3 Data

3.1 DATA TYPES

In programming we use different types of data. In day-to-day experiences we don't give a lot of thought to what kind of data we are using. If we are discussing the name and age of a person we might use values like *Josh* is eighteen years old, but in programming we must state what type of data *Josh* and the number 18 are so the values can be input and/or stored in the program. Visual Basic requires all values that are stored or input to be classified into one of the following data types:

Data Type	Memory Used	Value Range	Category	Example
Byte	1 byte	0 to 255 (unsigned)	numeric	27
Short	2 bytes	-32,768 to 32,767 (signed)	numeric	27000
Integer	4 bytes	-2,147,483,648 to 2,147,483,647 (signed)	numeric	27000000
Single	4 bytes	1.401298E-45 to 3.4028235E+38 for positive values and -3.4028235E+38 to -1.401298E-45 for negative values	numeric	27.5
Long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (9.2...E+18) (signed)	numeric	27.5E+12
Double	8 bytes	4.94065645841246544E-324 to 1.79769313486231570E+308 for positive values -1.79769313486231570E+308 to -4.94065645841246544E-324 for negative values	numeric	27.5E+200
Decimal	16 bytes	+/-79,228,162,514,264,337,593,543,950,335	numeric	27 billion
Char	2 bytes	A single character	non-numeric	"D"
String	Varies	One or more characters	non-numeric	"DOG"
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999 ***Now*** stores current date and time	non-numeric	12:23:00 on December 25, 2012
Boolean	2 bytes	True or False	logical	True
Object	4 bytes	Stores memory address of an object	object	

3.1.1 String

We have already worked with the String data type. In Chapter 2, we saw various uses of strings being printed in a **MessageBox**, a **TextBox**, and a **ListBox**. The String values can contain any characters—numeric or non-numeric, or a combination of these two—as long as it is enclosed in quotes.

Some examples of Strings:

"Kara"
"Julianna!"
"12 Smith Road"
"Patchogue, NY"
"744-2536"

3.1.2 Char

A subset of the String data type is a Char. A Char value is any single character. It can be a letter, number, or any special character including a space. In Visual Basic, Char values are enclosed in double quotes.

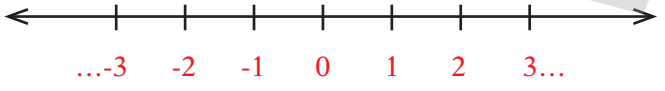
Some examples of Char values:

"A"
"@"
"7"
" "—this is a space in-between the double quotes

3.1.3 Integer

Numbers can be stored in two different categories. The Integer category includes positive and negative whole numbers (… -3, -2, -1, 0, 1, 2, 3 …).

That is, the whole numbers and their opposites.



Examples:

0 36 -218 -8 -2e4 7e5 4.12e6

These are numbers expressed in
an exponential or scientific notation.

-2e4 = -20000 7e5 = 700000 4.12e6 = 4120000

3.1.4 Double

The Double data type refers to real numbers. The real number category includes rational and irrational numbers that can be expressed using a decimal representation (3.14, 19.999999).

Examples:

7.0 -3.25 0.0 24.5 123.45 -15.2e2 -4.7e-3 3e-5

Note: -15.2e2 = -1520 -4.7e - 3 = -.0047 3e - 5 = .00003

Knowing the appropriate data type to store data is an important part of programming. For example if you were storing a value for your name ("Julianna") you would use a String. If you were storing a person's middle initial ("A") you would store as a Char. String and Char values MUST be enclosed in double quotes to distinguish them from variable names.

If you were storing Julianna's age you would use a Byte, Short, or Integer data type since age will be a whole number value. If you were storing someone's salary you would store the value as a Single or Double data type because currency or money values include a decimal expression.

In this book, the examples given will use the following:

String—for any non-numeric values
Char—for any single character
Integer—for any whole numbers
Double—for any values that would contain a decimal

EXERCISES

Determine which data type is best to use to store the following.

1. A body temperature
2. Your birthday
3. The area of a circle
4. The air temperature
5. Gender (M or F)
6. A 5-digit zip code
7. A town's name
8. A telephone number with hyphens
9. The price of an item
10. The number of students in a classroom

11. The grade (out of 100) on your next exam
12. Your average for the year
13. Your grade as either A, B, C, D, or F
14. An area code
15. Your salary
16. Your hourly rate for your job

3.2 VARIABLES

In algebra, quantities are referred to by names which we call variables. We use common variable names like x, y, z, a, b, etc., to represent numeric values. In programming, a variable is an item of data. The shortest variable name would be a single letter; however, using full names is preferred to describe the data. There are certain rules and guidelines for naming variables.

- Variable name must use a letter as the first character.
- Variable name can include any letters, numbers and underscores after the first character.
- Variable name can't use a space, period (.), exclamation mark (!), or non-numeric characters such as @, &, $, # in the name.
- Although variable names can contain hundreds of characters, thirty-two is the recommended maximum number of characters to use.
- Meaningful names instead of single characters are good practice.
- Variable name should not be a keyword (see Appendix B).
- Variable names are not case sensitive (*num* and *NUM* would be considered the same variable).

Some examples of variable names are:

x ab clue x3 dog name gpa full_name r2d2
first_name last_name grade_on_test final_exam_grade

**note the use of the underscore in the above*

firstName lastName gradeOnTest finalExamGrade

**note the use of lowercase letters with an uppercase letter in the above*

These are illegal variable names:

4tune	Dim	x+y	first name
begins with a number	*keyword*	*contains a + sign*	*contains a space*

A variable is a name given by the programmer to a memory storage location. To use a variable in a program, there must be space allocated in the computer RAM to hold the value assigned to that variable. This is accomplished by initially declaring a variable before it is used in the program. It actually allocates memory to hold the value of the variable.

