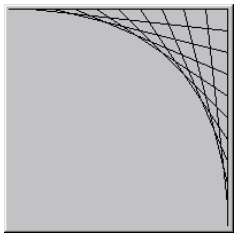


2.

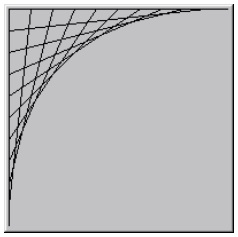
3.



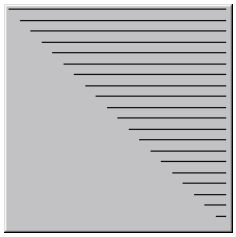
4.



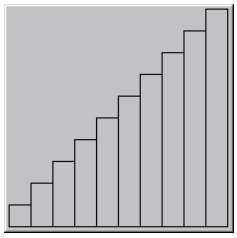
5.



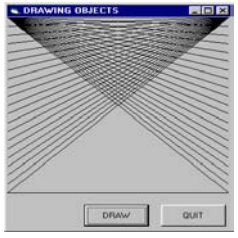
6.



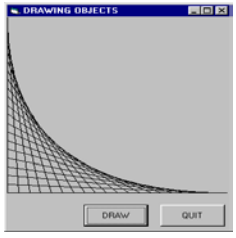
7.



8.



9.



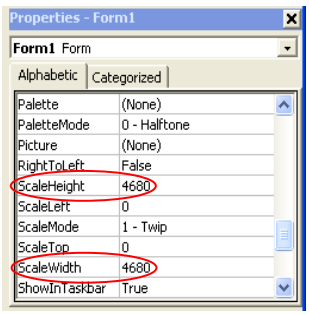
Writing a VB Program that Generates Line Art

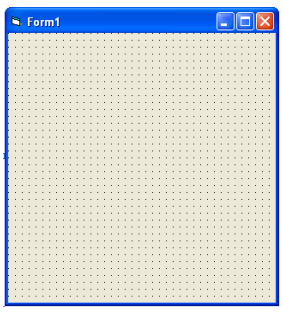
Before we begin writing code, we must ensure that the form (window) is properly prepared. For the sake of simplicity, a 100 × 100 grid is used in all our examples.

1. As described in class, launch Visual Basic and open a new "Standard EXE" project.

2. Stretch the form (window) so that the "drawing area" is square. To do this, stretch the width and the height of the form while watching the properties window. Stop when the form has the desired size and when the **ScaleHeight** and **ScaleWidth** properties have the same value. Note that the **ScaleMode** property by default has a value of "1-Twip." One ***twip*** is a tiny unit that is equivalent to one-twentieth of a printer's point or $1/1440^{th}$ of an inch (approximately

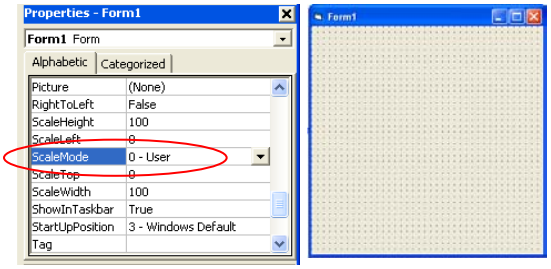
Use Google, MSDN help or visit http://msdn.microsoft.com for a complete explanation of the various units that can be used to describe co-ordinates on a form.



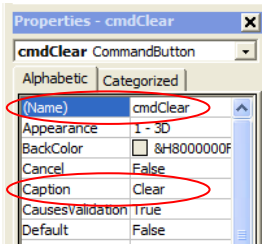


3. Change the **ScaleMode** property to "0-User" and the **ScaleHeight** and **ScaleWidth** properties to 100. This turns your form into a 100 × 100 grid, which is far more intuitive than a 4680 × 4680 grid.

4. Now enlarge the form somewhat to make room for a few command buttons.



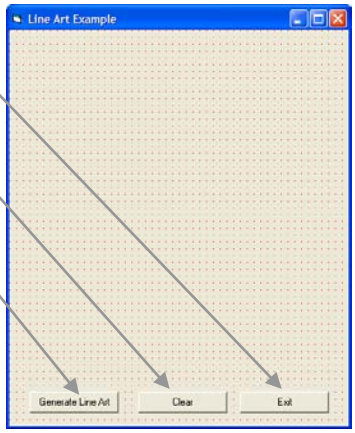
5. Using the properties window once again, change the "Caption" and "(Name)" properties to those suggested in the diagram at the right.



(Name): cmdClose
Caption: Exit

(Name): cmdClear
Caption: Clear

(Name): cmdDraw
Caption: Generate



6. Double-click the "cmdClose" button. This will automatically create the first statement and the last statement of a "**Sub**" (to be explained later) that will contain a simple instruction that will be executed every time the "cmdClose" button is clicked.

```
Private Sub cmdClose_Click()

End Sub
```

7. Type the instruction "**End**" between the first and last statements of the sub. Now the sub for the "cmdClose" button is complete. Every time the user clicks this button, the single instruction within the sub will be executed every time the user clicks the "Exit" button.

```
Private Sub cmdClose_Click()
    End
End Sub
```

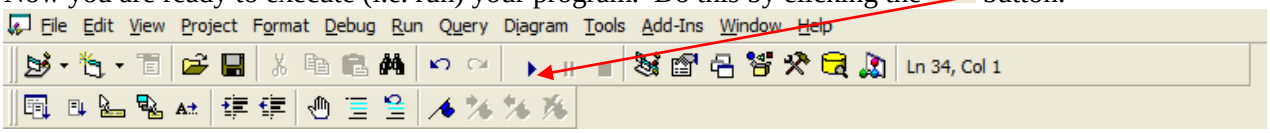
8. Now do the same for the "cmdClear" and the "cmdDraw" buttons. Double-click each button and type the code shown at the right.

```
Private Sub cmdClear_Click()
    Me.Cls
End Sub
```

9. Now add the command "**Option Explicit**" at the top of your program. By the time you are done, your program should look exactly like the following:

```
Option Explicit

Private Sub cmdClose_Click()
    End
End Sub

Private Sub cmdClear_Click()
    Me.Cls
End Sub

Private Sub cmdDraw_Click()
    Me.Line (0, 0)-(0, 100)
    Me.Line (0, 5)-(5, 100)
    Me.Line (0, 10)-(10, 100)
    Me.Line (0, 15)-(15, 100)
    Me.Line (0, 20)-(20, 100)
    Me.Line (0, 25)-(25, 100)
    Me.Line (0, 30)-(30, 100)
    Me.Line (0, 35)-(35, 100)
    Me.Line (0, 40)-(40, 100)
    Me.Line (0, 45)-(45, 100)
    Me.Line (0, 50)-(50, 100)
    Me.Line (0, 55)-(55, 100)
    Me.Line (0, 60)-(60, 100)
    Me.Line (0, 65)-(65, 100)
    Me.Line (0, 70)-(70, 100)
    Me.Line (0, 75)-(75, 100)
    Me.Line (0, 80)-(80, 100)
    Me.Line (0, 85)-(85, 100)
    Me.Line (0, 90)-(90, 100)
    Me.Line (0, 95)-(95, 100)
    Me.Line (0, 100)-(100, 100)
End Sub
```

10. Now you are ready to execute (i.e. run) your program. Do this by clicking the button.

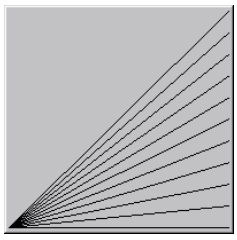


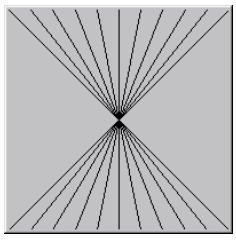
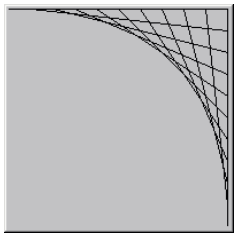
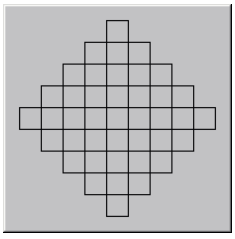
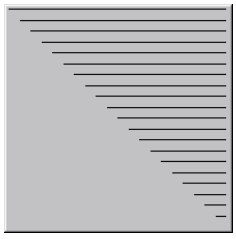
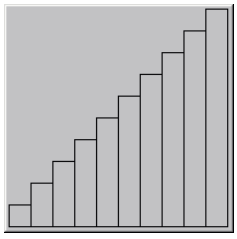
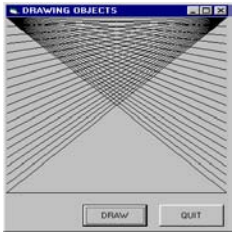
Questions

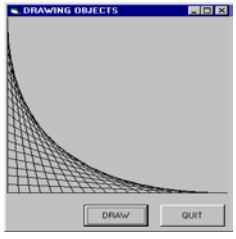
1. What is the difference between the **ScaleWidth** and **Width** properties of a form (or any other object)? What is the difference between the **ScaleHeight** and **Height** properties of a form (or any other object)?
2. Use Google, MSDN help, http://msdn.microsoft.com or any other source of information to define the terms twip, point, pixel and character. (These are some of the units that can be used as a **ScaleMode**. The others are inch, millimetre and centimetre, which hopefully, do not require an explanation.)

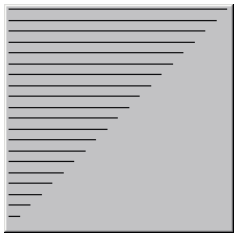
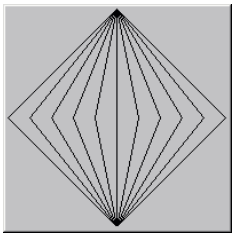
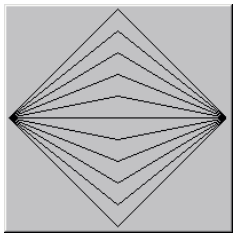
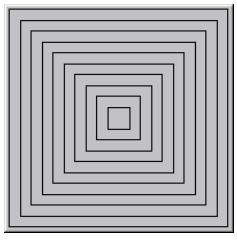
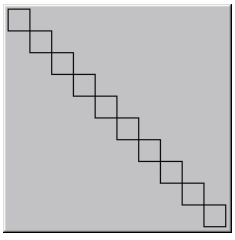
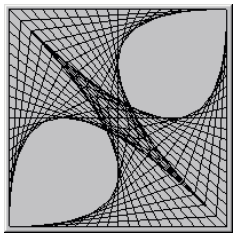
VB Programming Exercises

Now write VB programs to generate each of the following pictures.

1. 
2. 

3. 
4. 
5. 
6. 
7. 
8. 

9. 

10. 
11. 
12. 
13. 
14. 
15. 

Dramatically Reducing the Length of Line Art Programs

Introduction – Counted Loops

You probably noticed that the programs that you wrote to generate line art turned out to be extremely ***long*** and ***repetitive***. Fortunately, there is a programming concept known as ***repetition*** or ***looping*** that allows us to reduce dramatically the length of ***highly repetitive*** code. Instead of using the same statement (or very similar statements) multiple times in succession, we can list the statement ***once*** and ***specify the number of times it needs to be repeated***. Such a structure is called a ***counted loop***. In VB, counted loops are called "**For** … **Next**" loops.

