# Computer Programming in Excel VBA

# Part 1: An Introduction

by

Kwabena Ofosu, Ph.D., P.E., PTOE

## Abstract

This course is the first of a four-part series on computer programming in *Excel Visual Basic for Applications* (VBA), tailored to practicing engineers. In this course, a general overview of computers and computer programming languages is presented. A tour of the *Excel* VBA programming environment follows. The concept of variables is presented as it specifically relates to the VBA programming language. Several examples relevant to engineering are used to illustrate and demonstrate the concepts and methods learned in this class. Two mini-projects are used to demonstrate the programming concepts and methods in situations encountered by practicing engineers.

## TABLE OF CONTENTS

## List of Tables

**List of Figures**

# 1. INTRODUCTION

## 1.1 Computers

A computer can be defined as an electronic device that has the ability to accept, manipulate, process, store, retrieve, and output data according to programmed instructions. A computer consists of two basic parts a) the **Hardware**: the physical components of the computer, such as the monitor, keyboard, mouse, etc., and b) the **Software:** any set of instructions that tells the hardware what to do, e.g. web browsers, games, and word processors, etc.

The **central processing unit** (**CPU**) is the hardware component of a computer that carries out the instructions by performing the basic operations such as data input, arithmetic, logic, and output of results. The CPU is essentially the physical "brain" of the computer. The **memory** of a computer refers to the hardware used to store the instructions and data on a temporary or permanent basis. The **storage** of a computer refers to physical components and recording media used to retain the data in electronic format on a long term basis.

The **operating system** (**OS**) of a computer is software that manages and coordinates the computer's memory, storage, processes, and all of its software and hardware. Modern operating systems use a **Graphical User Interface** (**GUI,** pronounced "gooey") to enable a human user to interact with the computer. It is the GUI that enables a user to click on icons, select buttons, menus, checkboxes etc., and have results displayed back to the user on a screen using graphics, text, or some combination thereof. Currently, the most common operating systems in use for personal computers are **Microsoft Windows**, **Apple Mac OS X**, and **Linux**.

The flow of data during the interaction of a human user and a personal computer can be depicted schematically as in Figure 1. When a user selects a task to be performed, for example, by clicking with the mouse, using the keyboard, or by using some other input device, instructions are sent to the CPU via the memory to be processed and implemented. Computers are not designed to understand instructions in ordinary everyday language. The instructions sent to the CPU are written in a computer programming language, (also called a programming language). An element of the CPU called the compiler, converts these programmed instructions (commonly called **source code** or **code**) into **machine language**. Machine language consists of binary data (zeros and ones). Computers are built to understand and respond to machine language. The instructions can now be implemented and the results sent to an output device such as a monitor for display, or to a printer.

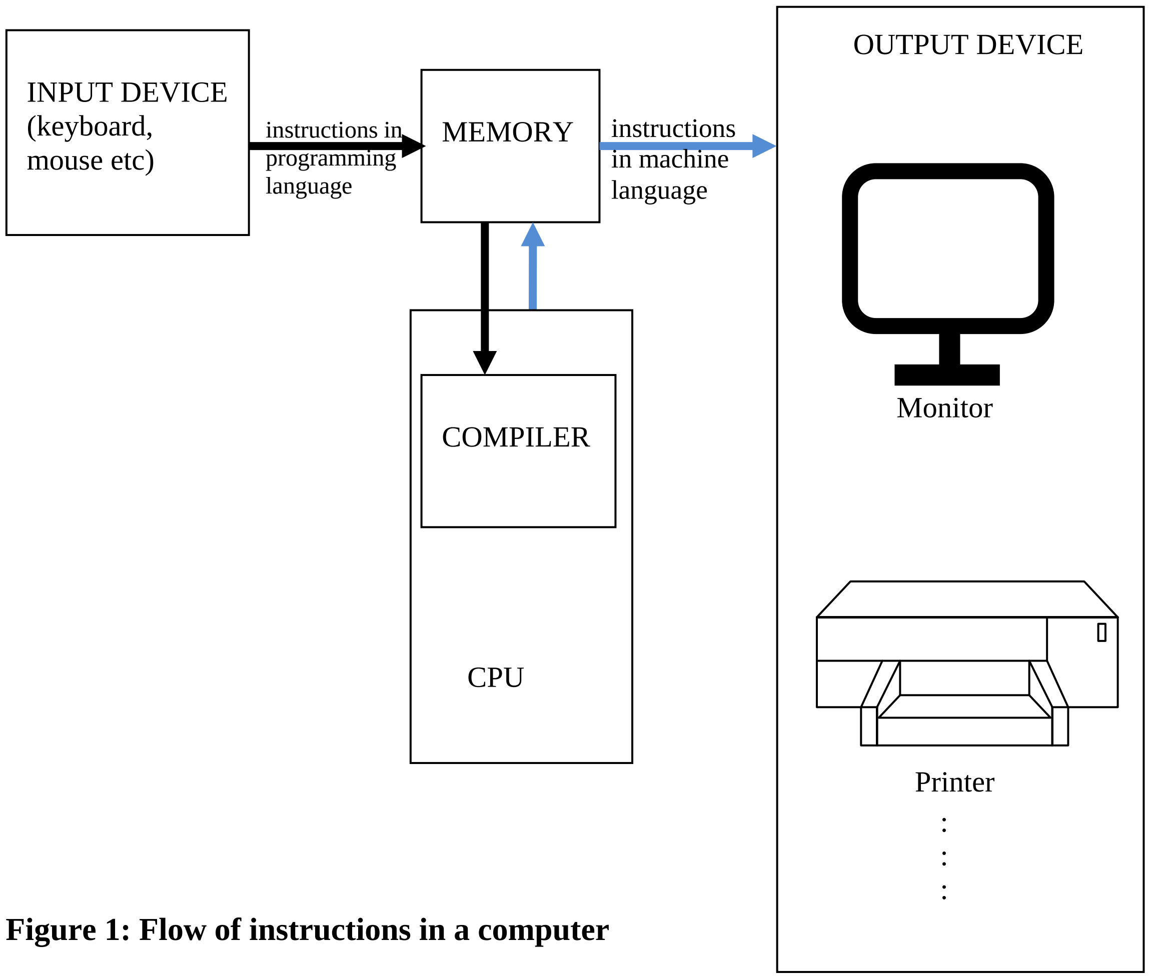

INPUT DEVICE (keyboard, mouse etc)
instructions in programming language
MEMORY
instructions in machine language
OUTPUT DEVICE
Monitor
COMPILER
CPU
Printer
:
:
:

**Figure 1: Flow of instructions in a computer**

## 1.2 Computer Programming

A **programming language** is a formal language specifically designed to communicate instructions to a computer. Programming languages can be used for many purposes, for example to create programs that control the behavior of a computer or device, and/or to implement algorithms accurately and efficiently.

Currently, there are literally hundreds of programming languages in use, each one developed to address a particular type of problem. Programming languages can be classified in many ways. One common approach to classification is based on the paradigm or approach to the programming. Some of the common classifications based on the approach to programming include **procedural programming**, **event-driven programming**, and **object-oriented programming**. Most modern programming languages, however, include elements from more than one classification. In procedural programming languages, the program (or programmer) specifies the sequence of operations, and program logic determines the next instruction to execute based on inputs from the user. In event-driven programming the user can press keys or buttons, mouse clicks, check boxes etc, and these user actions can cause an event to occur which triggers some specific procedure for which code has been written. In object-oriented programming the programmer builds the program by adding **objects** e.g. click buttons, check boxes etc through a GUI in the design environment. The objects have **properties** which can be manipulated e.g. color, or caption on a click button. An action associated with the object is called a **method**. Examples of methods that can be performed on an object based on the instructions include “Move”, “Print”, “Resize”, “Clear”, etc.

Some of the more common and widely used programming languages include:

- C (object-oriented programming language)
- C++ (object-oriented programming language)
- Java (object-oriented programming language)
- Visual Basic (event driven and object-oriented programming language)
- Python (object-oriented and procedural programming language)
- Ruby (object-oriented programming language)
- Pascal (procedural programming language)
- Matlab (procedural programming language)
- Fortran (procedural programming language), and many others.

The choice of what programming language to use on a specific project is determined by several factors such as:

- the operating system of the end user(s)
- the programming approach that is relevant, procedural programming or event-driven, etc
- how well the structure and features of the language are compatible with the project goals
- the level to which the program will be used to manipulate hardware components e.g. in video games
- the ease with which new features can be added to the existing program
- the skill set of the programming team
- support for, and community standing behind the language

Computer programming (or programming) is a comprehensive process of formulating a computing problem, developing a methodology to solve the problem, writing code in a specific programming language to implement the solution methodology, testing, debugging, maintaining the code, verifying and validating the results, and monitoring the consumption of computer resources over the entire process. The objective of programming, therefore, is to develop a series of instructions that can automate the performance of a specific task, or to solve some specific problem formulation in a timely and efficient manner. Therefore, programming involves a multidisciplinary as well as an interdisciplinary approach.

## 1.3 Relevance of Computer Programming

The question often arises as to why engineers and scientists should learn computer programming. Engineers, for instance, are generally required to take just an introductory level computing class as college freshmen. They generally do not receive any further formal instructions on the subject and may not be required to apply it in any other classes for the rest of their entire undergraduate work. Effectively, by graduation, the programming class is a distant memory of questionable relevance, and yet computer programming is becoming an increasingly advantageous and necessary skill set for engineers and scientists competing in the global economy. By writing computer programs to automate tedious and repetitive tasks such as design calculations, or preparing, processing and analyzing large amounts of data which would otherwise be done by hand, engineers and scientists can drastically increase their productivity and efficiency. Competence in computer programming predisposes engineers and scientists to pursue and develop more creative and innovative solutions than their peers. Knowledge, and some level of experience in computer programming enables engineers and scientists to communicate more effectively with full-time programmers and information technology professionals they may work with and collaborate with on various projects.

## 1.4 Planning Computer Programming Solutions

A 6-step procedure for planning and implementing computer programming solutions to a problem or research question is as follows:

1. Study the problem in detail and gain a good working understanding of the fundamental concepts and principles, and underlying theories. This may involve knowledge from a wide variety of fields such as engineering, physical sciences, mathematics, statistics, business, economics, government and many others.
2. Develop an algorithm for solving the problem. A flow chart of the algorithm will provide a detailed visual representation of the process from start to finish, including all inputs,

calculations, data processing, automated reasoning, outputs, reports, and feedback procedures.

3. Write **pseudo code** for your algorithm. Pseudo code is the instructions needed to implement the algorithm in ordinary everyday language. Test your algorithm by performing manual calculations to verify your results.
4. Write code to implement the algorithm in the target programming language following all syntax rules and requirements.
5. Test and debug the code intermittently to identify and resolve glitches and errors.
6. Test the program. Validate and verify the results. Make changes and updates as necessary where needed.

## 1.5 Visual Basic

*Visual Basic* (VB) is an event-driven programming language and integrated development environment (IDE) from *Microsoft*. It also has many elements of an object-oriented programming language. *Visual Basic* was derived from the earlier ***BASIC*** language. VB was designed to cater to beginner programmers and is therefore relatively easy to learn and use. VB can be used to develop programs from simple GUIs to large and very complex applications. Programming in VB is a combination of visually arranging components (called **controls**) on a **form**, specifying attributes and actions for the components, and writing code for additional functionality.

The VB language comes in several "dialects". One such dialect is ***Visual Basic for Applications*** (VBA). VBA is used as a macro or scripting language within the *Microsoft Office* suite of products. VBA is also used in many third-party products widely used by engineers and scientists such as *SolidWorks*, *AutoCAD*, and *ArcGIS*.

Other variations of VB include:

- *VBScript* which is used in Windows scripting and client-side web page scripting
- *Visual Basic .NET* is *Microsoft's* designated successor to Visual Basic 6.0
- *Microsoft Visual Studio Express* which is a free, scaled-down version of *Visual Basic .NET* for independent developers.

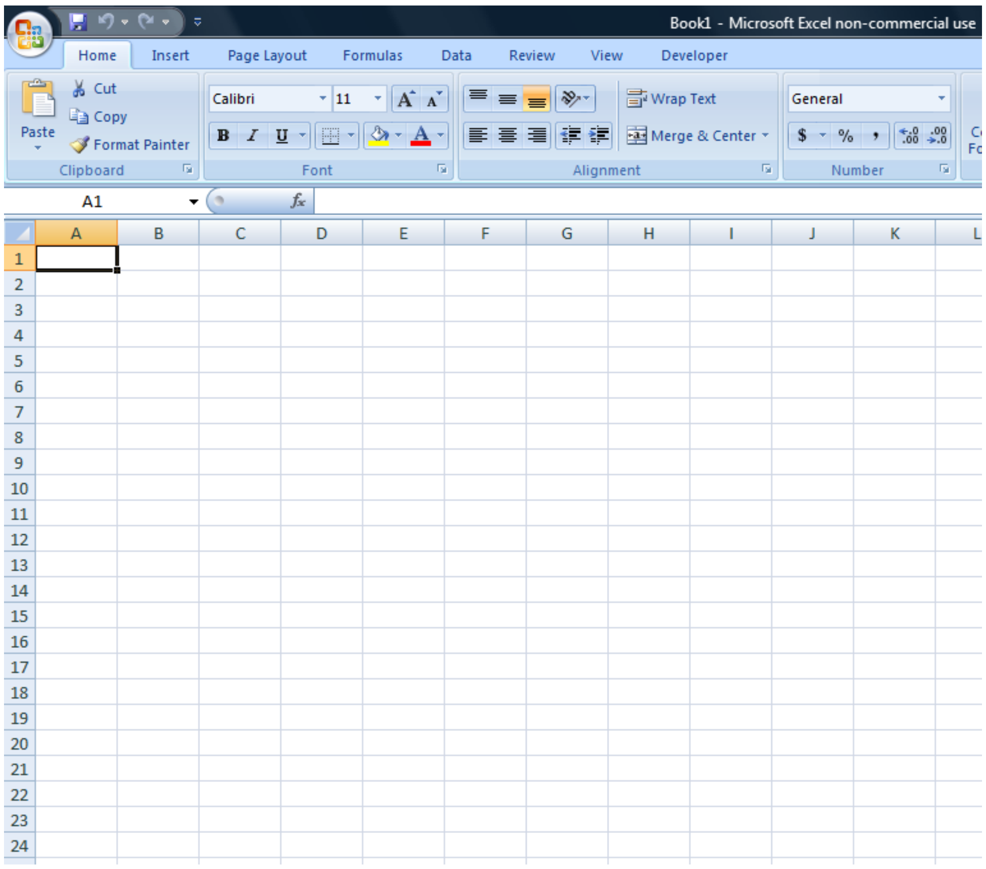
In this class the fundamentals of computer programming will be presented using the VBA language accessed from *Microsoft Excel*.

# 2. THE VBA ENVIRONMENT

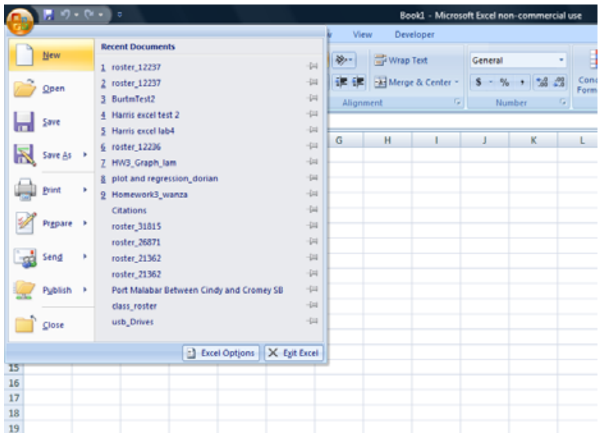
## 2.1 Opening the VBA Environment

If the user has never used the VBA environment in Excel before on their computer, it must be activated using the following steps:

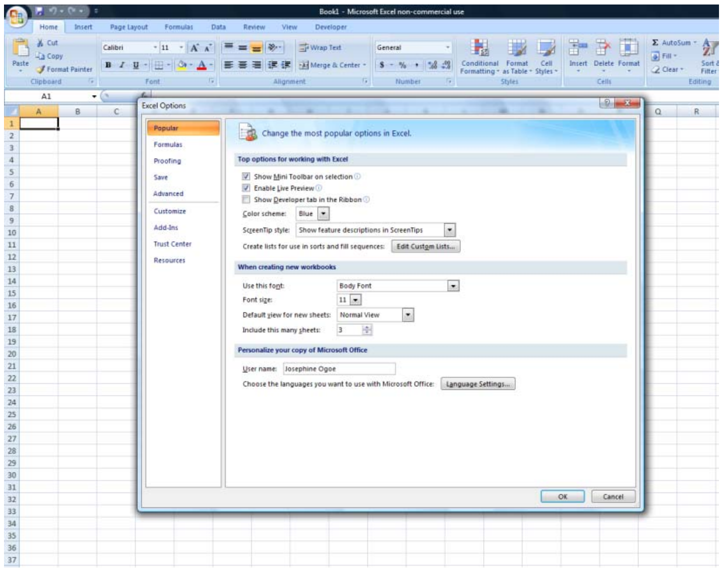
Step 1: Open a new session of Excel.



Step 2: Click on the Microsoft Office icon.



Step 3: Select **Excel Options**



Change the most popular options in Excel.

Top options for working with Excel

- [✓] Show Mini Toolbar on selection
- [✓] Enable Live Preview
- [ ] Show Developer tab in the Ribbon

Color scheme: Blue

ScreenTip style: Show feature descriptions in ScreenTips

Create lists for use in sorts and fill sequences: Edit Custom Lists...

When creating new workbooks

Use this font: Body Font

Font size: 11

Default view for new sheets: Normal View

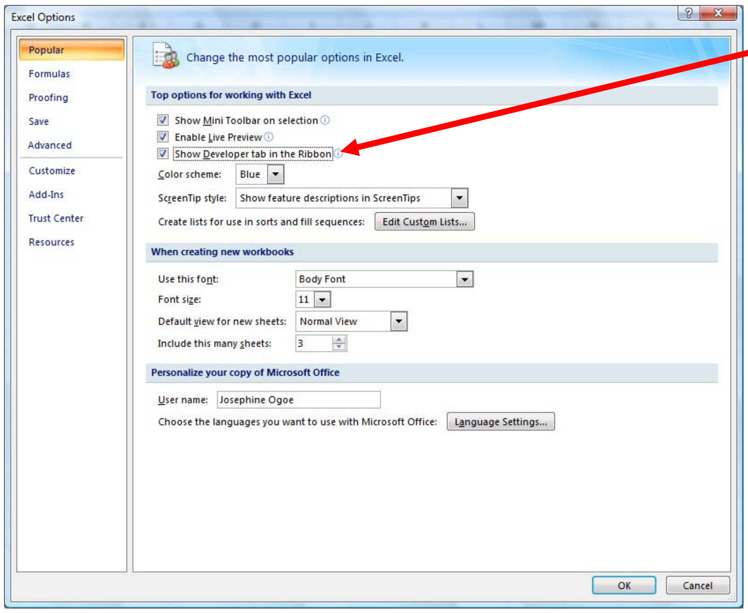
Include this many sheets: 3

Personalize your copy of Microsoft Office

User name: Josephine Ogoe

Choose the languages you want to use with Microsoft Office: Language Settings...

Step 4: Check the box to **Show Developer tab in the Ribbon**. (For Excel 2010 and above, users will select **Customize Ribbon**, and check the box beside **Developer**). Click OK.



Change the most popular options in Excel.

Top options for working with Excel

- ☑ Show Mini Toolbar on selection
- ☑ Enable Live Preview
- ☑ Show Developer tab in the Ribbon

Color scheme: Blue
ScreenTip style: Show feature descriptions in ScreenTips
Create lists for use in sorts and fill sequences: Edit Custom Lists...

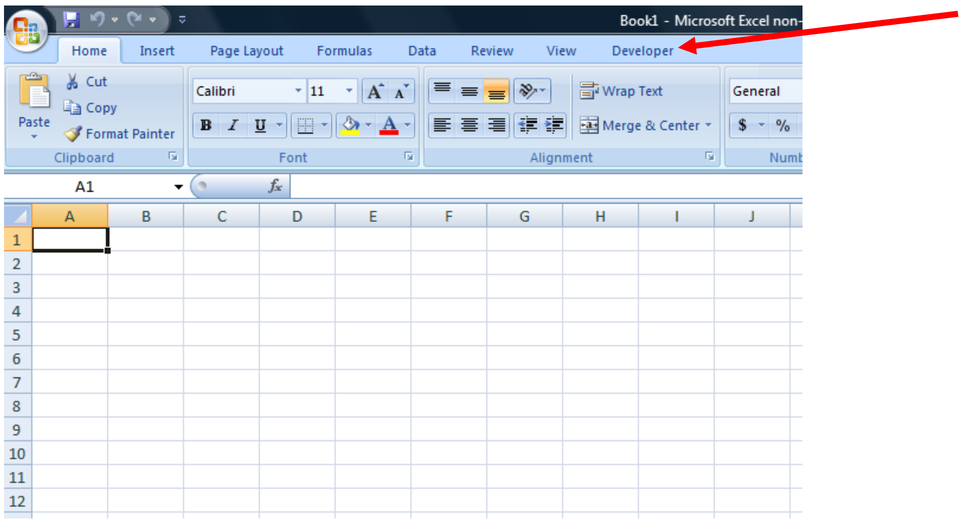
When creating new workbooks

Use this font: Body Font
Font size: 11
Default view for new sheets: Normal View
Include this many sheets: 3

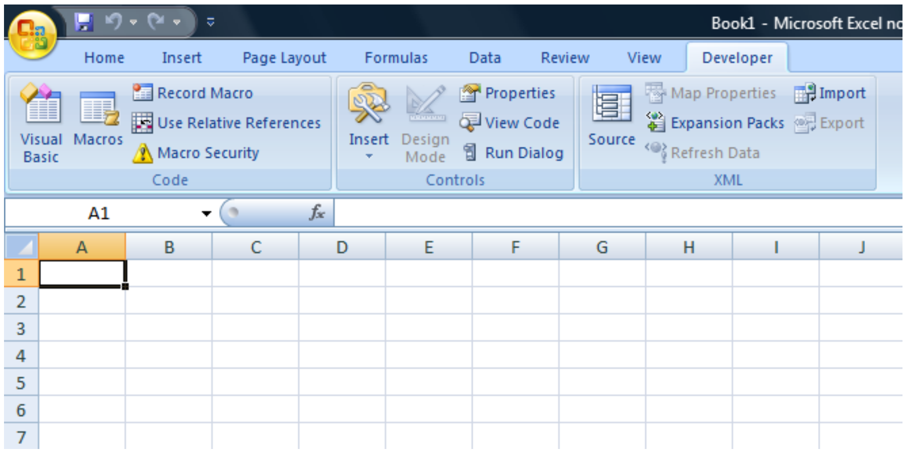
Personalize your copy of Microsoft Office

User name: Josephine Ogoe
Choose the languages you want to use with Microsoft Office: Language Settings...

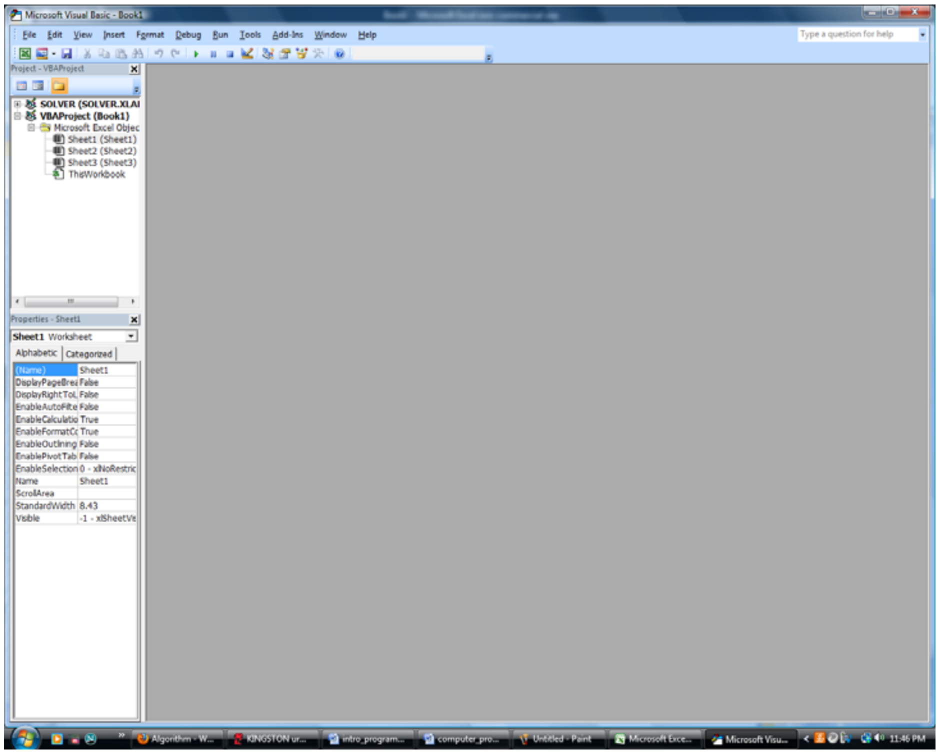
The **Developer** tab is now activated in the Excel session and will remain as such in perpetuity unless the user follows the procedure to deactivate it, or if the computer is for some reason reset to original factory settings.



Step 5: Click on **Developer** to view the contents of that tab.

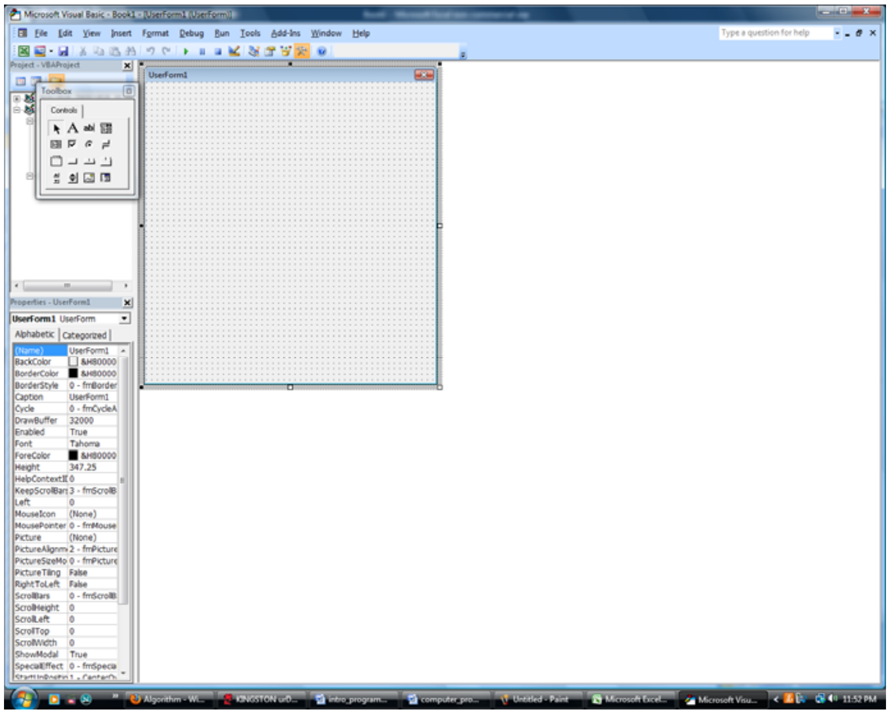


Step 6: Click on **Visual Basic** to enter the VBA environment.



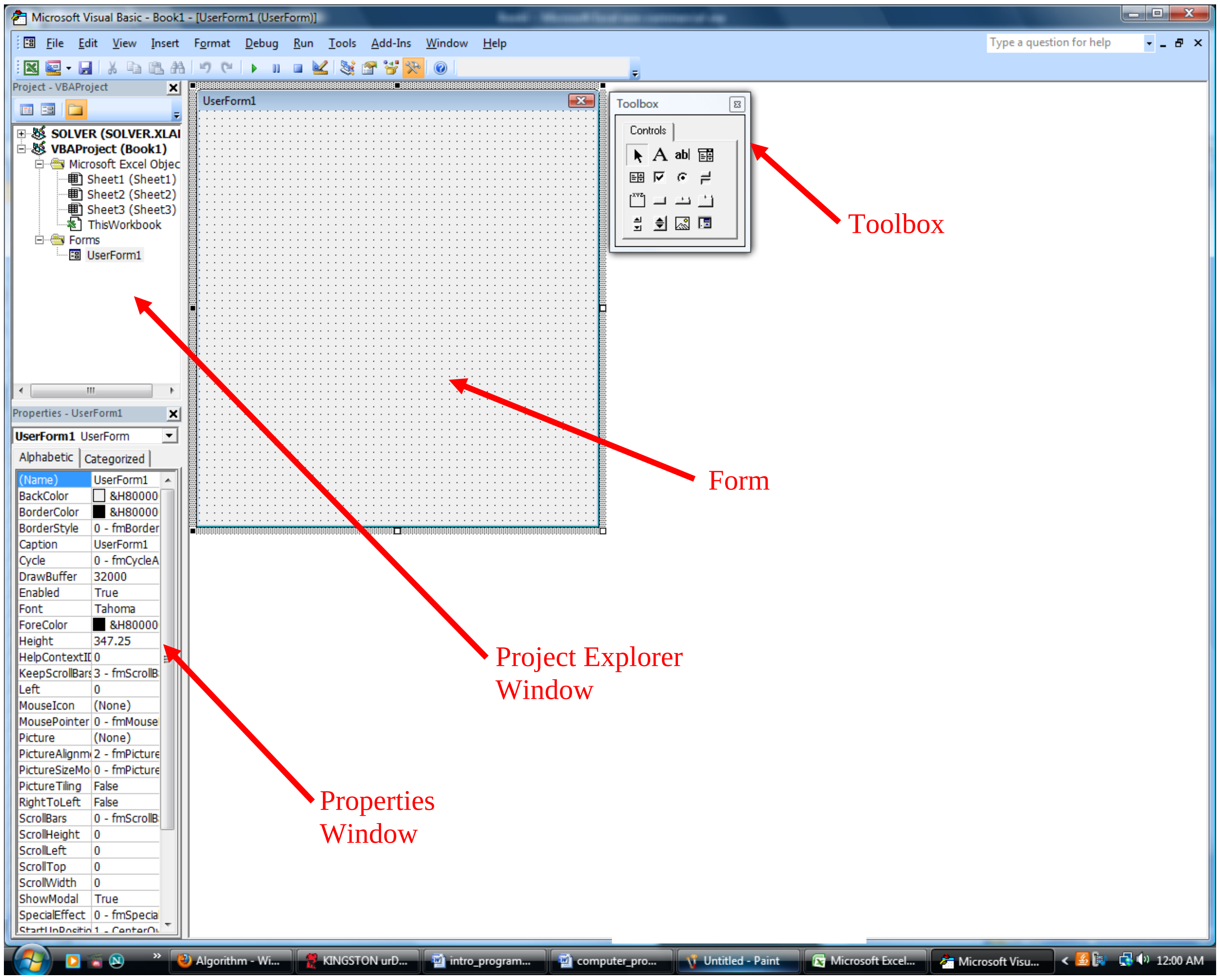
| | |
|---|---|
| (Name) | Sheet1 |
| DisplayPageBrea | False |
| DisplayRightToL | False |
| EnableAutoFilte | False |
| EnableCalculatio | True |
| EnableFormatCo | True |
| EnableOutlining | False |
| EnablePivotTab | False |
| EnableSelection | 0 - xlNoRestric |
| Name | Sheet1 |
| ScrollArea | |
| StandardWidth | 8.43 |
| Visible | -1 - xlSheetVe |

Step 7: Click on **Insert**, and then select **UserForm**. The user is now ready to begin developing a VBA application.



| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H80000 |
| BorderColor | &H80000 |
| BorderStyle | 0 - fmBorder |
| Caption | UserForm1 |
| Cycle | 0 - fmCycleA |
| DrawBuffer | 32000 |
| Enabled | True |
| Font | Tahoma |
| ForeColor | &H80000 |
| Height | 347.25 |
| HelpContextII | 0 |
| KeepScrollBar | 3 - fmScrollB |
| Left | 0 |
| MouseIcon | (None) |
| MousePointer | 0 - fmMouse |
| Picture | (None) |
| PictureAlignm | 2 - fmPicture |
| PictureSizeMo | 0 - fmPicture |
| PictureTiling | False |
| RightToLeft | False |
| ScrollBars | 0 - fmScrollB |
| ScrollHeight | 0 |
| ScrollLeft | 0 |
| ScrollTop | 0 |
| ScrollWidth | 0 |
| ShowModal | True |
| SpecialEffect | 0 - fmSpecia |
| StartUpPositio | 1 - CenterO |

## 2.2 A Tour of the VBA Environment



Properties - UserForm1

UserForm1 UserForm

Alphabetic | Categorized

| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H80000 |
| BorderColor | &H80000 |
| BorderStyle | 0 - fmBorder |
| Caption | UserForm1 |
| Cycle | 0 - fmCycleA |
| DrawBuffer | 32000 |
| Enabled | True |
| Font | Tahoma |
| ForeColor | &H80000 |
| Height | 347.25 |
| HelpContextID | 0 |
| KeepScrollBars | 3 - fmScrollB |
| Left | 0 |
| MouseIcon | (None) |
| MousePointer | 0 - fmMouse |
| Picture | (None) |
| PictureAlignm | 2 - fmPicture |
| PictureSizeMo | 0 - fmPicture |
| PictureTiling | False |
| RightToLeft | False |
| ScrollBars | 0 - fmScrollB |
| ScrollHeight | 0 |
| ScrollLeft | 0 |
| ScrollTop | 0 |
| ScrollWidth | 0 |
| ShowModal | True |
| SpecialEffect | 0 - fmSpecia |

<u>UserForm</u>

The **Form** or **UserForm** is the backbone upon which the application will be built. The toolbox contains **controls**. Controls are the text boxes, drop down menus, check boxes, push buttons etc. To build the user interface of the application controls are dragged from the toolbox onto the form and arranged as needed. The UserForm may be resized as needed by hovering over a handle to get the two headed arrow, clicking and dragging to desired size.

