VARIABLES

WHAT IS A VARIABLE?

A **variable** is a storage location in the computer's memory, used for holding information while the program is running. The information that is stored in a variable may change, hence the name variable.

There are three (3) basic types of information stored in computers:

1. STRINGS — Data with letters and/or characters
2. INTEGERS — Numbers without decimals
3. FLOATING POINT NUMBERS — Numbers with decimals

Variables are useful for temporarily holding data so you can work with it in some way. Here are a few things you can do with variables:

- Copy and store values entered by the user so they may be manipulated
- Perform arithmetic on values
- Test values to determine that they meet some criterion
- Temporarily hold and manipulate the value of a control property
- Remember information for later use in the program

DATA TYPES FOR VARIABLES

Variables can store many different types of data. How you declare the variable determines the type of data you can store in it. There are a number of different data types supported by Visual Basic. You need to become familiar with the limits of each of these – you don't want a user typing in a number that is greater than the data type you're using.

The following chart illustrates the most common data types used in Visual Basic:

DATA TYPE	DESCRIPTION	MEMORY (bytes)
Boolean	Holds a value that is either **True** or **False**	2
Char	Holds a single character (e.g. 'A', '!', etc.)	2
Integer	Holds an integer value between -2,147,483,648 to 2,147,483,647	4
Long	Holds an integer value between $-10E^{18}$ to $10E^{18}$	8
Single	Holds a real number up to 7 digits ($-3.40E^{38}$ to $3.40E^{38}$)	4
Double	Holds a real number up to 14 digits ($-1.80E^{308}$ to $1.80E^{308}$)	8
String	Holds string values which may consist of letters, numbers, symbols, etc.	10 + 2 bytes per character

DECLARING VARIABLES

- Before you use a variable, it must be declared.
- When you declare a variable, you are setting up a place in memory to hold information and you are assigning a name to the location. The computer will set aside a piece of memory to hold the information that you store with your variable.
- Although there are number of ways to declare variables, the **Dim** statement is the most commonly used to declare variables (think of **Dim** as short for **dimension**)

Here is the general form of a variable declaration:

```
Dim variableName As DataType
```

Here are a couple of examples of a variable declaration:

```
Dim age As Integer
Dim phoneNumber As String
```

You can also declare multiple variables of the same type with one declaration statement, as shown in the following example:

```
Dim length, width, height As Integer
```

RULES FOR NAMING VARIABLES

Just as there are rules and conventions for naming controls, there are rules and conventions for naming variables. The following are Visual Basic's naming rules for variables:

1. The first character must be a letter or an underscore (_) character
2. Maximum 255 characters
3. Cannot contain spaces or periods
4. Must contain only letters, numbers and underscore character (_); NO PUNCTUATION!
5. Cannot contain Visual Basic keywords, such as **If**, **Dim**, **Time**, **ElseIf**, etc.

ASSIGNING VALUES TO VARIABLES

Once a variable has been declared, you must assign values to the variables by using the equal sign (=). Variable assignment statements must be written so that the **identifier** (or **variable name**) is on the left side of the equal sign and the value is on the right. For example:

```
Dim age As Integer
age = 16
```

You may also specify a starting value in the Dim statement, as follows:

```
Dim age As Integer = 16
```

When a variable is first created it is assigned a default value. Variables with a numeric data type are assigned the value **0**. Boolean variables are initially assigned the value **False**. String variables are automatically assigned a special value called **Nothing.**

DATA TYPE	DEFAULT VALUE
Integer, Long, Single, Double	0
String	Nothing
Boolean	False

Depending on a variable's default value is not a good programming practice. Unless you are certain a variable will be assigned a value before being used in an operation, always initialize it. This principle is particularly true with string variables. Performing an operation on an uninitialized string variable often results in a runtime error, causing the program to halt execution because the value **Nothing** is invalid for many operations. To prevent such errors, always initialize string variables or make sure they are assigned a value before being used in other operations. A good practice is to initialize string variables with an empty string, as follows:

```
Dim name As String = String.Empty
```

OR

```
Dim name As String = ""
```

DECLARING AND INTIALIZING A *Char* DATA TYPE

Declaring and initializing a Char data type is a little different than declaring a String. If, for example, I wanted to declare a variable that will store a letter grade, I would first write my declaration statement:

```
Dim grade As Char
```

In order to initialize the variable, I will need to make enclose the value in double quotations followed by an uppercase C:

```
grade = "A"C
```

VARIABLE SCOPE: Local vs. Class-Level Variables

The placement of a variable declaration is important because it determines the **scope** of the variable. The scope of a variable is the set of statements that can access the variable.