Analogy – Adding Sugar to Coffee

Add Five Spoonfuls of Sugar to the Coffee	
Tedious, Long, Repetitive Method	***"For…Next" Loop Method (Counted Loop Method)***
' The following is not real VB. It is called ' "pseudo-code," which means false code. It is a ' mixture of VB and English and is a useful method ' for planning the overall structure of your programs. add 1st spoonful of sugar to the coffee add 2nd spoonful of sugar to the coffee add 3rd spoonful of sugar to the coffee add 4th spoonful of sugar to the coffee add 5th spoonful of sugar to the coffee	' The following is not real VB. It is called "pseudo-code," which means ' false code. It is a mixture of VB and English and is a useful method for ' planning the overall structure of your programs **For** I=1 **To** 5 add Ith spoonful of sugar to the coffee **Next** I **Note** In this example, the number of repetitions is exactly five. The value of "I" begins at 1 and increases by 1 after each repetition. After the first repetition, "I" becomes 2, after the second repetition, "I" becomes 3 after the third repetition, "I" becomes 4 and after the fourth repetition, "I" becomes 5. After the fifth repetition, the loop halts.

Using a Counted Loop to reduce the length of the Line Art Example Code on Page 8

Tedious, Long, Repetitive Method

```
Option Explicit

Private Sub cmdClose_Click()
    End
End Sub

Private Sub cmdClear_Click()
    Me.Cls
End Sub

Private Sub cmdDraw_Click()
    Me.Line (0, 0)-(0, 100)
    Me.Line (0, 5)-(5, 100)
    Me.Line (0, 10)-(10, 100)
    Me.Line (0, 15)-(15, 100)
    Me.Line (0, 20)-(20, 100)
    Me.Line (0, 25)-(25, 100)
    Me.Line (0, 30)-(30, 100)
    Me.Line (0, 35)-(35, 100)
    Me.Line (0, 40)-(40, 100)
    Me.Line (0, 45)-(45, 100)
    Me.Line (0, 50)-(50, 100)
    Me.Line (0, 55)-(55, 100)
    Me.Line (0, 60)-(60, 100)
    Me.Line (0, 65)-(65, 100)
    Me.Line (0, 70)-(70, 100)
    Me.Line (0, 75)-(75, 100)
    Me.Line (0, 80)-(80, 100)
    Me.Line (0, 85)-(85, 100)
    Me.Line (0, 90)-(90, 100)
    Me.Line (0, 95)-(95, 100)
    Me.Line (0, 100)-(100, 100)
End Sub
```

"For…Next" Loop Method (Counted Loop Method)

"**Option Explicit**" forces the programmer to declare all variables (see below).

This statement is called a ***variable declaration***. It is used to state the ***name*** and ***type*** of a variable. In this example, a variable called "I" is being declared.

This ***loop*** is used to repeat the statement "Me.**Line** (0,I)-(I,100)" twenty-one times. The value of "I" is set to 0 for the first iteration (repetition). After each repetition, the value of "I" is increased by five.

```
Option Explicit

Private Sub cmdClose_Click()
    End
End Sub

Private Sub cmdClear_Click()
    Me.Cls
End Sub

Private Sub cmdDraw_Click()

    Dim I As Integer

    For I = 0 To 100 Step 5
        Me.Line (0, I)-(I, 100)
    Next I

End Sub
```

Name of Object

Name of Event

Name of Sub

The variable "I" is called a ***loop counter variable*** (or simply a ***loop counter***). Why is this an appropriate name for such a variable?

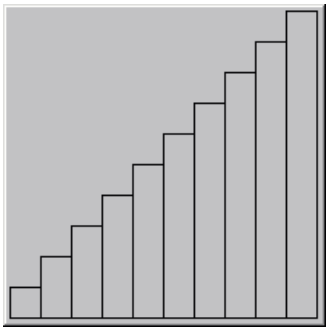
Exercises

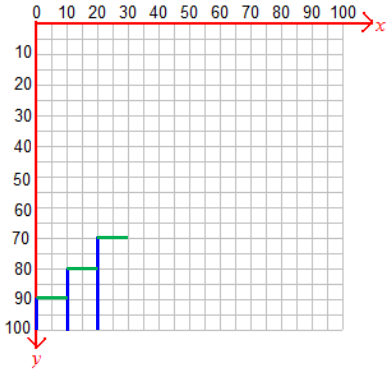
Use "For…Next" loops to generate the pictures in the exercises on page 9.

A Detailed Solution

Solution to Question 7 on Page 9

Planning the Solution





Vertical Lines		Horizontal Lines	
Start	**End**	**Start**	**End**
$(0,100)$	$(0,90)$	$(0,90)$	$(10,90)$
$(10,100)$	$(0,80)$	$(10,80)$	$(20,80)$
$(20,100)$	$(0,70)$	$(20,70)$	$(30,70)$
…	…	…	…
$(I,100)$	$(0,90-I)$	$(I,90-I)$	$(I+10,90-I)$
…	…	…	…
$(100,100)$	$(0,100)$	$(100,-10)$	$(100,-10)$

Writing the Code

Now that we have determined the patterns used to create the drawing, it's a simple matter to write the code. Two different methods are shown below.

Method 1

```
Private Sub cmdDrawBars_Click()

    Dim I As Integer

    'Draw the vertical line segments
    For I = 0 To 100 Step 10

        Me.Line (I, 100)-(I, 90 - I)

    Next I

    'Draw the horizontal line segments
    For I = 0 To 100 Step 10

        Me.Line (I, 90 - I)-(I + 10, 90 - I)

    Next I

    'Draw the single line segment at the bottom
    Me.Line (0, 100)-(100, 100)

End Sub
```

Method 2

```
Private Sub cmdDrawBars_Click()

    Dim I As Integer

    For I = 0 To 100 Step 10

        'Draw a vertical line segment
        Me.Line (I, 100)-(I, 90 - I)

        'Draw a horizontal line segment
        Me.Line (I, 90 - I)-(I + 10, 90 - I)

    Next I

    'Draw the single line segment at the bottom
    Me.Line (0, 100)-(100, 100)

End Sub
```

Question

Both methods presented above produce the same picture. Is there any difference between the two methods? Explain.

Using Grade 9 Math to Understand the Solution

Start
$(0,90)$
$(10,80)$
$(20,70)$
…
$(I,90-I)$
…
$(100,-10)$

⇒

x	y	First Differences
0	90	–
10	80	−10
20	70	−10
…	…	…
80	10	−10
90	0	−10
100	−10	−10

How are the x-co-ordinate and the y-co-ordinate related?

Since the first differences are constant, x and y must be ***linearly related***! Therefore, we can write an equation, in the form $y = mx + b$, that relates y to x.

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{80-90}{10-0} = \frac{-10}{10} = -1$$

$b = 90$ (since $(0,90)$ lies on the line)

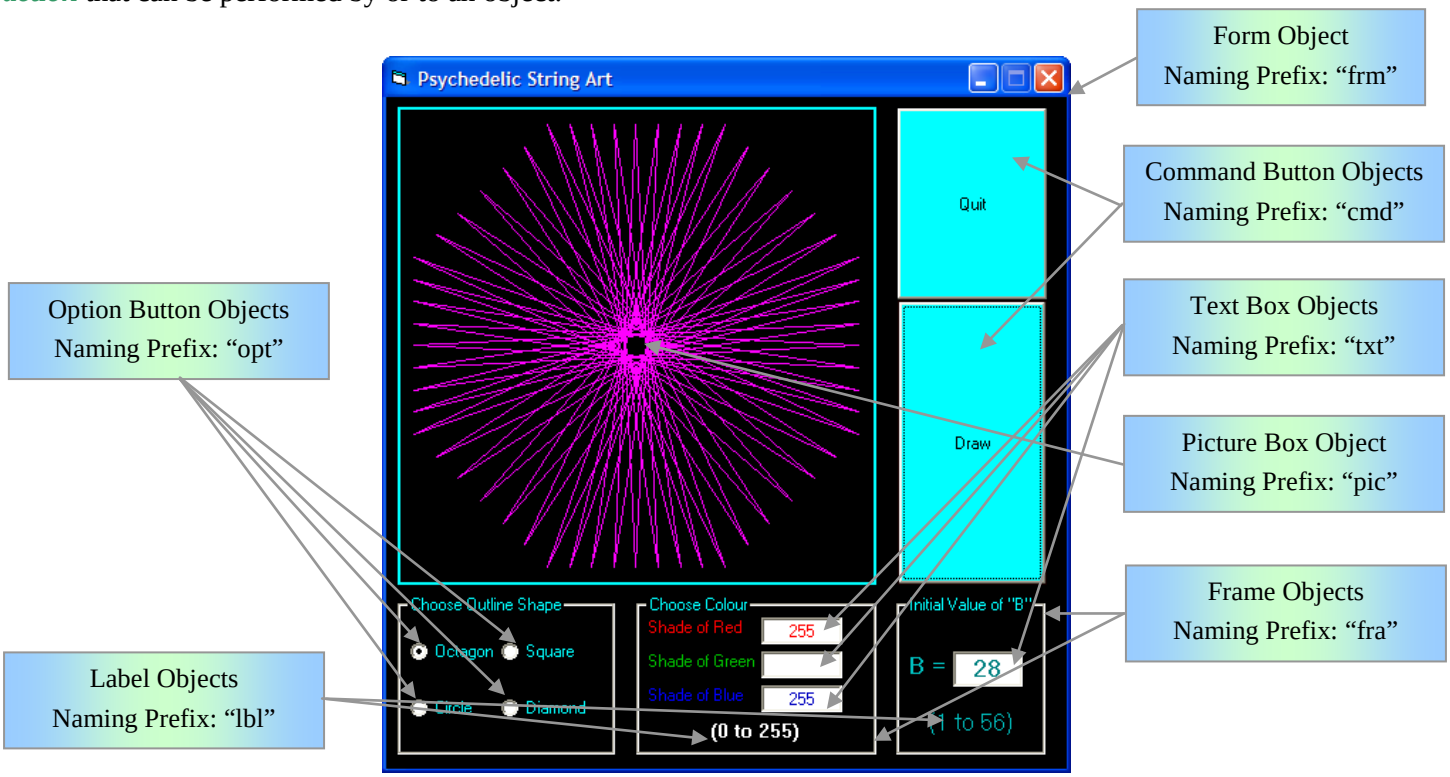
$\therefore y = -1x + 90$ or more simply, $y = 90 - x$

Circle Art

Objects, Properties and Methods

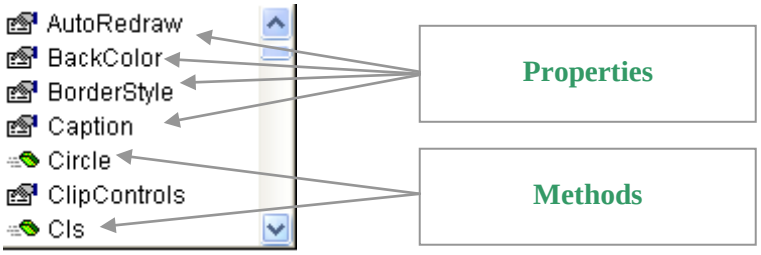
Object-oriented programming is centred on ***objects***. The diagram below shows examples of a few of the objects available in Visual Basic, along with their conventional naming prefixes. Objects tend to model real-world entities and tend to have extensive functionality. Each object is created from a template known as a ***class***. For example, the CommandButton class is used as a template to create as many command button objects as are desired by the programmer. A useful analogy is to think of a ***class as a cookie cutter*** and an ***object as a cookie***.