## Toolbox

Some of the controls that can be added from the toolbox include **label**, **text box**, **picture box**, **image**, **scrollbar**, **combo box** (drop down menu), **check box**, **radio button** (option button), and many others.

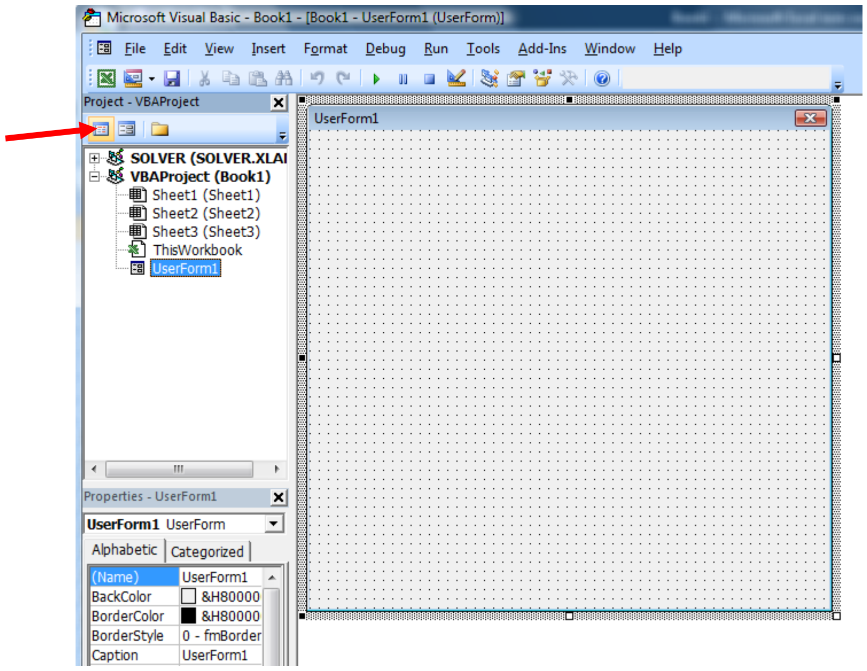
## Properties Window

The UserForm and controls are **objects**. Following from the fundamentals of object-oriented programming, the objects have properties and methods associated with them. By clicking on a control that has been placed on the UserForm, the properties of the object can be changed and modified from the **Properties Window**. The properties of a control listed in the Properties Window include Name, Caption, Font, Text Alignment, Value, and many others. The properties may be viewed alphabetically, or by category.

## Project Explorer Window

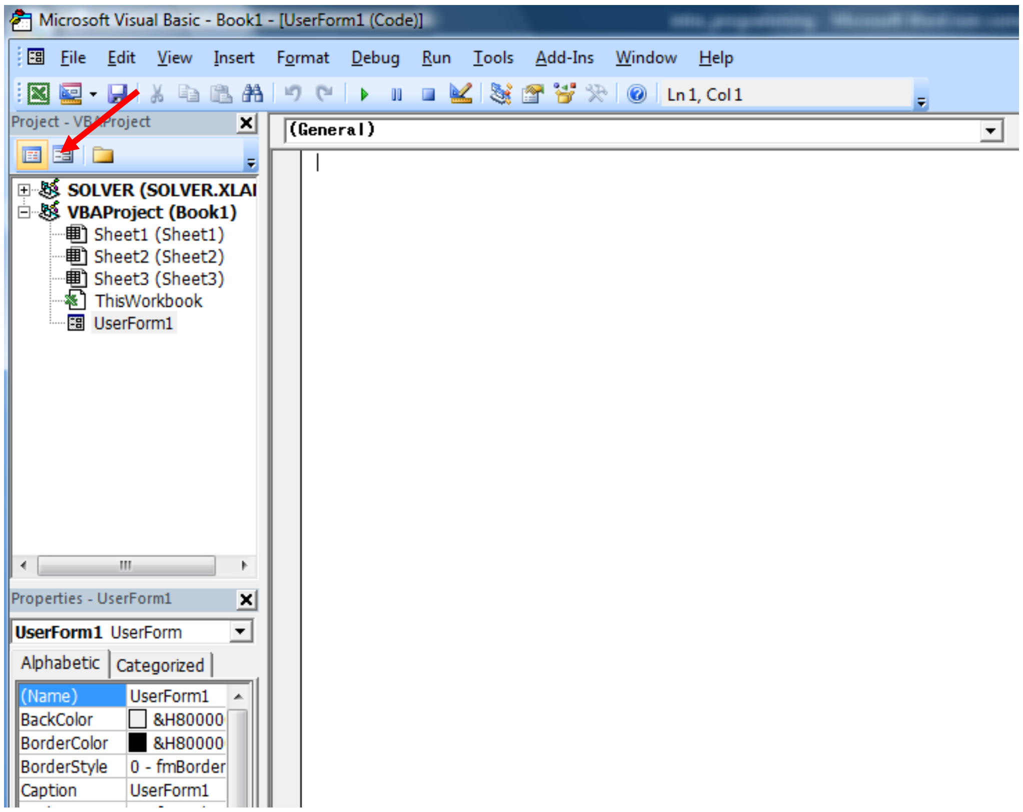
The **Project Explorer Window** (or Project Window) contains a list of files, spreadsheets, UserForms, etc. that are currently active and being used in the current project. The user may toggle between viewing an object and the code associated with it by clicking on the icons in the Project Window header bar.

In the Project Explorer Window, Select the UserForm and click on **View code**.



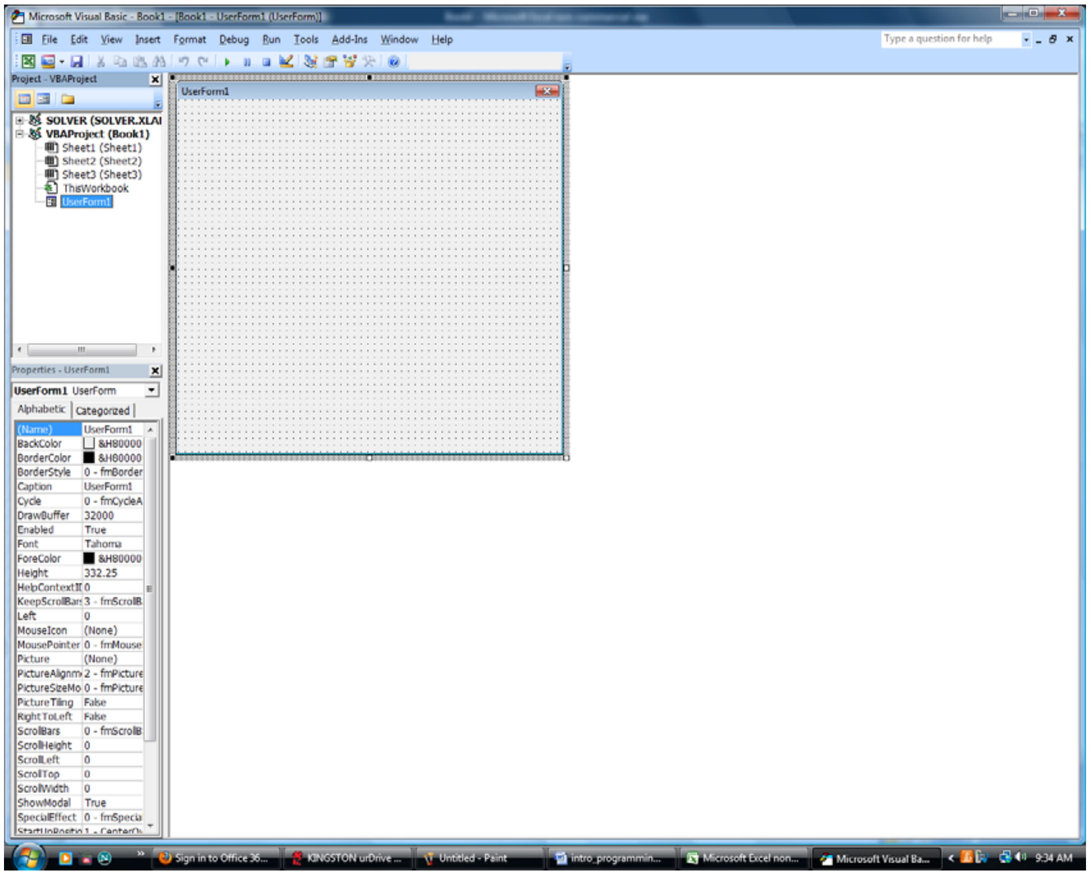
| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H80000 |
| BorderColor | &H80000 |
| BorderStyle | 0 - fmBorder |
| Caption | UserForm1 |

The current UserForm has no controls on it and no code has been written for it at this time therefore the code window appears blank.
To view the object again, click on **View object**



| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H80000 |
| BorderColor | &H80000 |
| BorderStyle | 0 - fmBorder |
| Caption | UserForm1 |

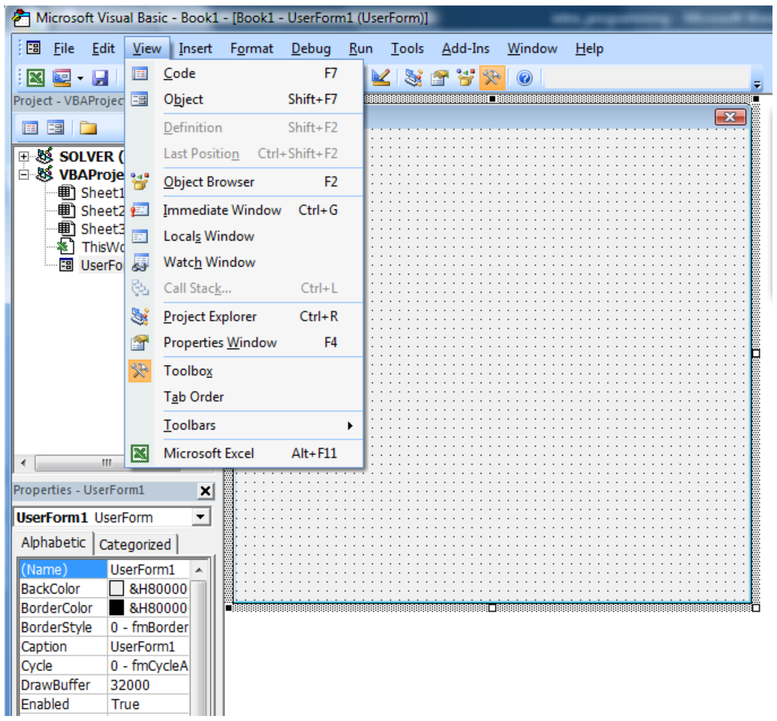
The form reopens (design view).



| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H80000 |
| BorderColor | &H80000 |
| BorderStyle | 0 - fmBorder |
| Caption | UserForm1 |
| Cycle | 0 - fmCycleA |
| DrawBuffer | 32000 |
| Enabled | True |
| Font | Tahoma |
| ForeColor | &H80000 |
| Height | 332.25 |
| HelpContextI | 0 |
| KeepScrollBar | 3 - fmScrollB |
| Left | 0 |
| MouseIcon | (None) |
| MousePointer | 0 - fmMouse |
| Picture | (None) |
| PictureAlignm | 2 - fmPicture |
| PictureSizeMo | 0 - fmPicture |
| PictureTiling | False |
| RightToLeft | False |
| ScrollBars | 0 - fmScrollB |
| ScrollHeight | 0 |
| ScrollLeft | 0 |
| ScrollTop | 0 |
| ScrollWidth | 0 |
| ShowModal | True |
| SpecialEffect | 0 - fmSpecia |

## View Menu

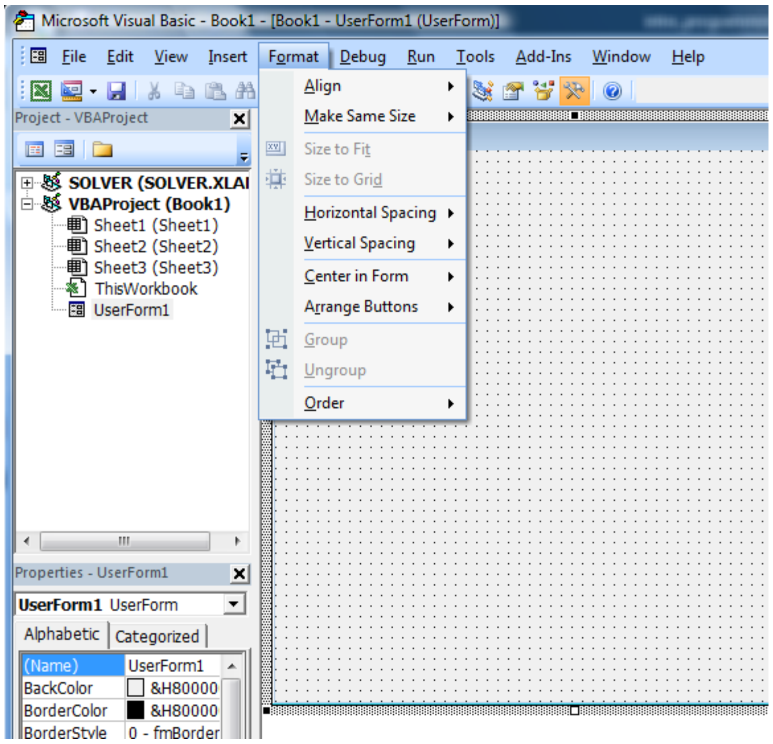
Other windows are available through the **View menu**. If a window is inadvertently closed, it may be reopened from the View menu



| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H80000 |
| BorderColor | &H80000 |
| BorderStyle | 0 - fmBorder |
| Caption | UserForm1 |
| Cycle | 0 - fmCycleA |
| DrawBuffer | 32000 |
| Enabled | True |

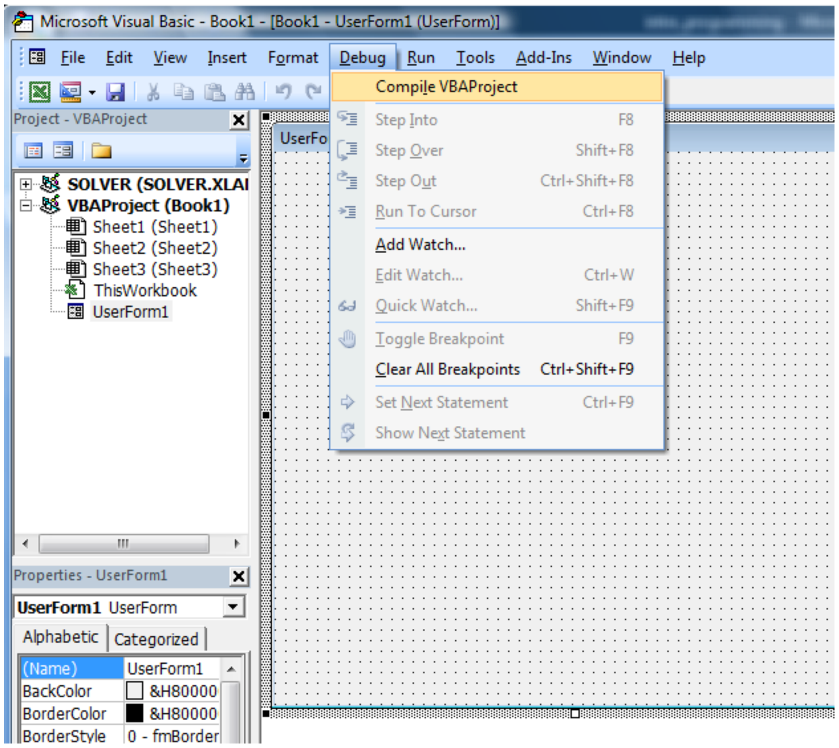
## Format Menu

The **Format menu** provides quick means to format, arrange, order, and align controls on a form.



## Debug Menu

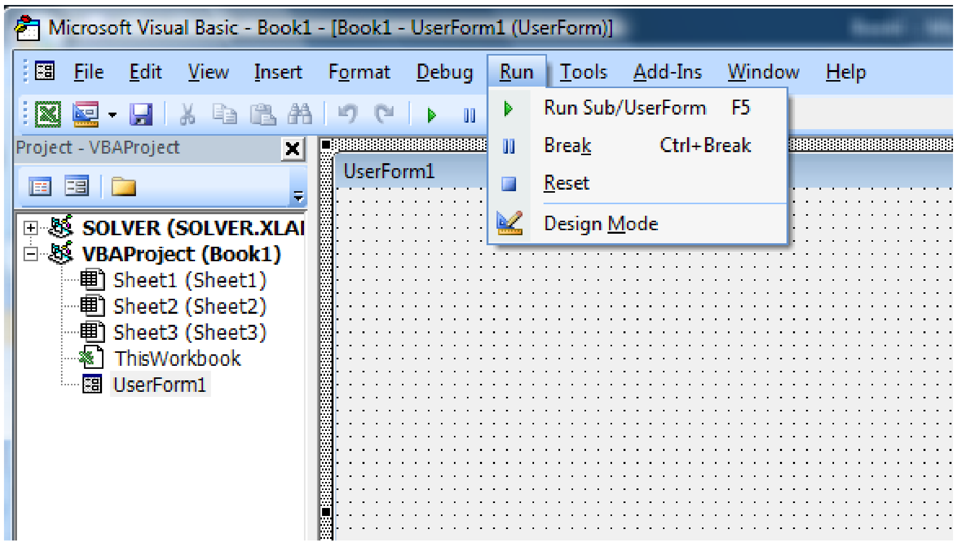
This menu provides means to check the code, identify and highlight errors in the code, and modify some areas in the code.



## Run Menu

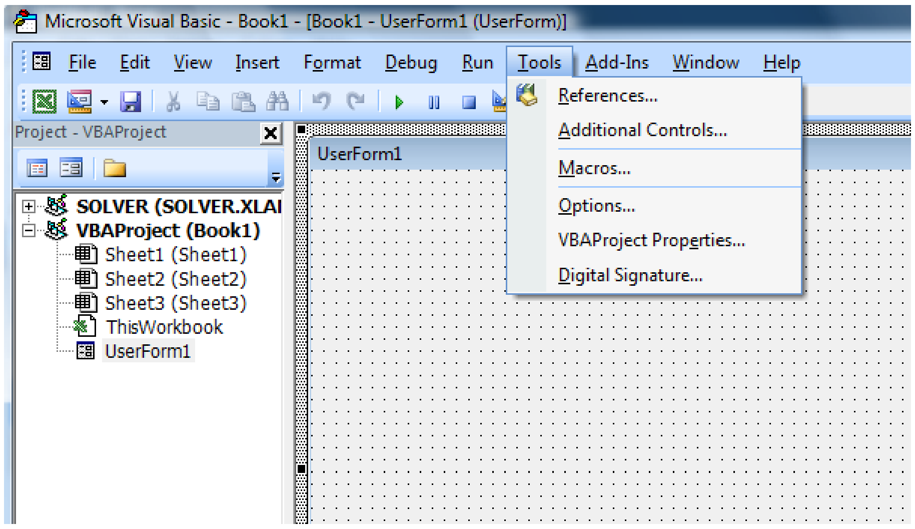
VBA has three modes, namely **design time**, **run time**, and **break time**. This VBA session is currently in design time (or design mode). This is the mode in which the application is being built and code is being written. When the application is actually running, being tested, or are codes being executed, it is it run time. Once in run time, if for some reason the program pauses or if there is a **run-time error**, the program goes into break time. In break time the program can be debugged, saved, updated, modified etc, and the run time can be restored from where it paused or it can be restarted. From break time the program can be reset and it will go back to design time for further work.

The **Run menu** enables a user to manually switch through the modes.



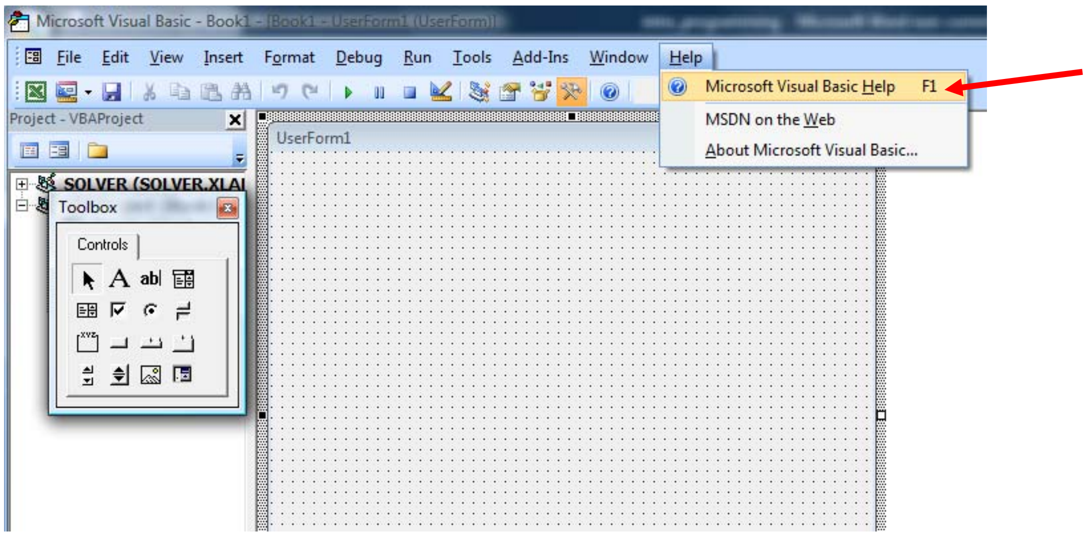
Tools Menu

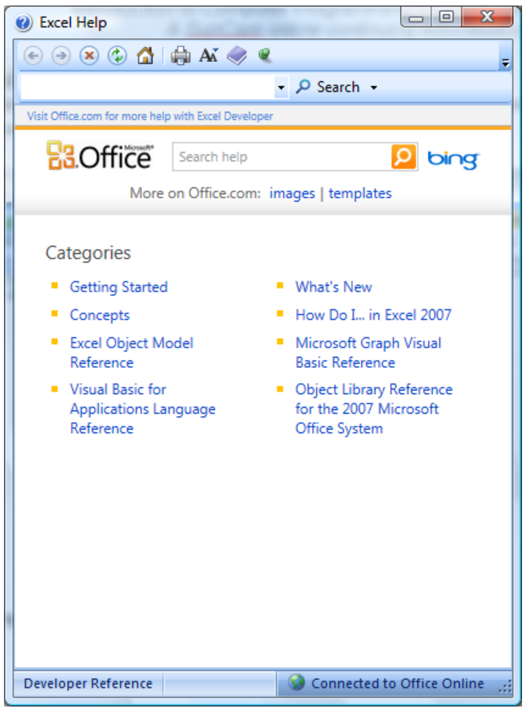
The **Tools menu** provides access to additional programming options, additional form controls, and setting up security features such as password protection, and locking the project from editing by a third party.



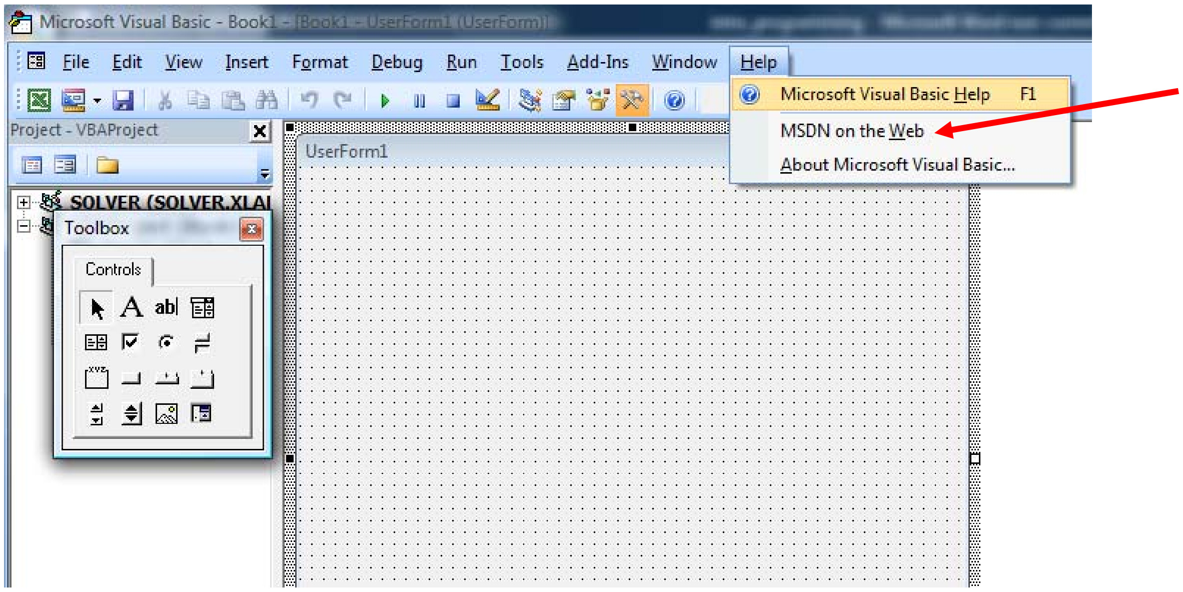
## Help

The **Help menu** provides access to vast help resources online such as search engines, user forums, libraries, and knowledge banks, through the Microsoft website, and the Microsoft Developers Network (MSDN). Click on each one.

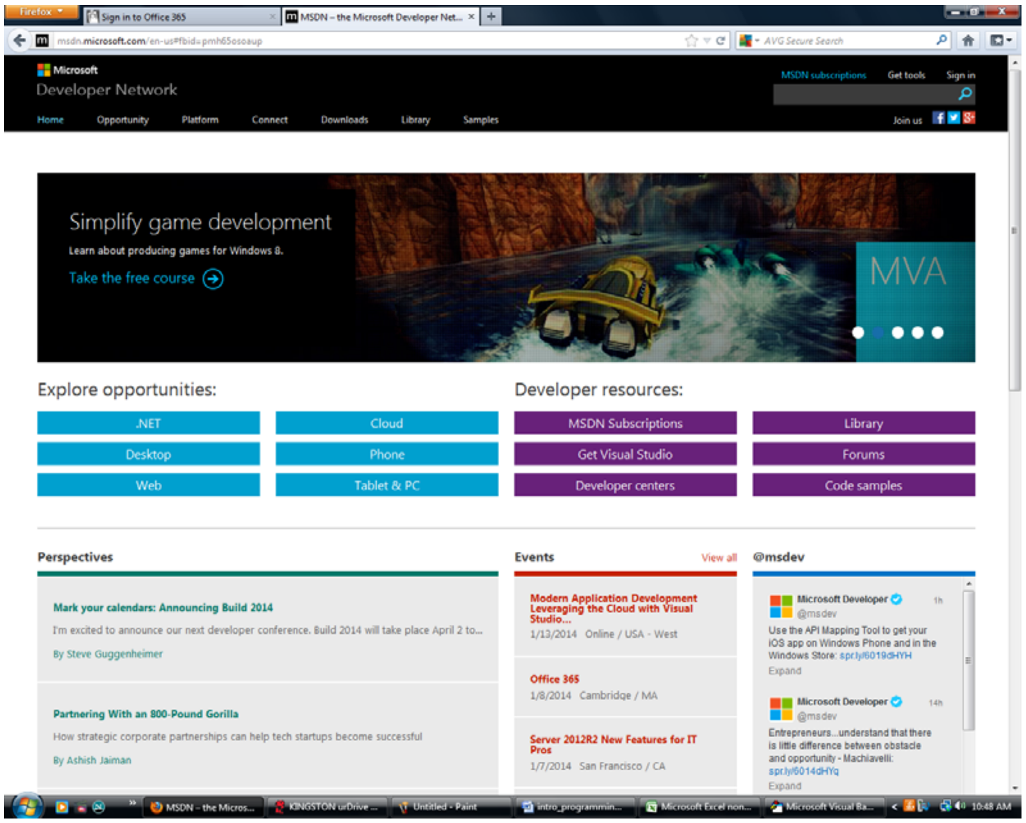



Excel Help
Search
Visit Office.com for more help with Excel Developer
Office
Search help
bing
More on Office.com: images | templates
Categories
Getting Started
Concepts
Excel Object Model Reference
Visual Basic for Applications Language Reference
What's New
How Do I... in Excel 2007
Microsoft Graph Visual Basic Reference
Object Library Reference for the 2007 Microsoft Office System
Developer Reference
Connected to Office Online

Click **Help**
Select **MSDN on the web**


Microsoft Visual Basic - Book1 - [Book1 - UserForm1 (UserForm)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Microsoft Visual Basic Help F1
MSDN on the Web
About Microsoft Visual Basic...
Project - VBAProject
SOLVER (SOLVER.XLAI
Toolbox
Controls
UserForm1

The Microsoft Developers Network (MSDN) website opens.