3.3 DECLARING VARIABLES

Declaration Statements are programming statements that are used for giving a variable a name and designating what type of data will be stored in it. When we declare a variable using a Dim (short for *Dimension*) statement, we allocate a piece of memory. In most cases this memory has the name of the variable, and can store a certain type of data that you specify. Declaration statements have the general syntax:

Dim *variable-name* **As** *data-type*

Examples:

```
Dim name As String
Dim gender As Char
Dim age As Integer
Dim salary As Double
```

If you choose to declare multiple variables in a single Dim statement you can do so, the one rule is that all variables declared in that line MUST be the same data type.

Dim *variable-name1, variable-name2…variable-nameN* **As** *data-type*

Examples:

```
Dim firstname, lastname As String
Dim age, weight As Integer
Dim payrate, raise, totalsalary As Double
```


The first time a value is stored in a variable, the variable is said to be initialized. In Visual Basic, by default, all numeric variables are initialized to zero and all String and Char values are initialized to zero-length strings (strings containing no characters); however, it is good practice to initialize all variables before they are used. In many other languages, numeric variables are not necessarily initialized to zero, nor strings to be empty. To gain experience in programming, initialize all variables before they are used in program statements.

Examples:

```
name = "" '-to initialize an empty string
age = 0 '-to initialize an integer to 0
salary = 0.0 '-to initialize a double to 0
```

A variable can also be initialized in the declaration statement using the syntax:

Dim *variable-name* **As** *data-type* = *value*

Examples:

```
Dim name As String = ""
Dim age As Integer = 0
Dim salary As Double = 0.0
```

EXERCISES

1. Which of the following are legal or illegal variable names? If illegal, state the reason why.

x	ab	cat	money$	a-b
x3	last bal	r2d2	sum.of	c3PO
of$er	2day	name	tax#	3po2

2. Write a declaration statement to declare the variable *tax* as a Double.

3. Write a declaration statement to declare the variable *day* as a String.

4. Write a declaration statement to declare the variable *total* as an Integer.

5. Write a declaration statement to declare the variables *first* and *last* as strings, using only one Dim statement.

6. Write a declaration statement to declare the variables *a*, *b*, and *c* as integers, using only one Dim statement.

7. Write the statement that will declare the variable *x* and assign a value of 57.

8. Write a statement that will assign the value stored in *x* into the variable *y*.

9. Write a statement that will assign the person's name (a String) of Kara to the variable *firstname*.

3.3.1 Constants

There is sometimes a need to protect the value of a variable throughout a program which means that once the variable is declared and assigned a value it is impossible to change the value throughout the program execution. These types of "protected" variables are referred to as constants. The syntax to declare them is different from other variable dimension statements. The word Const is used in place of Dim and the variable MUST be initialized to a value.

Const *variable-name* **As** *data-type* = *value*

The keyword Const indicates that you are declaring a constant, and the value following the equal sign indicates the value that it is assigned to permanently throughout the program execution.

Examples:

```
Const TAX As Double = 0.08645
Const PI As Double = 3.14159
Const HOURS As Integer = 40
Const AGE As Integer = 21
Const NAME As String = "Visual Basic"
```


Unlike a variable, a constant is generally named by using uppercase letters. The value of the constant will not change during the run of the program. Any attempt to modify the constant variable value throughout the program will result in an error. If you use a variable in your program code and the value never changes, you should consider storing the value as a constant.

There are several advantages of using constants. They can make the program code more readable. They can also save memory, since constants take less space. They also make program-wide changes easier to accomplish.

3.4 ARITHMETIC OPERATIONS

Visual Basic uses the arithmetic operations we are all familiar with. Some of the operator symbols are different from those we learned years ago. There are also a few additional operators that we will see.

Addition: takes the common symbol of +

```
12 + 3
```

Subtraction: takes the common symbol of -

```
8 - 5
```

Multiplication: in arithmetic, there are many ways to indicate multiplication, but in VB there is only one operator that will indicate multiplication, the symbol *

```
12 * 4
```

Division: Again, in VB, there is only one symbol that indicates division—the symbol /

```
20 / 4
```

Exponents: Since we cannot place an exponent in the upper portion of a line, in VB to indicate an exponent, we use the symbol ^ . To write 2^3 we would type the codes in as:

```
2 ^ 3
```

Parentheses: They have the same meaning as in arithmetic. We perform the operation(s) inside the parentheses first.

```
(4 + 2) - (8 - 3)
   add 4 and 2, subtract 8 and 3
   then subtract 6 and 5 thus producing 1
```

Two additional operators are available in Visual Basic.

They are **Integer Division** and **Mod**.

Integer Division: This operator will return only the integer portion of the quotient of two integers. The symbol for integer division is a backward slash \ . The following are comparisons of normal division and integer division:

Example:

20 / 8 = 2.5 while 20 \ 8 = 2
100 / 80 = 1.25 while 100 \ 80 = 1
20 / 4 = 5 while 20 \ 4 = 5
10 / 20 = .5 while 10 \ 20 = 0

Modulus Operator: The Modulus operator (Mod) returns the remainder of the quotient of two integers.

```
   1 r 4  <------  this number (4) is the modulus
8 | 12
```

Example:

12 Mod 8 = 4
10 Mod 3 = 1
20 Mod 5 = 0
10 Mod 15 = 10

3.4.1 Operator Precedence

Visual Basic follows the same rules of arithmetic that we have learned. These rules are called the order of operations. We commonly refer to this order as P E M D A S. Parentheses, Exponents, Multiplication or Division, Addition or Subtraction. These operations are performed left to right. Now, with Mod and \ included, the following chart shows the levels of priority for all of the above operations.

Priority Level	*Operator*	*Operator Name*
1st	**( )**	parentheses
2nd	**^**	exponents
3rd	*** , /**	multiplication or division
4th	****	integer division
5th	**Mod**	modulus (remainder)
6th	**+ , -**	addition or subtraction

EXERCISES

Calculate the values for the given expressions.