Each object is a collection of ***properties*** and ***methods***. A property is a ***characteristic*** of an object while a method is an ***action*** that can be performed by or to an object.



Understanding the Differences between Properties and Methods

Explain the differences between a ***property of an object*** and a ***method that can be performed on or by an object***.



Understanding the Differences between Variables and Objects

Variable

- A ***variable*** is a name that is used to represent a ***single value*** that is stored in the computer's main memory ("RAM").
- In VB, variables are created by using a "**Dim**" statement.

Object

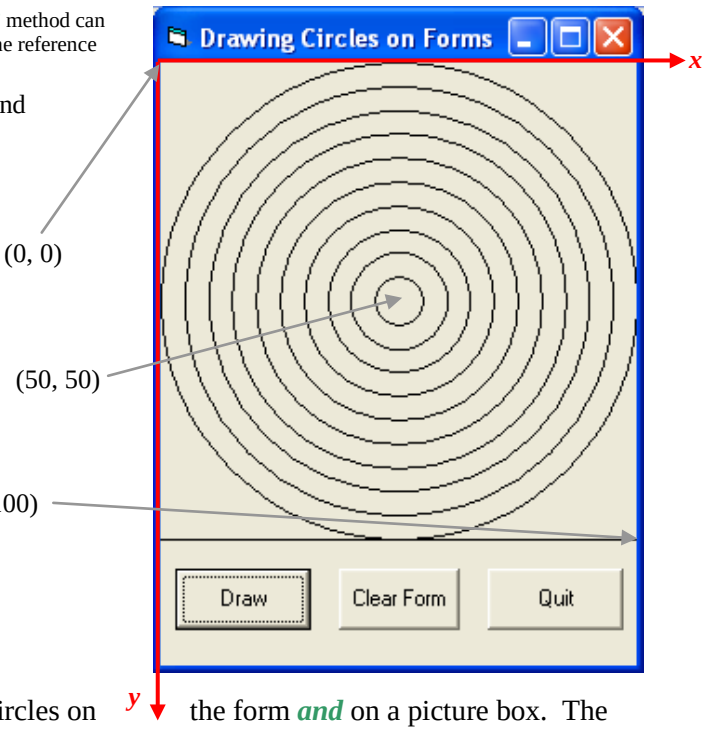
- An ***object*** is a collection of ***properties*** and ***methods***
- For the purposes of this course, objects in VB are created visually by using the form editor. "**Dim**" statements are not required for VB objects that can be created visually.
- To increase code readability, object names in VB should begin with conventional prefixes such as "cmd" and "frm." This practice is called "Hungarian notation" and is not used in most programming languages.

Drawing Circles on Forms and Other Objects

The circles shown here were drawn using the Visual Basic **"Circle"** method. The **"Circle"** method can be performed on forms, picture boxes and a few other objects. For more information, see the reference notes below, use MSDN help or visit http://msdn.microsoft.com.

The statement **"Me.Circle(X, Y) , R"** draws a circle with centre (X, Y) and radius R on the form.

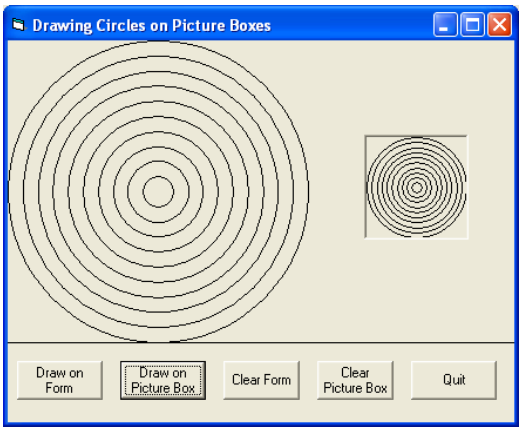
```
'This program is stored in the folder
'I:\Out\Nolfi\Ics3mo\Drawing, Graphics, Game Program
'                              Examples\Drawing Circles…
Option Explicit

Private Sub cmdDraw_Click()
   Dim Radius As Byte
   For Radius = 5 To 50 Step 5
     'Draw a circle with centre(50, 50) and radius "Radius"
      Me.Circle (50, 50), Radius
   Next Radius
End Sub

Private Sub cmdClear_Click()
   Me.Cls
End Sub

Private Sub cmdQuit_Click()
   End
End Sub
```



Now try the following extension of the above program. It draws circles on the form *and* on a picture box. The code and form are stored in **I:\Out\Nolfi\Ics3mo\Drawing and Graphics\Drawing Circles on …**

Questions

1. Describe how you would create the picture box shown at the right to ensure that the given code would work properly and to ensure that the grid is square.

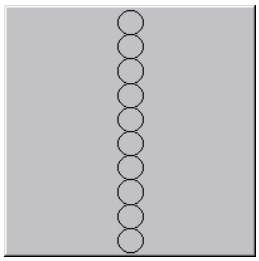


2. The code that is used to generate the circles on the form is identical to the code that is used to draw circles in the picture box. Why then, are the circles in the picture box much smaller?

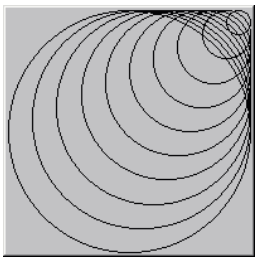
Exercises

Use "**For**…**Next**" loops to generate the pictures shown below.

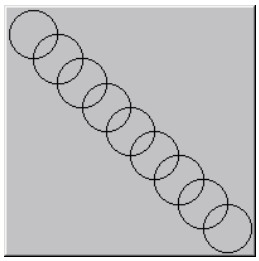
1.



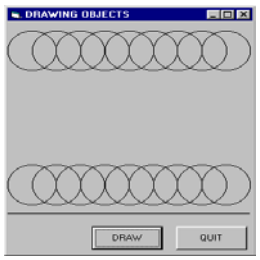
2.



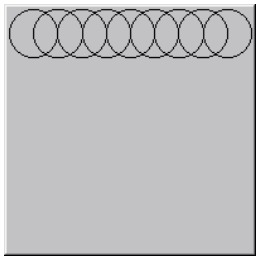
3.



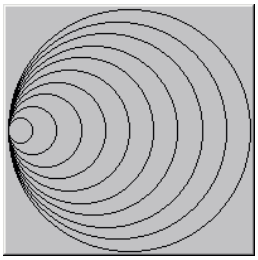
4.



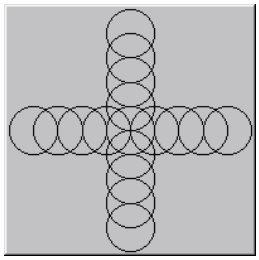
5.



6.

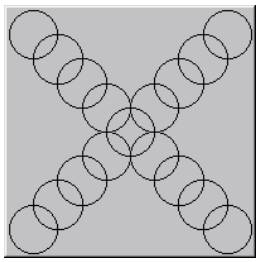


7.

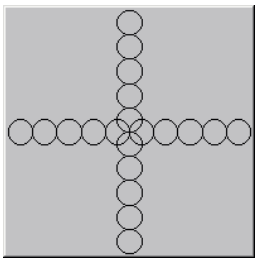


8.

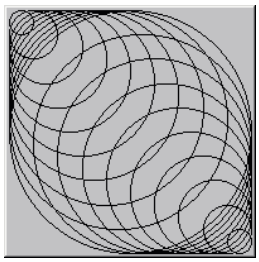
9.



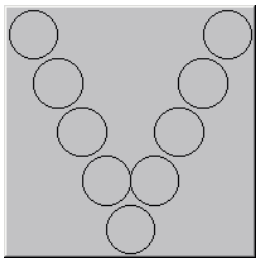
10.



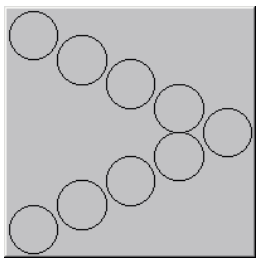
11.



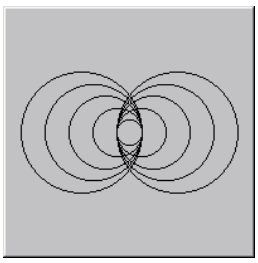
12.



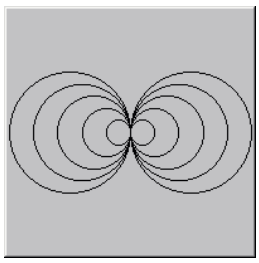
13.



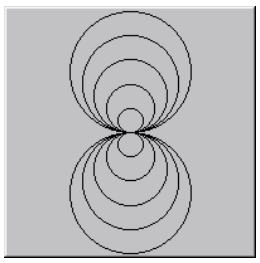
14.



15.



16.



Using the RGB Function to Include Colour in VB Line/Circle Drawings

Introduction – Primary Colours

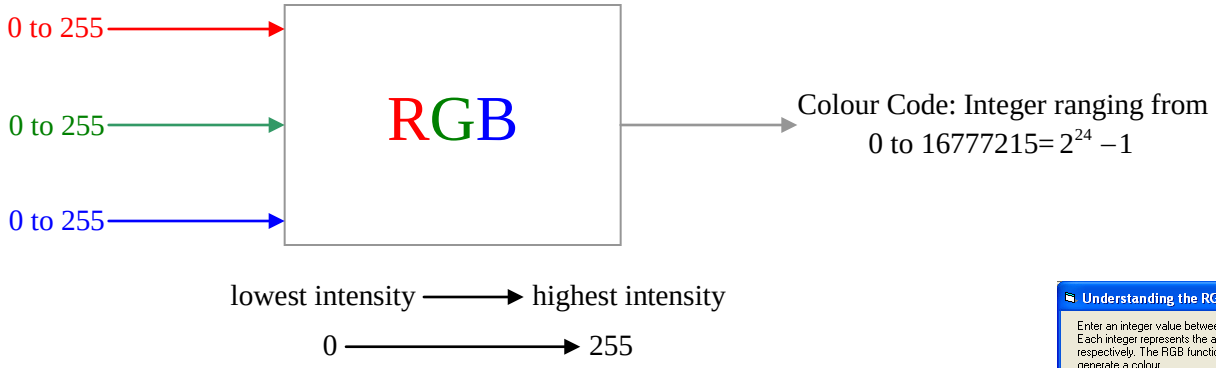
In art class you may have learned that ***red***, ***yellow*** and ***blue*** are ***primary colours***. This means that any other colour can be created by blending various amounts of these three colours.

For television and computer monitor purposes, however, ***red***, ***green*** and ***blue*** are used instead as the primary colours. This system of primary colours is called the ***RGB Colour Model***. (For more information, see RGB color model.)