Mark your calendars: Announcing Build 2014
I'm excited to announce our next developer conference. Build 2014 will take place April 2 to...
By Steve Guggenheimer

Partnering With an 800-Pound Gorilla
How strategic corporate partnerships can help tech startups become successful
By Ashish Jaiman

Modern Application Development Leveraging the Cloud with Visual Studio...
1/13/2014 Online / USA - West

Office 365
1/8/2014 Cambridge / MA

Server 2012R2 New Features for IT Pros
1/7/2014 San Francisco / CA

Microsoft Developer @msdev 1h
Use the API Mapping Tool to get your iOS app on Windows Phone and in the Windows Store: spr.ly/6019dHYH
Expand

Microsoft Developer @msdev 14h
Entrepreneurs...understand that there is little difference between obstacle and opportunity - Machiavelli: spr.ly/6014dHYq
Expand

## 2.3 First VBA Project

A simple project will be developed to demonstrate the use of forms and controls.

Problem Statement:
An engineer is required to attend up to four professional development events in a year in order to obtain approval for license renewal. Create a GUI on which the engineer can enter their personal information and a list of professional development events they attended and the hours they have earned over the renewal cycle. The GUI will have a button which when clicked will save the information entered onto a spreadsheet. Add another click button that clears all the information entered on the form as well as that saved to the spreadsheet. Add a third button that will terminate the program.

Solution:
Step 1: Plan the project

A proposed sketch of the user interface based on the requirements in the problem statement is as follows:

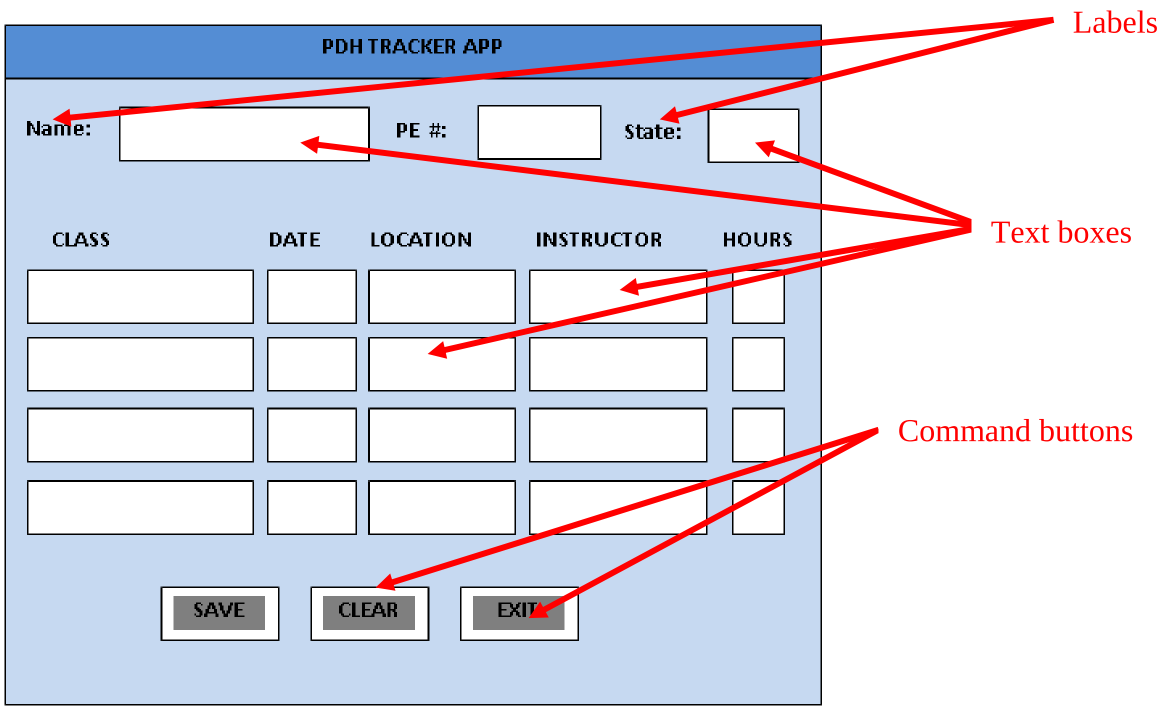


**Figure 2: Plan and sketch for First VBA Project**

Step 2: Algorithm.
Based on the problem narrative, the following flow chart is deduced.

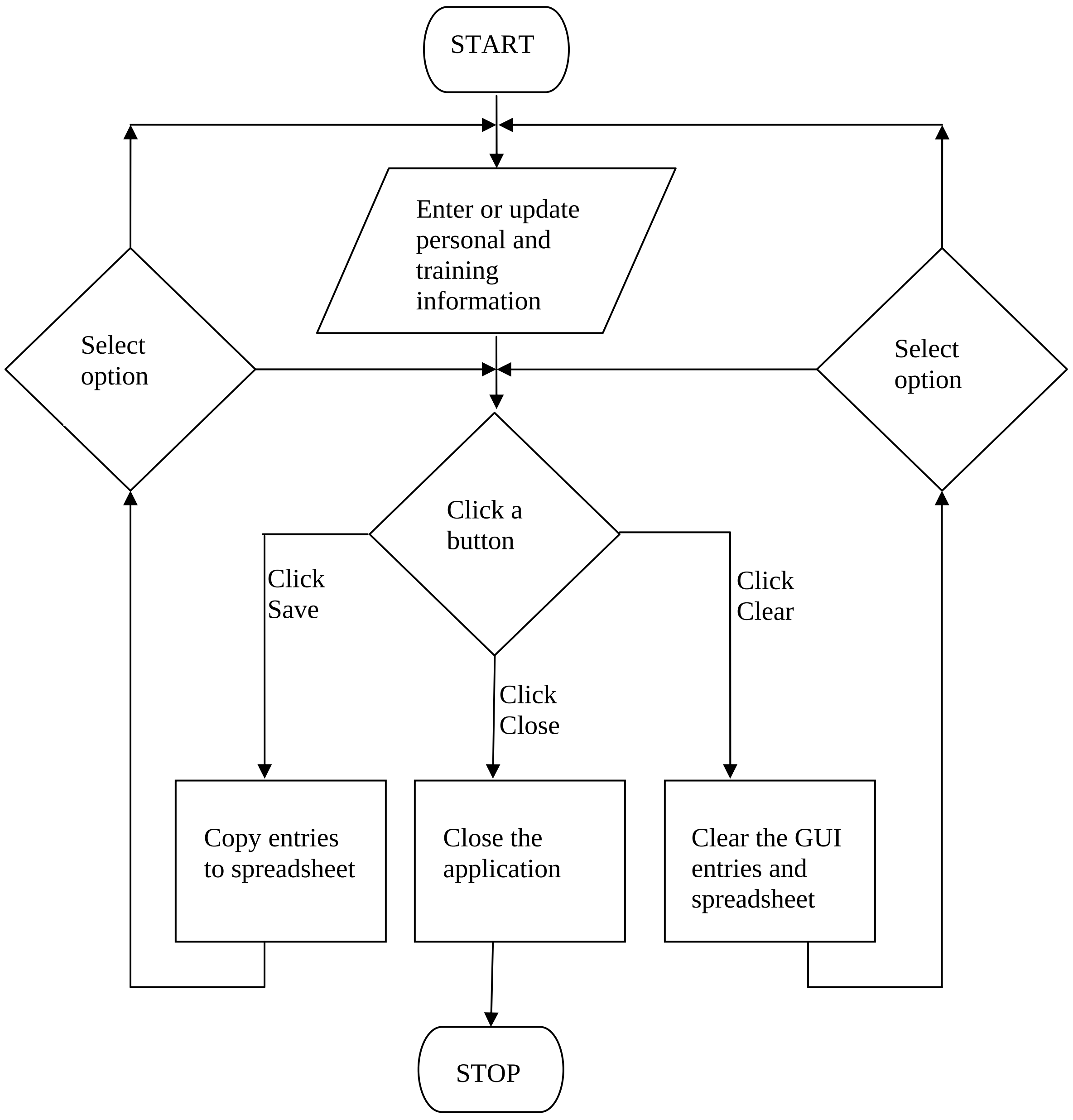
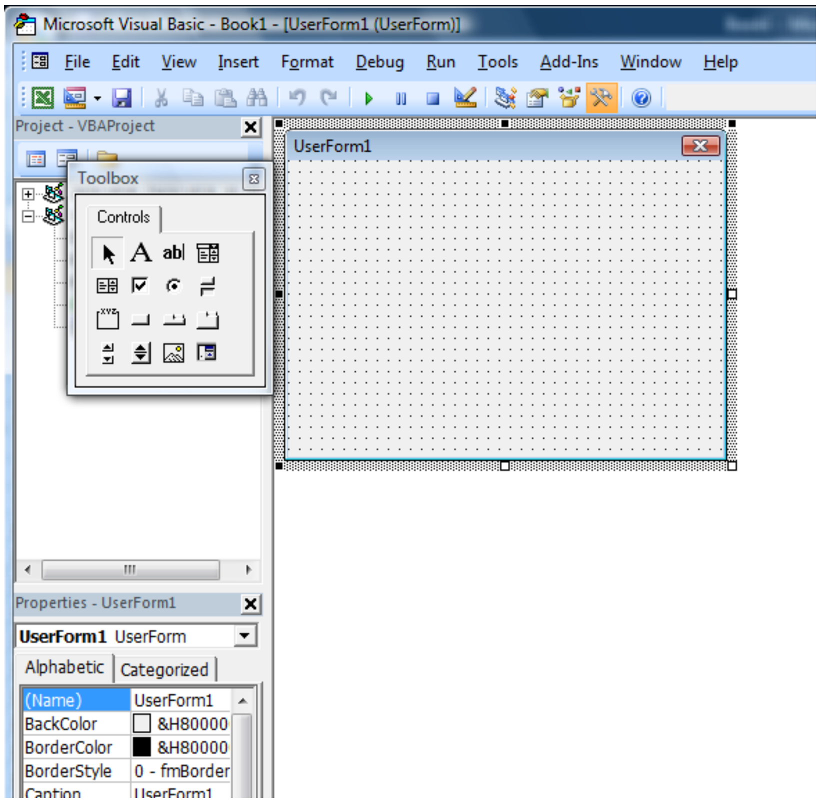

START
Enter or update personal and training information
Select option
Select option
Click a button
Click Save
Click Close
Click Clear
Copy entries to spreadsheet
Close the application
Clear the GUI entries and spreadsheet
STOP

**Figure 3: Algorithm for First VBA Project**

Step 3: Write pseudo code.
When the buttons are clicked on certain events are triggered.

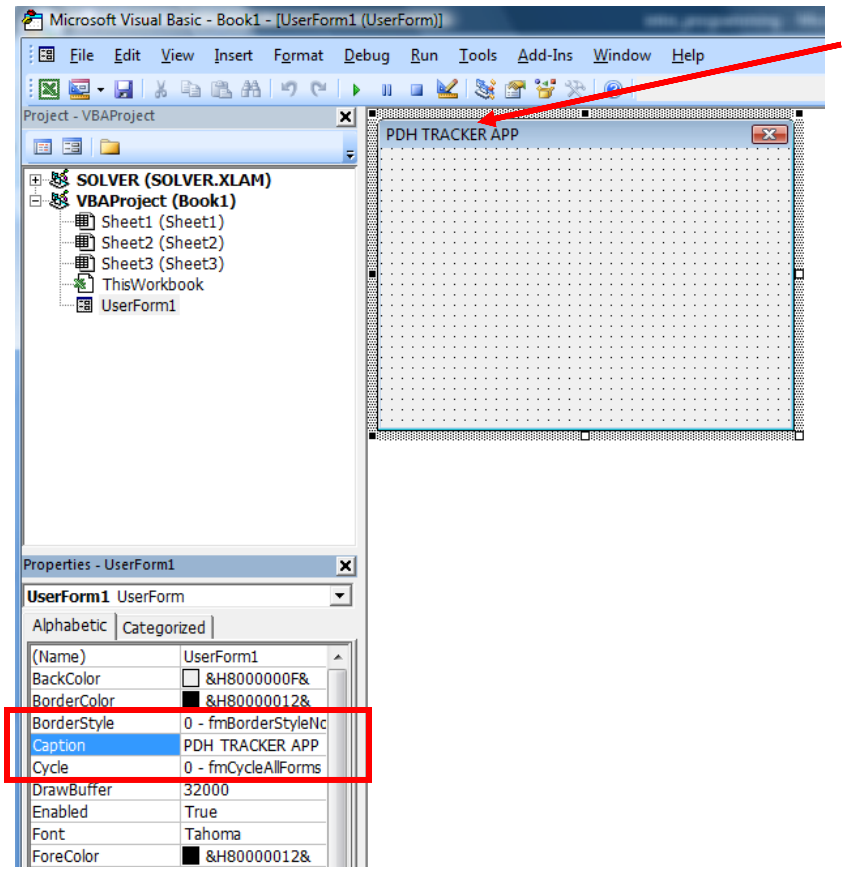
a) Save button: On click, copy entries on form to spreadsheet
b) Clear button: On click, clear all entries on the form and also on the spreadsheet
c) Exit button: On click, close the form

Step 4: Build the GUI and write the VBA code.
Create a new file in Excel and start a new session of VBA.



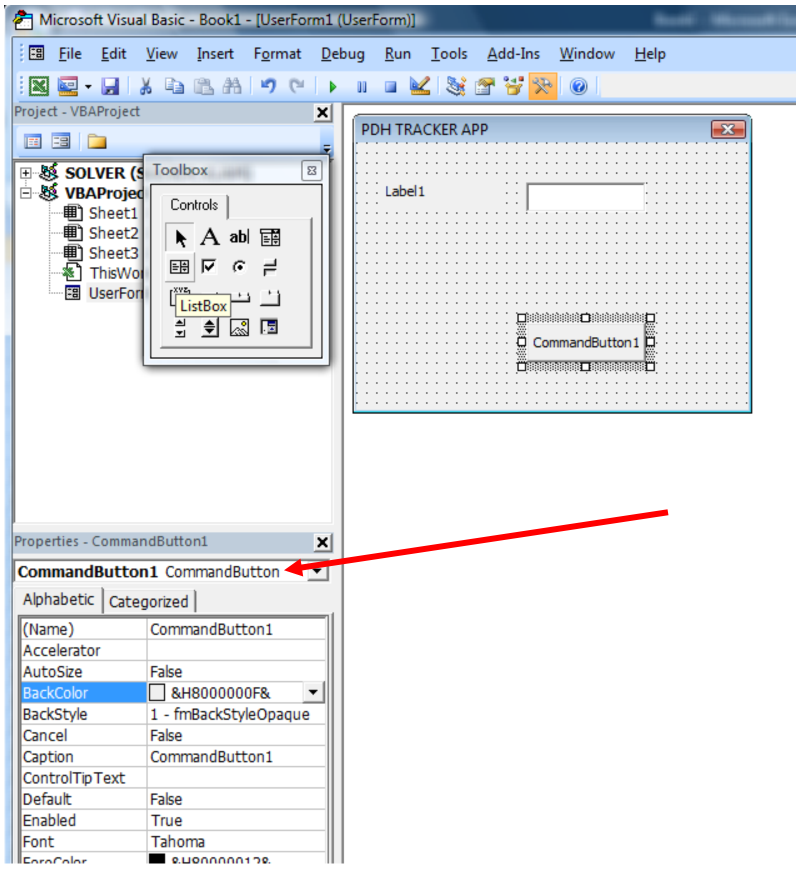
| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H80000 |
| BorderColor | &H80000 |
| BorderStyle | 0 - fmBorder |
| Caption | UserForm1 |

In the properties for the UserForm, look up the **Caption** property and change it to read "PDH TRACKER APP". Note that the Caption property determines the text in the form banner, but the name of the control (form) remains UserForm1. All controls can be manipulated in this manner. The actual name of the control can be changed in the **Name** property.



| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H8000000F& |
| BorderColor | &H80000012& |
| BorderStyle | 0 - fmBorderStyleNc |
| Caption | PDH TRACKER APP |
| Cycle | 0 - fmCycleAllForms |
| DrawBuffer | 32000 |
| Enabled | True |
| Font | Tahoma |
| ForeColor | &H80000012& |

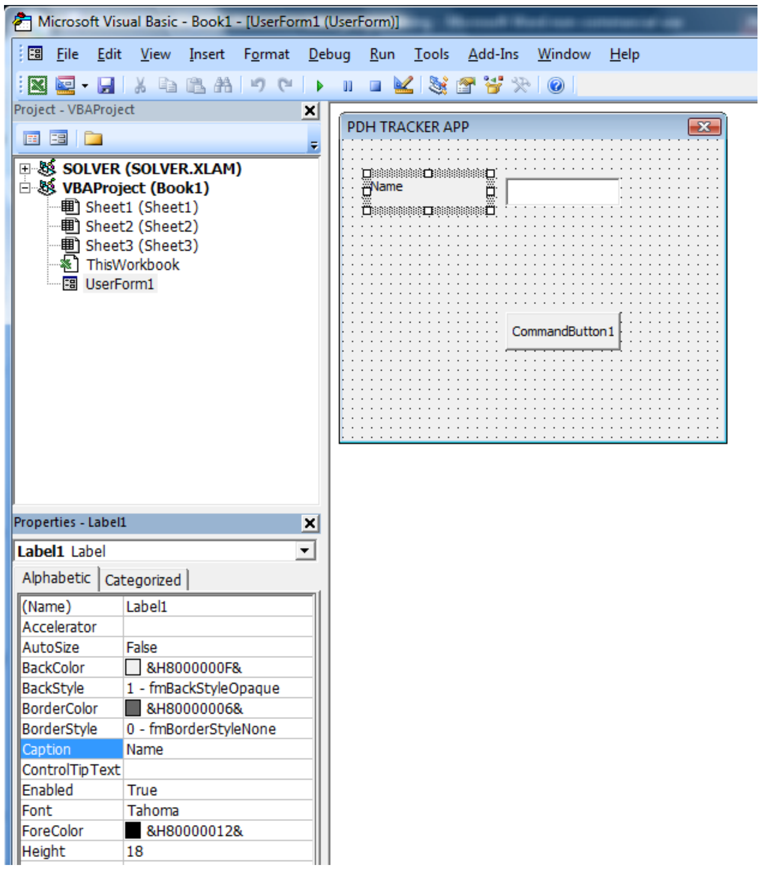
From the Toolbox click and drag one **label**, one **text box** and one **command button** onto the form.
Notice that the Properties Window is now showing the properties of the currently selected control.
Click on and drag the command button control to the desired location on the form.



CommandButton1 CommandButton

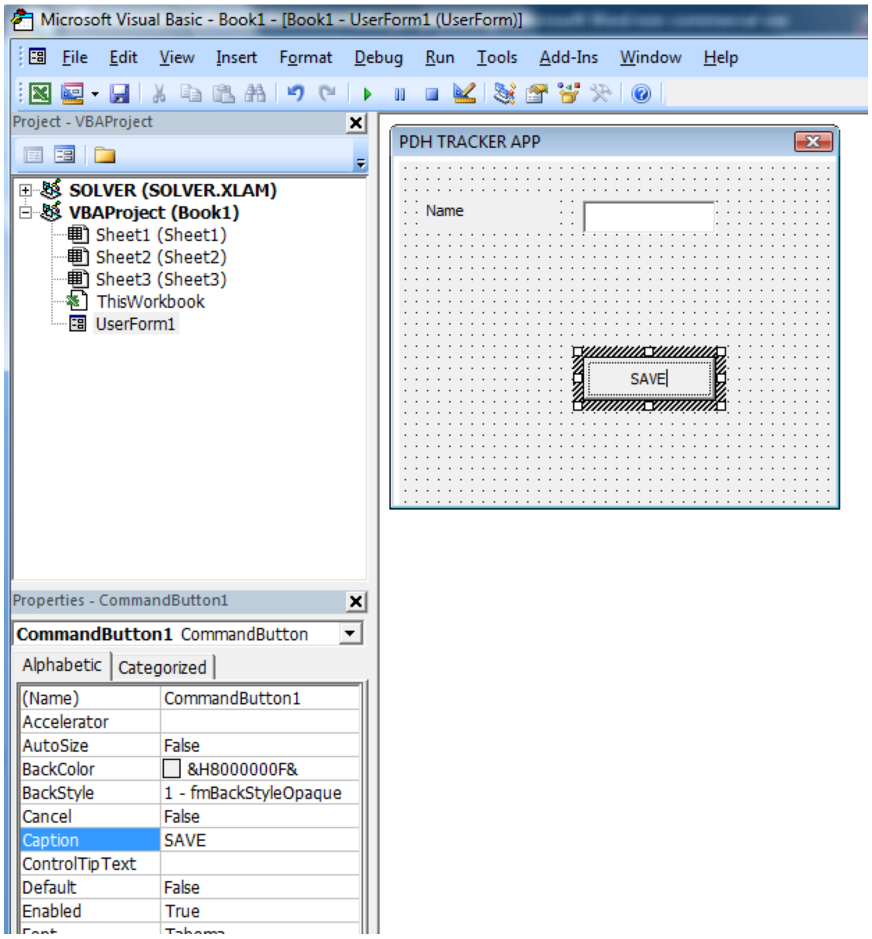
| | |
|---|---|
| (Name) | CommandButton1 |
| Accelerator | |
| AutoSize | False |
| BackColor | &H8000000F& |
| BackStyle | 1 - fmBackStyleOpaque |
| Cancel | False |
| Caption | CommandButton1 |
| ControlTipText | |
| Default | False |
| Enabled | True |
| Font | Tahoma |
| ForeColor | &H80000012& |

Select Label1.
In the Properties window change the Caption to read “Name”.
Alternately hover over the label control and then click on the Caption text “Label1” to highlight it.
Type over it the new Caption “Name”.



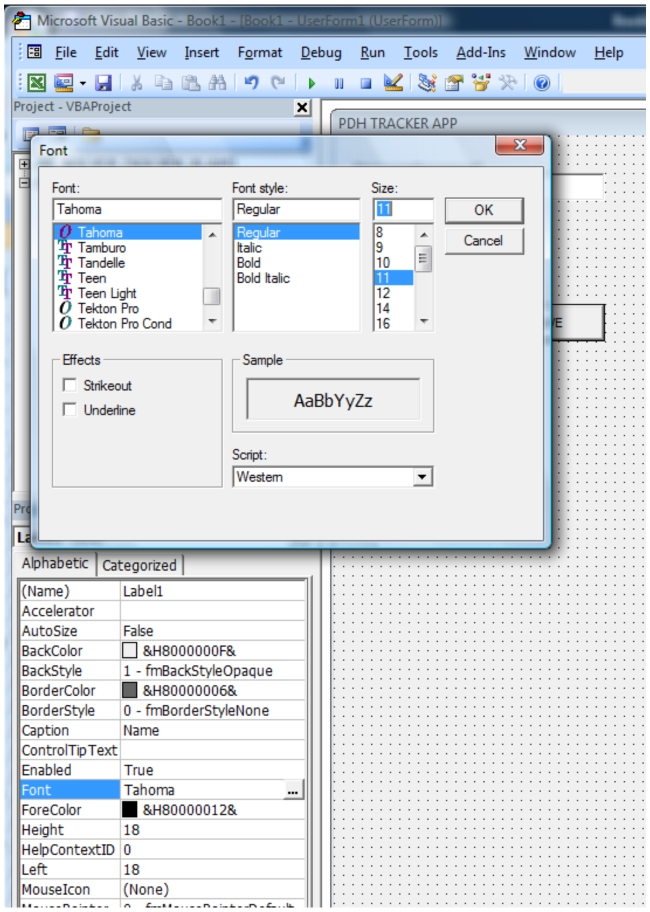
| | |
|---|---|
| (Name) | Label1 |
| Accelerator | |
| AutoSize | False |
| BackColor | □ &H8000000F& |
| BackStyle | 1 - fmBackStyleOpaque |
| BorderColor | ■ &H80000006& |
| BorderStyle | 0 - fmBorderStyleNone |
| Caption | Name |
| ControlTipText | |
| Enabled | True |
| Font | Tahoma |
| ForeColor | ■ &H80000012& |
| Height | 18 |

Change the command button caption to “Save”.


Microsoft Visual Basic - Book1 - [Book1 - UserForm1 (UserForm)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Project - VBAProject
SOLVER (SOLVER.XLAM)
VBAProject (Book1)
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook
UserForm1
PDH TRACKER APP
Name
SAVE
Properties - CommandButton1
CommandButton1 CommandButton
Alphabetic
Categorized

| | |
|---|---|
| (Name) | CommandButton1 |
| Accelerator | |
| AutoSize | False |
| BackColor | &H8000000F& |
| BackStyle | 1 - fmBackStyleOpaque |
| Cancel | False |
| Caption | SAVE |
| ControlTipText | |
| Default | False |
| Enabled | True |

Change the font size of the “Name” label to provide more clarity.


Microsoft Visual Basic - Book1 - [Book1 - UserForm1 (UserForm)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Project - VBAProject
PDH TRACKER APP
Font
OK
Cancel
Effects
Strikeout
Underline
Sample
AaBbYyZz
Alphabetic
Categorized

Font: Tahoma
Tahoma, Tamburo, Tandelle, Teen, Teen Light, Tekton Pro, Tekton Pro Cond

Font style: Regular
Regular, Italic, Bold, Bold Italic

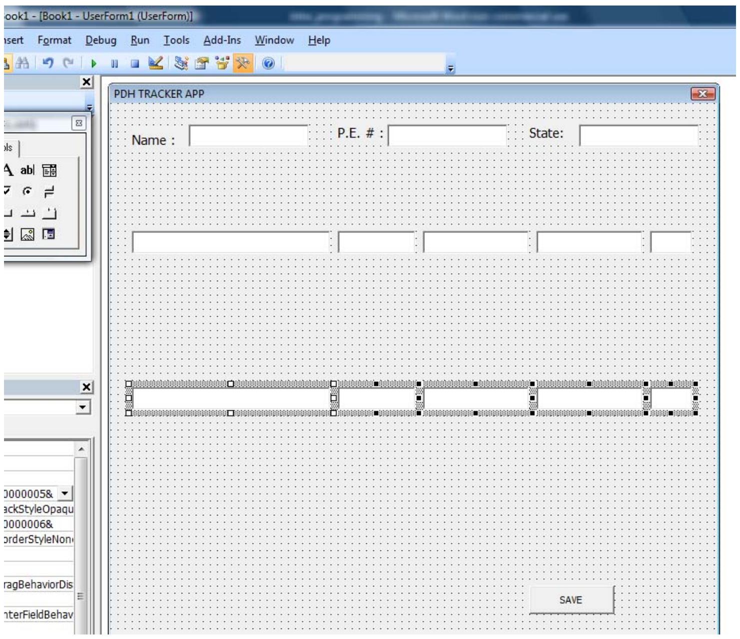
Size: 11
8, 9, 10, 11, 12, 14, 16

Script: Western

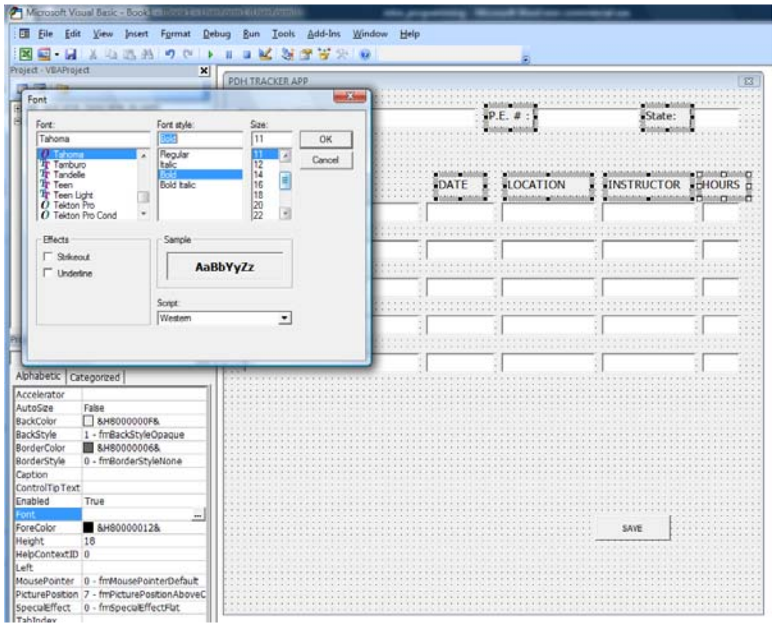
| | |
|---|---|
| (Name) | Label1 |
| Accelerator | |
| AutoSize | False |
| BackColor | □ &H8000000F& |
| BackStyle | 1 - fmBackStyleOpaque |
| BorderColor | ■ &H80000006& |
| BorderStyle | 0 - fmBorderStyleNone |
| Caption | Name |
| ControlTipText | |
| Enabled | True |
| Font | Tahoma |
| ForeColor | ■ &H80000012& |
| Height | 18 |
| HelpContextID | 0 |
| Left | 18 |
| MouseIcon | (None) |

Add more controls to the form, and click and drag on them to reposition and arrange them accordingly to match the layout from the earlier planning stage.
Alternately select a control and make copies of it by **Copy** and **Paste**.

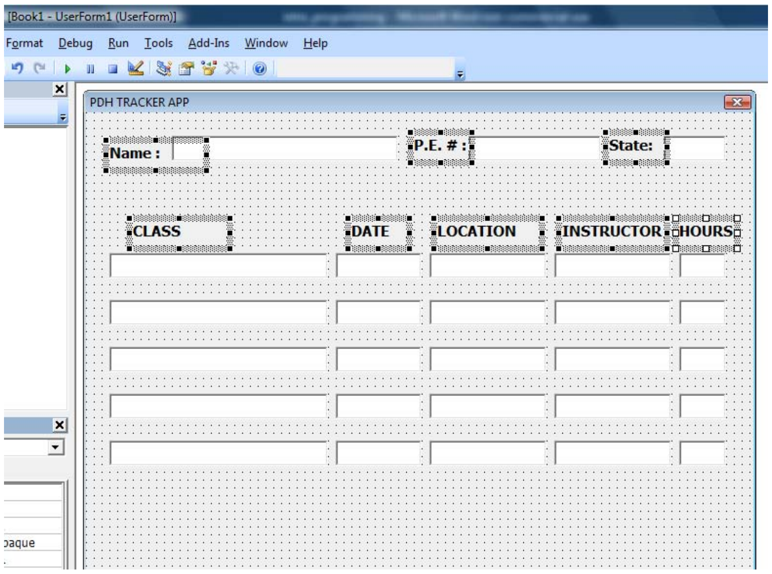
Multiple controls may be selected by holding down the **Shift** key and clicking on each control to select them. The multiple controls selected may be copied and pasted as one block.
The form as well as the controls may need to be resized accordingly.