LOCAL VARIABLES

A variable declared at the beginning of a procedure is accessible to any statement in that procedure. This means that any statement in the procedure can refer to the variable, change its value, and so on. This variable is said to be **local** to the procedure because its scope is limited to that procedure. Statements outside the procedure do not have access to the variable.

The following is an example of a local variable that is declared for a **Click** event procedure:

```
Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles btnOK.Click

    Dim x As Integer = 10
    Dim y As Integer = 30
    Dim z As Integer

    z = x + y

End Sub
```

Local variables

CLASS-LEVEL VARIABLES

When a variable needs to be accessed by several or all of the procedures in the Form class, the declaration should be placed in the General Declarations section of the form. This type of declaration is called a **class-level** or **module-level** variable whereby any statement in any procedure can refer to the variable or change its value. Class-level declarations should only be used when absolutely necessary. Anther important programming practice is to declare variables so that the scope is limited to where they are needed. This is good programming style because it produces cleaner code and helps eliminate the possibility of errors.

```
Private name As String = String.Empty
Private pi As Double = 3.14

Public Class Form1
```

Class-level variables

```
    Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles btnOK.Click

        Dim x As Integer = 10
        Dim y As Integer = 30
        Dim z As Integer

        z = x + y

    End Sub

End Class
```

Local variables

NAMED CONSTANTS

A **named constant** is like a variable whose content is read-only, and cannot be changed while the programming is running. Named constants look like regular variables except for the following differences:

- The word **Const** is used instead of **Dim**
- An initialization value is required

- By convention, all letters are uppercase
- Words in the name are separated by the underscore character

The general form of a named constant declaration is:

```
Const CONSTANT_NAME As DataType = Value
```

The keyword **Const** tells Visual Basic you are declaring a named constant instead of a variable. The value give after the **=** sign is the value of the constant throughout the program's execution.

A value must be assigned when a named constant is declared or an error will result. An error will also result if any statements in the program attempt to change the contents of a named constant.

So, for example, if I wanted to use a named constant to store an interest rate of 4.5%, it would look something like this:

```
Const INTEREST_RATE As Single = 0.045
```

PROGRAMMING CONVENTIONS AND DOCUMENTATION

Programming conventions become extremely important in a team environment. If the code is easy to read, understand, and maintain, any programmer should easily and quickly be able to find what they need to know. Coding conventions can include:

- Standardized formats for labelling and commenting code.
- Naming conventions for objects, variables, and procedures.
- Guidelines for spacing, formatting, and indenting.
- Appropriate internal and external documentation for their software.

COMMENT CODE

Comment code is text added to code that is used to explain and clarify program code for other programmers. Comments have no effect on the way an application runs. In Visual Basic applications, a single quotation mark (') must begin a comment. Anything after the single quotation mark is considered a comment for that line of the program. The following will apply to all programs you write from now on:

- Every variable and process must be commented.

- In the general declarations of each form put your name, the course code, the due date, and the name/brief description of the program name. Here's an example of a properly documented program.

```
'NAME:        Jack Black
'COURSE:      TIK 2O1
'DATE:        January 1, 1999
'PROGRAM:     Calculator Program


'Declare global variables
```

```
    Private name As String = String.Empty
    Private pi As Double = 3.14

    Public Class Form1
```

```
        Private Sub btnOK_Click(ByVal sender As System.Object, ByVal e As
            System.EventArgs) Handles btnOK.Click

            'Declare local variables
            Dim x As Integer = 10
            Dim y As Integer = 30
            Dim z As Integer

            'Add the two values and output to the label
            z = x + y
            lblAnswer.Text = z.ToString

        End Sub

End Class
```

Other coding conventions that you must follow include:

- White space improves readability, so indent all code inside loops and conditional statements. Use blank lines to improve readability.
- All objects should be named with a consistent prefix that makes it easy to identify the type of object (e.g. **btn** for Button controls)
- All variables must be declared at the top of the procedure or form in which they are used and commented appropriately