The RGB Function in VB

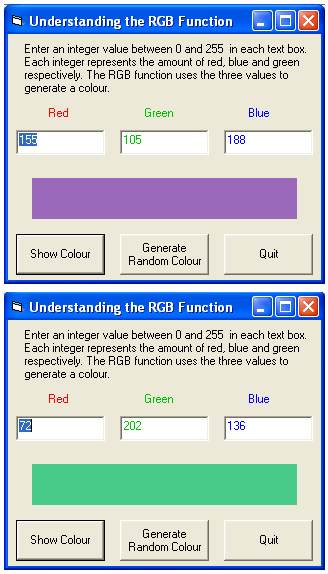
VB uses a 24-bit RGB colour model. This means that each colour is represented by a group of 24 ***binary digits*** (i.e. zeros and ones). For example, the colour black is represented by the code "000000000000000000000000" and the colour white is represented by the code "111111111111111111111111." When converted from binary (base 2) form to decimal (base 10) form, these numbers are 0 and 16777215 respectively. Therefore, a 24-bit colour model allows for the representation of over 16 million colours!

To "blend" the primary colours red, green and blue, VB uses a function called, of all things, "RGB." The RGB function requires three integers (whole numbers), each of which must be between 0 and 255 inclusive. Each number represents the intensity of the primary colour, with 0 representing the lowest intensity and 255 representing the highest. Once the three values are given to the RGB function, it produces a single whole number that represents the particular colour. This is shown pictorially below.



A VB Program that helps you to understand the RGB Colour Model

Load the VB project "RGB.vbp" found in the folder **I:\Out\Nolfi\Ics3m0\Drawing, Graphics, Game Program Examples\RGB**. Run the program and experiment with both the "Show Colour" and "Generate Random Colour" buttons. This program is also useful for finding values of "red," "green" and "blue" for desired colours.



How to use the RGB Function

1. *Examples involving the Assigning of a Specific Colour*

```
Me.ForeColor = RGB(72, 202, 136)  'Change the form's foreground colour
Me.BackColor = RGB(72, 202, 136)  'Change the form's background colour
```

2. *Examples involving the Assigning of a Random Colour*

```
Me.ForeColor = RGB(Int(Rnd*256), Int(Rnd*256), Int(Rnd*256))
Me.ForeColor = RGB(Int(Rnd*256), Int(Rnd*256), Int(Rnd*256))
```

Note on Random Numbers

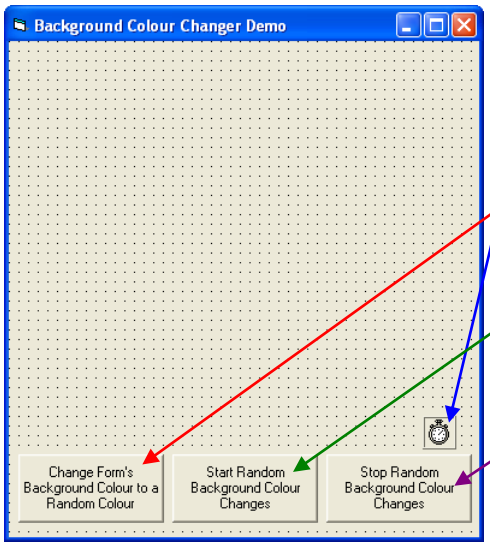
At this point, it suffices to say the VB expression "Int(Rnd*256)" produces a random integer (whole number) between 0 and 255 inclusive. In the next unit we shall take a far more detailed look

Using Timer Controls in your VB Programs

Example – Change Background Colour of Form at Regular Intervals

Load the VB project "ChangeBackColor.vbp" found in the folder **I:\Out\Nolfi\Ics3m0\Drawing, Graphics, Game Program Examples\Timer Examples**. When you run the program, you will discover that that there are ***two ways*** to change the background colour of the form.

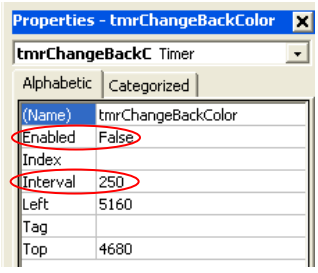
- The background colour of the form can be changed ***manually*** by clicking the button "Change Form's Background Colour …" Every time this button is clicked, the form's "BackColor" property is changed to a random value.
- The background colour of the form can also be changed ***automatically*** at regular intervals. This is accomplished by using a ***timer control*** (see below). A timer control's only purpose is to generate a "Timer" event at regular intervals. The interval is set by the programmer using the "Interval" property, whose value is a time specified in ***milliseconds***. For example, if the "Interval" property is set to 250 and the "Enabled" property is set to "True," the timer will generate a "Timer" event every 250 milliseconds (i.e. every 1/4 of a second). Every time that a "Timer" event is generated, the "**Sub**" corresponding to the timer is executed.



This object is known as a ***timer control***. It is used to execute code ***automatically*** at regular intervals. Timer controls are only visible at ***design-time***. At ***run-time***, they are invisible.

```
Private Sub cmdChangeBackColor_Click()
    Me.BackColor = RGB(Int(Rnd * 256), Int(Rnd * 256), Int(Rnd * 256))
End Sub

'Start the timer by setting the "Enabled" property to "True"
Private Sub cmdStart_Click()
    tmrChangeBackColor.Enabled = True
End Sub

'Stop the timer by setting the "Enabled" property to "False"
Private Sub cmdStop_Click()
    tmrChangeBackColor.Enabled = False
End Sub
```

```
'The following sub is automatically executed at regular intervals as long as the
'timer object's "Enabled" property is set to "True." The time (in milliseconds) that
'elapses between executions of this sub is stored in the "Interval" property.

Private Sub tmrChangeBackColor_Timer()
    Me.BackColor = RGB(Int(Rnd * 256), Int(Rnd * 256), Int(Rnd * 256))
End Sub
```



The name of the ***object*** is "tmrChangeBackColor"

The name of the ***event*** is "Timer"

The name of the "Sub" is "tmrChangeBackColor_Timer" This sub is executed automatically every time the "tmrChangeBackColor" object generates the "Timer" event.

Example – Drawing Shapes One at a Time

Create your own Artistic Design – Assignment to be Handed in!

Description of Assignment

Now it is time to put into practice what you have learned in this unit. You will create an artistic design of your own that consists of lines and circles. If you have the desire to earn a very high mark (i.e. 4+, 4++), then you need to go beyond what we have learned and include ellipses (ovals), arcs and whatever else you fancy!

What you must hand in

(a) Before you write your VB code, you must use ***a pencil and graph paper*** to design your art. It is not necessary to sketch every line and circle that will be included. It suffices to sketch enough to reveal the patterns that will be used to create a VB program that is as short as possible. Alternatively, you may use "**Geometers' Sketchpad**" instead of pencil and paper if you wish. If you do not know how to use Geometers' Sketchpad, it is worthwhile learning how because it is a valuable tool in math courses.

(b) You must also include tables of values with your pencil and paper sketch (or with your Geometers' Sketchpad sketch). Once again, it is only necessary to include enough points in your tables to reveal patterns!

(c) Obviously, you must also hand in your VB program. Make sure that you submit both your ".vbp" file and your ".frm" file(s). If you hand in only your ".vbp" file, there will be nothing for me to mark!

Evaluation Guide

Categories	*Criteria*	*Descriptors*					*Level*	*Average*
		Level 4	Level 3	Level 2	Level 1	Level 0		
Knowledge and Understanding (KU)	**Ability to Distinguish between Constant and Variable Information.**	Extensive	Good	Moderate	Minimal	Insufficient		
Application (APP)	**Loops used Wherever Possible** To what degree are repetitive steps implemented using "For…Next" loops?	Very High	High	Moderate	Minimal	Insufficient		
	Declaration of Variables To what degree are the variables declared with appropriate data types?	Very High	High	Moderate	Minimal	Insufficient		
	Pencil and Paper Sketch and Table of Values To what degree has the student employed a logical, thorough and organized debugging method?	Very High	High	Moderate	Minimal	Insufficient		
Thinking, Inquiry and Problem Solving (TIPS)	**Degree to which Design differs from Examples** To what degree has the student created a design that differs significantly from the examples given in the course notes?	Very High	High	Moderate	Minimal	Insufficient		
	Inclusion of Elements not Explicitly Taught To what degree has the student included elements not covered in the unit? (e.g. arcs, ellipses, etc.)	Very High	High	Moderate	Minimal	Insufficient		
Communication (COM)	**Indentation of Code** **Insertion of Blank Lines in Strategic Places** (to make code easier to read)	Very Few or no Errors	A Few Minor Errors	Moderate Number of Errors	Large Number of Errors	Very Large Number of Errors		
	Descriptiveness of Identifier Names Variables, Constants, Objects, Functions, Subs, etc **Inclusion of Property Names with Object Names** (**e.g.** 'txtName.Text' instead of 'txtName' alone) **Clarity of Code** How easy is it to understand, modify and debug the code? **Adherence to Naming Conventions** (**e.g.** use "txt" for text boxes, "lbl" for labels, etc.)	Masterful	Good	Adequate	Passable	Insufficient		

Animations in VB

Animated GIFs on Web Pages

If you have surfed the Internet, then surely you have seen what is known as an ***animated GIF***. "GIF" is a ***lossless*** image compression format that very effectively reduces the size of pictures that contain only a few solid colours (such as comic strip characters or cartoons). GIF also supports simple animations. If you find or create an animated GIF that you would like to include in a Visual Basic program, however, you would soon discover that Microsoft does not provide any controls that support animated GIFs. How can this problem be solved? There are two strategies that you could apply.

1. *A Lazy but Intelligent Method – Find a Third Party VB Control that Supports Animated GIFs*

Find a third party VB control that supports animated GIFs. Besides the controls that are listed in the standard VB toolbox, many others are available through the **"Components…"** option in the **"Project"** menu (see ***Yet another Lazy Method – Use a Windows Media Player Control***). By searching the list of components, you might find one that supports animated GIFs. If you do not find one, you might be able to download such a component from the Internet. The CNET site www.download.com is a good source of shareware and freeware. If you use such a site to download a component such as a GIF animator control, once the component is installed it will appear in the "Components…" dialogue box in VB.

2. *The Brute Force Method*

If you search high and low and are unable to find a VB control that supports the type of animation that you would like to use, then you can simply write your own VB code. Advanced VB programmers would know how to write their own controls. For the purposes of this course, however, we can write some simple code that involves a picture box or image control, a timer and a control array of picture boxes or image controls.

To gain insight into this method, load, run and study **I:\Out\Nolfi\Ics3m0\Drawing, Graphics, Game Program Examples\Animation\Flag Animation.vbp**. Also, see the note that begins on the next page, which is entitled "Using Timer Controls in your VB Programs."

Note: If you use method 2, you must first extract all the frames from your animated GIF. This can be accomplished using a good image editor or animation program.

Other types of Animations

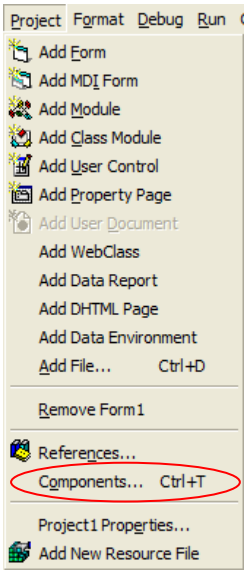
1. *Another Lazy Method – Find a Third Party VB Control that Supports other types of Animations*

Web pages support a host of other animations such as "Adobe Macromedia Flash" animations. Such animations can be used as long as you can find or develop VB controls that support them.