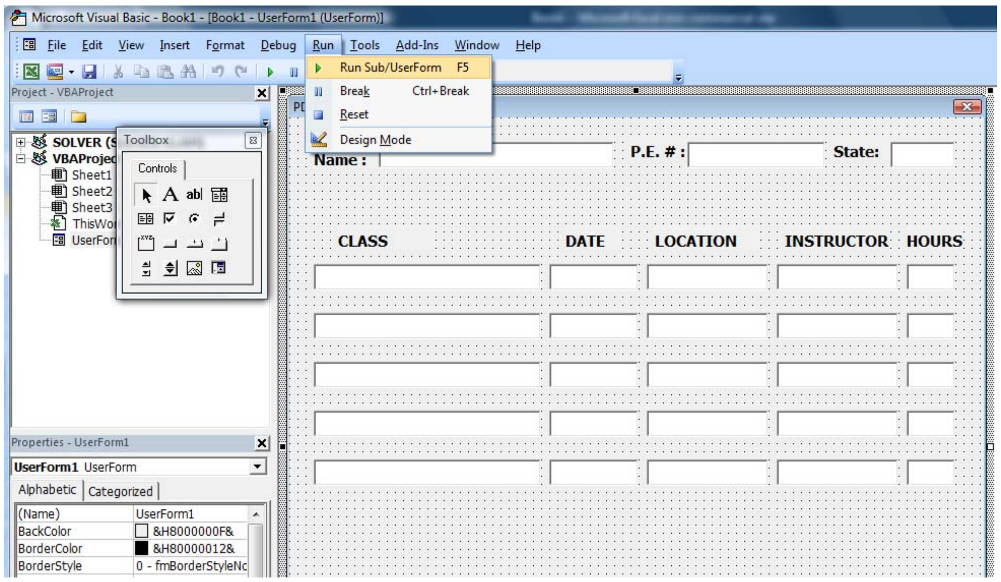
The properties of multiple controls can be changed in one operation given the same property is being changed on all the controls selected.
Select all the column header labels.
In the Properties Window select **Font**.
Change the **Font Style** to **Bold**.



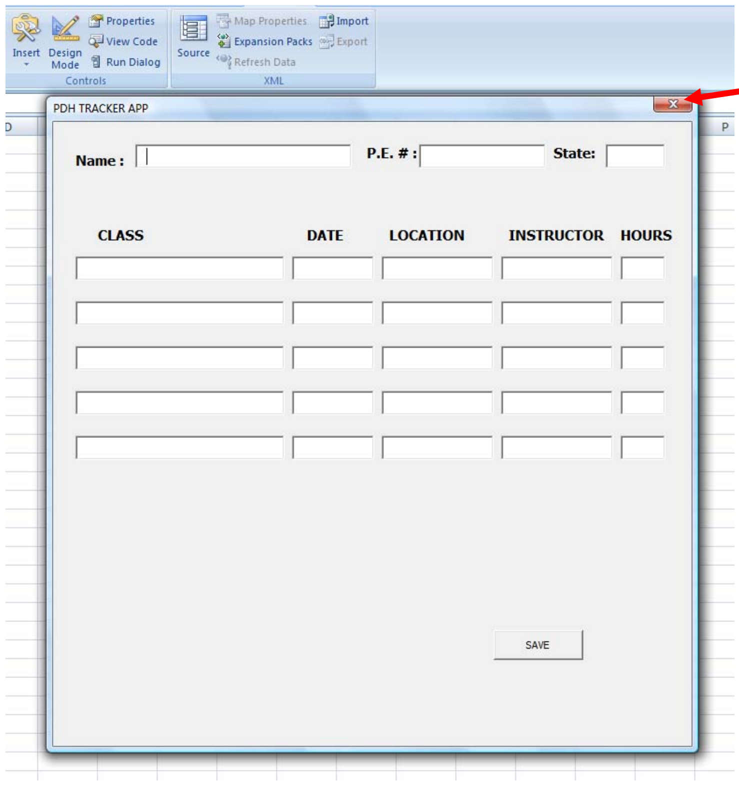
| | |
|---|---|
| Accelerator | |
| AutoSize | False |
| BackColor | &H8000000F& |
| BackStyle | 1 - fmBackStyleOpaque |
| BorderColor | &H80000006& |
| BorderStyle | 0 - fmBorderStyleNone |
| Caption | |
| ControlTipText | |
| Enabled | True |
| Font | |
| ForeColor | &H80000012& |
| Height | 18 |
| HelpContextID | 0 |
| Left | |
| MousePointer | 0 - fmMousePointerDefault |
| PicturePosition | 7 - fmPicturePositionAboveC |
| SpecialEffect | 0 - fmSpecialEffectFlat |


[Book1 - UserForm1 (UserForm)]
Format Debug Run Tools Add-Ins Window Help
PDH TRACKER APP
Name :
P.E. # :
State:
CLASS
DATE
LOCATION
INSTRUCTOR
HOURS

At this time the application may be tested.
Click on the **Run** menu and Select **Run Sub/ UserForm**, or simply click on the Run button in the Toolbar.

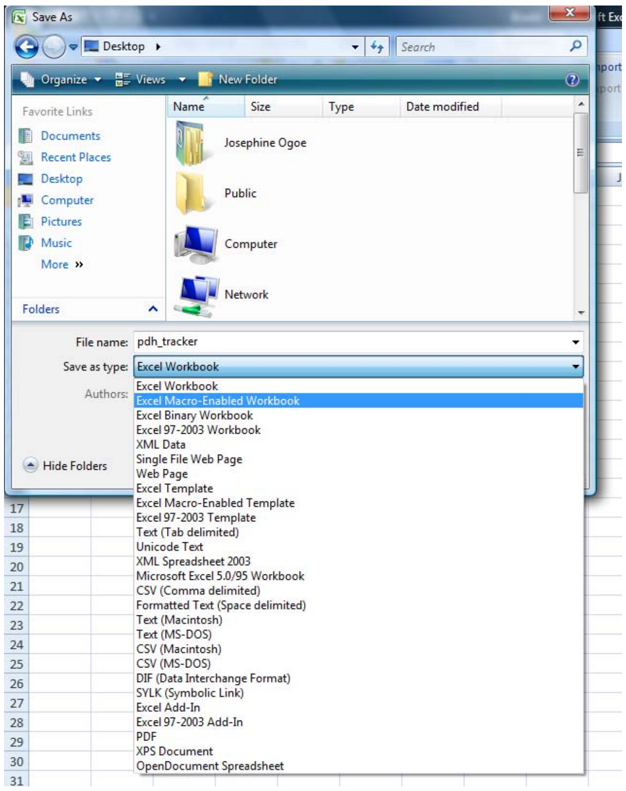


The UserForm goes into run time. If code has been written, it will be executed when the appropriate button is clicked on.



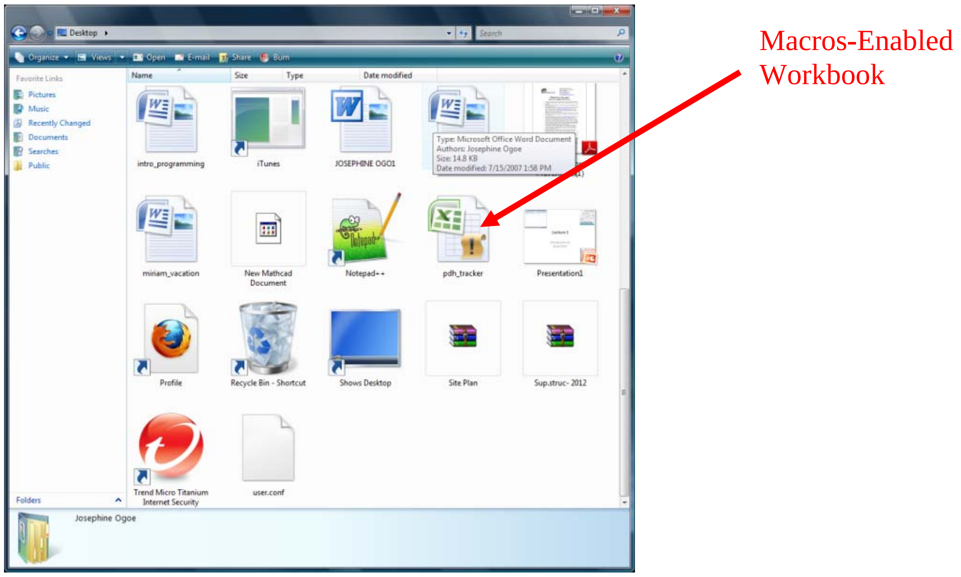
Click to leave run time and return to design time

So far the file and VBA application have not been saved.
To save, from the VBA environment, click on Save.
Select a folder, give the file a name.
In the **Save as Type**, select **Excel Macros-Enabled Workbook**.
Click on OK.



File name: pdh_tracker
Save as type: Excel Workbook

The Macros-Enabled Workbook is an Excel file which has VBA code and applications added by the user. Macros–Enabled Workbooks can be distinguished from other Excel files by their logo.



At this time, code can now be written for the controls. From the flow chart it can be seen that the events in this program are associated with the command buttons. Code needs to written to control the behavior of the controls once a command button is clicked on.

## The Exit Button

Upon clicking on this button in run time the form closes. The VBA syntax to close a form is

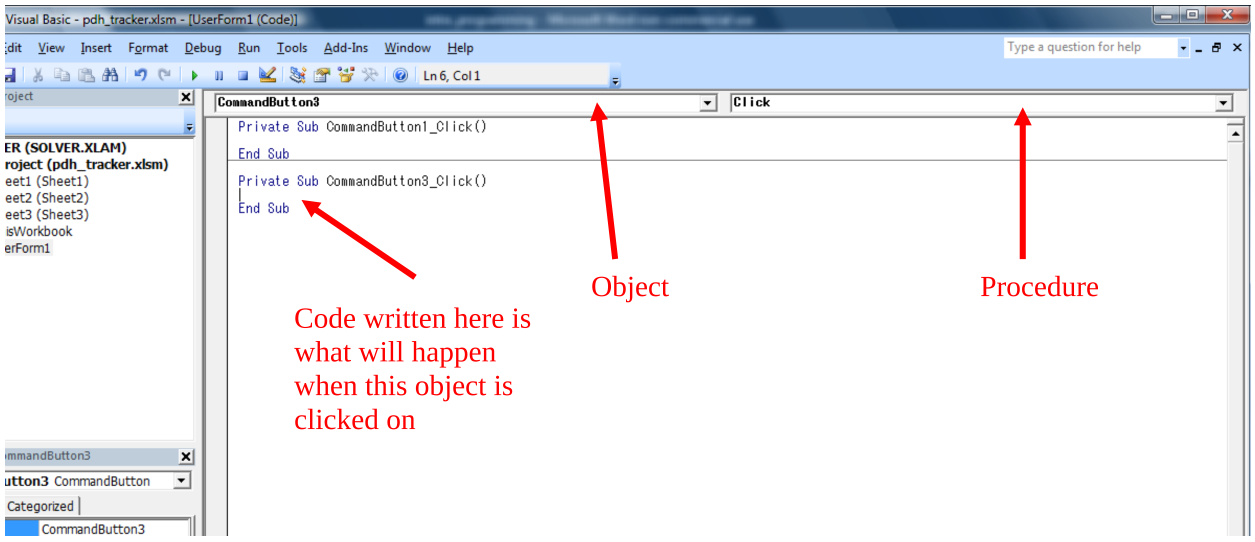
*Unload Formname*

where
*Formname* is the name of the form.

The name of the form may be looked up by clicking on the form and reviewing the Name property in the Properties Window. In this case the correct code would be

*Unload UserForm1*

To associate this code with the Exit button, select the Exit button and click on View Code, or simply double click on the Exit button to go straight to the code window for procedures related to this control.
The Exit button in this project is *CommandButton3*.

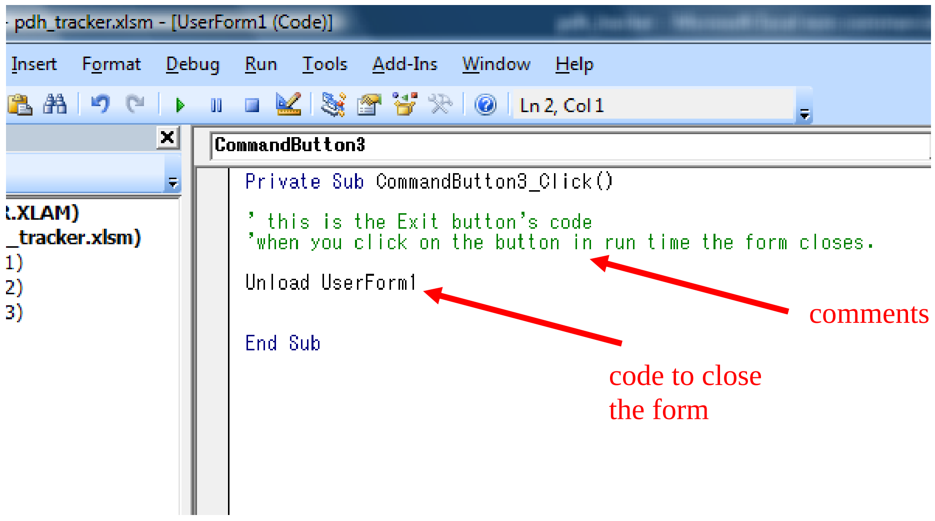


```
Private Sub CommandButton1_Click()
End Sub

Private Sub CommandButton3_Click()
|
End Sub
```

Visual Basic - pdh_tracker.xlsm - [UserForm1 (Code)]
Edit View Insert Format Debug Run Tools Add-Ins Window Help
Type a question for help
Ln 6, Col 1
CommandButton3
Click
ER (SOLVER.XLAM)
roject (pdh_tracker.xlsm)
eet1 (Sheet1)
eet2 (Sheet2)
eet3 (Sheet3)
isWorkbook
erForm1
mmandButton3
utton3 CommandButton
Categorized
CommandButton3
Object
Procedure
Code written here is what will happen when this object is clicked on

Type the code to close the UserForm in the click event procedure for *CommandButton3*. Add a **comment** line to remind the programmer what this code will do when executed.

In VBA to create a comment type the apostrophe. Anything typed to the right of the apostrophe on that line becomes a comment. The compiler recognizes comments to be informational statements by and for the programmer and does not attempt to execute them. Comments are extremely useful particularly in large and complex programs. Comments provide a simple and plain way for a third party to follow what the program is doing by reading them. The rule of thumb is to write as many comments as necessary.


pdh_tracker.xlsm - [UserForm1 (Code)]
Insert Format Debug Run Tools Add-Ins Window Help
Ln 2, Col 1
CommandButton3
.XLAM)
_tracker.xlsm)
1)
2)
3)
comments
code to close the form

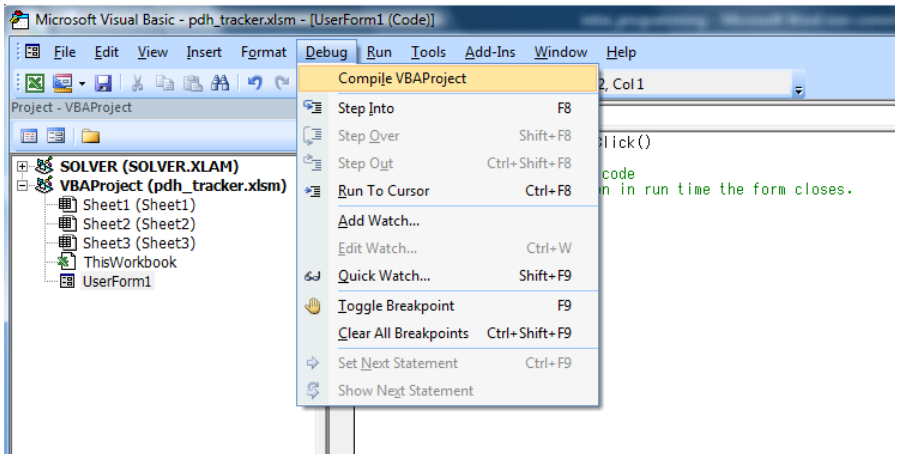
```
Private Sub CommandButton3_Click()

' this is the Exit button's code
'when you click on the button in run time the form closes.

Unload UserForm1

End Sub
```

In some cases, an instruction (command) may be too long to fit laterally on the screen and it would be desirable for the programmer to type the command over multiple lines. This can be done by creating a **line break**. In VBA, a line break is created by typing a space (Spacebar on keyboard), followed by underscore (Shift + hyphen key), followed by enter (Enter key) to go to the next line, even though technically, to the compiler, the cursor is still on the same line.

Before running the code check for any **syntax errors**. A syntax error is an error that occurs due to typing incorrect code as per the rules of the programming language being used. Syntax errors may be caused by typographical errors, misspelling, omission of characters etc.

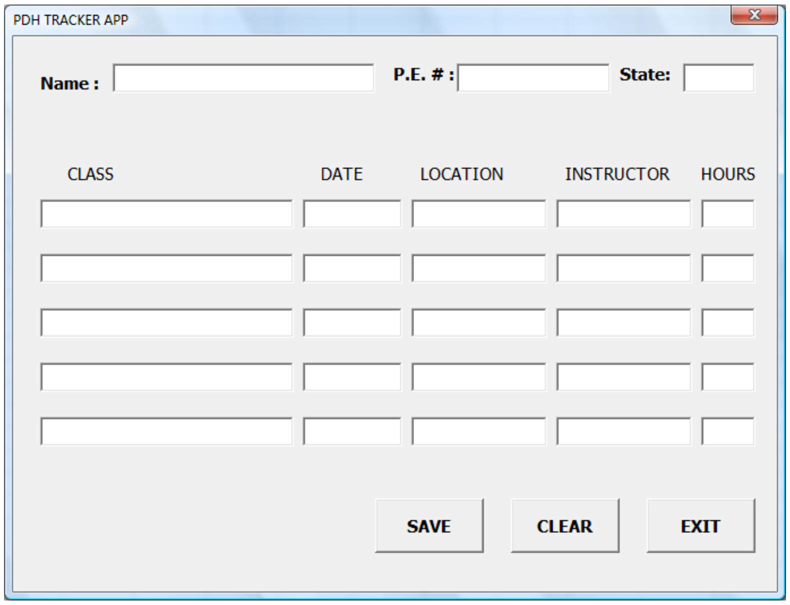
Checking for syntax errors can be done rapidly by selecting the **Debug** menu, and selecting **Compile VBAProject**.

If the code contains errors or omissions, those areas in the code will be flagged for attention.

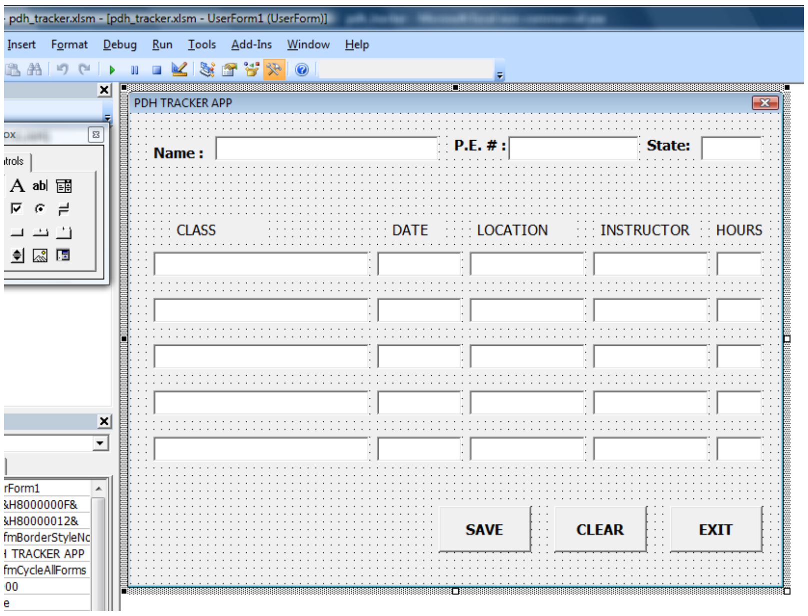


Once the Compile VBAProject produces no further errors, the program may now be tested.

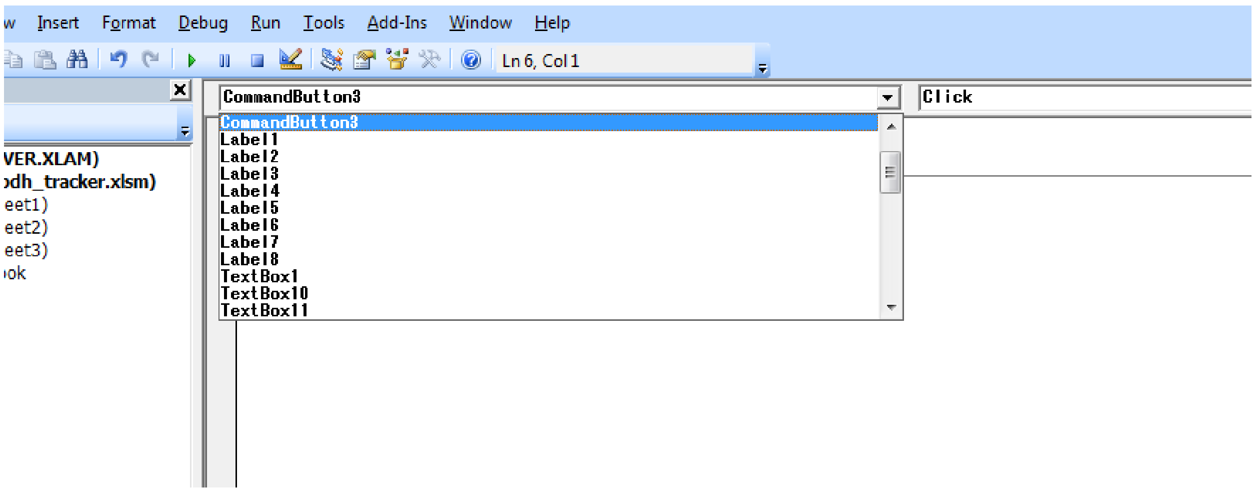
Click on Run to activate the form.



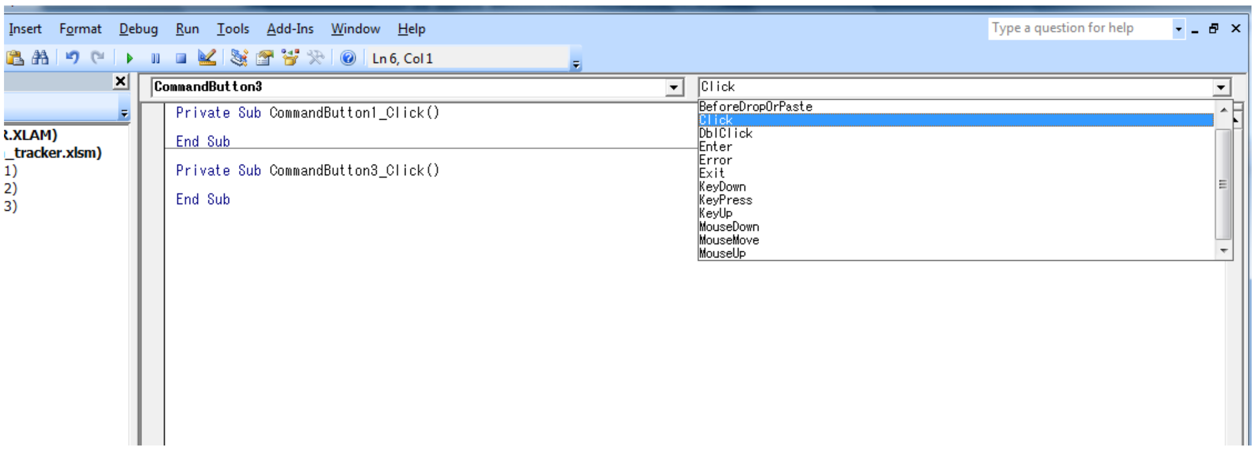
Now click on the Exit button. The form closes as expected, and sends the user back to design time.


pdh_tracker.xlsm - [pdh_tracker.xlsm - UserForm1 (UserForm)]
Insert Format Debug Run Tools Add-Ins Window Help
PDH TRACKER APP
Name :
P.E. # :
State:
CLASS
DATE
LOCATION
INSTRUCTOR
HOURS
SAVE
CLEAR
EXIT

To view other objects in this project click on the objects drop down.



To see other procedures and associated code relevant to any object, click on the events dropdown.



```
Private Sub CommandButton1_Click()
End Sub
Private Sub CommandButton3_Click()
End Sub
```

The Save Button

The Save button shall save the entries on the form to a spreadsheet. In other words each text box entry on the form will saved to a cell on the spreadsheet. The reference to a cell on a spreadsheet is as follows

*Sheets("sheetname").cells(i, j)*

where

*sheetname* is the name of the spreadsheet
*i* is the row number of the cell on the spreadsheet
*j* is the column number of the cell on the spreadsheet

So for example, the cell D3 on spreadsheet "Sheet1" would be referenced as

*Sheets("Sheet1").cells(3, 4)*

Following from the principles of object-oriented programming, a spreadsheet is an object to which properties and methods can be associated. So the value in the cell would be a property of that cell and would be referenced as

*Sheets("sheetname").cells(i, j).value*

In a similar manner a control on a form will be referenced as

*UserFormname.controlname*

and the value it holds as

*UserFormname.controlname.value*

So for a Textbox1 on a UserForm1, the reference to the text box and to its value would respectively be

*UserForm1.TextBox1*

and

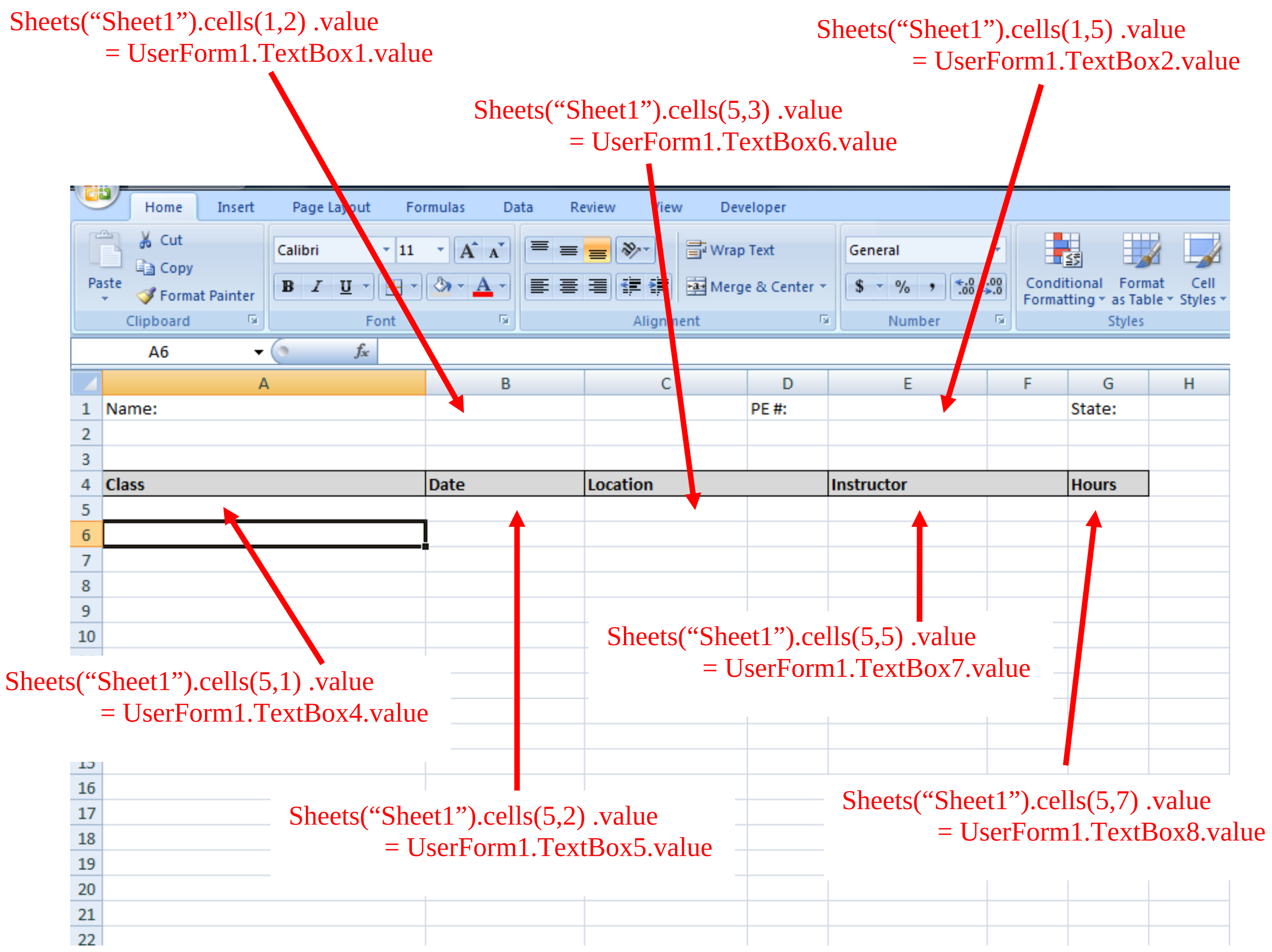
*UserForm1.TextBox1.value*

To save a text box value to a cell on the spreadsheet the **assignment operator**, the equals sign, is used, as follows.