	Expression	*Value*
1.	24 / 4 * 3	_____
2.	24 * 4 * 3	_____
3.	24 - 4 + 3	_____
4.	5 + 2 * 4	_____
5.	10 / 2 * 5	_____
6.	12 / 10	_____
7.	12 \ 10	_____
8.	15 \ 20	_____
9.	15 Mod 2	_____
10.	18 Mod 7	_____
11.	15 Mod 20	_____
12.	3 ^ 2 + 4 * 5	_____
13.	(5 + 2) * -4	_____

14. 8 + 12 * 3 - 7 _____

15. 13 - 4 ^ 2 _____

16. 10 * (5 / 2) _____

17. 45 - (8 \ 3) ^ 3 * 5 _____

18. 8 + 13 Mod 5 _____

19. 16 Mod 9 \ 2 _____

3.5 ASSIGNMENT STATEMENTS

Assignments are statements that tell the computer to store a value into a variable. Assignment statements always have an equal (=) sign and one variable name immediately to the left of the equal sign. The right side of the equal sign can be a value or an expression. The value or expression on the right of the equal sign is assigned to the variable on the left of the equal sign. In other words, the value of the variable on the left side of the equal sign changes to become equal to the value on the right side.

In algebra we could see either

x = 3
or
3 = x

In Visual Basic, 3 = x is ***not*** a valid expression. An assignment statement is used to assign a value or resulting value of a calculation to a variable. The assignment statement has the following syntax:

variable = expression

Examples

```
x = 5                  'assigns the value of 5 to variable x
tax = .0675            'assigns the value of .0675 to the variable tax
x = y                  'assigns the value in y to variable x
y = x                  'assigns the value in x to the variable y
a = b + c              'assigns the value of the sum of b and c to a
x = "HELLO"            'assigns the string HELLO to variable x
a = "Visual Basic"     'assigns the string Visual Basic to the variable a
```

EXERCISES

Write the following as Visual Basic Statements.

1. a = 2xy
2. c = 3(a + b)
3. d = 5y2
4. e = (6)(8)
5. $f = \frac{81 + 82}{2}$
6. y = 3x - 7
7. A = lw
8. $A = s^2$
9. A = ½bh
10. A = ½ h (b1 + b2)
11. $A = \pi r^2$
12. d = rt
13. P = 2l + 2w
14. P = a + b + c
15. $Q = \frac{4}{5} + \frac{D}{2}$
16. $d = \frac{1}{2}at^2$

3.5.1 Type Conversions (Casting)

When an assignment statement is used, the expression on the right side of the = sign may need to be converted to the data type of the variable to the left of the = sign.

Function	Description
CBool()	Converts the expression to a Boolean (True or False) value. The expression must be a number. 0 represents False, any other value returns True, a string that represents a number or the strings "True" or "False".
CByte()	Converts a numeric or string expression to a Byte value. The value must be between 0-255. All fractional values are rounded.
CChar()	Converts a string expression into a char using the first character in the string.
CDate()	Converts a valid string expression to a Date.
CDbl()	Converts a numeric or string expression to a Double value. The expression must be numeric (contain only numbers and a decimal point).
CDec()	Converts a numeric or string expression to a Decimal value.
CInt()	Converts a numeric or string expression to an Integer value.
CLng()	Converts a numeric or string expression to a Long value. The expression must be numeric (contain only numbers and a decimal point).
CObj()	Converts an expression to an object.
CShort()	Converts a numeric or string expression to a Short value. The expression must be numeric.
CSng()	Converts a numeric or string expression to a Single value. The expression must be numeric (contain only numbers and a decimal point).
CStr()	Converts a numeric, Boolean, or Date expression to a String.

Examples:

CInt(*String*) casting a String to an Integer
CDbl(*String*) casting a String to a Double
CStr(*Integer or Double*) casting a numeric value to a String

note: CInt("123") becomes 123 'the number one-hundred-twenty-three (integer)
CStr(123) becomes "123" 'the string of characters 1 2 3

Although the above may look alike, they are different

**When using casting, as in the following:

CInt("212 Main Street")

an error will occur due to the fact that the compiler is unable to convert Main Street to a numeric value.

All data that is entered by the user through a **TextBox** or an **InputBox** (section 3.6) is a String value and if it is not to be stored as a String value it must be converted using one of the conversion functions.

Casting to an Integer

```
Dim age As Integer
age = CInt(txtAge.text)
```

Casting to a Double

```
Dim salary As Double
salary = CDbl(txtSalary.text)
```

Casting to a String

```
Dim age As Integer
Dim s As String
age = 21
s = CStr(age)
```


EXERCISES

1. Write the statement that will assign the String variable name to be the value input in txtName.

2. Write the statement that will assign the Integer variable grade to be the value input in txtGrade.

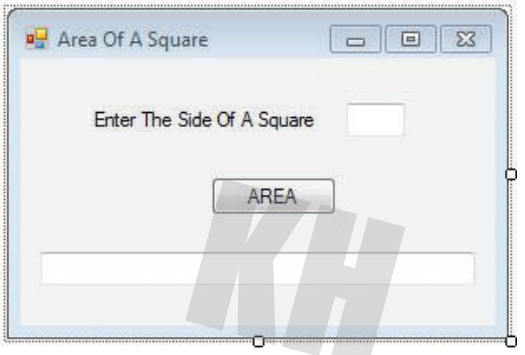
3. Write the statement that will assign the Double variable salary to be the value input in txtSalary.

Some Programming Examples

Example 1: Find the area of a square given the side.

The interface should contain a **TextBox** named txtIn for the input and a **TextBox** named txtOut for output. A **Label** should accompany the **TextBox** prompting the user. A **Button** named btnArea is also needed. You will also note that the *Text* property of **Form1** was changed to Area Of A Square.