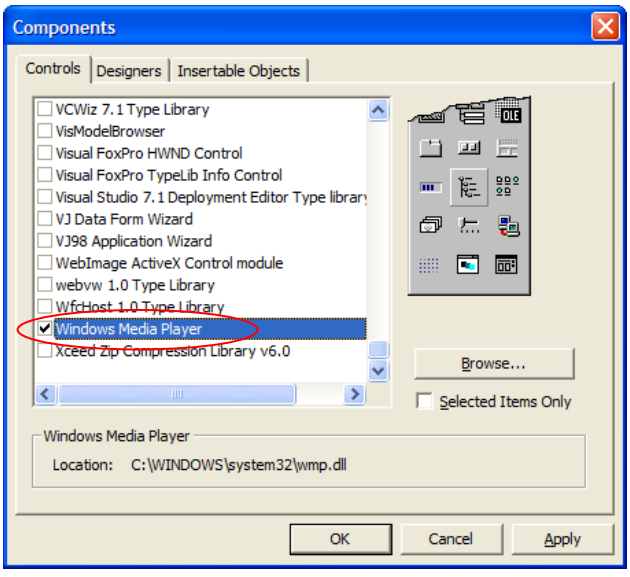
2. *Yet another Lazy Method – Use a Windows Media Player Control*

You can use "Windows Media Player" controls on your VB forms.

1. Choose "Components" from the "Project" menu.



2. In the "Components" dialogue box, choose the "Controls" tab. Scroll down to find "Windows Media Player" and check its check box.



3. Once steps 1 and 2 are completed, the Windows Media Player icon will appear in the tool box.

Once you have the Windows Media Player control on your form, you will need to study its properties and methods to learn how to use it.

Question

You will notice that the "Flag Animation" program mentioned above has two forms, one of which the user will never see. Explain the purpose of this "invisible" form. Why would it be a poor idea to load all the animation frames from files each time they need to be displayed?

Other Graphics Examples

The folder **"I:\Out\Nolfi\Ics3m0\Drawing, Graphics, Game Program Examples"** contains a large number of examples of many simple programming techniques that you would likely use in your video game programs. Study each example carefully to learn many simple tricks that will help you create programs that will dazzle your non-programmer friends!

Example – Canadian Flag Animation Program

Load and run the program

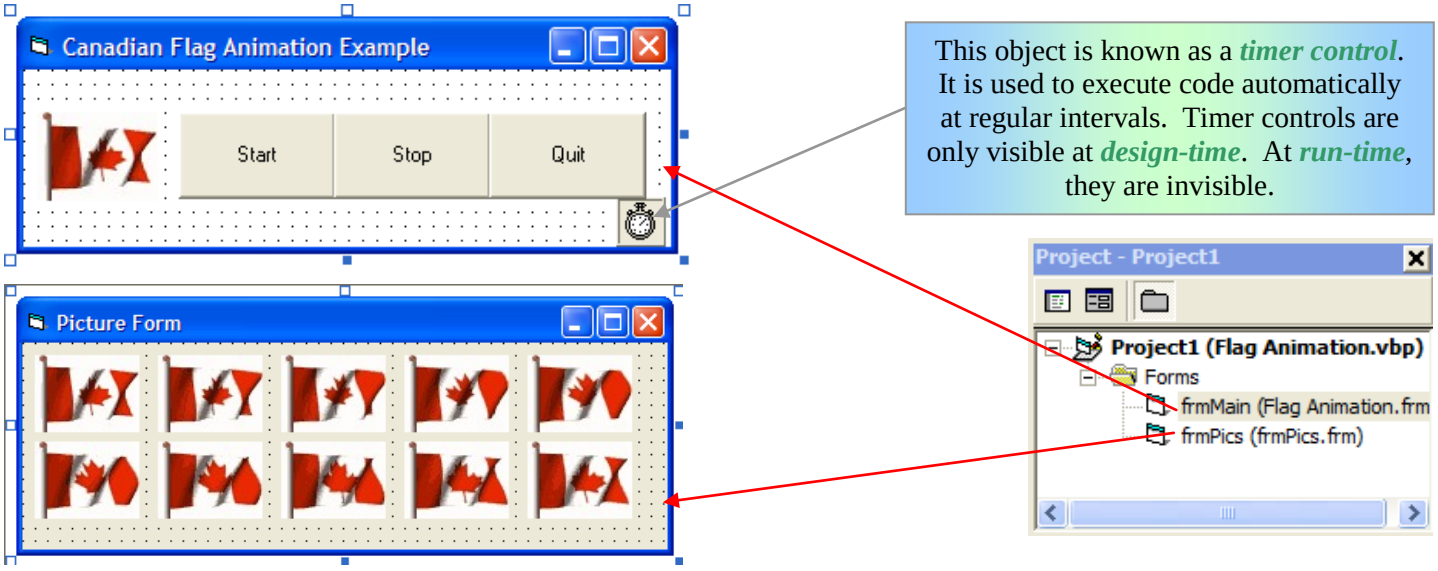
I:\Out\Nolfi\Ics3m0\Drawing, Graphics, Game Program Examples\Animation\Flag Animation.vbp.

Then read the notes below to understand how the flag animation is accomplished.

Using Multiple Forms in a VB Project

As shown below, it is possible to create a VB project that contains two or more forms. Carefully take note of the following important points.

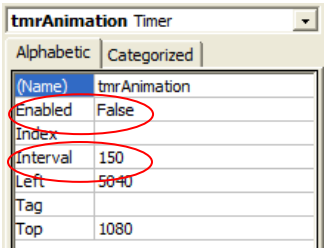
1. The project file "Flag Animation.vbp" stores important information regarding all the files required for the project. The filename extension ".vbp" stands for "Visual Basic Project."
2. Please keep in mind that ".vbp" files ***do not store any code***. The VB code used for your forms is actually stored in files with a ".frm" extension, which naturally, stands for "form." The flag animation project contains two forms, "frmMain," which is stored in the file "Flag Animation.frm" and "frmPics," which is stored in the file "frmPics.frm."
3. When the user runs this VB project, he/she will only see the "frmMain" form. The "frmPics" form is loaded into main memory (RAM) but it remains hidden from the user. The purpose of the "frmPics" form is to store the still pictures (known as "frames") used for the animation.
4. Whenever you create a VB project, it is best to store all the required files in a single folder.



Timer Controls

As stated above, a ***timer control*** is used to ***execute code at regular intervals without user intervention***. Applications of this include the following:

- Update a picture at regular intervals (e.g. traffic camera picture, second hand of clock)
- Cancel an action that is taking a long time to complete (i.e. every so often check if the user has clicked the "Cancel" button)
- Check periodically for new email messages
- Almost anything that you can imagine that occurs at regular intervals



Shown at the right is the property list for the timer used in the flag animation example. There are only two properties that are of great interest to us.

Enabled: When this property is set to **False**, the timer control does not generate the "Timer" event. This means that no code is executed at regular intervals. If this property is set to **True**, the timer control will generate the "Timer" event at regular intervals as specified by the "Interval" property.

Interval: This property specifies how much time (in milliseconds) should elapse between calls to the timer control's "Timer" event. If "Enabled" is set to **True** and "Interval" is set to a positive value, the "Timer" event will be generated every "Interval" milliseconds. Otherwise, the "Timer" event is not generated.

The following is the code used in the "frmMain" form. Note that there is no code needed for the "frmPics" form because it serves entirely as a convenient storage area for the individual frames in the animation.

```
Option Explicit

'The following variable keeps track of the frame
'that is displayed in the "imgFlag" image control.
'The value of "Frame" ranges from 0 to 9 inclusive.
Dim Frame As Byte

Private Sub Form_Load()
    Load frmPics 'Load "frmPics" but do not show it.
    Frame = 1
End Sub

Private Sub cmdStart_Click()
    tmrAnimation.Enabled = True
End Sub

Private Sub cmdStop_Click()
    tmrAnimation.Enabled = False
End Sub

'The following Sub is called automatically every 150 ms
'whenever the timer "tmrAnimation" is enabled.
Private Sub tmrAnimation_Timer()
    'Display current frame.
    imgFlag.Picture = frmPics.imgFrame(Frame).Picture
    'Calculate next frame to be displayed.
    Frame = (Frame + 1) Mod 10
End Sub

Private Sub cmdQuit_Click()
    End
End Sub
```

Force variable declarations.

Examples of Various *Events*

- **Click**: Press and release a mouse button
- **Load**: Form is loaded into RAM
- **Timer**: Interval for timer control has elapsed

To shorten the code considerably, a structure known as a ***control array*** is used (see next page). Using this idea allows us to use ***one statement*** to access a large number of objects. In this example, we are able to access any of the flag animation frames just by altering the value of the "Frame" variable.

- **tmrAnimation**: name of ***object***
- **Timer**: name of ***event***
- **tmrAnimation_Timer**: name of **Sub**
- **Sub:** Subroutine, a set of instructions that performs a specific task for a main routine, requiring direction back to the proper place in the main routine on completion of the task.

How the Value of the "Frame" Variable Changes

Value of "Frame"	Calculation	New Value of "Frame"
0	(0+1) **Mod** 10	1
1	(1+1) **Mod** 10	2
2	(2+1) **Mod** 10	3
3	(3+1) **Mod** 10	4
4	(4+1) **Mod** 10	5
5	(5+1) **Mod** 10	6
6	(6+1) **Mod** 10	7
7	(7+1) **Mod** 10	8
8	(8+1) **Mod** 10	9
9	(9+1) **Mod** 10	10
10	(10+1) **Mod** 10	0

This is called a ***trace chart*** or a ***memory map***. It is a tool that is used to figure out how the values of variables change as a program executes.

The **Mod** operator is used to calculate the ***remainder*** obtained when one integer quantity is divided by another. For example,

31 **Mod** 7 = 3

because 7 divides into 31 four times with three left over.

Note that the backslash ("\") is used to compute the ***quotient*** obtained when one integer quantity is divided by another. For example,

31 \ 7 = 4

The forward slash is used for general division. For example,

31 / 7 = 4.42857142857143

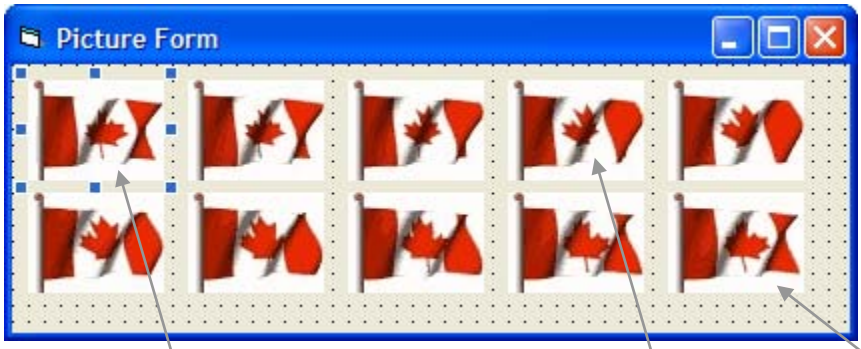
By using the above table we can see clearly how the "**Mod**" operator can be used to "cycle through" a range of integers. In the flag animation example, the "**Mod**" operator is used to cycle through the integers 0, 1, 2, …, 9. Once 10 is reached, the value of "Frame" becomes zero once again and the cycle starts anew.

Control Arrays

An ***array*** is a programming structure that allows us to use a ***single name*** to access any member of a collection of variable or objects. Since each member of the collection has the same name, a number called the ***index*** or ***subscript*** is used to identify any particular member, which is called an ***element***. Arrays make it easy to process large amounts using loops. When arrays are used to process large amounts of data, the code tends to be much shorter than equivalent code written without the use of arrays. A much more detailed study of arrays will be done in unit 3 of this course.