*destination* = *origin*

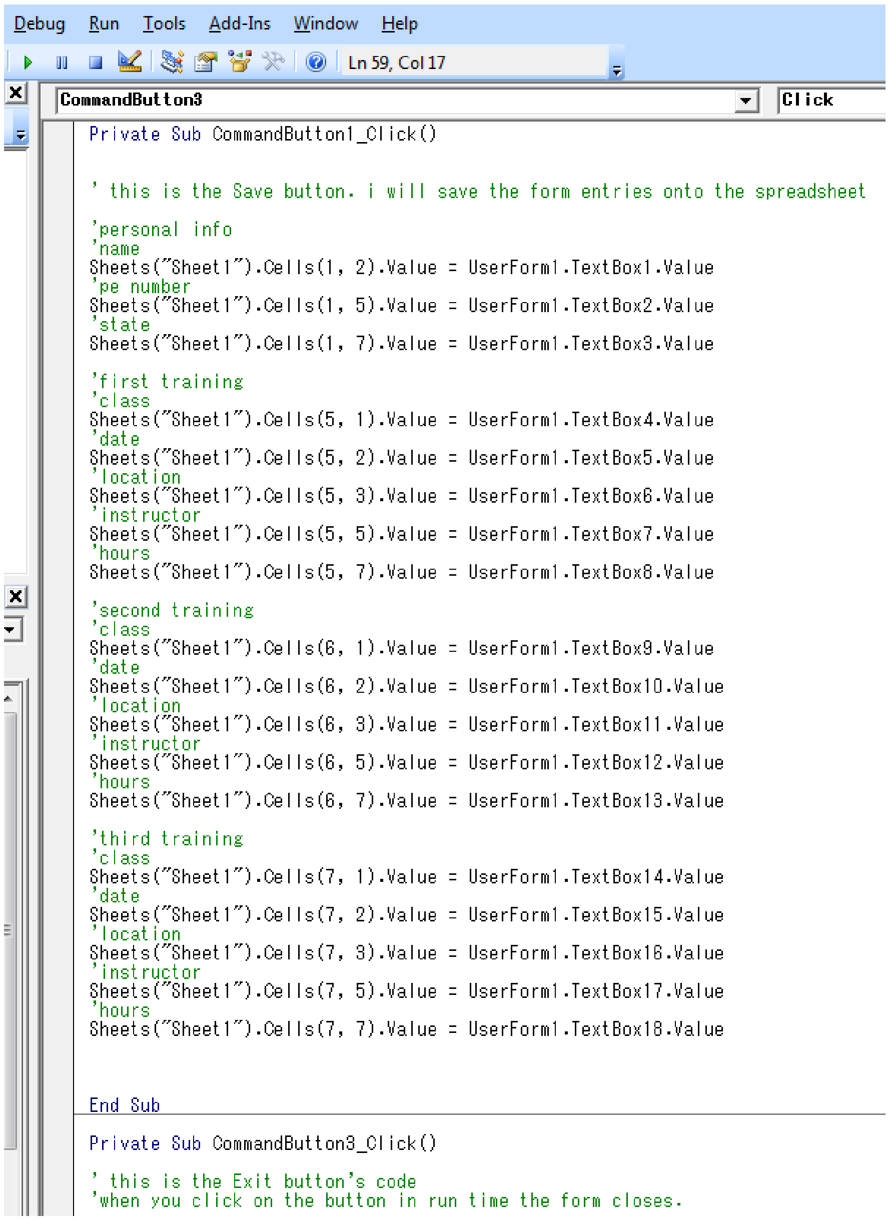
In other words the value of the right hand side of the equality sign will always be assigned to the destination on the left hand side, and never vice versa.

Based on the following framework on the spreadsheet, and the framework established on the UserForm1, some of the control-to-cell assignments will be as follows



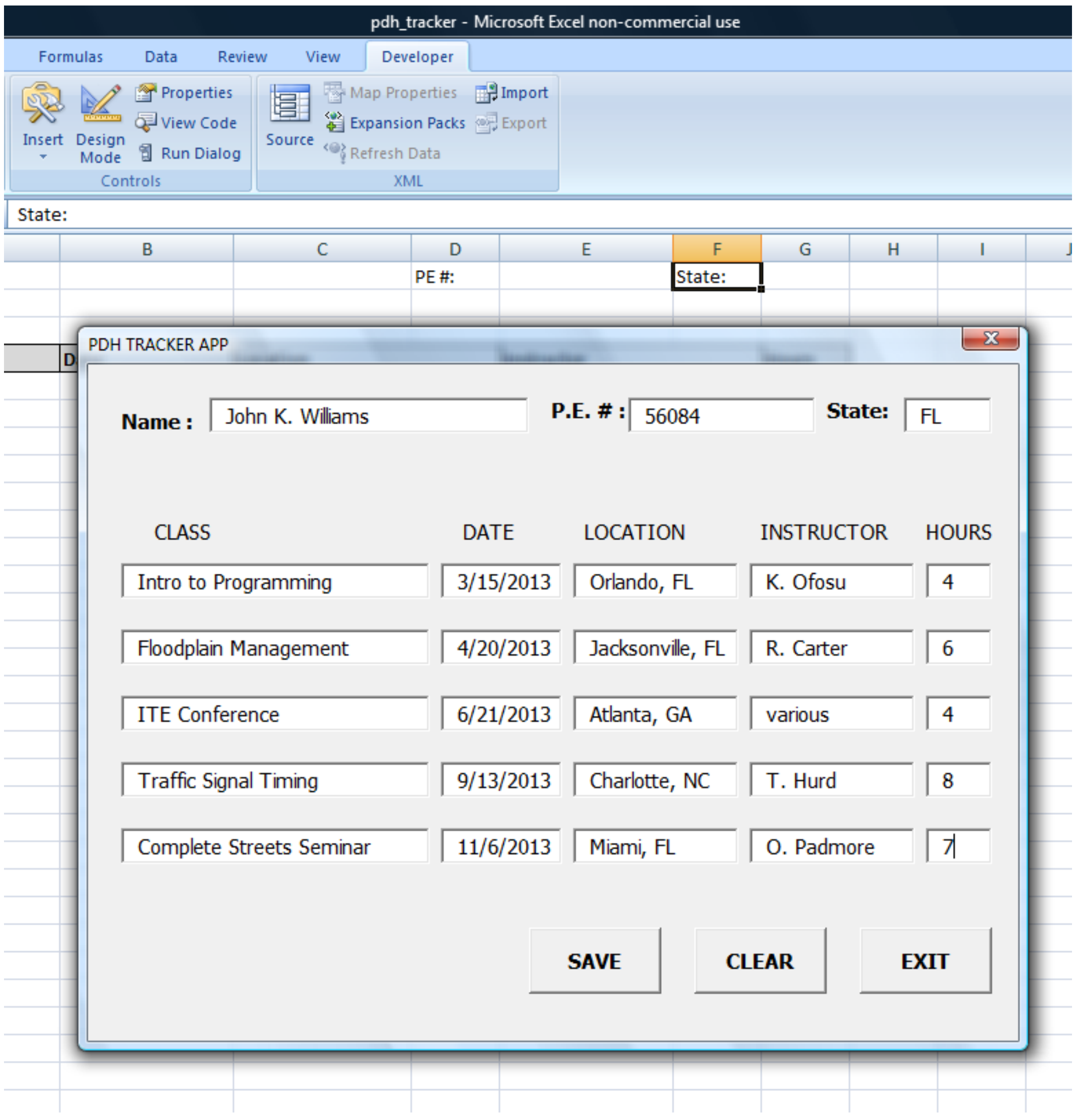
```
Sheets("Sheet1").cells(1,2) .value
    = UserForm1.TextBox1.value
Sheets("Sheet1").cells(1,5) .value
    = UserForm1.TextBox2.value
Sheets("Sheet1").cells(5,3) .value
    = UserForm1.TextBox6.value
Sheets("Sheet1").cells(5,5) .value
    = UserForm1.TextBox7.value
Sheets("Sheet1").cells(5,1) .value
    = UserForm1.TextBox4.value
Sheets("Sheet1").cells(5,2) .value
    = UserForm1.TextBox5.value
Sheets("Sheet1").cells(5,7) .value
    = UserForm1.TextBox8.value
```

The code for the Save button will therefore be a compilation of these control-to-cells assignments.
In the design window double click on the Save button to open its **click Procedure**.
In the click procedure, list all the assignment statements.
Add comments to enhance clarity and for a third party to be able to review the code with minimal problems.



```
Private Sub CommandButton1_Click()

' this is the Save button. i will save the form entries onto the spreadsheet

'personal info
'name
Sheets("Sheet1").Cells(1, 2).Value = UserForm1.TextBox1.Value
'pe number
Sheets("Sheet1").Cells(1, 5).Value = UserForm1.TextBox2.Value
'state
Sheets("Sheet1").Cells(1, 7).Value = UserForm1.TextBox3.Value

'first training
'class
Sheets("Sheet1").Cells(5, 1).Value = UserForm1.TextBox4.Value
'date
Sheets("Sheet1").Cells(5, 2).Value = UserForm1.TextBox5.Value
'location
Sheets("Sheet1").Cells(5, 3).Value = UserForm1.TextBox6.Value
'instructor
Sheets("Sheet1").Cells(5, 5).Value = UserForm1.TextBox7.Value
'hours
Sheets("Sheet1").Cells(5, 7).Value = UserForm1.TextBox8.Value

'second training
'class
Sheets("Sheet1").Cells(6, 1).Value = UserForm1.TextBox9.Value
'date
Sheets("Sheet1").Cells(6, 2).Value = UserForm1.TextBox10.Value
'location
Sheets("Sheet1").Cells(6, 3).Value = UserForm1.TextBox11.Value
'instructor
Sheets("Sheet1").Cells(6, 5).Value = UserForm1.TextBox12.Value
'hours
Sheets("Sheet1").Cells(6, 7).Value = UserForm1.TextBox13.Value

'third training
'class
Sheets("Sheet1").Cells(7, 1).Value = UserForm1.TextBox14.Value
'date
Sheets("Sheet1").Cells(7, 2).Value = UserForm1.TextBox15.Value
'location
Sheets("Sheet1").Cells(7, 3).Value = UserForm1.TextBox16.Value
'instructor
Sheets("Sheet1").Cells(7, 5).Value = UserForm1.TextBox17.Value
'hours
Sheets("Sheet1").Cells(7, 7).Value = UserForm1.TextBox18.Value


End Sub
Private Sub CommandButton3_Click()

' this is the Exit button's code
'when you click on the button in run time the form closes.
```

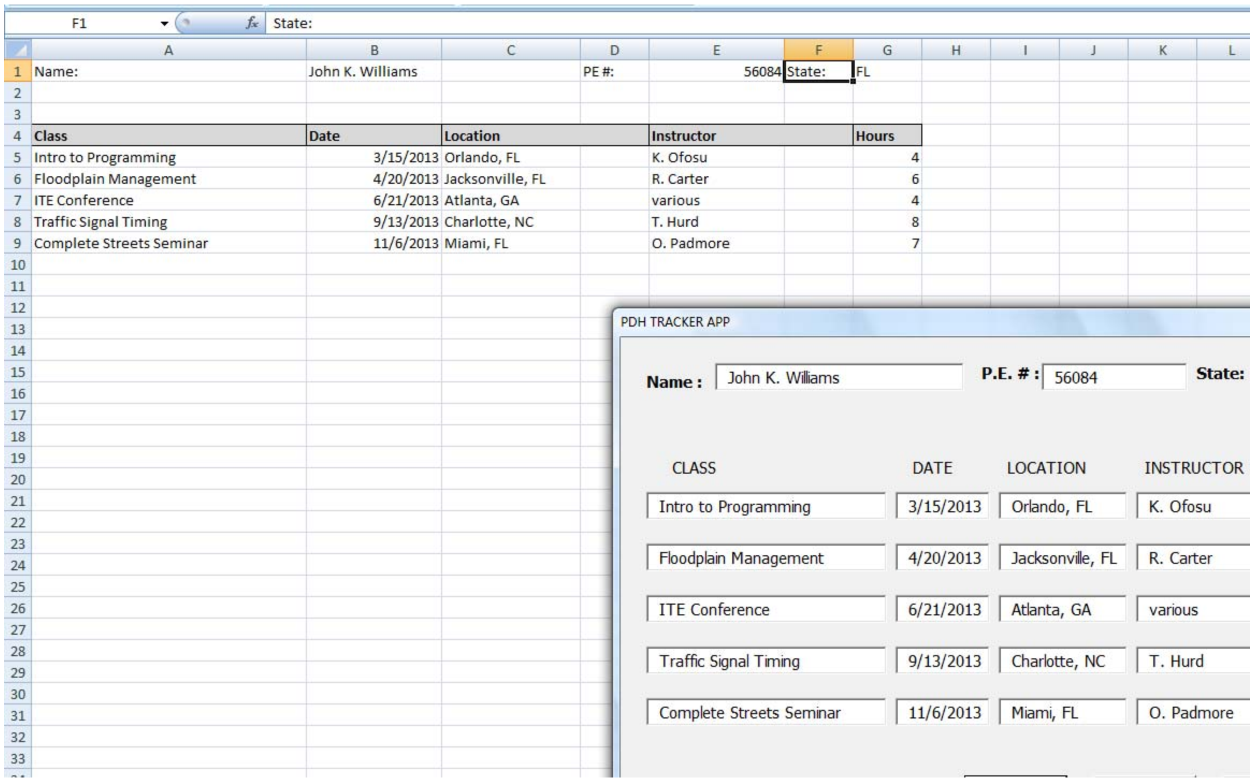
Test the Save button.
First, enter information in the "PDH TRACKER APP" GUI.



Name : John K. Williams  P.E. # : 56084  State: FL

| CLASS | DATE | LOCATION | INSTRUCTOR | HOURS |
|---|---|---|---|---|
| Intro to Programming | 3/15/2013 | Orlando, FL | K. Ofosu | 4 |
| Floodplain Management | 4/20/2013 | Jacksonville, FL | R. Carter | 6 |
| ITE Conference | 6/21/2013 | Atlanta, GA | various | 4 |
| Traffic Signal Timing | 9/13/2013 | Charlotte, NC | T. Hurd | 8 |
| Complete Streets Seminar | 11/6/2013 | Miami, FL | O. Padmore | 7 |

Click on the Save button.
Inspect the spreadsheet "Sheet1" to confirm that all commands have been correctly executed.



F1 State:

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Name: | John K. Williams | | PE #: | 56084 | State: | FL |
| 2 | | | | | | | |
| 3 | | | | | | | |
| 4 | Class | Date | Location | | Instructor | | Hours |
| 5 | Intro to Programming | 3/15/2013 | Orlando, FL | | K. Ofosu | | 4 |
| 6 | Floodplain Management | 4/20/2013 | Jacksonville, FL | | R. Carter | | 6 |
| 7 | ITE Conference | 6/21/2013 | Atlanta, GA | | various | | 4 |
| 8 | Traffic Signal Timing | 9/13/2013 | Charlotte, NC | | T. Hurd | | 8 |
| 9 | Complete Streets Seminar | 11/6/2013 | Miami, FL | | O. Padmore | | 7 |

PDH TRACKER APP

Name : John K. Williams   P.E. # : 56084   State:

| CLASS | DATE | LOCATION | INSTRUCTOR |
|---|---|---|---|
| Intro to Programming | 3/15/2013 | Orlando, FL | K. Ofosu |
| Floodplain Management | 4/20/2013 | Jacksonville, FL | R. Carter |
| ITE Conference | 6/21/2013 | Atlanta, GA | various |
| Traffic Signal Timing | 9/13/2013 | Charlotte, NC | T. Hurd |
| Complete Streets Seminar | 11/6/2013 | Miami, FL | O. Padmore |

The test is a success.
Close out and return to design time.

The Clear Button

The Clear button shall clear all entries in each control on the form. It will then go to the spreadsheet and clear the data that has been saved there from the form.

There are number of ways to clear the entry in a control, or change the object value property value to “blank”.

- *Sheets(“sheetname”).cells(i, j).ClearContents*,
  e.g. *Sheets(“Sheet1”).cells(7,1).ClearContents*

- *Sheets(“sheetname”).cells(i, j).value = Null* and
  *UserForm.Controlname.value = Null*
  e.g. *Sheets(“Sheet1”).cells(7,1).value = Null*

- *Sheets(“sheetname”).cells(i, j).value = “”* and
  *UserForm.Controlname.value = “”*
  e.g. *UserForm1.TextBox8.value = “ ”*

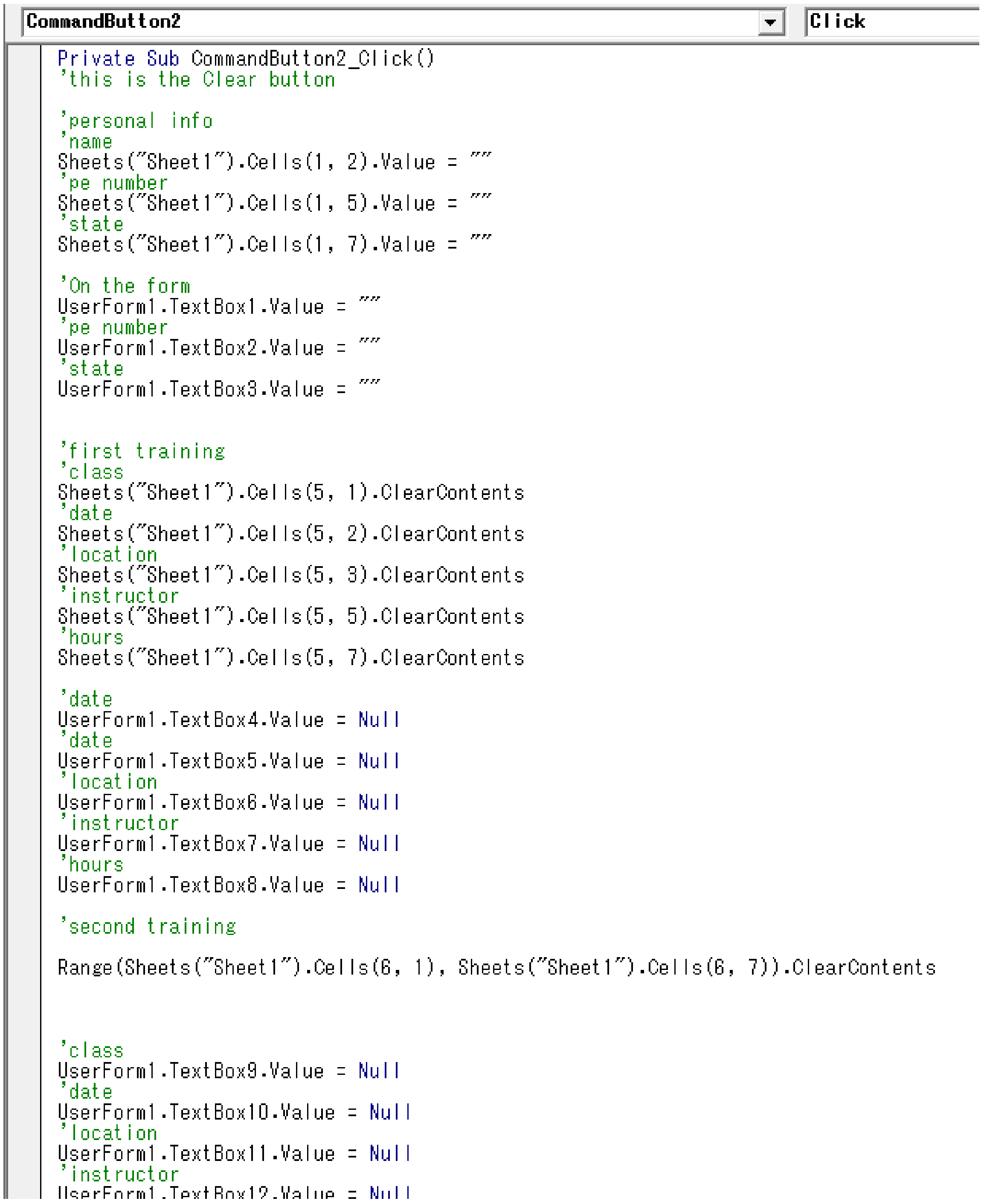
Alternately, for the spreadsheet, a **range** of cells can be specified and then contents erased. The syntax would be

Range(*Sheets(“sheetname”).cells(i, j), Sheets(“sheetname”).cells(i, j)).ClearContents*

So for range A5:G9 on the spreadsheet, the syntax will be

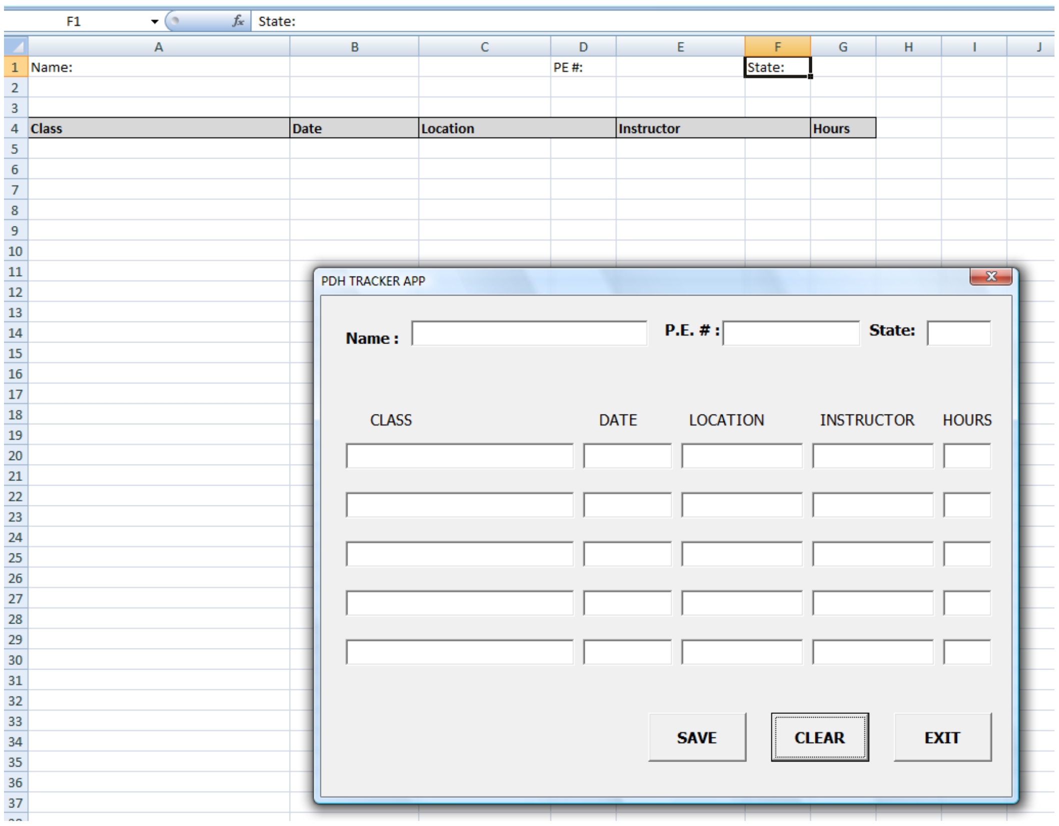
*Range(Sheets(“Sheet1”).cells(5, 1), Sheets(“Sheet1”).cells(9, 7)).ClearContents*

A combination of all of the above will be used to clear the spreadsheet and the form entries in this example.
Double click on the Save button from design time to open up the procedure for it.
Type the code to clear the contents of all controls on the form.
Also clear the spreadsheet entries.



```
Private Sub CommandButton2_Click()
'this is the Clear button

'personal info
'name
Sheets("Sheet1").Cells(1, 2).Value = ""
'pe number
Sheets("Sheet1").Cells(1, 5).Value = ""
'state
Sheets("Sheet1").Cells(1, 7).Value = ""

'On the form
UserForm1.TextBox1.Value = ""
'pe number
UserForm1.TextBox2.Value = ""
'state
UserForm1.TextBox3.Value = ""


'first training
'class
Sheets("Sheet1").Cells(5, 1).ClearContents
'date
Sheets("Sheet1").Cells(5, 2).ClearContents
'location
Sheets("Sheet1").Cells(5, 3).ClearContents
'instructor
Sheets("Sheet1").Cells(5, 5).ClearContents
'hours
Sheets("Sheet1").Cells(5, 7).ClearContents

'date
UserForm1.TextBox4.Value = Null
'date
UserForm1.TextBox5.Value = Null
'location
UserForm1.TextBox6.Value = Null
'instructor
UserForm1.TextBox7.Value = Null
'hours
UserForm1.TextBox8.Value = Null

'second training

Range(Sheets("Sheet1").Cells(6, 1), Sheets("Sheet1").Cells(6, 7)).ClearContents



'class
UserForm1.TextBox9.Value = Null
'date
UserForm1.TextBox10.Value = Null
'location
UserForm1.TextBox11.Value = Null
'instructor
UserForm1.TextBox12.Value = Null
```

Click on the **Debug** menu, and select **Compile VBAProject**.
Correct any syntax errors highlighted.
If none, launch the form to test it.
Fill out the information requested.
Click on Save to copy the form information onto the spreadsheet.
Click on the Clear button to clear the information on the form and on the spreadsheet.



The test is a success.
Click Exit to close the form and return to design time.
Save the workbook
Close out of Excel.

# 3. WORKING WITH VARIABLES

## 3.1 Variables

A variable is area of computer memory allocated to hold data of a certain type. A simplistic analogy is a mailbox in a corporate office staff room. The mailbox for each employee must have a unique identification and will be restricted by its physical size to holding a certain kind or type of mail delivery. Once a variable is set up, its content will become the value of the variable. When the variable is called for some calculation, its value will be inserted into the calculation process. In programming, variables are necessary to facilitate calculations and data manipulations that otherwise will be difficult or inefficient to do by directly using the values of the form controls.

## 3.2 Data Types

VBA supports several data types. Each data type has a range of values it can store, and consumes a specific amount of computer memory in doing so. Programmers must choose the appropriate data type, keeping in mind the impact it will have on the computer's resources, and the speed and smooth running of programs. Table 1 summarizes the data types supported by VBA.

**Table 1: Variable data types**

| Data Type | Storage | Range of Values |
|---|---|---|
| Byte | 1 byte | 0 to 255 |
| Integer | 2 bytes | -32,768 to 32,767 |
| Long | 4 bytes | -2,147,483,648 to 2,147,483,648 |
| Single | 4 bytes | -3.402823E+38 to -1.401298E-45 for negative values<br>1.401298E-45 to 3.402823E+38 for positive values. |
| Double | 8 bytes | -1.79769313486232e+308 to -4.94065645841247E-324 for negative values<br>4.94065645841247E-324 to 1.79769313486232e+308 for positive values. |

**Table 1 (continued): variable data types**

| Data Type | Storage | Range of Values |
|---|---|---|
| Currency | 8 bytes | -922,337,203,685,477.5808 to 922,337,203,685,477.5807 |
| Decimal | 12 bytes | +/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use<br>+/- 7.9228162514264337593543950335 (28 decimal places). |
| String (fixed length) | Length of string | 1 to 65,400 characters |
| String (variable length) | Length + 10 bytes | 0 to 2 billion characters |
| Date | 8 bytes | January 1, 100 to December 31, 9999 |
| Boolean | 2 bytes | True or False |
| Object | 4 bytes | Any embedded object |
| Variant (numeric) | 16 bytes | Any value as large as Double |
| Variant (text) | Length+22 bytes | Same as variable-length string |

### 3.3 Rules for Naming Variables

In VBA variable names must meet the following requirements:

- It must be less than 255 characters
- No space, period, punctuation, or special characters are allowed in the variable name
- It must begin with an alphabetical character
- It must not be a reserved word in the VBA language e.g. Sub, TextBox, UserForm, etc.
- It must be a unique name in the procedure in which it is being used

Table 2 show examples of acceptable versus inadmissible variable names.

**Table 2: Variable names**

| **Valid Variable Name** | **Unacceptable Variable Name** |
|---|---|
| My_House | My.House (dot not allowed) |
| ThisMonth | 1NewCar (must start with alphabetical) |
| Long_Lastname_isALLOWED | Me&You (& special character not allowed) |
| Car99 | Employee SSN (space not allowed) |

### 3.4 Variable Name Conventions

As programs get large and complex it becomes advantageous to establish a name convention to manage the variables. Following a name convention can make it easier to identify the type of the variable. It will also make the code easier to figure out when it is reviewed by a third party or revisited after an extended period of time. For programs and applications developed by multiple individuals or teams of programmers, adopting a common name convention for variables will provide a common platform for communication and prevent confusion.

Different programming languages are amenable to different name conventions. One common convention used in VBA is to append a three letter prefix that indicates the data type of that variable, in lower case letters, and the variable name proper starting with an uppercase letter.

Optionally, especially for larger and yet more complex programs, the prefix can be extended to include the **scope** (described in Section 3.6) of the variable. Table 3 shows examples of commonly used prefixes based on data type.

**Table 3: Variable name convention**

| Data Type | Prefix | Example |
|---|---|---|
| Byte | byt | bytSpeed |
| Single | sng | sngAngle |
| Integer | int | intCounter |
| Long | lng | lngTraffic_Volume |
| Double | dbl | dbl85Percentile |
| Currency | cur | curUnitPrice |
| String | str | strFirstname |
| Boolean | bln | blnOnTime |
| Object | obj | objSpreadsheet |
| Variant | var | varAddress |

## 3.5 Variable Declaration

Variable declaration is the process of assigning a name and type to a variable. By default, variable declaration is required before a variable can be used in a procedure, otherwise a run-time error will occur. When a variable is declared, an area of memory is reserved for it and assigned a name, the name of the variable. It is then ready to accept and store data of the type specified in the declaration statement.

The syntax for variable declaration in VBA is as follows:

*Dim variablename As datatype*

Examples:

1. A variable that will hold an integer value

*Dim intQuantity As Integer*

2. A variable that will hold an unit price

*Dim curUnitPrice As Currency*

3. A variable that will hold a text string

*Dim intComment As String*

Multiple variables can be declared at once provided they are of the same type.

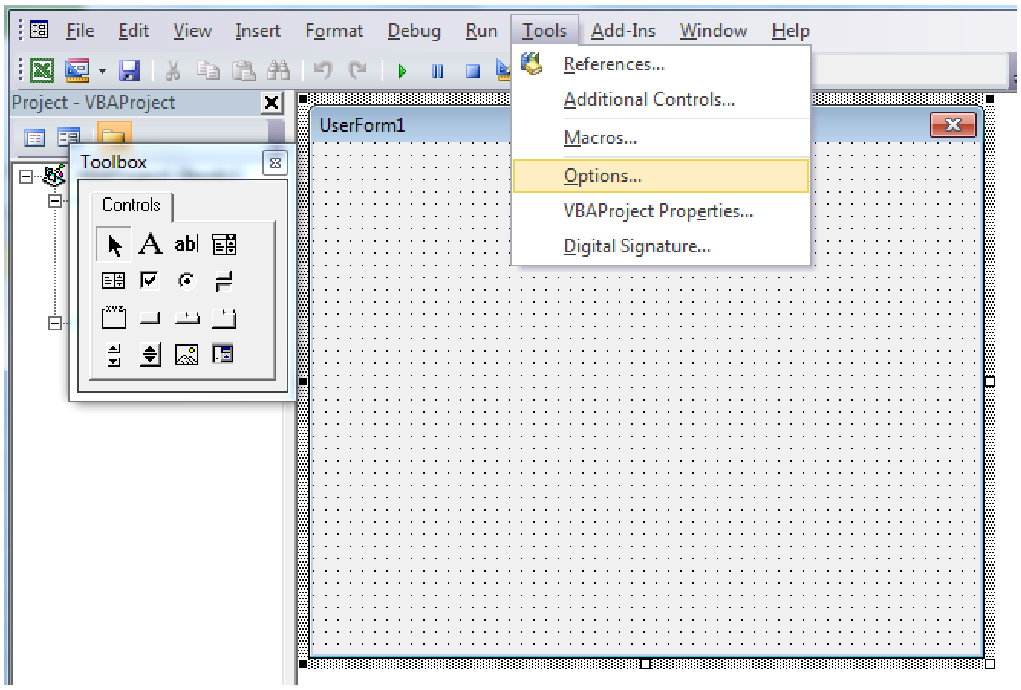
Example: Three string variables

*Dim intComment, intReport, intFeedback  As String*

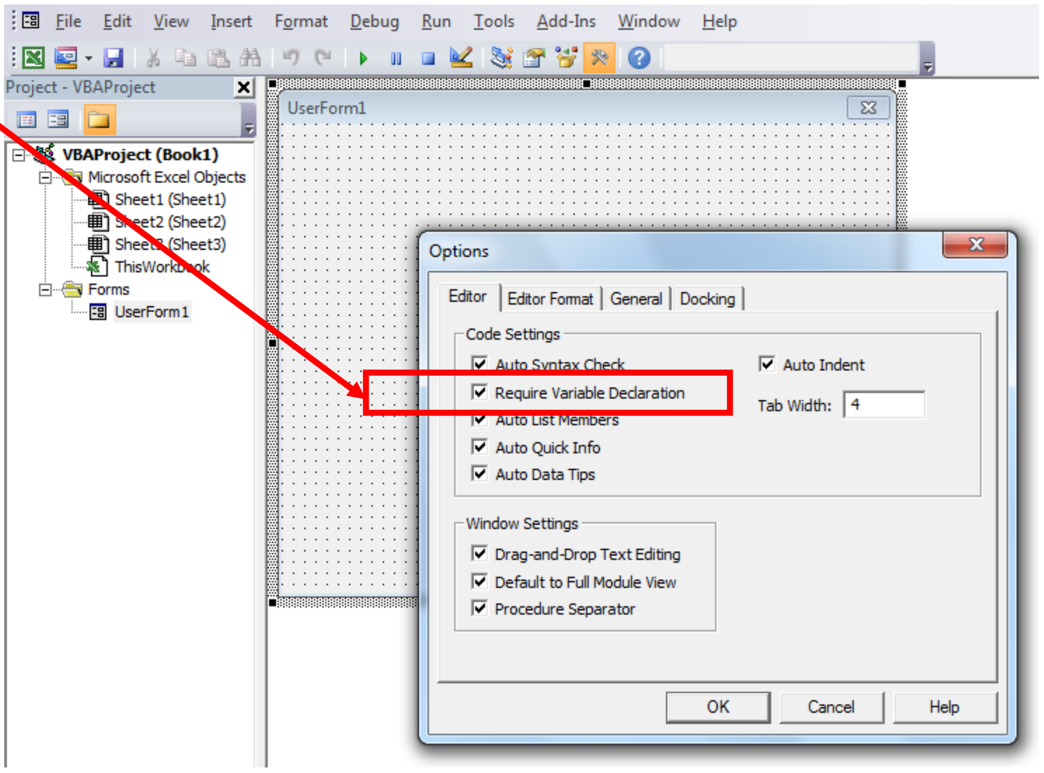
The default data type in VBA is variant. In other words if a data type is not specified it will be a variant type by default.