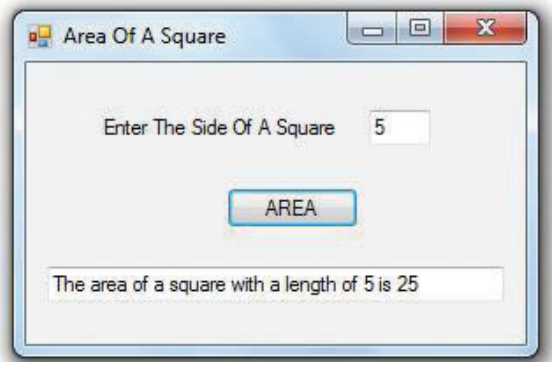
The interface:



The code for the above example:

```
'area of a square
Public Class Form1
    Private Sub btnArea_Click(...
        Dim side, area As Integer          'declare side and area as Integers
        side = CInt(txtIn.Text)            'cast value in TextBox to an Integer
        area = side ^ 2                    'calculate the area
        'display the results in a TextBox
        txtOut.Text = "The area of a square with a length of " & CStr(side) & " is " & CStr(area)
    End Sub
End Class
```

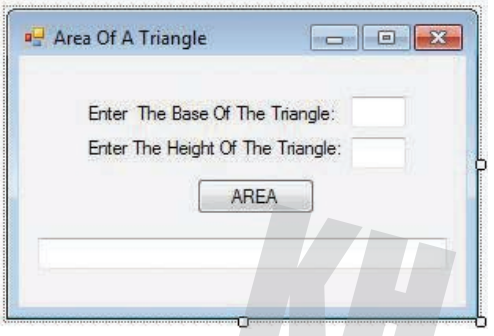
The result:



Example 2: Find the area of a triangle given the base and height.

The interface should contain two **TextBoxes** named txtBase and txtHeight for the input and a **TextBox** named txtOut for output. Two **Labels** should accompany each **TextBox** prompting the user. A **Button** named btnArea is also needed. You will also note that the *Text* property of **Form1** was changed to Area Of A Triangle.

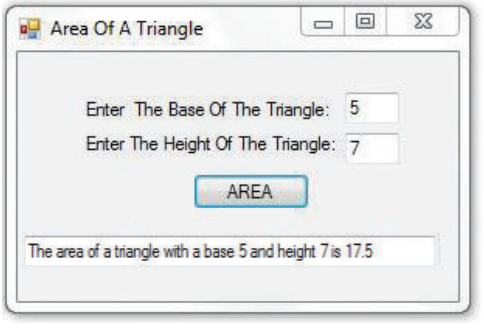
The interface:



The code:

```
'area of a triangle
Public Class Form1
    Private Sub btnArea_Click(...
        Dim base, height As Integer          'declare base and height as Integers
        Dim area As Double                   'since division by 2 could produce a decimal
        base = CInt(txtBase.Text)            'cast value in both TextBoxes to Integers
        height = CInt(txtHeight.Text)
        area = (base * height) / 2           'calculate the area
        'display the results in a TextBox
        txtOut.Text = "The area of a triangle with a base " & CStr(base) & " and height " &
        CStr(height) & " is " & CStr(area)
    End Sub
End Class
```

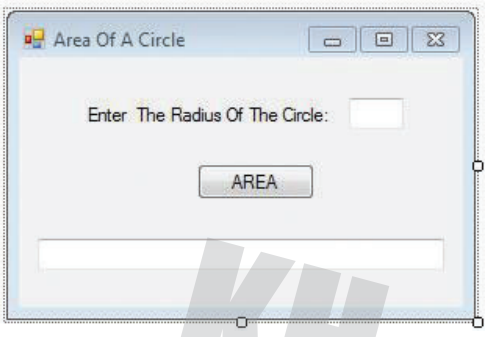
The results:



Example 3: Find the area of a circle given the radius.

The interface should contain a **TextBox** named txtRadius for the input and a **TextBox** named txtOut for output. A **Label** should accompany the **TextBox** prompting the user. A **Button** named btnArea is also needed. You will also note that the *Text* property of **Form1** was changed to Area Of A Circle.

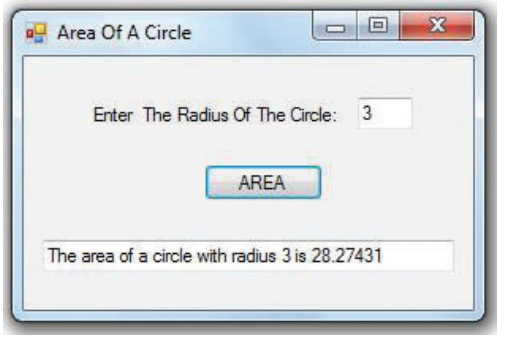
The interface:



The code:

```
'area of a circle
Public Class Form1
    Private Sub btnArea_Click(...
        Dim radius As Integer                'declare radius as Integer
        Const PI As Double = 3.14159         'declare PI a constant
        Dim area As Double                   'since multiplying by a decimal produces a decimal
        radius = CInt(txtRadius.Text)        'cast value in TextBox to Integer
        area = PI * radius ^ 2               'calculate the area
        'display the results in a TextBox
        txtOut.Text = "The area of a circle with radius " & CStr(radius) & " is " & CStr(area)
    End Sub
End Class
```

The result:



3.6 INPUT BOXES

As mentioned in section 2.3.7, input boxes are an alternative to text boxes for inputting values into your program. Like text boxes, all values input are String data types so if the input value is to be used as a number, character, or Boolean value it is necessary to convert the data in the **InputBox** using the Conversion Functions in section 3.5.1.

Here are samples of

The syntax for an **InputBox** is:

variable = InputBox(*prompt, title, default value*)

or

variable = Microsoft.VisualBasic.InputBox(*prompt, title, default value*)

prompt—is the instructions to the user
title—the header at the top of the window
default value—a value that will be input if none other is entered by the user

To create an **InputBox** to enter a value that is to be stored as an Integer:

```
Dim num As Integer
num = CInt(Microsoft.VisualBasic.InputBox("Enter a whole number", "Input number", 0))
```

or the shorter format

```
Dim age As Integer
age = CInt(InputBox("Enter your age"))
```

Note that the conversion function **CInt()** must be used to convert the String value that is being input into an Integer value.