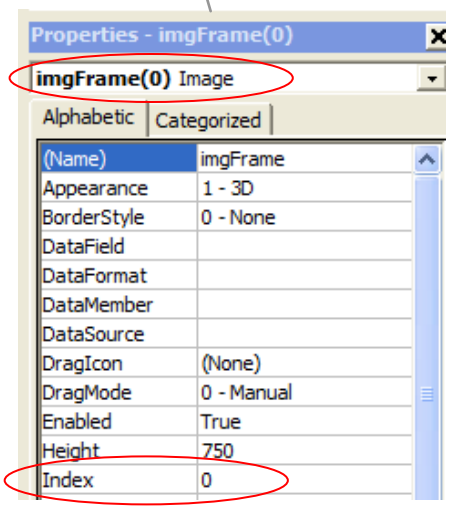
A ***control array*** is an array of ***VB controls***. A ***control*** in VB, speaking very loosely, is an object on a form that acts in the same manner as a control on a machine. A control on a machine is used to guide it and regulate it. Similarly, a control on a form is used to "guide" and "regulate" a program.

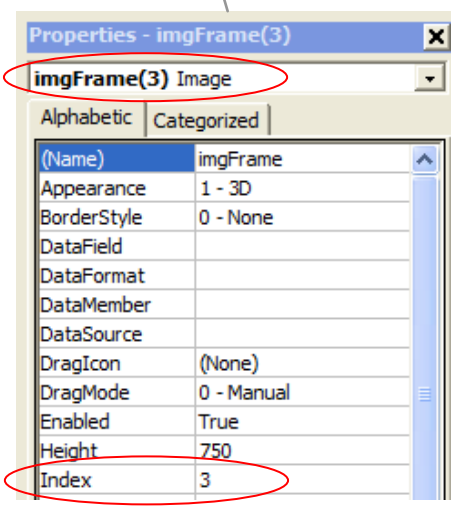
The idea of an array is similar to that of street addresses. All buildings on Kennedy Road North, for example, have the same street name. If you were to tell someone who is unfamiliar with Brampton that Central Peel is located on Kennedy Road North, then he/she would only have a vague idea of its location. On the other hand, if you specify the complete address, 32 Kennedy Road North, then the school would be much easier to find.

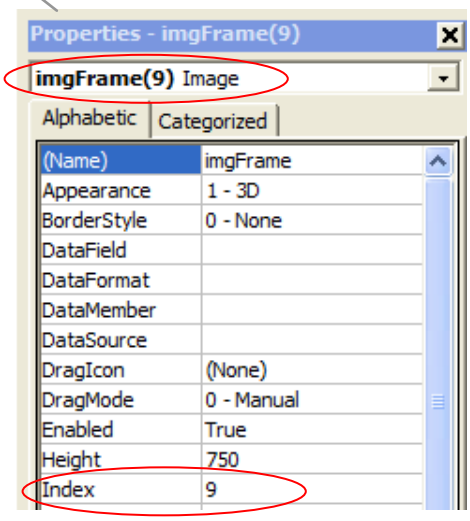


The series of image controls on the form at the left is a ***control array of image controls***. The following is a summary of its features:

- Each image control has the ***same name***, that is, "imgFrame." The name of an array is analogous to a ***street name***.
- To distinguish one ***element*** of the control array from another, its ***index*** is used.
- The ***index*** or ***subscript*** of a particular ***element*** is an integer that uniquely identifies it. The index of an element is analogous to a ***street number***.







How to Create a Control Array

There are ***three ways*** to create a control array at ***design-time***.

- Copy an existing control and then paste it onto the form.
- Assign the same name to more than one control.
- Set the control's "Index" property to a non-negative integer value.

Important Terminology

Syntax

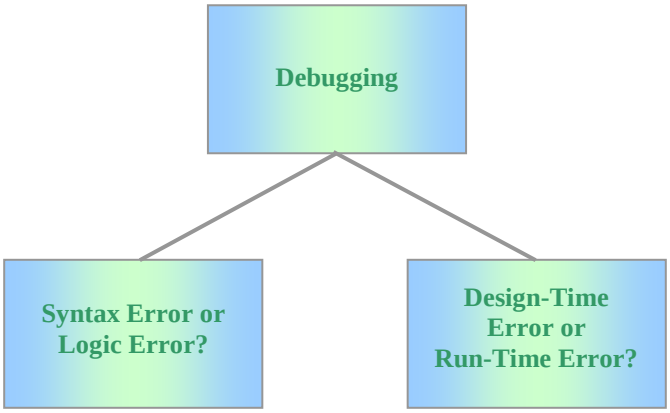
Linguistics: The study of the rules for the formation of grammatical sentences in a language.
Computer Science: The grammatical rules and structural patterns governing the ordered use of appropriate words and symbols for creating valid programming statements using a programming language.

Logic

Philosophy: The study of the principles of reasoning, especially of the structure of propositions as distinguished from their content and of method and validity in deductive reasoning.
Computer Science: The non-arithmetic operations performed by a computer, such as sorting, comparing and matching, that involve "yes-no" or "true-false" decisions.

Debug

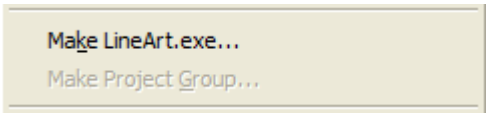
Computer Science: Locate and correct errors in a computer program.



Compile

Computer Science: Use a computer program called a ***compiler*** to translate ***source code***, written in a particular programming language such as Visual Basic, into computer-readable ***machine code*** that can be executed by a CPU.

To compile a program in VB, use the "Make…" option from the "File" menu. For example, if your project were called "LineArt.vbp," there would be an option in the "File" menu "Make LineArt.exe." In Windows operating systems, the filename extension ".exe" stands for "executable." Executable files contain machine code that can be executed directly by a processor (CPU).



Syntax Error

Computer Science: An invalid use of the grammatical rules governing the structure of programming statements. Programs that contain syntax errors cannot be compiled.

In VB, statements that contain syntax errors are displayed in bright red.

```
For I = 0 To 120 Stp 10
```

The VB keyword "**Step**" is misspelled. Therefore, this statement contains a ***syntax error***. Another way of expressing this is to say that the statement is ***syntactically invalid***.

Logic Error

Computer Science: A programming error that causes a program to behave in an unexpected or unpredictable manner (i.e. a bug).

Unfortunately, the VB software development environment (or any other for that matter) cannot detect and correct logic errors. The only way to correct such errors is to use your problem solving skills in conjunction with the debugging strategies that we shall be learning throughout this course.

Design-Time Error
Computer Science: A programming error that occurs while a program is being designed (which obviously means that the program is ***not*** running).

Run-Time Error
Computer Science: An error that occurs while a program is executing. Run-time errors are also known as ***exceptions***.

Argument
Mathematics: A variable in a logical or mathematical expression whose value determines the value of the dependent variable. For instance, if $f(x) = y$, then x is called the ***independent variable*** or the ***argument***.
Computer Science: A value that is passed to a function or some other programming construct. For example, the RGB function in VB takes three arguments, each of which is an integer from 0 to 255 inclusive.

The "For … Next" Statement

Syntax

The "**For … Next**" statement repeats a group of statements a specified number of times.

```
For counter = start To end [Step step]
    [statements]
    [Exit For]
    [statements]
Next [counter]
```

The square brackets are used to denote ***optional arguments*** or ***statements***.

The "**For …Next**" statement syntax has these parts:

Part	Description
counter	**Required**. Numeric variable used as a loop counter. The variable cannot be a **Boolean** or an array element.
start	**Required**. Initial value of *counter*.
end	**Required**. Final value of *counter*.
step	**Optional**. Amount *counter* is changed each time through the loop. If not specified, *step* defaults to one.
statements	**Optional**. One or more statements between **For** and **Next** that are executed the specified number of times.

Remarks

The *step* argument can be either positive or negative. The value of the *step* argument determines loop processing as follows:

Value	Loop executes if
Positive or 0	*counter* <= *end*
Negative	*counter* >= *end*

After all statements in the loop have executed, *step* is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute initially), or the loop is exited and execution continues with the statement following the **Next** statement.

Tip

Changing the value of *counter* while inside a loop can make it more difficult to read and debug your code.

Any number of **Exit For** statements may be placed anywhere in the loop as an alternate way to exit. **Exit For** is usually used after the evaluating of some condition, for example **If...Then**, and transfers control to the statement immediately following **Next**.

You can nest **For...Next** loops by placing one **For...Next** loop within another. Give each loop a unique variable name as its *counter*. The following construction is correct:

```
For I = 1 To 10
  For J = 1 To 10
    For K = 1 To 10
      ...
    Next K
  Next J
Next I
```

Note: If you omit *counter* in a **Next** statement, execution continues as if *counter* were included. If a **Next** statement is encountered before its corresponding **For** statement, an error occurs.

Line Method

Draws lines and rectangles on an object.

Syntax

object.**Line** [**Step**] (*x1*, y1) [**Step**] – (*x2*, *y2*)**,** [*colour*]**,** [**B**][**F**]

This symbol ***is not*** a subtraction sign! In the context of the "**Line**" method, the "–" symbol means "to." A line is drawn ***from*** the point with co-ordinates (*x1*, y1) ***to*** the point with co-ordinates (*x2*, *y2*).

The **Line** method syntax has the following object qualifier and parts:

Part	Description
object	***Optional***. Object expression that evaluates to a **Form** object, **PictureBox** control, **Printer** object, **Printers** collection, **Forms** collection, **PropertyPage** object, **UserControl** object or a **UserDocument** object. If *object* is omitted, the **Form** with the ***focus*** is assumed to be ***object***.
Step	***Optional***. Keyword specifying that the starting point co-ordinates are relative to the current graphics position given by the **CurrentX** and **CurrentY** properties.
(*x1*, *y1*)	***Optional***. **Single** values indicating the coordinates of the starting point for the line or rectangle. The **ScaleMode** property determines the unit of measure used. If omitted, the line begins at the position indicated by **CurrentX** and **CurrentY**.
Step	***Optional***. Keyword specifying that the end-point co-ordinates are relative to the line starting point.
(*x2*, *y2*)	***Required***. **Single** values indicating the co-ordinates of the end-point for the line being drawn.
colour	Optional. **Long** integer value indicating the RGB colour used to draw the line. If omitted, the **ForeColor** property setting is used. You can use the **RGB** function or **QBColor** function to specify the colour.
B	***Optional***. If included, causes a box to be drawn using the coordinates to specify opposite corners of the box.
F	***Optional***. If the **B** option is used, the **F** option specifies that the box is filled with the same colour used to draw the box. You cannot use **F** without **B**. If **B** is used without **F**, the box is filled with the current **FillColor** and **FillStyle**. The default value for **FillStyle** is transparent.

Remarks

To draw connected lines, begin a subsequent line at the end point of the previous line.

The width of the line drawn depends on the setting of the **DrawWidth** property. The way a line or box is drawn on the background depends on the setting of the **DrawMode** and **DrawStyle** properties.

When **Line** executes, the **CurrentX** and **CurrentY** properties are set to the end point specified by the arguments. This method cannot be used in a "**With**…**End**" block.