The variable declaration requirement may be deactivated (or reactivated) as follows:

In the VBA Environment click on **Tools.**
Select **Options.**



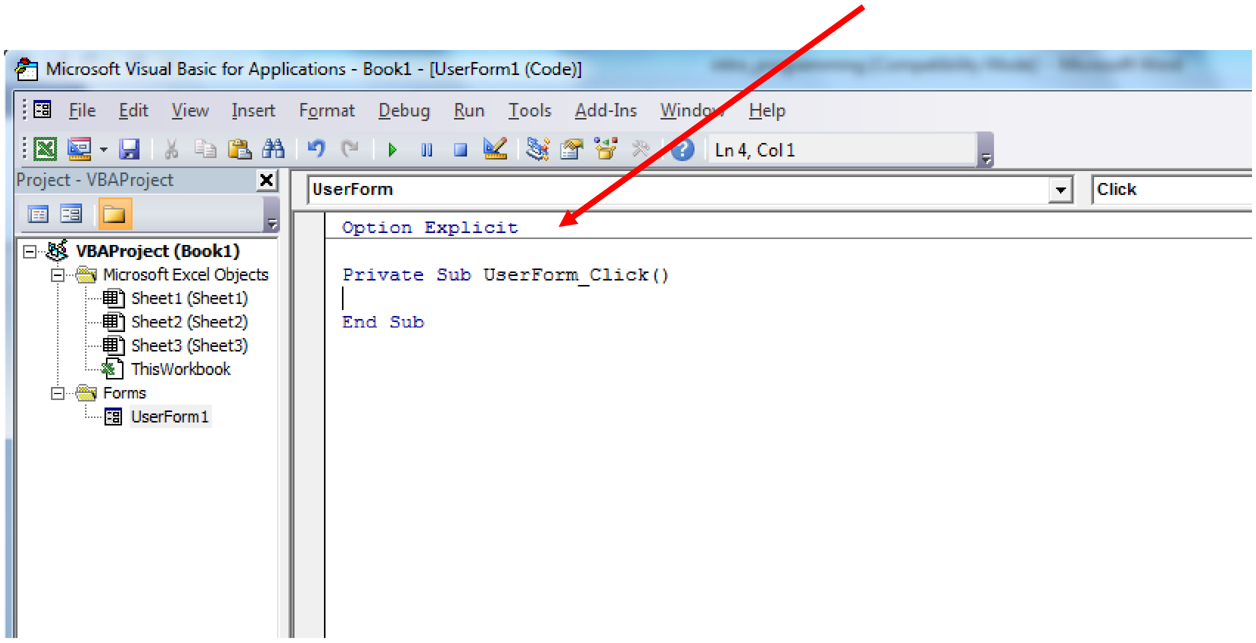
Uncheck (or check) **Require Variable Declaration.**
Click OK.



Code Settings
- [✓] Auto Syntax Check
- [✓] Require Variable Declaration
- [✓] Auto List Members
- [✓] Auto Quick Info
- [✓] Auto Data Tips
- [✓] Auto Indent
- Tab Width: 4

Window Settings
- [✓] Drag-and-Drop Text Editing
- [✓] Default to Full Module View
- [✓] Procedure Separator

Alternately, in the code window typing **Option Explicit** at the top of the window will require that all variables be declared before using them.



```
Option Explicit

Private Sub UserForm_Click()

End Sub
```

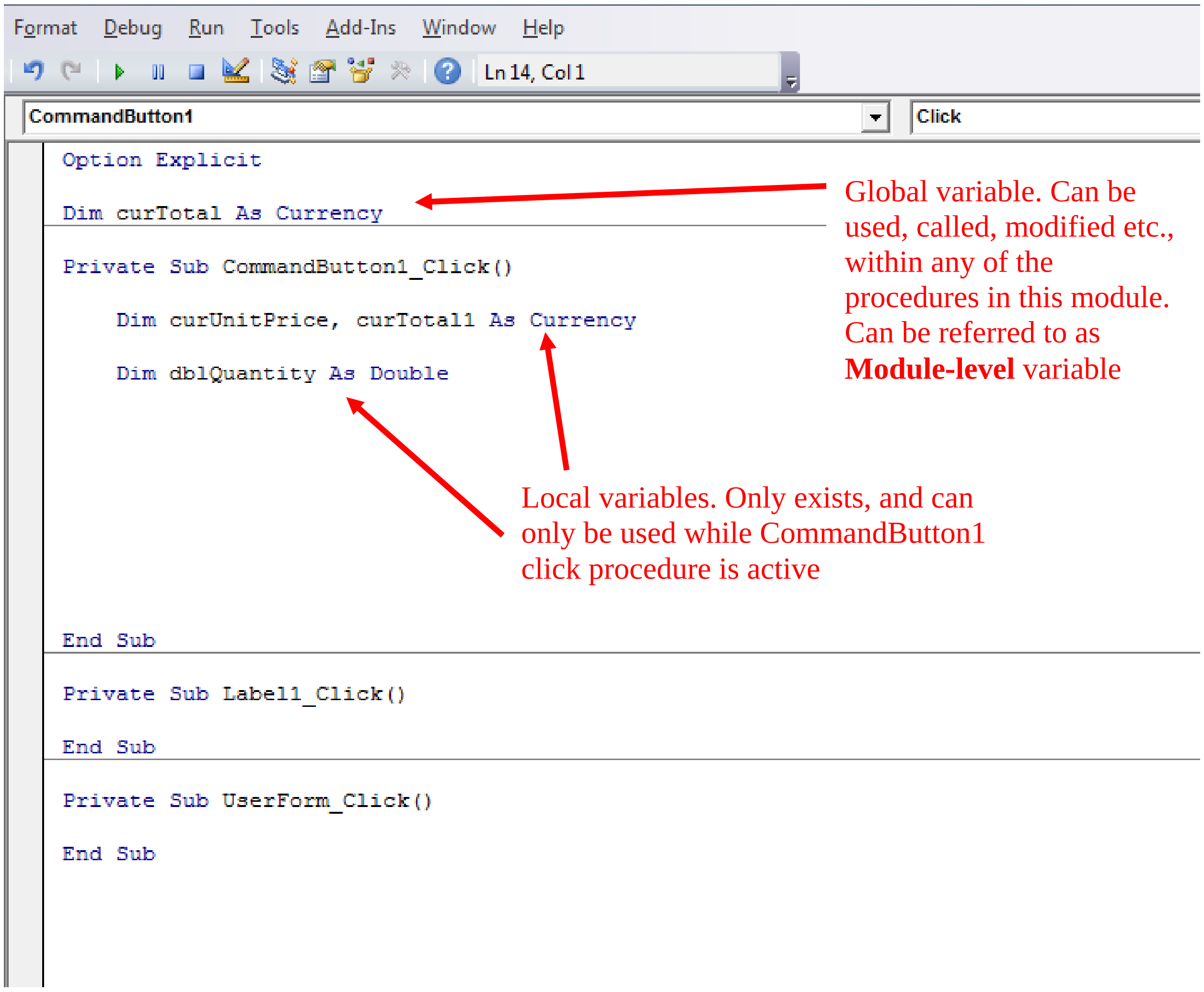
Variable declaration facilitates writing clean, strongly typed code, and simplifies debugging. Variable declaration saves overall programming time and eliminates the source of the most common and unnecessary bugs. In this course the variable declaration requirement will be applied at all times.

## 3.6 Scope of a Variable

Variables may have two kinds of "lifetime", or **scope**.

A variable that is declared within a procedure or **function** is called a **local variable**, and only exists while that procedure or function is executing. Once the procedure completes execution, the variable ceases to exist.

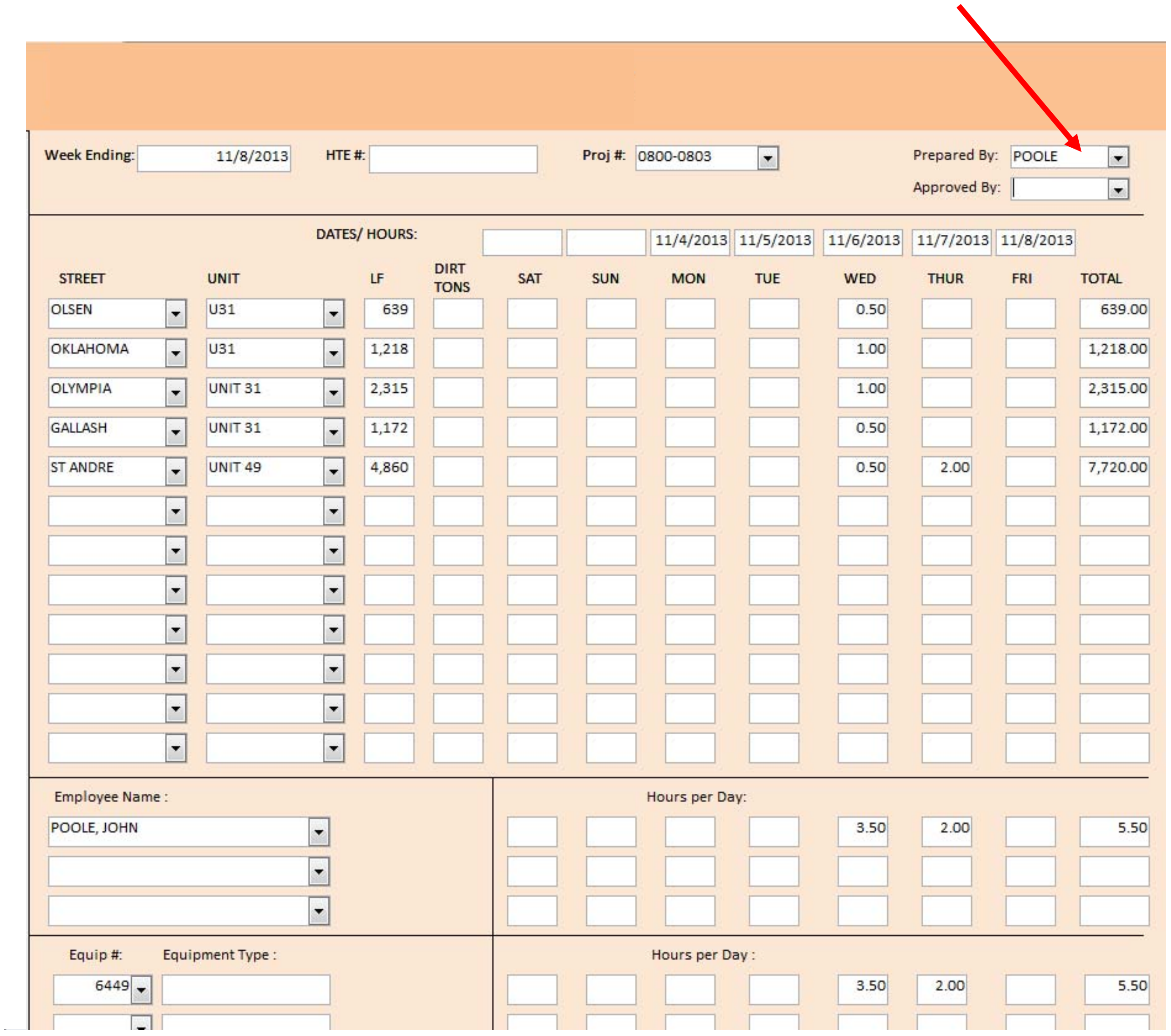
A **global variable** on the other hand exists independently of any specific function or procedure. Any function or procedure in the module may use it or modify it.  A global variable stays “alive” as long as the VBA module (e.g. an open form) it is associated with is active. Global variables are declared outside of any procedure or function, at the top of the code window, below the *Option Explicit* line.


Format Debug Run Tools Add-Ins Window Help
Ln 14, Col 1
CommandButton1
Click
Global variable. Can be used, called, modified etc., within any of the procedures in this module. Can be referred to as **Module-level** variable
Local variables. Only exists, and can only be used while CommandButton1 click procedure is active

```
Option Explicit

Dim curTotal As Currency

Private Sub CommandButton1_Click()

    Dim curUnitPrice, curTotal1 As Currency

    Dim dblQuantity As Double




End Sub

Private Sub Label1_Click()

End Sub

Private Sub UserForm_Click()

End Sub
```

Some applications may consist of multiple modules (forms), and it may be necessary that a variable (global variable) manipulated in one form be available or called in or from another active module, or vice versa.

For example, consider a maintenance activity form filled out by a field crew leader.



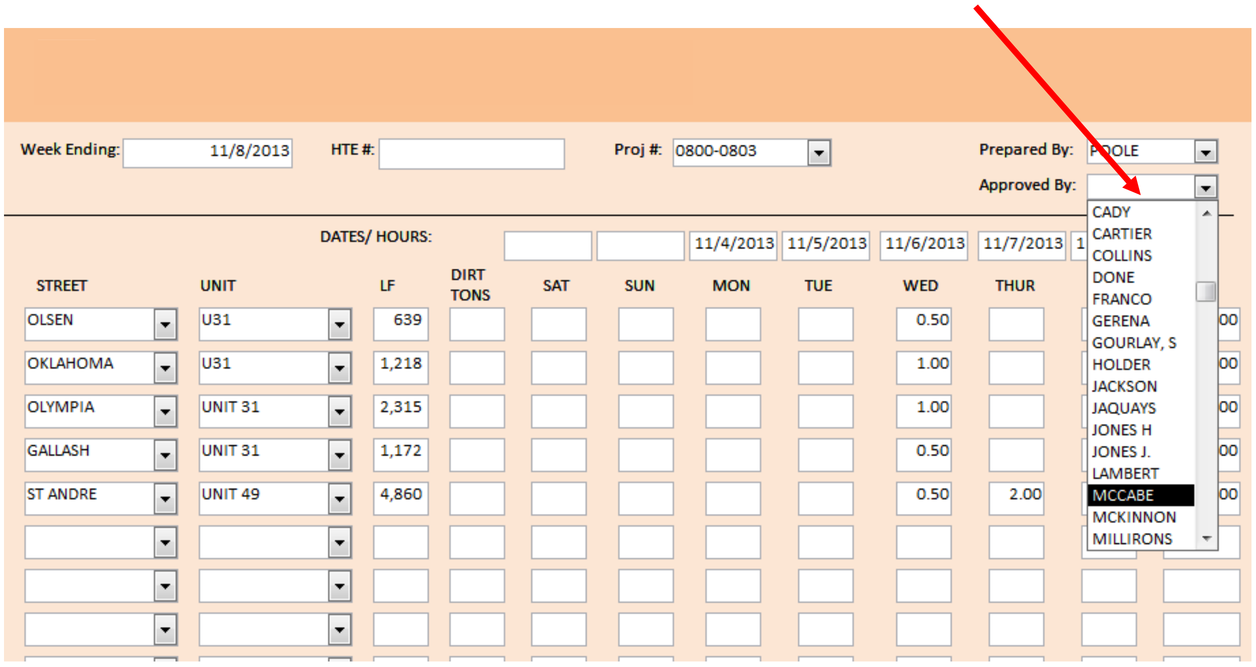
Week Ending: 11/8/2013 HTE #: Proj #: 0800-0803 Prepared By: POOLE
Approved By:

| STREET | UNIT | LF | DIRT TONS | SAT | SUN | MON | TUE | WED | THUR | FRI | TOTAL |
|---|---|---|---|---|---|---|---|---|---|---|---|
| DATES/ HOURS: | | | | | | 11/4/2013 | 11/5/2013 | 11/6/2013 | 11/7/2013 | 11/8/2013 | |
| OLSEN | U31 | 639 | | | | | | 0.50 | | | 639.00 |
| OKLAHOMA | U31 | 1,218 | | | | | | 1.00 | | | 1,218.00 |
| OLYMPIA | UNIT 31 | 2,315 | | | | | | 1.00 | | | 2,315.00 |
| GALLASH | UNIT 31 | 1,172 | | | | | | 0.50 | | | 1,172.00 |
| ST ANDRE | UNIT 49 | 4,860 | | | | | | 0.50 | 2.00 | | 7,720.00 |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| | | | | | | | | | | | |

| Employee Name : | Hours per Day: | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| POOLE, JOHN | | | | | 3.50 | 2.00 | | 5.50 |
| | | | | | | | | |
| | | | | | | | | |

| Equip #: | Equipment Type : | Hours per Day : | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 6449 | | | | | | 3.50 | 2.00 | | 5.50 |
| | | | | | | | | | |

At a later stage, the completed form shall be approved by a supervisor.

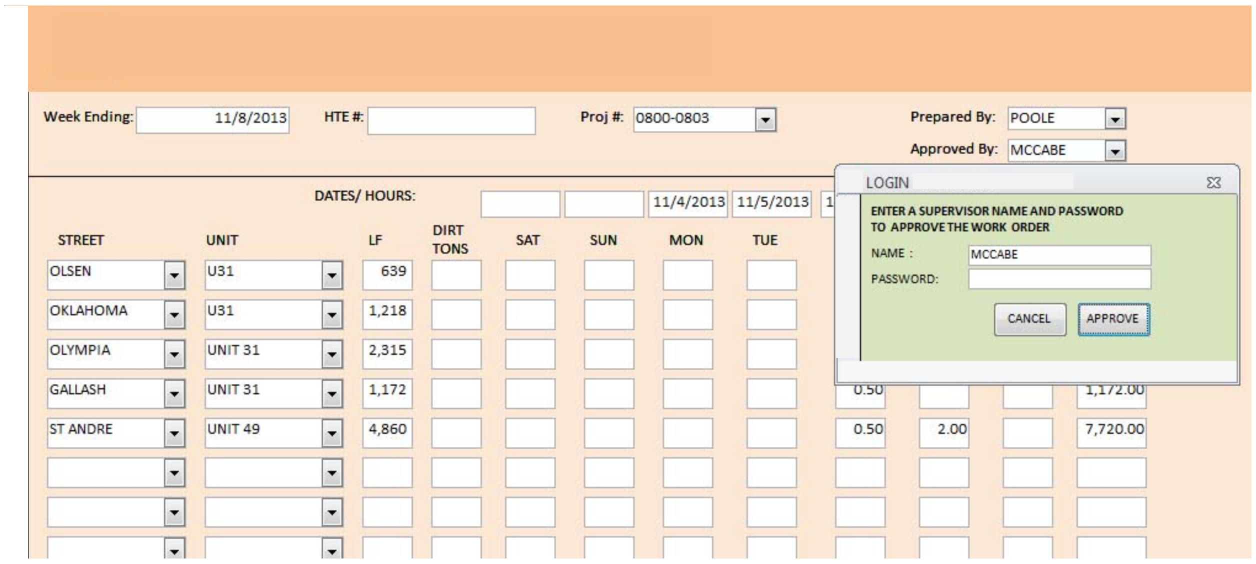


Week Ending: 11/8/2013 HTE #: Proj #: 0800-0803 Prepared By: POOLE
Approved By:

CADY
CARTIER
COLLINS
DONE
FRANCO
GERENA
GOURLAY, S
HOLDER
JACKSON
JAQUAYS
JONES H
JONES J.
LAMBERT
MCCABE
MCKINNON
MILLIRONS

| | | | DATES/ HOURS: | | | 11/4/2013 | 11/5/2013 | 11/6/2013 | 11/7/2013 | |
|---|---|---|---|---|---|---|---|---|---|---|
| STREET | UNIT | LF | DIRT TONS | SAT | SUN | MON | TUE | WED | THUR | |
| OLSEN | U31 | 639 | | | | | | 0.50 | | 00 |
| OKLAHOMA | U31 | 1,218 | | | | | | 1.00 | | 00 |
| OLYMPIA | UNIT 31 | 2,315 | | | | | | 1.00 | | 00 |
| GALLASH | UNIT 31 | 1,172 | | | | | | 0.50 | | 00 |
| ST ANDRE | UNIT 49 | 4,860 | | | | | | 0.50 | 2.00 | 00 |

Upon the supervisor selecting their last name from the supervisor list, a log-in form opens asking the supervisor to enter their name and password.



Week Ending: 11/8/2013 HTE #: Proj #: 0800-0803 Prepared By: POOLE
Approved By: MCCABE

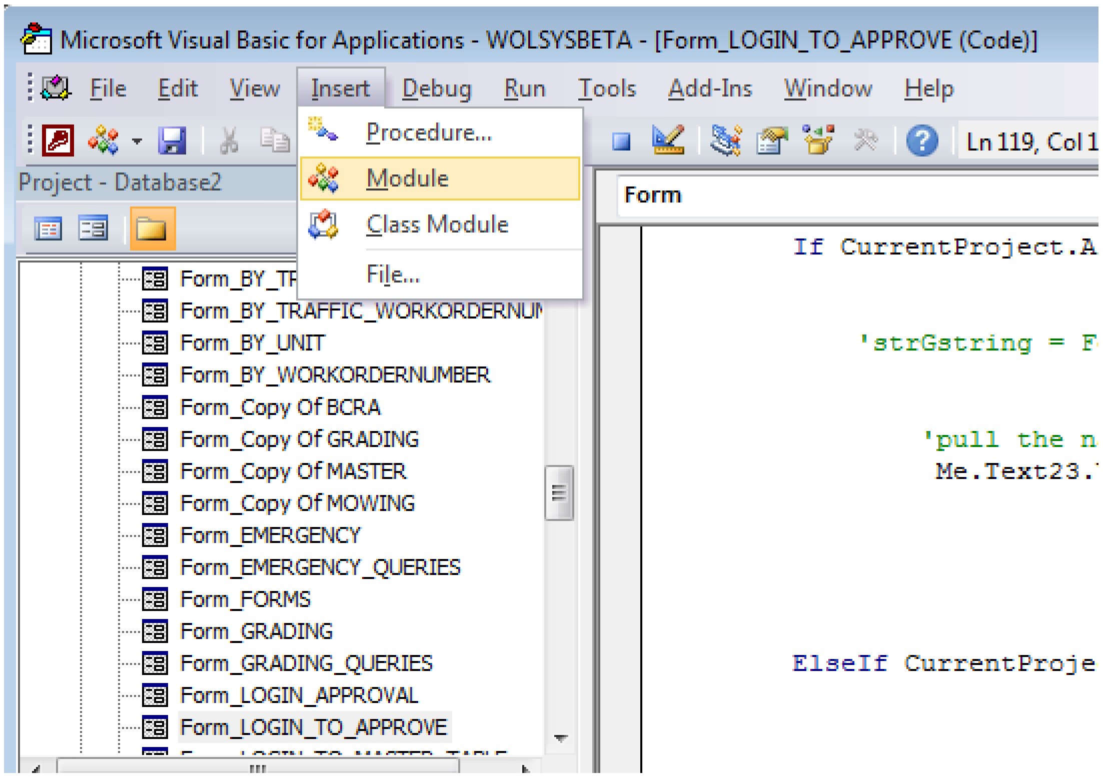
LOGIN
ENTER A SUPERVISOR NAME AND PASSWORD TO APPROVE THE WORK ORDER
NAME : MCCABE
PASSWORD:
CANCEL APPROVE

| | | | DATES/ HOURS: | | | 11/4/2013 | 11/5/2013 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| STREET | UNIT | LF | DIRT TONS | SAT | SUN | MON | TUE | | | | |
| OLSEN | U31 | 639 | | | | | | | | | |
| OKLAHOMA | U31 | 1,218 | | | | | | | | | |
| OLYMPIA | UNIT 31 | 2,315 | | | | | | | | | |
| GALLASH | UNIT 31 | 1,172 | | | | | | 0.50 | | | 1,172.00 |
| ST ANDRE | UNIT 49 | 4,860 | | | | | | 0.50 | 2.00 | | 7,720.00 |

Upon correctly entering the password, the approval will be confirmed, otherwise it will be denied.

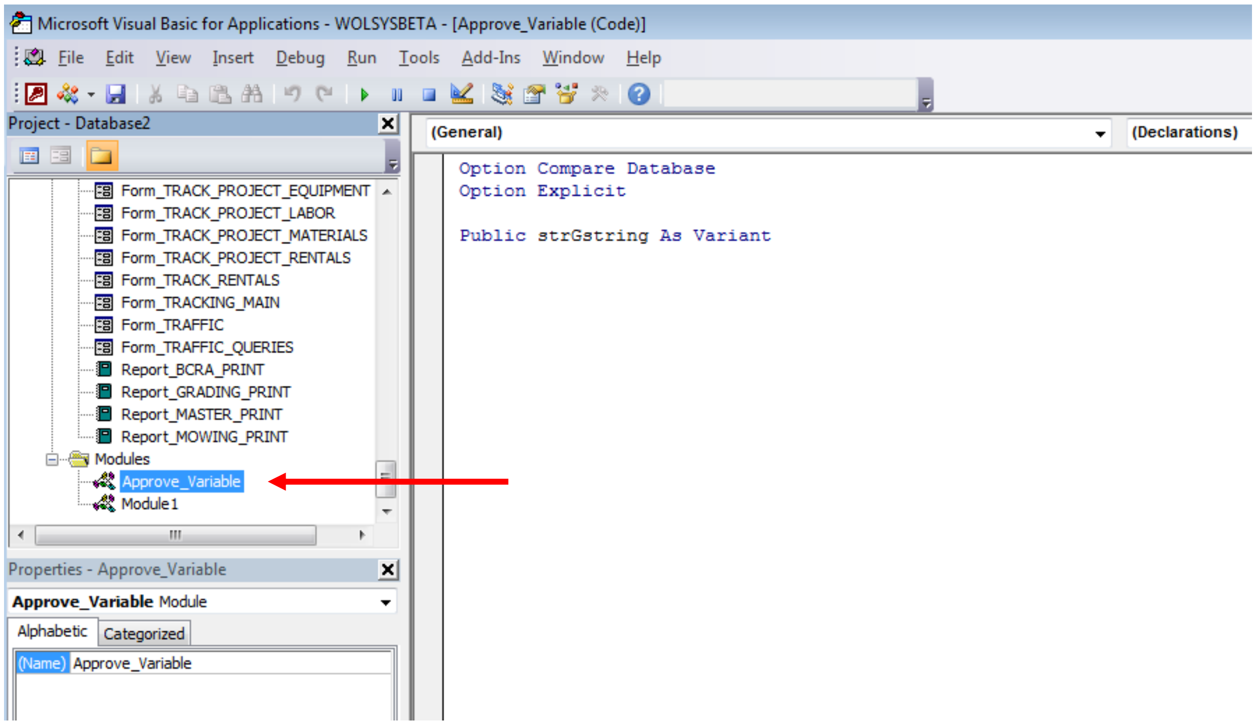
For this to work, a global variable is needed for the two forms to be able to exchange information. In this case a global variable named *strGstring*, a string type, will be used.