To input a **Double** value:

```
Dim num As Double
num = CDbl(Microsoft.VisualBasic.InputBox("Enter a real number", "Input number", 0))
```

or the shorter format

```
Dim salary As Double
salary = CDbl(InputBox("Enter the salary"))
```

3.7 FORMATTING OUTPUT

Numeric and date values can be formatted in output statements so that they appear as a specific type of value. Although these values can be formatted by using a ToString() (see Chapter 7) method along with a special format description, we will show a simple way to format numbers, currency, dates, times, and percents.

The function name precedes the formatted variable name enclosed in parenthesis.

Function	Description
FormatCurrency	Returns an expression formatted as a currency value
FormatPercent	Returns an expression formatted as a percentage
FormatNumber	Returns an expression formatted as a number
FormatDateTime	Returns an expression formatted as a date or time

Assuming we declare a variable

```
Dim num As Double
```

and assign a value to variable num

```
num = 123.45678
```

Formatting the Number To A Specified Number of Decimals

FormatNumber(*variable, number of decimal places*)

FormatNumber(num, 2) ⟶ 123.46
FormatNumber(num, 3) ⟶ 123.457
FormatNumber(num, 1) ⟶ 123.5

Formatting Using Currency

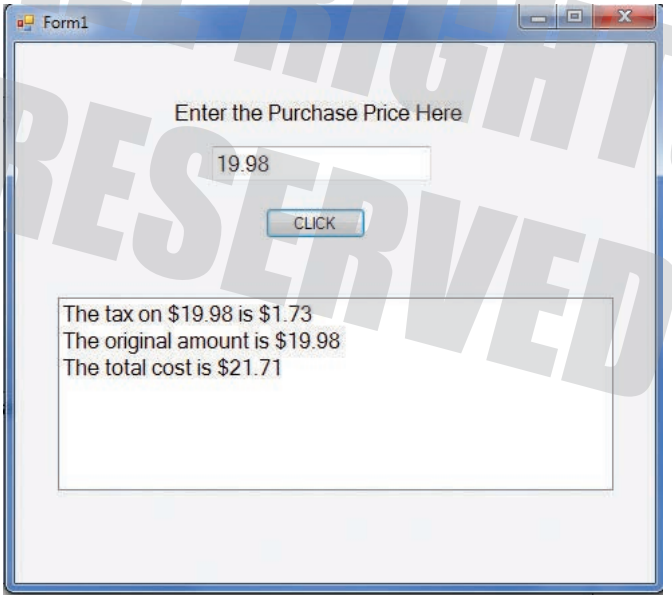
Works similar to FormatNumber, but it displays the $ sign before the number

FormatCurrency(*variable, number of decimal places*)—if number of decimal places is omitted it defaults to 2 decimal places

FormatCurrency(num, 2) ⟶ $123.46
FormatCurrency(num) ⟶ $123.46
FormatCurrency(num,0) ⟶ $123

Example:

```
'tax on a purchase

Private Sub btnTax_Click(...
    Dim amount, tax, total As Double    `declare amount and tax as Integers

    amount = CDbl(txtPurchase.Text)     'cast value in TextBox to a Double

    tax = amount * 0.08645              'calculate the tax
    total = amount + tax                'calculate total

    'display the results formatted as dollars and cents in a TextBox
    lstOut.Items.Add("The tax on " & FormatCurrency(amount, 2) & " is " &
    FormatCurrency(tax, 2))
    lstOut.Items.Add("The original amount is " & FormatCurrency(amount, 2))
    lstOut.Items.Add("The total cost is " & FormatCurrency(total, 2))

End Sub
```



Format as a Percent

Works similar to FormatNumber, but: multiplies *num* by 100 first, then puts "%" at the end of the number.

FormatPercent(*variable, number of decimal places*) remember num = 123.45678

FormatPercent(num, 2) ⟶ 12345.68%
FormatPercent(num, 1) ⟶ 12345.7%

Formatting Using Date or Time

Date/Time Formats

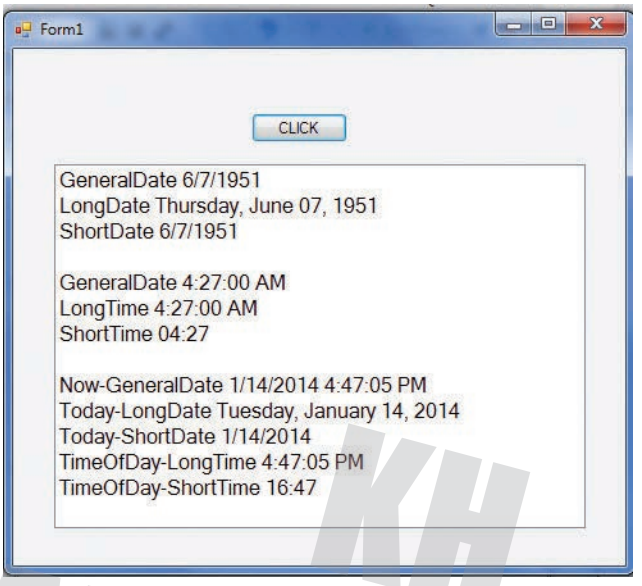
Format Value	Description and Examples
DateFormat.GeneralDate	If the expression contains a date, it formats it the same as ShortDate (3/21/95). If contains a time, it formats it the same as LongTime ("02:07:18 PM").
DateFormat.LongDate	Displays the day of the week, the month, the day, and the year. The time is not reported. (Sunday April 12, 1998)
DateFormat.ShortDate	Displays the month, day, and year in MM/DD/YYYY format. ("12/25/2013")
DateFormat.LongTime	Displays the hours, minutes, seconds, and AM or PM. The date is not reported. ("02:07:18 PM")
DateFormat.ShortTime	Displays just the hours and minutes in military time (24 hour format). The date is not reported. ("13:54")

Date/Time Functions (These may be used as the first argument to a FormatDateTime function.)