Circle Method

Draws a circle, ellipse or arc on an object.

Syntax

object.**Circle** [**Step**] (*x*, *y*)**,** *radius*, [*colour*, *start*, *end*, *aspect*]

The **Circle** method syntax has the following object qualifier and parts:

Part	Description
object	***Optional***. Object expression that evaluates to a **Form** object, **PictureBox** control, **Printer** object, **Printers** collection, **Forms** collection, **PropertyPage** object, **UserControl** object or a **UserDocument** object. If *object* is omitted, the **Form** with the ***focus*** is assumed to be ***object***.
Step	***Optional***. Keyword specifying that the center of the circle, ellipse or arc is relative to the current coordinates given by the **CurrentX** and **CurrentY** properties of *object*.
(*x*, *y*)	***Required***. **Single** values indicating the coordinates for the center point of the circle, ellipse or arc. The **ScaleMode** property of *object* determines the units of measure used.
radius	***Required***. **Single** value indicating the radius of the circle, ellipse or arc. The **ScaleMode** property of *object* determines the unit of measure used.
colour	***Optional***. **Long** integer value indicating the RGB colour of the circle's outline. If omitted, the value of the **ForeColor** property is used. You can use the **RGB** function or **QBColor** function to specify the colour.
start, *end*	***Optional***. Single-precision values. When an arc or a partial circle or ellipse is drawn, *start* and *end* specify (in radians) the beginning and end positions of the arc. The range for both is –2 π radians to 2 π radians. The default value for *start* is 0 radians; the default for *end* is 2 π radians.
aspect	***Optional***. Single-precision value indicating the aspect ratio of the circle. The default value is 1.0, which yields a perfect (non-elliptical) circle on any screen.

Remarks

To fill a circle, set the **FillColor** and **FillStyle** properties of the object on which the circle or ellipse is drawn. Only a closed figure can be filled. Closed figures include circles, ellipses or pie slices (arcs with radius lines drawn at both ends).

When drawing a partial circle or ellipse, if *start* is negative, **Circle** draws a radius to *start*, and treats the angle as positive; if *end* is negative, **Circle** draws a radius to *end* and treats the angle as positive. The **Circle** method always draws in a counter-clockwise (positive) direction.

The width of the line used to draw the circle, ellipse, or arc depends on the setting of the **DrawWidth** property. The way the circle is drawn on the background depends on the setting of the **DrawMode** and **DrawStyle** properties.

When drawing pie slices, to draw a radius to angle 0 (giving a horizontal line segment to the right), specify a very small negative value for *start*, rather than zero.

You can omit an argument in the middle of the syntax, but you must include the argument's comma before including the next argument. If you omit an optional argument, omit the comma following the last argument you specify.

When **Circle** executes, the **CurrentX** and **CurrentY** properties are set to the centre point specified by the arguments.

This method cannot be used in a "**With**…**End**" block.

PSet Method

Sets a point (pixel) on an object to a specified colour.

Syntax

object.**PSet** [**Step**] (*x*, *y*), [*colour*]

The **PSet** method syntax has the following object qualifier and parts:

Part	Description
object	***Optional***. Object expression that evaluates to a **Form** object, **PictureBox** control, **Printer** object, **Printers** collection, **Forms** collection, **PropertyPage** object, **UserControl** object or a **UserDocument** object. If *object* is omitted, the **Form** with the ***focus*** is assumed to be ***object***.
Step	***Optional***. ***Keyword*** specifying that the coordinates are relative to the current graphics position given by the **CurrentX** and **CurrentY** properties.
(*x*, *y*)	***Required***. **Single** values indicating the horizontal (*x*-axis) and vertical (*y*-axis) coordinates of the point to set.
colour	***Optional***. **Long** integer value indicating the **RGB** colour specified for point. If omitted, the current **ForeColor** property setting is used. You can use the **RGB** function or **QBColor** function to specify the colour.

Remarks

The size of the point drawn depends on the setting of the **DrawWidth** property. When **DrawWidth** is 1, **PSet** sets a single pixel to the specified colour. When **DrawWidth** is greater than 1, the point is centred on the specified coordinates.

The way the point is drawn depends on the setting of the **DrawMode** and **DrawStyle** properties.

When **PSet** executes, the **CurrentX** and **CurrentY** properties are set to the point specified by the arguments.

To clear a single pixel with the **PSet** method, specify the coordinates of the pixel and use the **BackColor** property setting as the *colour* argument.

This method cannot be used in a "**With**…**End**" block.

RGB Function

Returns a "**Long**" whole number representing an RGB (red, green, blue) colour value.

Syntax

RGB(*red*, *green*, *blue*)

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

Part	Description
red	Required; **Variant (Integer)**. Number in the range 0–255 inclusive that represents the **red** component of the colour.
green	Required; **Variant (Integer)**. Number in the range 0–255 inclusive that represents the **green** component of the colour.
blue	Required; **Variant (Integer)**. Number in the range 0–255 inclusive that represents the **blue** component of the colour.

Remarks

Application methods and properties that accept a colour specification expect that specification to be a number representing an RGB colour value. An RGB colour value specifies the relative intensity of red, green and blue to cause a specific colour to be displayed.
The value for any argument to **RGB** that exceeds 255 is assumed to be 255.
The following table lists some standard colours and the red, green and blue values they include:

Colour	Red Value	Green Value	Blue Value
Black	0	0	0
Blue	0	0	255
Green	0	255	0
Cyan	0	255	255
Red	255	0	0
Magenta	255	0	255
Yellow	255	255	0
White	255	255	255

Using Colour Properties

Many of the controls in Visual Basic have properties that determine the colours used to display the control. Keep in mind that some of these properties also apply to controls that are not graphical. The following table describes the colour properties.

Property	Description
BackColor	Sets the background colour of the form or control used for drawing. If you change the "**BackColor**" property after using graphics methods to draw, the graphics are erased by the new background colour.
ForeColor	Sets the colour used by graphics methods to create text or graphics in a form or control. Changing "**ForeColor**"does not affect text or graphics already created.
BorderColor	Sets the colour of the border of a shape control.
FillColor	Sets the colour that fills circles created with the "**Circle**" method and boxes created with the "**Line**" method.

Defining Colours

The colour properties can use any of several methods to define the colour value. The "RGB" and "QBColor" functions described above are two different ways. This section discusses two other methods.

- Using defined constants
- Using direct colour settings

Using Direct Colour Settings

Using the RGB function or the intrinsic constants to define colour are indirect methods. They are indirect because Visual Basic interprets them into the single approach it uses to represent colour. If you understand how colours are represented in Visual Basic, you can assign numbers to colour properties and arguments that specify colour directly. In most cases, it's much easier to enter these numbers in hexadecimal form.

The valid range for a normal RGB colour is 0 to 16,777,215 (&HFFFFFF&). Each colour setting (property or argument) is a 4-byte integer. The high byte of a number in this range equals 0. The lower 3 bytes, from least to most significant byte, determine the amount of red, green and blue, respectively. The red, green and blue components are each represented by a number between 0 and 255 (&HFF).

Consequently, you can specify a colour as a hexadecimal number using this syntax: *&HRRGGBB&*.

e.g. txtName.BackColor = &HFF0000&

The *BB* specifies the amount of blue, *GG* the amount of green, and *RR* the amount of red. Each of these fragments is a two-digit hexadecimal number from 00 to FF. The median value is 80. Thus, the following number specifies grey, which has the median amount of all three colours:

&H808080&

Setting the most significant bit to 1 changes the meaning of the colour value: It no longer represents an RGB colour, but an environment-wide color specified through the "Windows Control Panel." The values that correspond to these system-wide colours range from &H80000000& to &H80000015&.

Note: Although you can specify over 16 million different colours, systems with old video cards may not be able to display them accurately.

Using Defined Constants

You do not need to understand how colour values are generated if you use the intrinsic constants listed in the "Object Browser." In addition, intrinsic constants do not need to be declared. For example, you can use the constant "**vbRed**" whenever you want to specify red as a colour argument or colour property setting:

e.g. txtName.BackColor = vbRed

The tables below summarize the defined intrinsic colour constants available in VB.

Colours

Constant	*Value*	*Description*
vbBlack	&H0	Black
vbRed	&HFF	Red
vbGreen	&HFF00	Green
vbYellow	&HFFFF	Yellow
vbBlue	&HFF0000	Blue
vbMagenta	&HFF00FF	Magenta
vbCyan	&HFFFF00	Cyan
vbWhite	&HFFFFFF	White

System Colours

Constant	*Value*	*Description*	*Constant*	*Value*	*Description*
vbScrollBars	&H80000000	Scroll bar color	**vbHighlightText**	&H8000000E	Text color of items selected in a control
vbDesktop	&H80000001	Desktop color	**vbButtonFace**	&H8000000F	Color of shading on the face of command buttons
vbActiveTitleBar	&H80000002	Color of the title bar for the active window	**vbButtonShadow**	&H80000010	Color of shading on the edge of command buttons
vbInactiveTitleBar	&H80000003	Color of the title bar for the inactive window	**vbGrayText**	&H80000011	Grayed (disabled) text
vbMenuBar	&H80000004	Menu background color	**vbButtonText**	&H80000012	Text color on push buttons
vbWindowBackground	&H80000005	Window background color	**vbInactiveCaptionText**	&H80000013	Color of text in an inactive caption
vbWindowFrame	&H80000006	Window frame color	**vb3DHighlight**	&H80000014	Highlight color for 3D display elements
vbMenuText	&H80000007	Color of text on menus	**vb3DDKShadow**	&H80000015	Darkest shadow color for 3D display elements
vbWindowText	&H80000008	Color of text in windows	**vb3DLight**	&H80000016	Second lightest of the 3D colors after **vb3Dhighlight**
vbTitleBarText	&H80000009	Color of text in caption, size box, and scroll arrow	**vb3DFace**	&H8000000F	Color of text face
vbActiveBorder	&H8000000A	Border color of active window	**vb3Dshadow**	&H80000010	Color of text shadow
vbInactiveBorder	&H8000000B	Border color of inactive window	**vbInfoText**	&H80000017	Color of text in ToolTips
vbApplicationWorkspace	&H8000000C	Background color of multiple-document interface (MDI) applications	**vbInfoBackground**	&H80000018	Background color of ToolTips
vbHighlight	&H8000000D	Background color of items selected in a control			

Summary of Unit 1

1. The ***most important lesson*** of the entire course is _____

   _____

   _____.

2. If you choose to ignore the most important lesson of the course, you are likely to _____

   _____.

3. ***George Polya's four steps*** of problem solving are _____

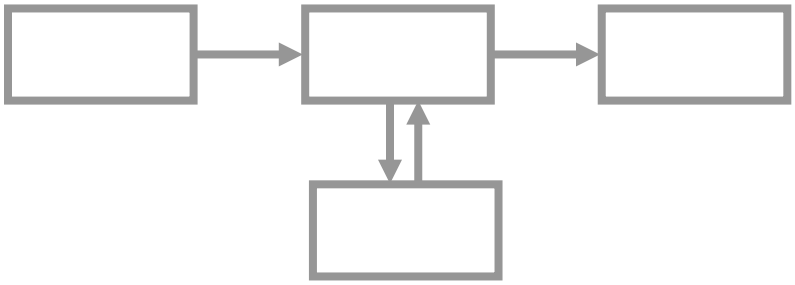
   _____.