The global variable must be declared within a standard module.
In the VBA window, click on Insert, and select Module



```
        If CurrentProject.A

            'strGstring = F

              'pull the n
               Me.Text23.

        ElseIf CurrentProje
```

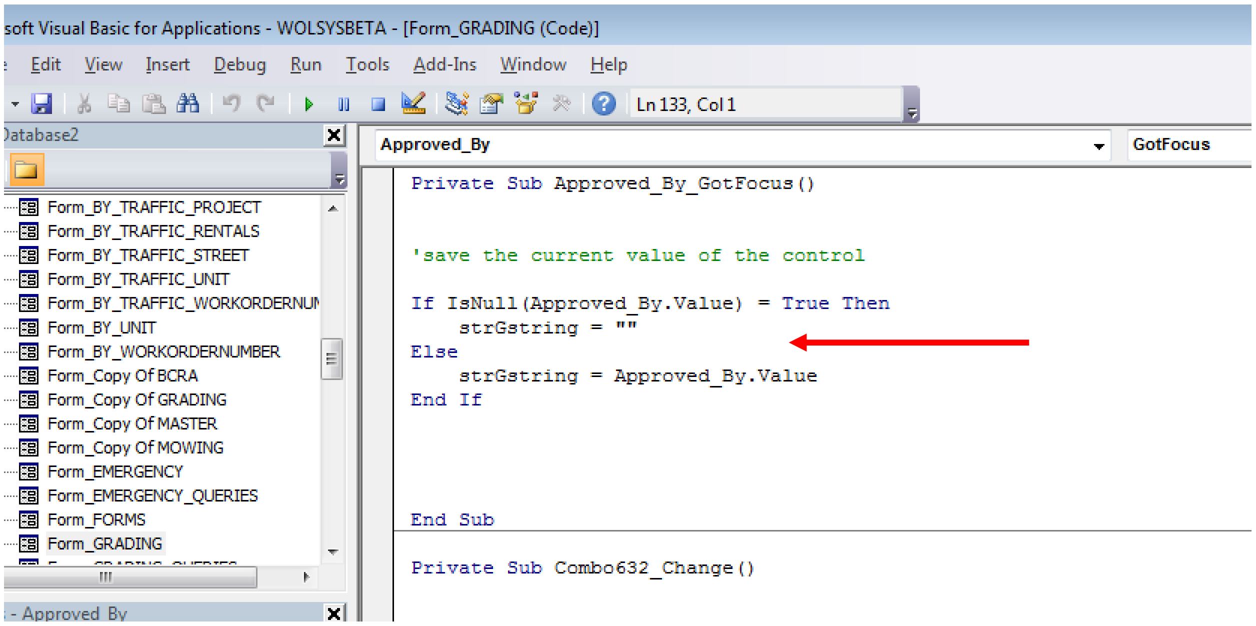
A new standard module is created.
Rename the module as needed.
Click on View Code to open the code window for the new standard module.
Under the *Option Explicit* line, declare the global variable using the keyword *Global* or *Public*.



```
Option Compare Database
Option Explicit

Public strGstring As Variant
```

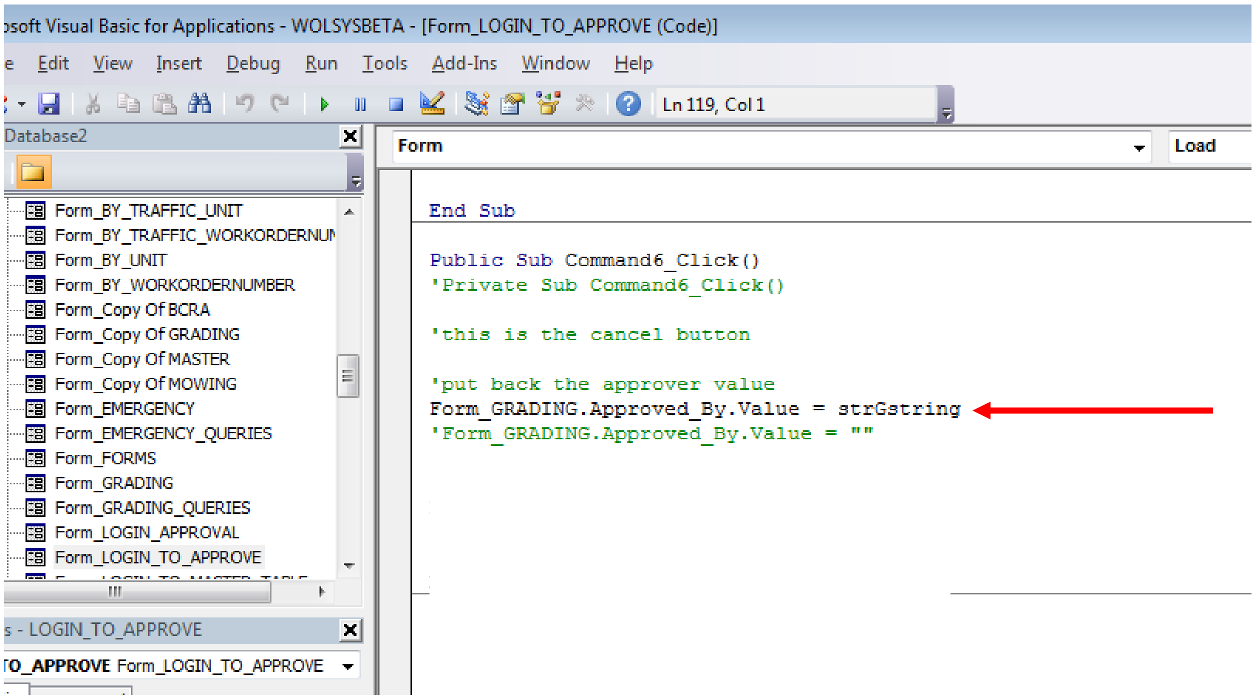
The variable *strGstring* is now available for manipulation from and by any module (form) in the workbook.

In this example, the value will be assigned with the *Approved_By* control entry on the main form.



```
Private Sub Approved_By_GotFocus()


'save the current value of the control

If IsNull(Approved_By.Value) = True Then
    strGstring = ""
Else
    strGstring = Approved_By.Value
End If



End Sub

Private Sub Combo632_Change()
```

The variable is then called by the log-in form and will be used for confirmation in the log-in and password process.



```
End Sub

Public Sub Command6_Click()
'Private Sub Command6_Click()

'this is the cancel button

'put back the approver value
Form_GRADING.Approved_By.Value = strGstring
'Form_GRADING.Approved_By.Value = ""
```

### 3.7 Order of Operations

The arithmetic operations that can be performed in VBA include addition, subtraction, multiplication, division, and exponentiation. The order of operations determines the order in which arithmetic operations will be performed in a mathematical expression. The **order of precedence** for arithmetic operations is as follows:

| | |
|---|---|
| 1st. | Exponentiation |
| 2nd. | Multiplication and division |
| 3rd. | Addition and subtraction |

Operations of the same level are performed from left to right.

Example: The expression $2+3*6^2$

Under the order of precedence the exponentiation will be conducted first.
Next, the result of the exponentiation will be multiplied by the adjacent value.
Third, the addition will be performed.
The result therefore will be 110.

If the above result was not the original intent, but rather to conduct the addition first, then **parenthesis** will have to be inserted appropriately to control the calculation. In the VBA code, type:

```
(2+3)*6^2
```

This yields the result 180.

It must be noted that VBA does not support implied algebraic operations so for example the expression 3(X-Y) will not be syntactically valid in VBA. It will have to be typed as 3*(X-Y) where X and Y are variables that have been declared earlier in the code.

### 3.8 Second VBA Project

Problem Statement:
Create a calculator to assist a construction engineer in calculating material costs based on the unit price of the material and the quantity of material to be ordered. The calculator will have three line items with a subtotal for each item. The calculator will have a click button to calculate

the subtotal and another click button to calculate the grand total. Add a button that will clear all entries if the user would like to start afresh. Provide a button that will close the application when clicked on.

Solution:

A sketch of the application is as follows

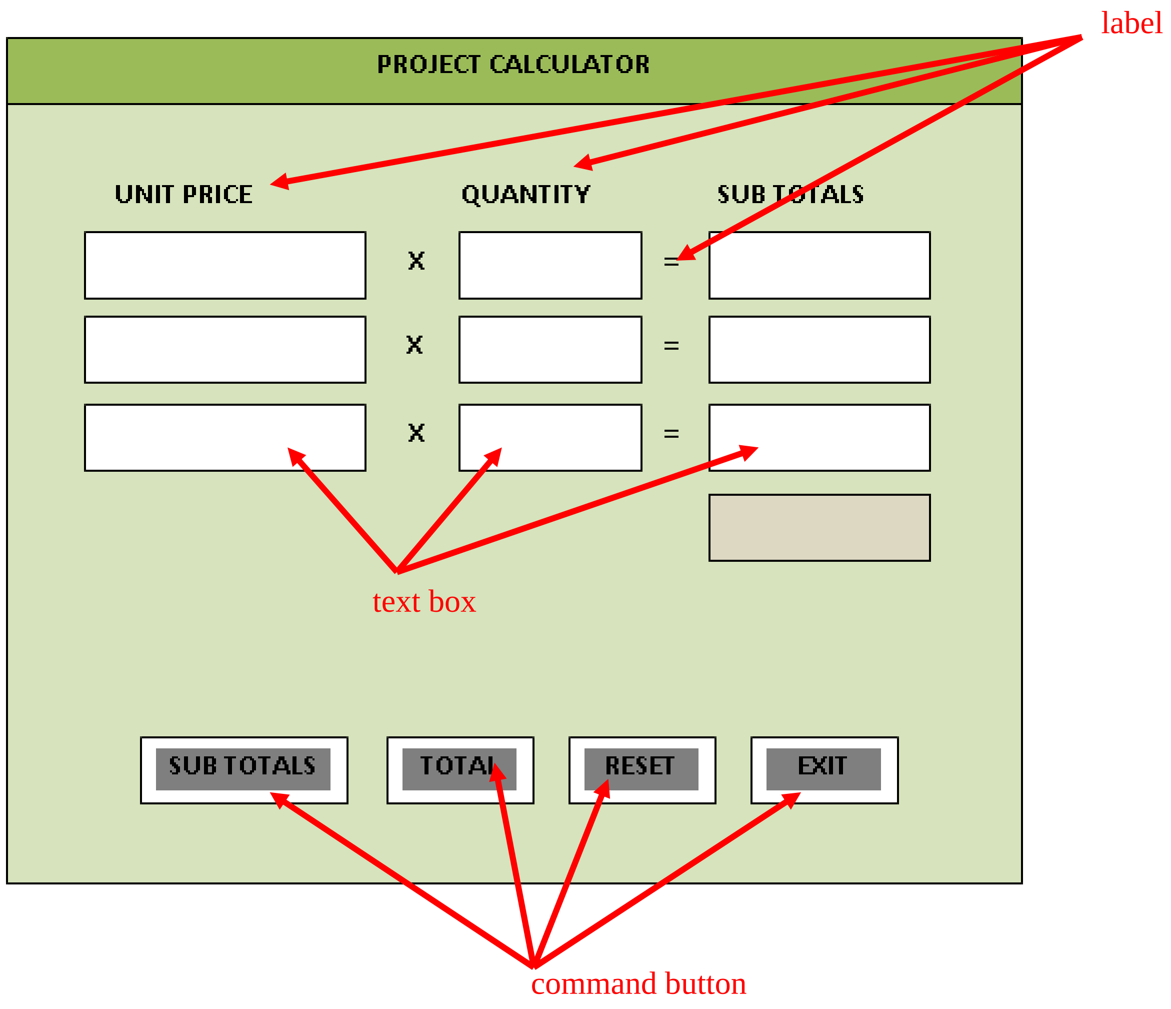


**Figure 4: Plan and sketch for Second VBA Project**

Algorithm for calculation of Subtotals

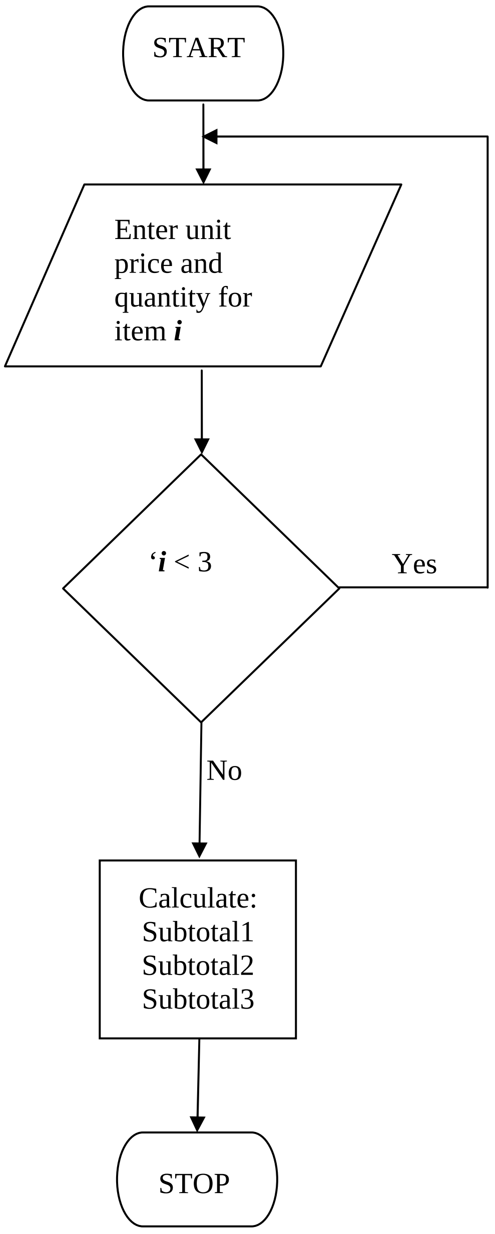
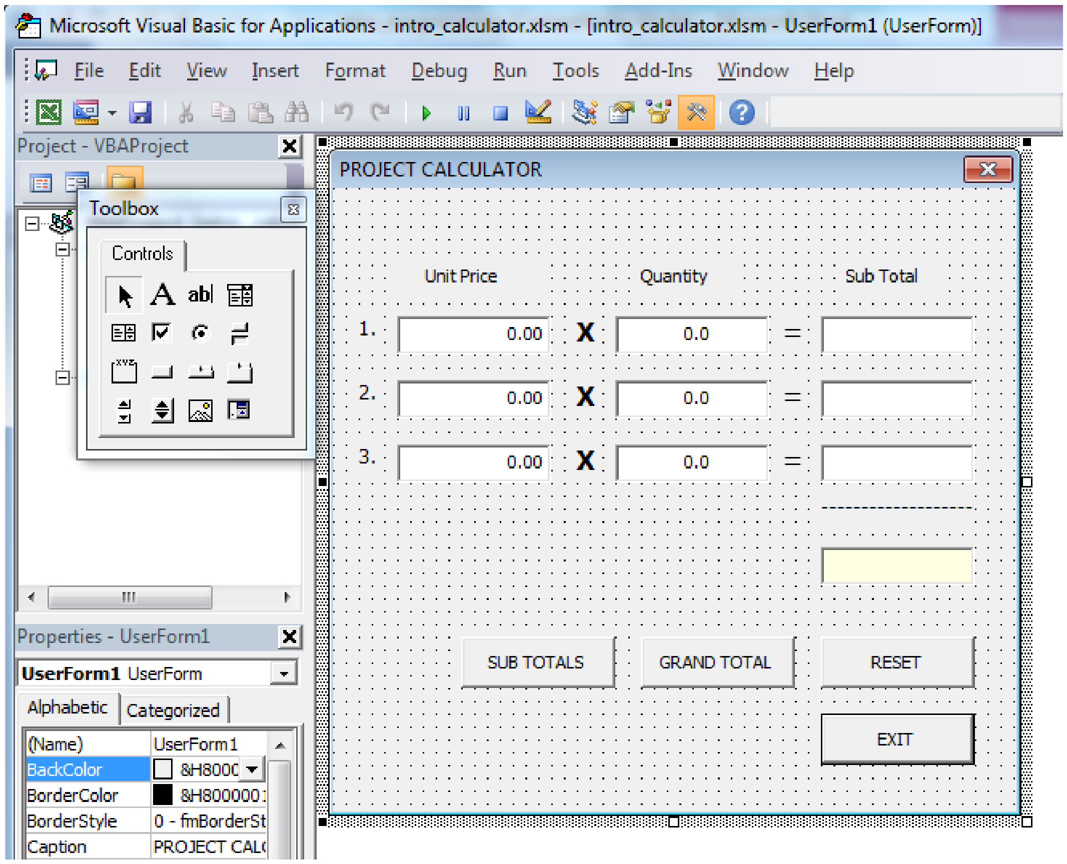

START
Enter unit price and quantity for item *i*
'*i* < 3
Yes
No
Calculate: Subtotal1 Subtotal2 Subtotal3
STOP

**Figure 5: Algorithm for Second VBA Project**

Open a VBA session.
Click on **Insert**, select **UserForm.**
Click on the **UserForm.**
In the **Properties Window**, change **Caption** property of the UserForm to “Project Calculator”.
Click on the **UserForm.**
Drag and drop labels, text boxes, and command buttons and arrange them as in the sketch.
Select a command button.
In the **Properties Window**, change the **Caption** property to match the sketch.
Repeat for all command buttons.
Change the **Caption** property of all labels to match the sketch.
Click on the Total text box and change the **BackColor** property to a color of your choice.
Select all Unit Price text boxes. Change **Value** property to “0.00”.
Select all Quantity text boxes. Change **Value** property to “0.0”.
Save your file as an Excel Macros-Enabled Workbook

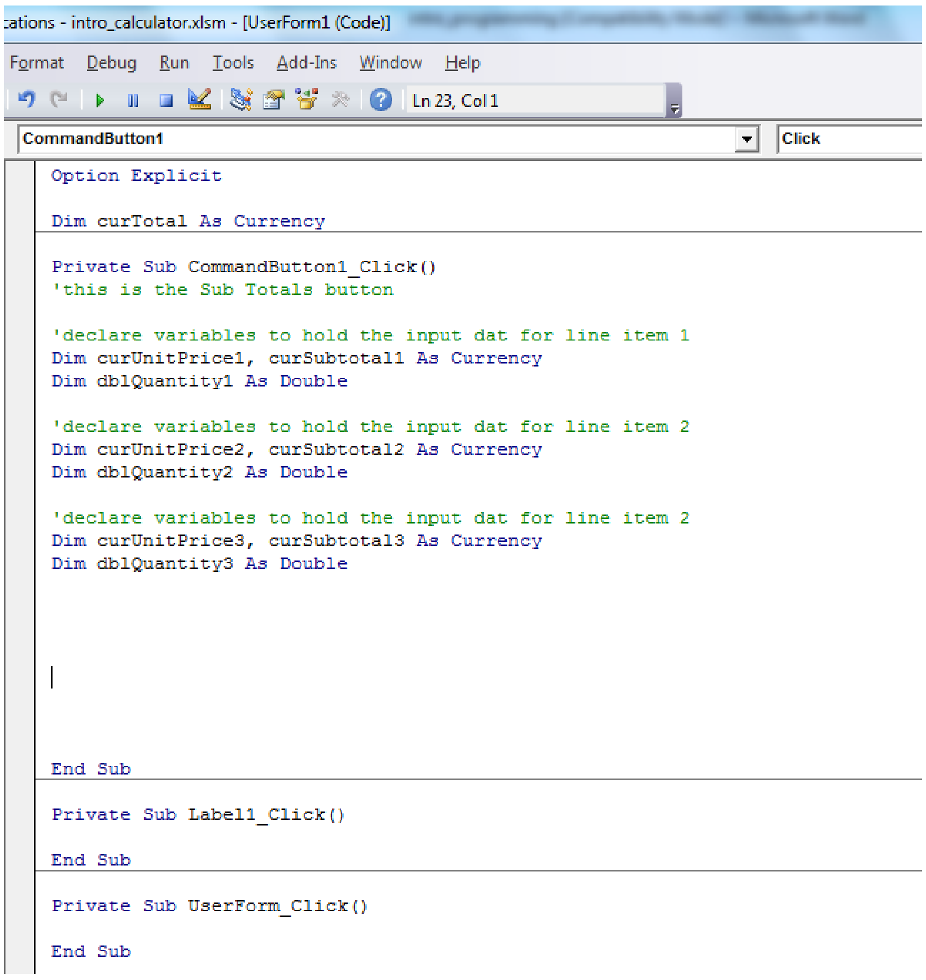

Microsoft Visual Basic for Applications - intro_calculator.xlsm - [intro_calculator.xlsm - UserForm1 (UserForm)]

| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H8000 |
| BorderColor | &H800000 |
| BorderStyle | 0 - fmBorderSt |
| Caption | PROJECT CAL |

Now code shall be written for each command button.

Sub Totals button
On clicking this button, the values entered by the user need to be stored as variables and used to calculate the subtotal for each line item.

Double click on the **Sub Totals** button (*CommandButton1*) to open the code window.
In the click procedure for *CommandButton1* (*Private Sub CommandButton1_Click( )* ), declare variables (using appropriate names of your choice and following variable name rules and convention) to hold Unit Price, Quantity and Subtotal for each line item.
Declare a global variable that will hold the sum of the subtotals.
Save your work.


cations - intro_calculator.xlsm - [UserForm1 (Code)]
Format Debug Run Tools Add-Ins Window Help
Ln 23, Col 1
CommandButton1
Click

```
Option Explicit

Dim curTotal As Currency

Private Sub CommandButton1_Click()
'this is the Sub Totals button

'declare variables to hold the input dat for line item 1
Dim curUnitPrice1, curSubtotal1 As Currency
Dim dblQuantity1 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice2, curSubtotal2 As Currency
Dim dblQuantity2 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice3, curSubtotal3 As Currency
Dim dblQuantity3 As Double




End Sub

Private Sub Label1_Click()

End Sub

Private Sub UserForm_Click()

End Sub
```

Now the variables need to be assigned values. In other words e.g., "…where is the value for *curUnitPrice1* coming from?" It is coming from the *Textbox1* value. So the *Textbox1* value needs to be assigned to the *curUnitPrice1 variable*.

Assignment is done using the assignment operator, the equals ( = ) sign. In this case:

*curUnitPrice1 = UserForm1.TextBox1.value*

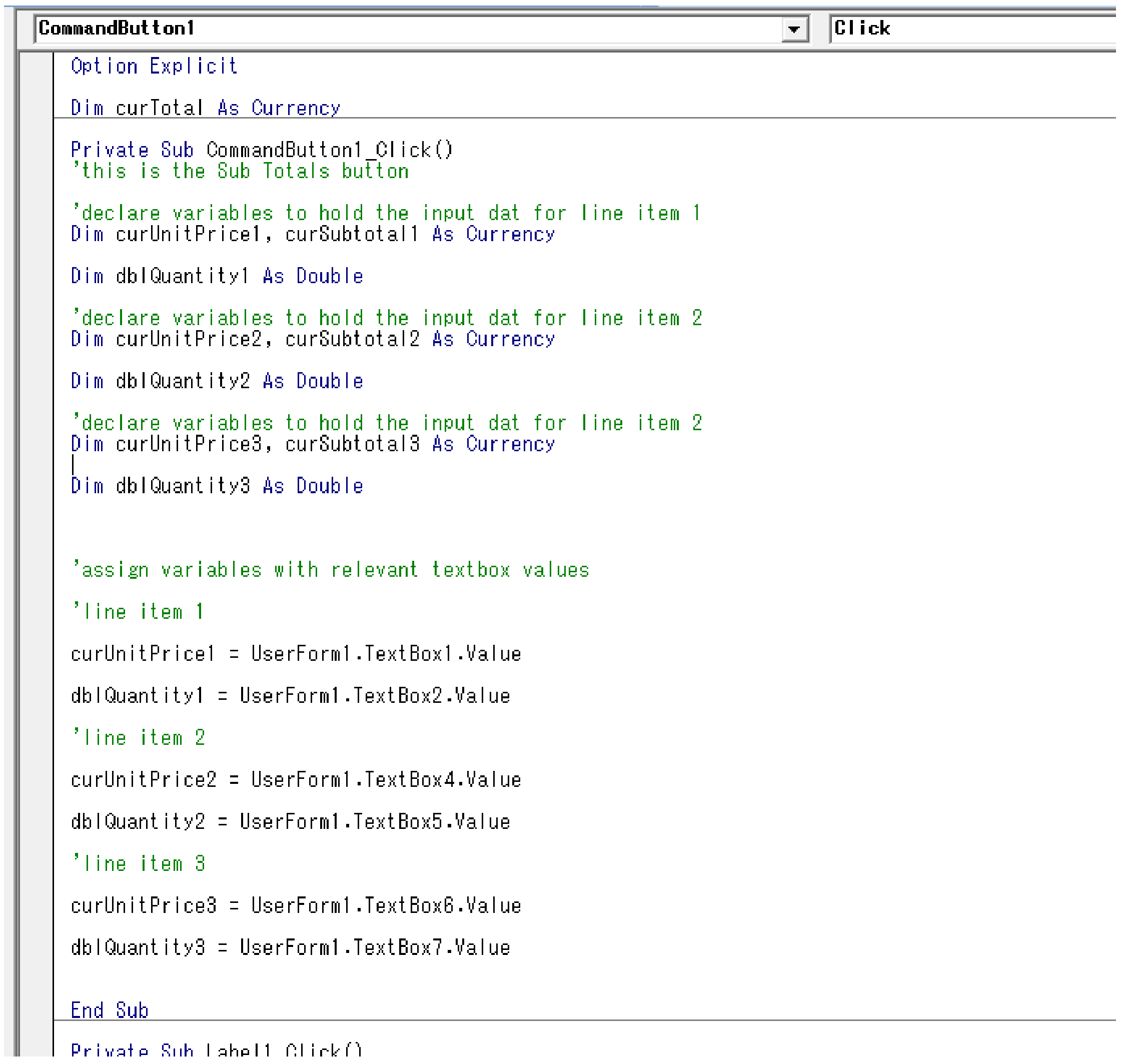
Note that the origin will always be on the right hand side and the destination on the left hand side, and never vice versa. In other words the right hand side information is being sent to the left hand side for storage.

Consider the above code was reversed to

*UserForm1.TextBox1.value = curUnitPrice1*

It would mean the value stored in the variable *curUnitPrice* is being sent to the *TextBox1*, and since *Textbox1* is a control on the form, the value of the variable *curUnitPrice* would be displayed on the form in *Textbox1*. This is a diametrically opposite result from the original intent. However it is important to note that there will be cases where such a result is desired. In this problem for example, after calculating the grand total using the relevant variable(s), it shall be displayed on the form. So the "reverse" assignment statement will be the correct one in that case.

Returning to the variable assignment step, assign the values for all variables that will be used in the calculation of the subtotals when the **Sub Totals** button is clicked.



```
Option Explicit

Dim curTotal As Currency

Private Sub CommandButton1_Click()
'this is the Sub Totals button

'declare variables to hold the input dat for line item 1
Dim curUnitPrice1, curSubtotal1 As Currency

Dim dblQuantity1 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice2, curSubtotal2 As Currency

Dim dblQuantity2 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice3, curSubtotal3 As Currency

Dim dblQuantity3 As Double


'assign variables with relevant textbox values

'line item 1

curUnitPrice1 = UserForm1.TextBox1.Value

dblQuantity1 = UserForm1.TextBox2.Value

'line item 2

curUnitPrice2 = UserForm1.TextBox4.Value

dblQuantity2 = UserForm1.TextBox5.Value

'line item 3

curUnitPrice3 = UserForm1.TextBox6.Value

dblQuantity3 = UserForm1.TextBox7.Value


End Sub

Private Sub Label1_Click()
```

The calculations for the subtotal of each line item involves multiplying the corresponding variables holding the Unit Price and the Quantity for that line item.

```
curSubTotal1 = curUnitPrice1 * dblQuantity1
```

The result shall be displayed on the form in the appropriate textbox.

```
UserForm1.TextBox3.value = curSubTotal1
```

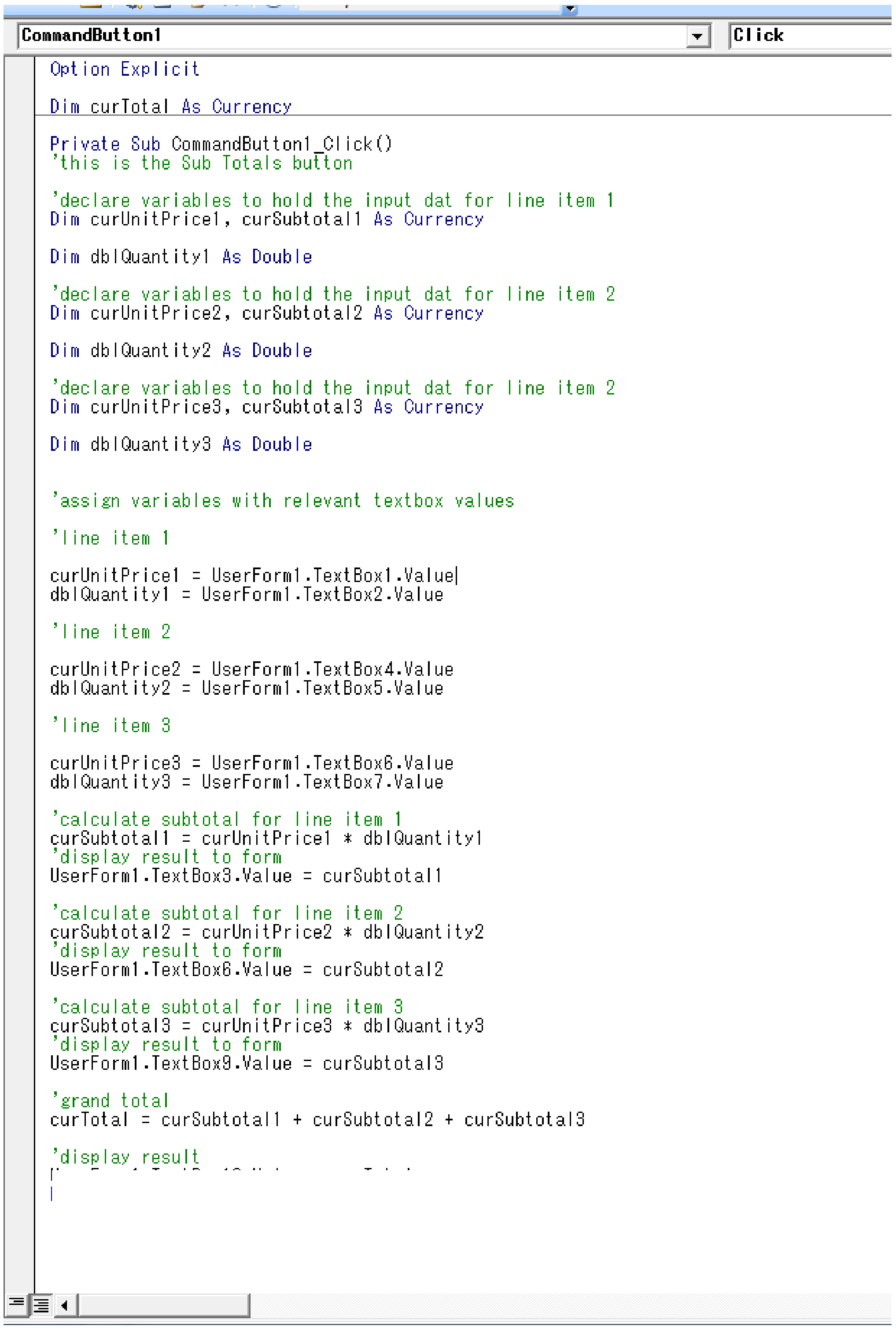
Repeat the process for the other line items.

The grand total is the sum of the sub totals.

```
curTotal = curSubTotal1+ curSubTotal2+ curSubTotal3
```

The grand total button will be used to display the grand total on the form. This is therefore the end of the Subtotals procedure (clicking on *CommandButton1*). As with all procedures it shall run from top to bottom. Once it ends, the local variables will cease to exist. The *curTotal* variable however will continue to “live” as long as the form is active. This is because it is a global variable (module-level) and continues to exist with its current value regardless of which procedure is executing or not, even though its value came from one such procedure which has run to termination.