Function Name	Description
Now	Returns the current day and time from the system clock.
Today	Returns the current date from the system clock (without the time).
TimeOfDay	Returns the current time from the system clock (without the date).


```
Private Sub btnPress_Click(...
    Dim day, hour As DateTime
    day = "6/7/51"
    hour = "4:27"
    lstOut.Items.Add("GeneralDate " & FormatDateTime(day, DateFormat.GeneralDate))
    lstOut.Items.Add("LongDate " & FormatDateTime(day, DateFormat.LongDate))
    lstOut.Items.Add("ShortDate " & FormatDateTime(day, DateFormat.ShortDate))
    lstOut.Items.Add(" ")
    lstOut.Items.Add("GeneralDate " & FormatDateTime(hour, DateFormat.GeneralDate))
    lstOut.Items.Add("LongTime " & FormatDateTime(hour, DateFormat.LongTime))
    lstOut.Items.Add("ShortTime " & FormatDateTime(hour, DateFormat.ShortTime))
    lstOut.Items.Add(" ")
    lstOut.Items.Add("Now-GeneralDate " & FormatDateTime(Now, DateFormat.GeneralDate))
    lstOut.Items.Add("Today-LongDate " & FormatDateTime(Today, DateFormat.LongDate))
    lstOut.Items.Add("Today-ShortDate " & FormatDateTime(Today, DateFormat.ShortDate))
    lstOut.Items.Add("TimeOfDay-LongTime " & FormatDateTime(TimeOfDay, DateFormat.
      LongTime))
    lstOut.Items.Add("TimeOfDay-ShortTime " & FormatDateTime(TimeOfDay, DateFormat.
      ShortTime))
End Sub
```

The output of the above:



3.8 MATH FUNCTIONS

There are functions pertaining to mathematical concepts that can be used in a variety of ways. Many of these are beyond the scope of an introductory class, but it is beneficial for the person who might choose to continue their programming education to examine the following functions.

Function	*Description*	*Syntax*
Math.Abs	Returns the absolute value of a number.	Math.Abs(number)
Math.Ceiling	Returns an integer greater than or equal to a number.	Math.Ceiling(number)
Fix	Returns the integer part of a number. If the number is negative, a negative number greater than or equal to the number is returned.	Fix(number)
Math.Floor	Returns largest integer less than or equal to a number.	Math.Floor(number)

Function	Description	Syntax
Int	Return the integer part of a number. But if the number is negative, it returns the negative number less than or equal to the number.	Int(number)
Math.Max	Returns the largest of two decimal numbers.	Math.Max(num1,num2)
Math.Min	Returns the smallest of two decimal numbers.	Math.Min(num1,num2)
Math.Pow	Returns a number raised to the power value.	Math.Pow(number,power)
Rnd	Returns a random number of **single** datatype.	Rnd(number)
Math.Sign	Returns the sign of a number.	Math.Sign(number)
Math.Sqrt	Return the square root of a number.	Math.sqrt(number)
Math.Log	Returns the logarithm for a number.	Math.Log(number)
Math.Log10	Return the base 10 logarithmic value for a number.	Math.Log10(number)
Math.Sin	Returns the Sine value for a angle. This function returns NaN if the angle is positive or negative infinity value or it is not a number.	Math.Sin(value)
Math.Asin	Returns the Arc Sine angle for a sine value. The value should be in the range of '-1' and '1'.	Math.Asin(value)
Math.Sinh	Returns the Hyperbolic Sine value for an angle. If the value is negative or positive infinity or NaN, it returns an equivalent value in Double datatype.	Math.Sinh(value)
Math.Cos	Returns the Cosine value for an angle. If the specified angle is positive or negative infinity or Not a Number, the value returned is 'NaN'.	Math.Cos(value)
Math.Acos	Returns the Arc Cosine value for a Cosine value.	Math.Acos(value)
Math.Cosh	Returns the hyperbolic Cosine value for an angle. If the angle is positive or negative infinity returns a positive infinity value. If the angle is 'NaN' returns the same.	Math.Cosh(value)
Math.Tan	Returns the tangent value for an angle. If an angle is positive, negative infinity or Nan, the function returns 'NaN'.	Math.Tan(value)
Math.Atan	Returns the arc tangent value for a tangent value.	Math.Atan(value)
Math.Tanh	Returns the hyperbolic tangent value for a angle.	Math.Tanh(value)

Illustrations of Math Functions *(Note: expression in the explanations below refers to a calculation, variable or numeric value)*

Math.Abs Function

The Abs is short for the absolute value. It examines an expression and returns the absolute value (non-negative value) of the expression. The absolute value is defined as the distance from 0, which is a positive value, whereas the direction can be positive or negative or 0.

Syntax: Math.Abs(number)

Examples:
Math.Abs(3) is 3
Math.Abs(-3) is 3
Math.Abs(0) is 0
Math.Abs(4.79) is 4.79
Math.Abs(-4.79) is 4.79

Math.Ceiling Function

The Ceiling will examine an expression and return an Integer that is the next highest Integer if the expression is a decimal, or returns the Integer itself if the expression is an integer. ***Math.Ceiling()*** returns an Integer greater than or equal to the specified number.

Syntax: Math.Ceiling(Number)

Examples:
Math.Ceiling(2.3) is 3
Math.Ceiling(-2.4) is -2
Math.Ceiling(-.9) is 0
Math.Ceiling(.4) is 1
Math.Ceiling(5) is 5
Math.Ceiling(-5) is -5

Math.Floor Function

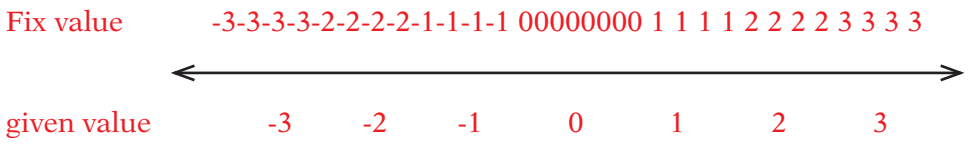
The Floor function returns just the reverse of what the Ceiling function produces. It examines an expression and returns the lower Integer value of an expression, or the number itself if it happens to be an Integer.