4. A ***program*** is _____.

   Code is _____.

   A ***programming language*** is _____.

   An ***algorithm*** is _____.

5. 

6. For ***two-dimensional computer graphics***, the origin of the Cartesian co-ordinate system is located at the _____ of the screen. The ***x*-axis** is _____ and runs along the very _____ of the screen. The *x*-co-ordinates increase as we move toward the _____. The ***y*-axis** is _____ and runs along the _____ of the screen. The *y*-co-ordinates increase as we move _____ the screen. The orientation of the ***y*-axis** is somewhat unusual because we are accustomed to seeing it _____ in math class. For ***three-dimensional computer graphics*** everything is the same as in the two-dimensional case except that there is an additional _____ called the _____ that is directed _____ the screen.

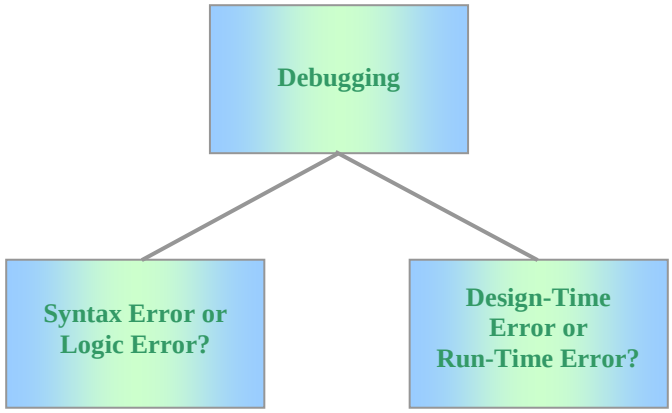
7. To draw a line segment in VB, we can use the _____ method. This method requires two pieces of information, namely the _____ of the line segment. To draw a circle in VB, we can use the _____ method. This method also requires two pieces of information, the _____ of the circle and the _____ of the circle.

8. When we access a property or method, we use the formats ***objectName.propertyName*** or ***objectName.methodName***. In the case of a ***form object***, we can use the word _____ as a substitute for the name of the form. This is helpful because no changes in the VB _____ are required if we ever decide to change the name of the form.

9. A ***variable*** is _____.
If "**`Option Explicit`**" is used, all variables must be _____. This means that the _____ and the _____ of the variable are specified using the _____ statement. Although in VB it is possible to use variables without declaration, it is a very bad idea because _____
_____.

10. An ***object*** is a collection of _____ and _____. By ***convention***, object names should begin with suggested prefixes such as _____ for command buttons and _____ for forms. Also by convention, "CamelCase" is used for variable and object names. This makes the names much easier to _____. For example, it is much easier to _____ the variable name "NumberOfGamesSold" than the name "numberofgamessold."

11. ***Debugging*** a program means to _____. The VB development environment has several features and tools that help us to ***debug*** our programs. For example, a _____ can be set by clicking in the left margin of the code editor window. This allows the programmer to _____. In addition, VB displays ***syntactically invalid*** statements in bright _____. Unfortunately, _____ _____ cannot be detected by VB. These errors will manifest themselves while a program is _____.



12. The CPU (central processing unit) of a computer is also known as a _____. CPUs cannot execute programs written in higher level languages such as Visual Basic, C++ and Java. Therefore, higher level language _____ code must be translated into _____ code using a special program called a _____.

13. Unless an object's "AutoRedraw" property is set to "**`True`**," any image that is drawn on the object will _____
_____.

14. We used the _____, _____ and _____ properties to create a more convenient 100×100 grid on a form or a picture box. By default, distance on an object is measured using a tiny unit called a _____, which is equal to _____. Distance can also be measured using _____, _____, _____, _____, _____ and _____.

15. To avoid using a massive number of identical or similar statements, a programming concept called ***repetition*** is used. In this unit, we have learned to use _____ loops to implement this concept in VB.

16. In VB, certain objects are called ***controls*** because _____.

17. Whenever it is necessary to process a reasonably large number of controls all of which are of the same type, it is convenient to use a _____. The advantage of using such a structure is that each control will have the same _____, making it possible to access all the controls using a single statement. Since all the controls in such a structure have the same name, it is necessary to have a method of distinguishing one from another. To do this, an integer value called an _____ or a _____ is used. The _____ of an ***element*** of a ***control array*** is similar to the _____ in a person's address.

18. VB programs are subdivided into structures know as _____, or simply _____ for short. (See question 19 for a hint.)

19. The "Subs" that we have encountered in VB so far are known as ***event procedures*** because _____ _____. Such subs (short for "subroutine") are named automatically by joining the name of the _____ to an _____, which is then joined to the name of the _____. The VB event monitor waits for an _____ to occur on a particular _____. When this happens, the corresponding _____ is called, which means that certain code is executed.

20. An easy way to specify a colour in VB is to use the _____ function. The RGB function requires three _____, each of is an integer ranging from _____ to _____. In binary, these two values are represented by the eight-bit codes _____ and _____.

21. The purpose of a video card (also known as a graphics card or a graphics adaptor) is to _____ _____. Graphics cards require RAM (random access memory) because _____ _____. Among other capabilities, video cards can be used to set the _____ _____ of a monitor. This determines the number of _____ in each row and each column. The total number of _____ on the screen is determined by multiplying the _____ _____.

22. There are four properties of objects that deal with colour, namely _____, _____, _____ and _____. Colours can be specified in a variety of ways. VB has built-in _____ such as "`vbWhite`" for commonly used colours. For other colours, it is easy to specify the required colour by using the _____ function.

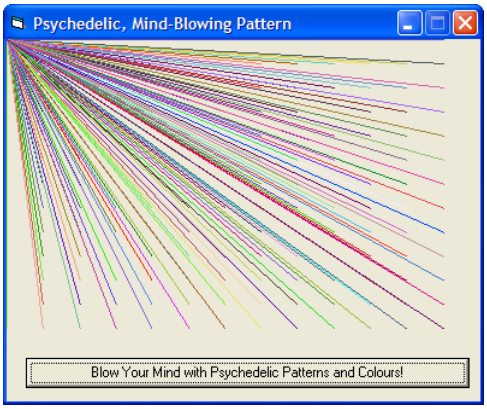
23. A loop within another loop is called a _____ loop. If the outer loop repeats *m* times and the inner loop repeats *n* times, then the total number of repetitions is _____.

24. In computer science, the term ***argument*** refers to _____. This is similar to the mathematical meaning of this term, which is _____.

25. In technical documents about programming, square brackets are used to denote _____ _____. The square brackets should _____ be used in the actual code. They simply mean that the programmer may include the arguments or statements if he/she ***wishes*** to do so. It is not _____ to include items that are enclosed in square brackets.

Enhancement Problems

1. The following example will help you to begin learning about nested loops. In addition, you will see the "RGB" function and the "Line" method in action. Study the code and then answer questions (a) to (g).




```
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'You can use this program to learn about nested loops
'and generating random integers. The random integers
'are used to generate random colours.
'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Option Explicit

Private Sub cmdBlowYourMind_Click()
    Dim I As Integer, J As Integer
    Dim ShadeOfRed As Integer
    Dim ShadeOfGreen As Integer
    Dim ShadeOfBlue As Integer
    For I = 0 To 120 Step 10
        For J = 0 To 120 Step 10
            ShadeOfRed = Int(Rnd * 256)
            ShadeOfGreen = Int(Rnd * 256)
            ShadeOfBlue = Int(Rnd * 256)
            Me.ForeColor = RGB(ShadeOfRed, _
                               ShadeOfGreen, ShadeOfBlue)
            Me.Line (0, 0)-(I, J)
        Next J
    Next I

End Sub
```

Questions

(b) You will find a copy of this program on the "Courses" page of www.misternolfi.com or in the following folder: **I:\Out\Nolfi\Ics3mo\Drawing, Graphics, Game Program Examples\Psychedelia**

(c) Run the program and click on the "Blow Your Mind…" button a few times. Then minimize the program and immediately restore it. Repeat the above steps but this time, set the "**AutoRedraw**" property of the form to "**False**." What do you notice? Write a brief explanation of the "AutoRedraw" property.

(d) Using a piece of graph paper, explain the order in which this program plots the lines. Do not forget to label the axes and to orient them in the same manner as they are oriented on the screen. In addition, be sure to check the values of the **ScaleWidth** and **ScaleHeight** properties before you scale your axes.

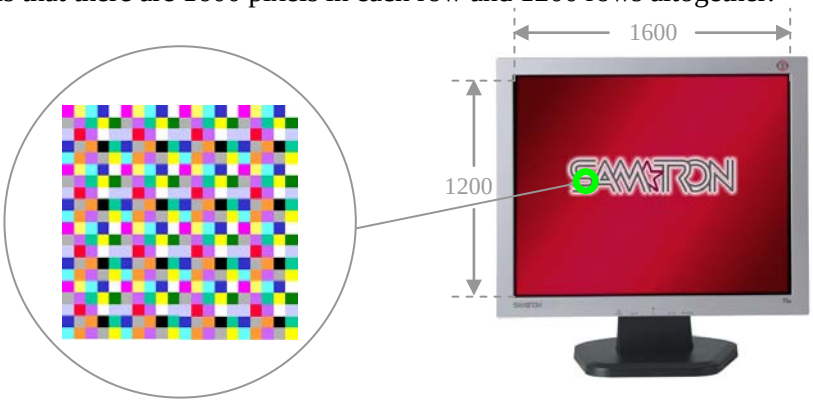
(e) Explain the purpose of the "**ScaleMode**" property.

(f) The RGB function requires integer ***arguments*** in the range 0 to 255 inclusive. Explain how this program uses the "Int" and the "Rnd" functions to generate random integers between 0 and 255.

(g) Modify the above program so that a Timer object is used to change the pattern instead of a command button. (See page 17 for more details).

Questions 2, 3 and 4 deal with how a display screen (i.e. a monitor screen) is organized into ***pixels***. The colour of each pixel (short for "picture element") is determined by a ***binary code*** (sequence of zeros and ones). The ***video card*** (also known as ***graphics card*** or ***graphics adaptor***) sets the colour of each pixel according to the binary code stored for each pixel. The ***screen resolution*** determines the number of pixels in each row and the total number of rows of pixels. For example, a screen resolution of 1600 × 1200 means that there are 1600 pixels in each row and 1200 rows altogether.

This is a ***magnified*** section of an image on a computer monitor screen. Each small square is called a ***pixel*** ("picture element"), which is the smallest addressable segment of the picture. The colour of each pixel is determined by a binary code.



2. How many different colours can be displayed by a video card set to each of the following modes? In each case, show your work. (The first one is done for you as an example.)

(a) Four-bit colour

The possible binary codes are
0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111.
Therefore, only 16 colours can be displayed using 4-bit colour. (Shortcut: $2^4 = 16$)

(b) Eight-bit colour

(c) Sixteen-bit colour

(d) Twenty-four bit colour

(e) Thirty-two bit colour

(f) Sixty-four bit colour

3. How much memory is required to display a full-screen image if the video card is set to a screen resolution of 800 pixels × 600 pixels and to 32-bit colour mode? Show your work! (**Note:** 1 KB=1024 bytes, 1 MB=1024 KB)

What is the largest screen resolution possible if a video card with 32 MB of RAM is set to 32-bit colour mode? Show your work!