Syntax: Math.Floor(Number)

Examples:
Math.Floor(2.3) is 2
Math.Floor(-2.4) is -3
Math.Floor(-.9) is -1
Math.Floor(.4) is 0
Math.Floor(5) is 5
Math.Floor(-5) is -5

Fix Function

The Fix function will examine an expression and return only an Integer part of the number if the number is positive. But when the number is negative, it returns the next higher Integer value of the expression. The diagram below attempts to show how the Fix works.



Syntax: Fix(Number)

Examples: Fix(2.4) is 2
Fix(-2.4) is -2
Fix(2) is 2
Fix(-2) is -2

Int Function

The Int function examines an expression and returns the Integer portion of that number. If the number is negative, a negative number less than or equal to that number is returned.

Syntax: Int(Number)

Examples: Int(2.4) is 2
Int(-2.4) is -3

Math.Max Function

The Max function is used to compare two values. It will examine two given and returns the larger of two decimal numbers

Syntax: Math.Max(num1, num2)

Examples: Math.Max(2,6) is 6
Math.Max(5.6, 2.91) is 5.6
Math.Max(5, -7) is 5
Math.Max(-4, -6) is -4

Math.Min Function

The Min function is used to compare two values. It will examine the two given values and returns the smallest of two decimal numbers.

Syntax: Math.Min(num1, num2)

Examples: Math.Min(2,6) is 2
Math.Min(5.6, 2.91) is 2.91
Math.Min(5,-7) is -7
Math.Min(-4,-6) is -6

Math.Pow Function

Pow is the power function. It takes two numbers and raises the first number to the power indicated by the second number. It is the same as using the ^ symbol with a number. We have already taken numbers to a power by using the exponent symbol.

Syntax: Math.Pow(Number,Power)

Examples: Math.Pow(2,3) is 8
Math.Pow(3,2) is 9
Math.Pow(4,2) is 16
Math.Pow(16,.5) is 4

Math.Sqrt Function

Math.Sqrt() returns the square root of a number.

Syntax: Math.Sqrt(Number)

Examples: Math.Sqrt(16) is 4
Math.Sqrt(100) is 10
Math.Sqrt(2) is 1.414213562

Math.Sign Function

Math.Sign() returns the sign of the specified number in the argument. The value returned is either -1, 1, or 0: -1 if the number is negative, 1 if the argument is positive, and 0 if the argument is 0.

Syntax: Math.Sign(Number)

Examples: Math.Sign(5) is 1
Math.Sign(-5) is -1
Math.Sign(0) is 0

Math.Log Function

Math.Log() finds the logarithm for the specified value.

Syntax: Math.Log(Value)

Examples: Math.Log(1) is 0
Math.Log(0) is -Infinity

Math.Log10 Function

Math.Log10() finds the base 10 Logarithmic value for a number.

Syntax: Math.Log10(Value)

Examples: Math.Log10(100) is 2
Math.Log10(1000) is 3

Math.Sin Function

Math.Sin() is used to find the Sine value for an angle in Radians. This returns NaN if the given angle is positive or negative infinity.

Syntax: Math.Sin(Value in radians)

Examples: Math.Sin(1) is 0.841470984807897
Math.Sin(2) is 0.909297426825682

Math.Asin Function

Math.Asin() is used to find the Arc Sine angle for a sine value. The value should be in the range of -1 and 1 or NaN will be returned.

Syntax: Math.Asin(Value)

Examples: Math.Asin(6) is NaN
Math.Asin(0) is 0
Math.Asin(1) is 1.5707963267949

Math.Cos Function

Math.Cos() is used to find the Cosine value for an angle in Radians. If the angle is positive or negative infinity the return value is NaN.

Syntax: Math.Cos(Value in Radians)

Examples: Math.Cos(0) is 1
Math.Cos(.5) is 0.87758256189037276
Math.Cos(1) is 0.54030230586813976
Math.Cos(2) is -0.41614683654714241

Math.Acos Function

Math.Acos() is used to find the Arc Cosine value for a Cosine value.

Syntax: Math.Acos(Value)

Examples: Math.Acos(1) is 0
Math.Acos(0) is 1.5707963267948966
Math.Acos(.5) is 1.0471875511965979
Math.Acos(-2) is NaN

Math.Tan Function

Math.Tan() is a mathematical function that is used to find the tangent value for an angle in Radians. If the given angle is positive, negative infinity or NaN, the function returns NaN.

Syntax: Math.Tan(Value in Radians)

Example: Math.Tan(0) is 0111
Math.Tan(1) is 1.557077246549023
Math.Tan(2) is -2.1850398632615189

Math.Atan Function

Math.Atan() is used to find the arc tangent value for the given tangent value.

Syntax: Math.Atan(Value)

Examples: Math.Atan(1) is 0.785398163397448
Math.Atan(0) is 0
Math.Atan(1.55707724654903) is 0.9999351015877854

Rnd Function

Rnd() returns a random number of a Double data-type. Many game programs use the concept of random numbers. Visual Basic has a variety of ways in which to generate the simulations of random numbers.

Syntax: Rnd(Number)

Examples: First, it is necessary to randomize the random number generator by using the following statement:

```
Randomize()      'initialize the random number generator
```

And then to actually generate a number, use Rnd(). This will produce a random number (a double) as small as 0 and less than 1.

```
Dim value As Double
Randomize()
value = Rnd()
txtOut.Text = value
```

The output will produce a value from 0 to .999999999.

These values are not always appropriate to use. It is often necessary to generate the numbers 0 and 1 to simulate the toss of a coin, or the numbers 1 to 6 to simulate the roll of a die. This is accomplished by multiplying and adding values to the Rnd().

Example: To simulate the roll of a die:

```
Randomize()
Dim value As Integer
'Generate random value between 1 and 6
value = Int((6 * Rnd()) + 1)
txtOut.Text = value
```


Examining this line: value = Int((6 * Rnd()) + 1)

```
6 * Rnd()
```

takes a number between 0 - .99999999 and multiplies it by 6 which now becomes a value from 0 - 5.9999999

adding 1 to the value now makes it: 1 - 6.9999999

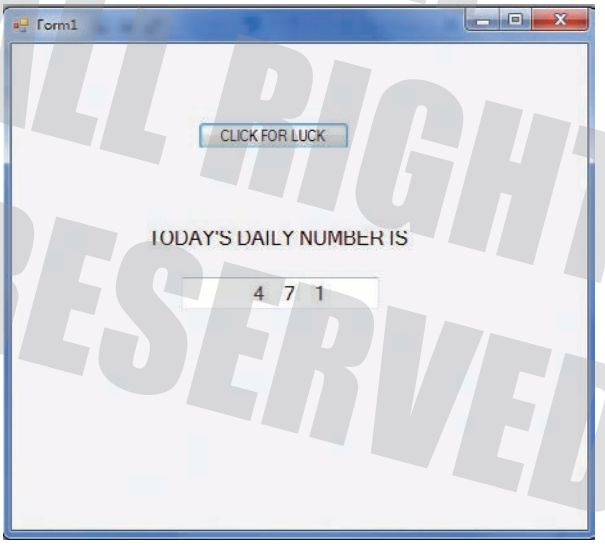
Int: the mathematical function which takes only the integer value so new value will be an integer value from 1 - 6

Random numbers are used in many applications of games and cards. We also see the use of random numbers in the generation of Lotto, Lottery, the Daily Number, Pick Three, etc.

The following is a program example:

A simulation of the NYS Daily Number - randomly choosing three numbers from 0 to 9, and making a three-digit representation of a number from 0 to 999.

```
'to simulate the NYS daily number
Private Sub btnNumber_Click(...
    Dim num1, num2, num3 As Integer
    Randomize()
    txtOut.Clear() 'clears the text box
    num1 = Int(Rnd() * 10)
    num2 = Int(Rnd() * 10) 'generate each to be digit to be values 0 to 9
    num3 = Int(Rnd() * 10)
    'constructing textbox
    txtOut.Text = " " & CStr(num1) & " " & CStr(num2) & " " & CStr(num3)

End Sub
```



3.9 VISUAL BASIC DESIGN GUIDELINES

Try to put yourself in the place of the user and think about how your program will be used. Some basic guidelines for developing a good, user-friendly interface, as well as better code writing:

General Form Design

- Place information toward the top left or top center of the screen, because we usually read from right to left.
- Try to provide some blank space on the screen.
- Make sure controls are surrounded by blank space and not next to each other.

- Keep **Buttons** within the same size if in a column or a row.
- Don't make the **Form** larger than the screen size.

Labels and Text Boxes

- Provide every **TextBox** used for input with a **Label**.
- The proximity of a **Label** should be either above or to the left of the **TextBox** they are describing.
- Use only one or two fonts that are easy to read.
- Use a font size of usually 10 or 12 to make it more easily readable.
- Be consistent with upper and lowercase letters.
- Do not use wild or special fonts that may be hard to read or that may not be available on the user's computer.
- Don't use a lot of underlining or italics.

Buttons

- The text in a **Button** should be clearly labeled.

Colors

- Generally use colors to create a soothing effect.
- Use a contrasting color to emphasize headings or titles.

And for the programming code portion of your project:

Code

- Name your controls to indicate their purpose using the three-letter prefixes txt(**TextBox**), lbl(**Label**), btn(**Button**), lst(**ListBox**).
- Use variable names that are similar to their representations like avg to store an average value.
- Place comments throughout your program code.

3.10 SUMMARY

All values that are stored in a Visual Basic program are considered to be a specific data type. The most common data types used in programs are String (a collection of characters), Char (a single character), Integer (whole numbers), and Double (real numbers).

When using a variable to store a value the variable must be declared using a Dimension statement.

Dim *variablename* As *data-type*

Variables that are declared as constants contain values that cannot change throughout the program execution.

Variables can be used in arithmetic assignment statements. All resulting variables are stated on the *left* side of the = sign while all calculations are on the *right* side of the = sign.

result = a + b

Visual Basic calculations follow the same order of mathematical operator precedence as commonly accepted in mathematics. This order is known by the acronym PEMDAS.

All data that is input to a Visual Basic program is String data. In order to change these values to another data type (Integer, Double…) conversion or casting functions must be used.

Output can be formatted using a variety of formatting functions. One of the most common output format function used is when displaying money values using FormatCurrency.

For more complex mathematic operations (square root, exponents, ...) Visual Basic provides a set of Math functions that can be used in assignment statements and calculations.

PROGRAMMING PROBLEMS

PROBLEM 1

Your Uncle Thaddeus has gone into the perfume business and has decided to sell his perfume in pyramid shaped bottles. He has asked you to write a program that given the dimensions of the pyramid bottle you can calculate and output the volume in cubic inches for his pyramid shaped perfume bottles. Also include a picture of your bottle on your interface.

PROBLEM 2

Your Uncle Thaddeus has decided to cash in his piggy bank. His bank contains only pennies, nickels, dimes, and quarters. Write a program to figure out how much money your Uncle Thaddeus has in his piggy bank. You can use textboxes or input boxes for input (your choice). Display the amount in a **ListBox** remembering to use **FormatCurrency** because the output is a money value.

PROBLEM 3

Uncle Thaddeus Pizzeria needs a program to calculate the number of slices that can be cut from any size pizza.

The input for the program will be the diameter of the pizza in inches. The program will calculate the number of slices based on the "rule" that each slice should have an area of 21.5 inches squared.

You can test your program with the following:

Diameter of Pie	*Number of Slices (number is rounded)*
18 inches	12
14 inches	7
8 inches	2

PROBLEM 4

You have decided to open a store based on one of your favorite themes. Your store will offer only a limited number of items—four.

Create a VB project that displays your merchandise and corresponding prices. Allow your customers to enter the quantities of each item and then calculate and display the total amount for their purchase including a 5% sales tax. Remember to use **FormatCurrency** in your output!