Click Debug.
Select Compile VBA Project.
Address any errors that are highlighted.
If none, save the workbook as an Excel Macros-Enabled Workbook.



```
Option Explicit

Dim curTotal As Currency

Private Sub CommandButton1_Click()
'this is the Sub Totals button

'declare variables to hold the input dat for line item 1
Dim curUnitPrice1, curSubtotal1 As Currency

Dim dblQuantity1 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice2, curSubtotal2 As Currency

Dim dblQuantity2 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice3, curSubtotal3 As Currency

Dim dblQuantity3 As Double


'assign variables with relevant textbox values

'line item 1

curUnitPrice1 = UserForm1.TextBox1.Value
dblQuantity1 = UserForm1.TextBox2.Value

'line item 2

curUnitPrice2 = UserForm1.TextBox4.Value
dblQuantity2 = UserForm1.TextBox5.Value

'line item 3

curUnitPrice3 = UserForm1.TextBox6.Value
dblQuantity3 = UserForm1.TextBox7.Value

'calculate subtotal for line item 1
curSubtotal1 = curUnitPrice1 * dblQuantity1
'display result to form
UserForm1.TextBox3.Value = curSubtotal1

'calculate subtotal for line item 2
curSubtotal2 = curUnitPrice2 * dblQuantity2
'display result to form
UserForm1.TextBox6.Value = curSubtotal2

'calculate subtotal for line item 3
curSubtotal3 = curUnitPrice3 * dblQuantity3
'display result to form
UserForm1.TextBox9.Value = curSubtotal3

'grand total
curTotal = curSubtotal1 + curSubtotal2 + curSubtotal3

'display result
```

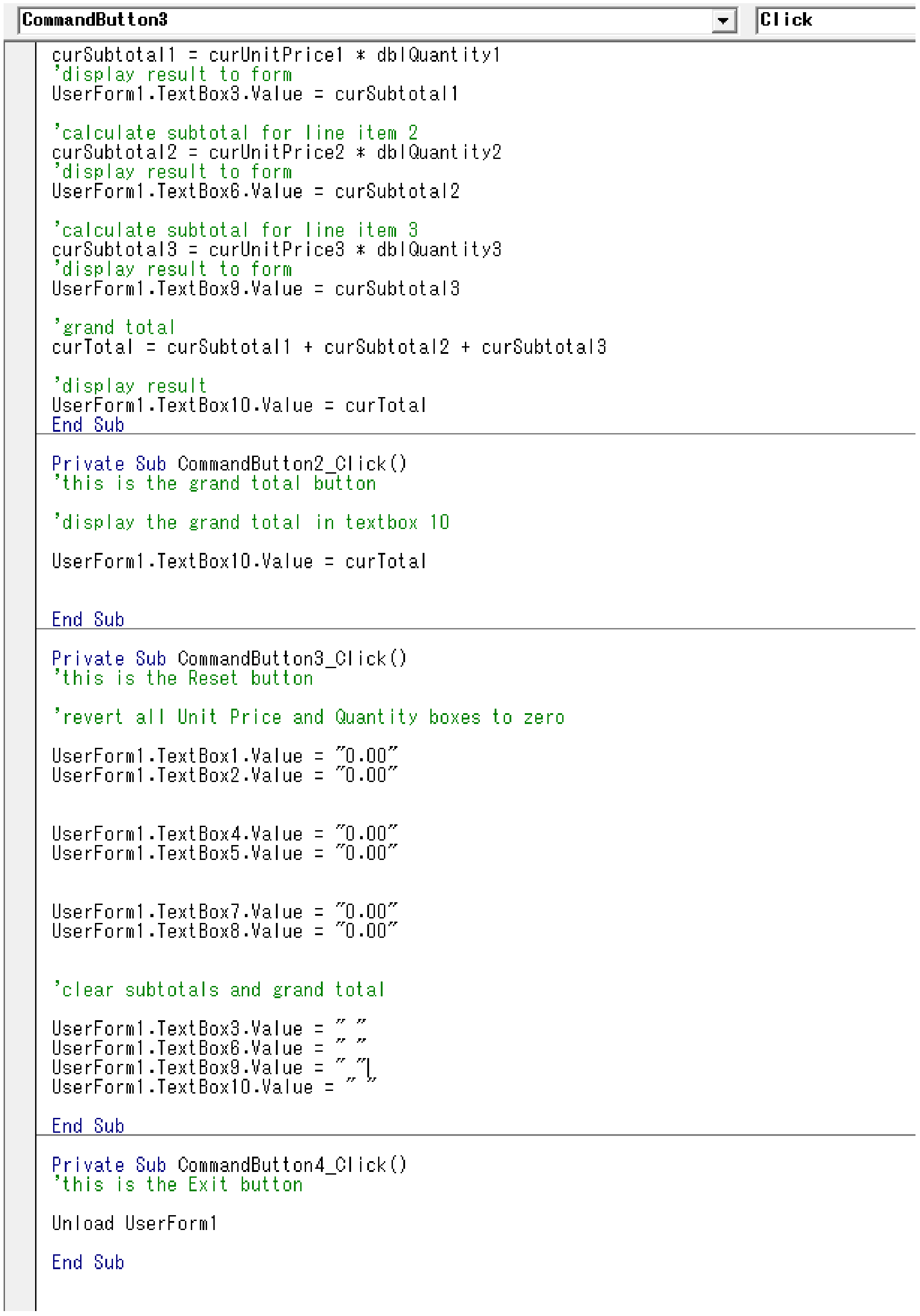
Total button
The grand total has already been calculated in the Subtotals procedure using a global variabe. Since the form is still active, that global variable still exists and holds the value we are looking for. Assign the *curTotal* variable to its display text box.
Double click on the **Grand Total** button to go *Commanbutton2_click ( )* procedure.

Reset button
The **Reset** button will revert the input text boxes to zeros. It will then clear contents in the subtotals and grand total text boxes.

Exit button
The **Exit** button closes the application.

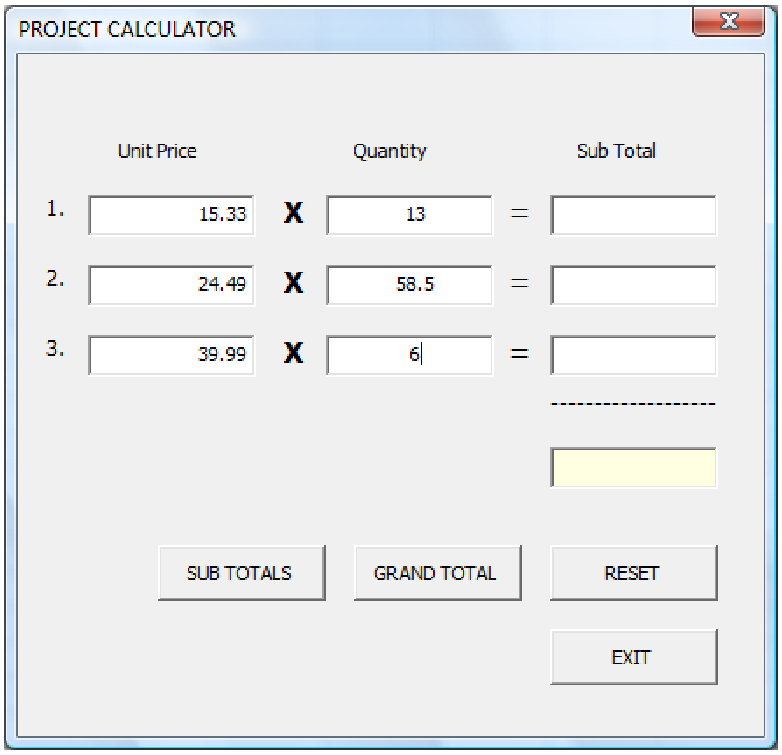
Click Debug
Select Compile VBA Project

If errors are highlighted, address them. If not, save your work.
Test the application.



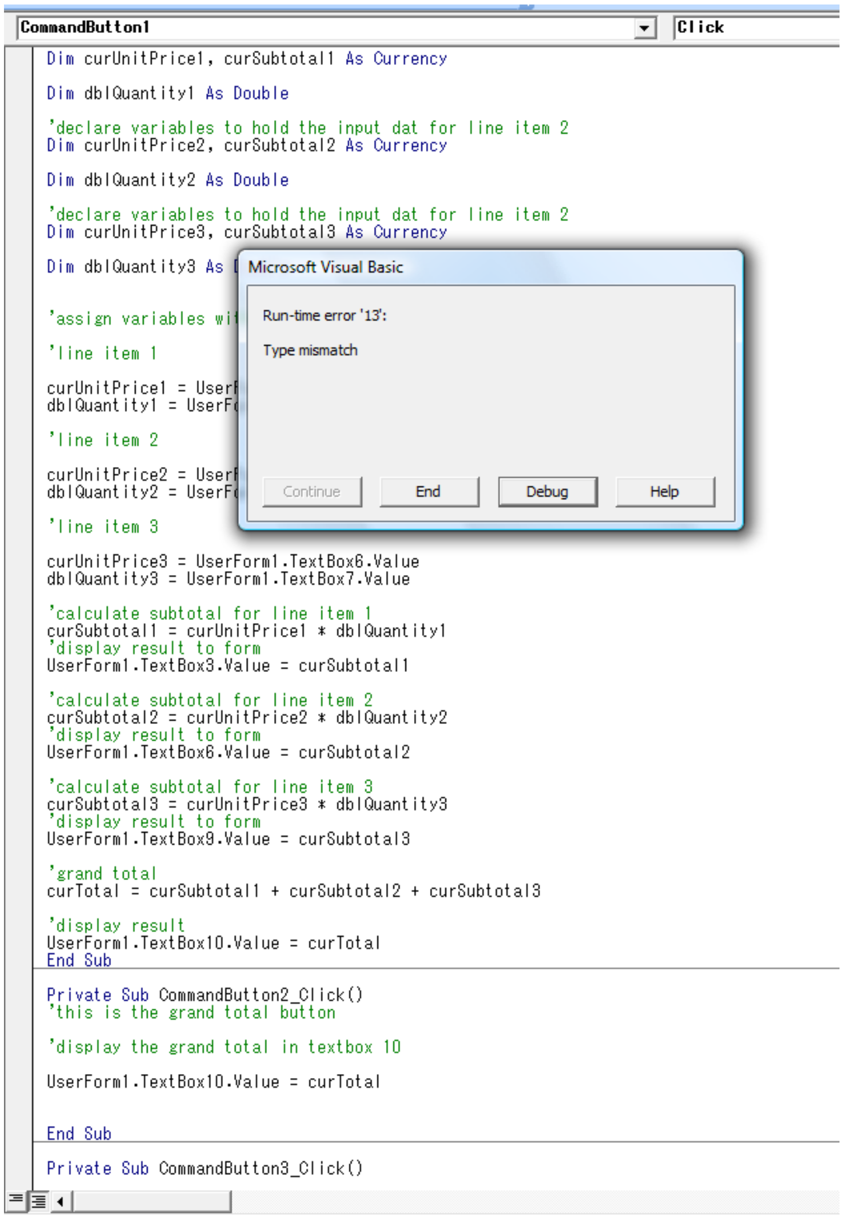
```
curSubtotal1 = curUnitPrice1 * dblQuantity1
'display result to form
UserForm1.TextBox3.Value = curSubtotal1

'calculate subtotal for line item 2
curSubtotal2 = curUnitPrice2 * dblQuantity2
'display result to form
UserForm1.TextBox6.Value = curSubtotal2

'calculate subtotal for line item 3
curSubtotal3 = curUnitPrice3 * dblQuantity3
'display result to form
UserForm1.TextBox9.Value = curSubtotal3

'grand total
curTotal = curSubtotal1 + curSubtotal2 + curSubtotal3

'display result
UserForm1.TextBox10.Value = curTotal
End Sub
--------------------------------------------------
Private Sub CommandButton2_Click()
'this is the grand total button

'display the grand total in textbox 10

UserForm1.TextBox10.Value = curTotal


End Sub
--------------------------------------------------
Private Sub CommandButton3_Click()
'this is the Reset button

'revert all Unit Price and Quantity boxes to zero

UserForm1.TextBox1.Value = "0.00"
UserForm1.TextBox2.Value = "0.00"


UserForm1.TextBox4.Value = "0.00"
UserForm1.TextBox5.Value = "0.00"


UserForm1.TextBox7.Value = "0.00"
UserForm1.TextBox8.Value = "0.00"


'clear subtotals and grand total

UserForm1.TextBox3.Value = " "
UserForm1.TextBox6.Value = " "
UserForm1.TextBox9.Value = " "
UserForm1.TextBox10.Value = " "

End Sub
--------------------------------------------------
Private Sub CommandButton4_Click()
'this is the Exit button

Unload UserForm1

End Sub
```

Fill in some data
Click on the **Sub Totals** button



| | Unit Price | | Quantity | | Sub Total |
|---|---|---|---|---|---|
| 1. | 15.33 | X | 13 | = | |
| 2. | 24.49 | X | 58.5 | = | |
| 3. | 39.99 | X | 6 | = | |

A **Run-time error** occurs. It says "Type mismatch". This means a variable is being asked to hold a data type different from the type in its declaration statement.



CommandButton1 | Click

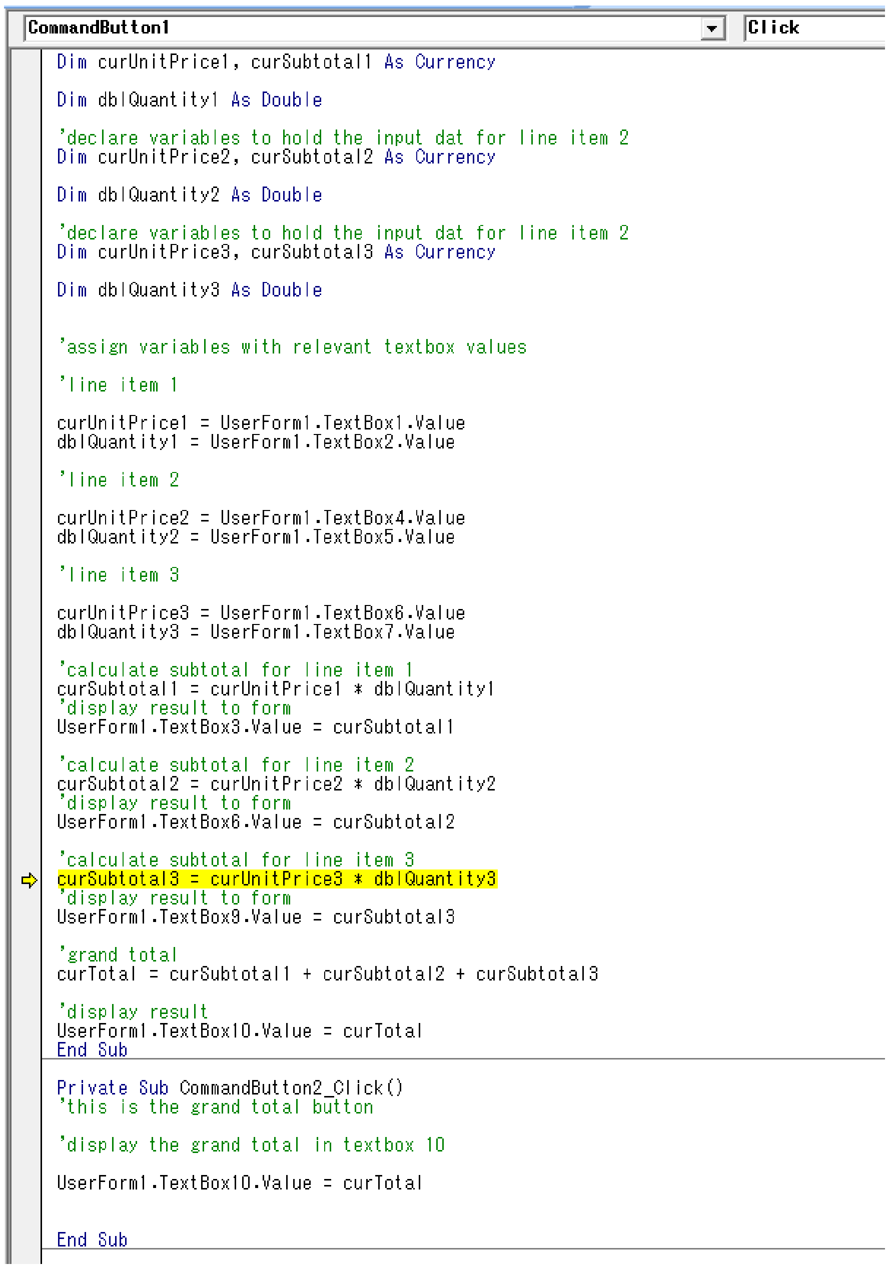
```
Dim curUnitPrice1, curSubtotal1 As Currency

Dim dblQuantity1 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice2, curSubtotal2 As Currency

Dim dblQuantity2 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice3, curSubtotal3 As Currency

Dim dblQuantity3 As [

'assign variables wi

'line item 1

curUnitPrice1 = UserF
dblQuantity1 = UserF

'line item 2

curUnitPrice2 = UserF
dblQuantity2 = UserF

'line item 3

curUnitPrice3 = UserForm1.TextBox6.Value
dblQuantity3 = UserForm1.TextBox7.Value

'calculate subtotal for line item 1
curSubtotal1 = curUnitPrice1 * dblQuantity1
'display result to form
UserForm1.TextBox3.Value = curSubtotal1

'calculate subtotal for line item 2
curSubtotal2 = curUnitPrice2 * dblQuantity2
'display result to form
UserForm1.TextBox6.Value = curSubtotal2

'calculate subtotal for line item 3
curSubtotal3 = curUnitPrice3 * dblQuantity3
'display result to form
UserForm1.TextBox9.Value = curSubtotal3

'grand total
curTotal = curSubtotal1 + curSubtotal2 + curSubtotal3

'display result
UserForm1.TextBox10.Value = curTotal
End Sub

Private Sub CommandButton2_Click()
'this is the grand total button

'display the grand total in textbox 10

UserForm1.TextBox10.Value = curTotal


End Sub

Private Sub CommandButton3_Click()
```

Microsoft Visual Basic

Run-time error '13':

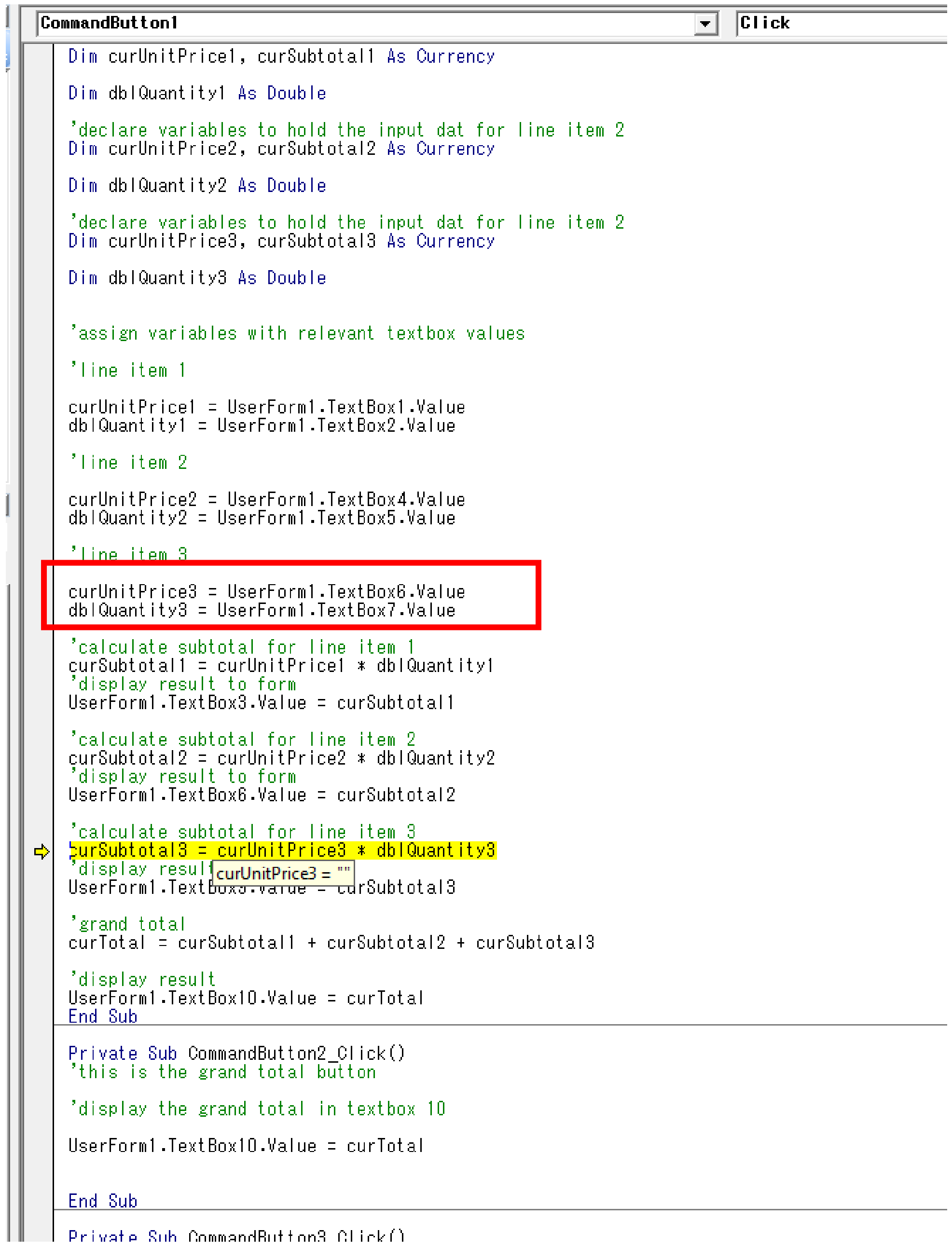
Type mismatch

Click **Debug** to go to the line of code where the compiler encountered the problem. The line where the problem caused the execution to **break** (pause) is highlighted by the compiler.



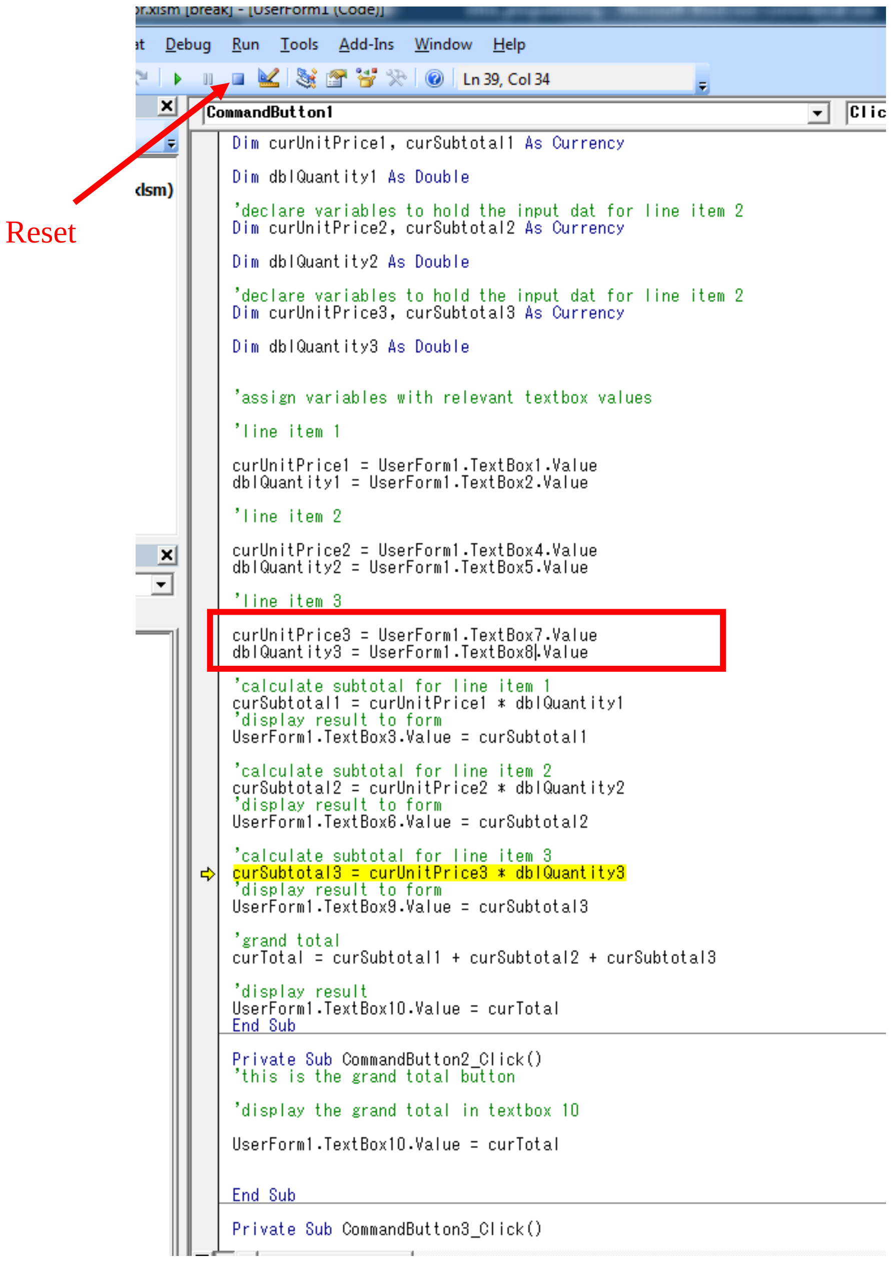
```
Dim curUnitPrice1, curSubtotal1 As Currency

Dim dblQuantity1 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice2, curSubtotal2 As Currency

Dim dblQuantity2 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice3, curSubtotal3 As Currency

Dim dblQuantity3 As Double


'assign variables with relevant textbox values

'line item 1

curUnitPrice1 = UserForm1.TextBox1.Value
dblQuantity1 = UserForm1.TextBox2.Value

'line item 2

curUnitPrice2 = UserForm1.TextBox4.Value
dblQuantity2 = UserForm1.TextBox5.Value

'line item 3

curUnitPrice3 = UserForm1.TextBox6.Value
dblQuantity3 = UserForm1.TextBox7.Value

'calculate subtotal for line item 1
curSubtotal1 = curUnitPrice1 * dblQuantity1
'display result to form
UserForm1.TextBox3.Value = curSubtotal1

'calculate subtotal for line item 2
curSubtotal2 = curUnitPrice2 * dblQuantity2
'display result to form
UserForm1.TextBox6.Value = curSubtotal2

'calculate subtotal for line item 3
curSubtotal3 = curUnitPrice3 * dblQuantity3
'display result to form
UserForm1.TextBox9.Value = curSubtotal3

'grand total
curTotal = curSubtotal1 + curSubtotal2 + curSubtotal3

'display result
UserForm1.TextBox10.Value = curTotal
End Sub

Private Sub CommandButton2_Click()
'this is the grand total button

'display the grand total in textbox 10

UserForm1.TextBox10.Value = curTotal


End Sub
```

Hovering over the variable names, the cursor tip shows the current value of the variable. In this case *curUnitPrice 3* shows a null value. Yet a value was assigned to it, or so it was thought.



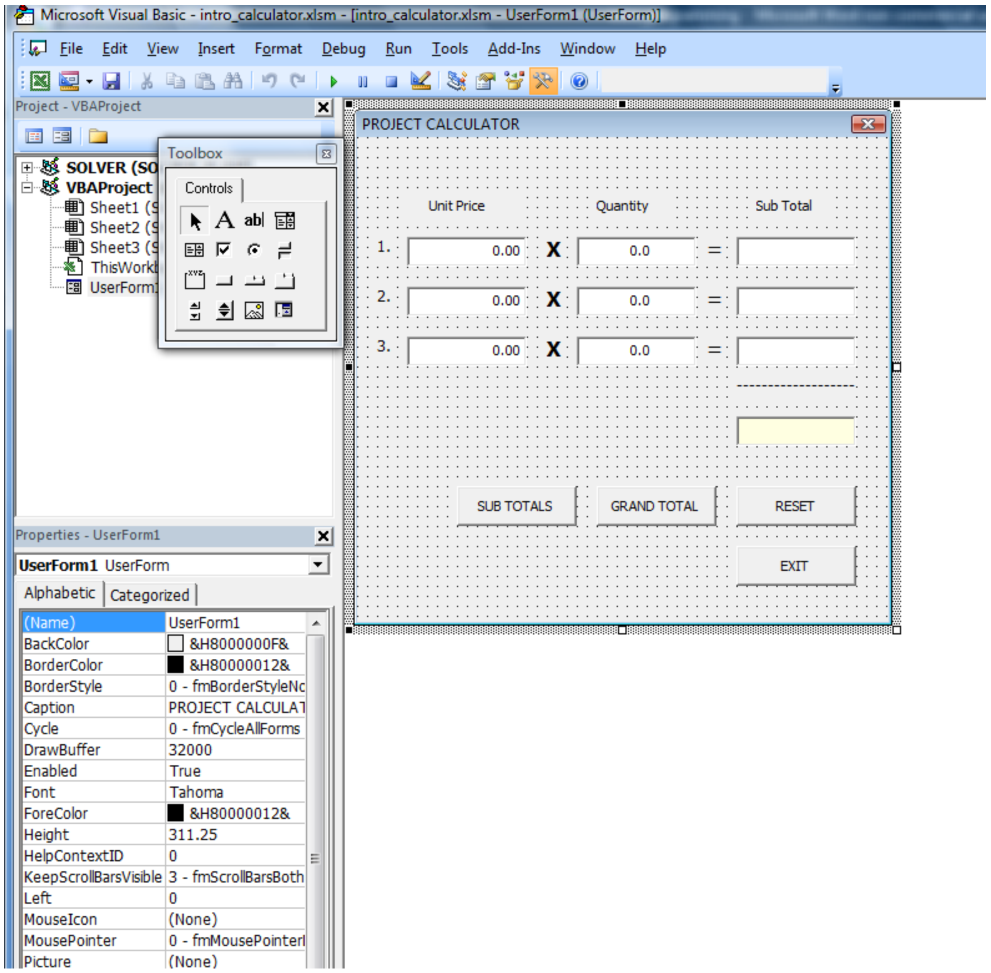
```
Dim curUnitPrice1, curSubtotal1 As Currency

Dim dblQuantity1 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice2, curSubtotal2 As Currency

Dim dblQuantity2 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice3, curSubtotal3 As Currency

Dim dblQuantity3 As Double


'assign variables with relevant textbox values

'line item 1

curUnitPrice1 = UserForm1.TextBox1.Value
dblQuantity1 = UserForm1.TextBox2.Value

'line item 2

curUnitPrice2 = UserForm1.TextBox4.Value
dblQuantity2 = UserForm1.TextBox5.Value

'line item 3

curUnitPrice3 = UserForm1.TextBox6.Value
dblQuantity3 = UserForm1.TextBox7.Value

'calculate subtotal for line item 1
curSubtotal1 = curUnitPrice1 * dblQuantity1
'display result to form
UserForm1.TextBox3.Value = curSubtotal1

'calculate subtotal for line item 2
curSubtotal2 = curUnitPrice2 * dblQuantity2
'display result to form
UserForm1.TextBox6.Value = curSubtotal2

'calculate subtotal for line item 3
curSubtotal3 = curUnitPrice3 * dblQuantity3
'display result
UserForm1.TextBox9.Value = curSubtotal3

'grand total
curTotal = curSubtotal1 + curSubtotal2 + curSubtotal3

'display result
UserForm1.TextBox10.Value = curTotal
End Sub

Private Sub CommandButton2_Click()
'this is the grand total button

'display the grand total in textbox 10

UserForm1.TextBox10.Value = curTotal


End Sub

Private Sub CommandButton3_Click()
```

Study the code carefully. *curUnitPrice3* was assigned the *TextBox6* value, this is incorrect. It should be assigned the value in *TextBox7*. *dblQuantity3* shall be assigned the value from T*extBox8*, not *TextBox7*.
Compile your code again through the **Debug** menu
After making all of these changes, click on the **Reset** button in the VBA toolbar.



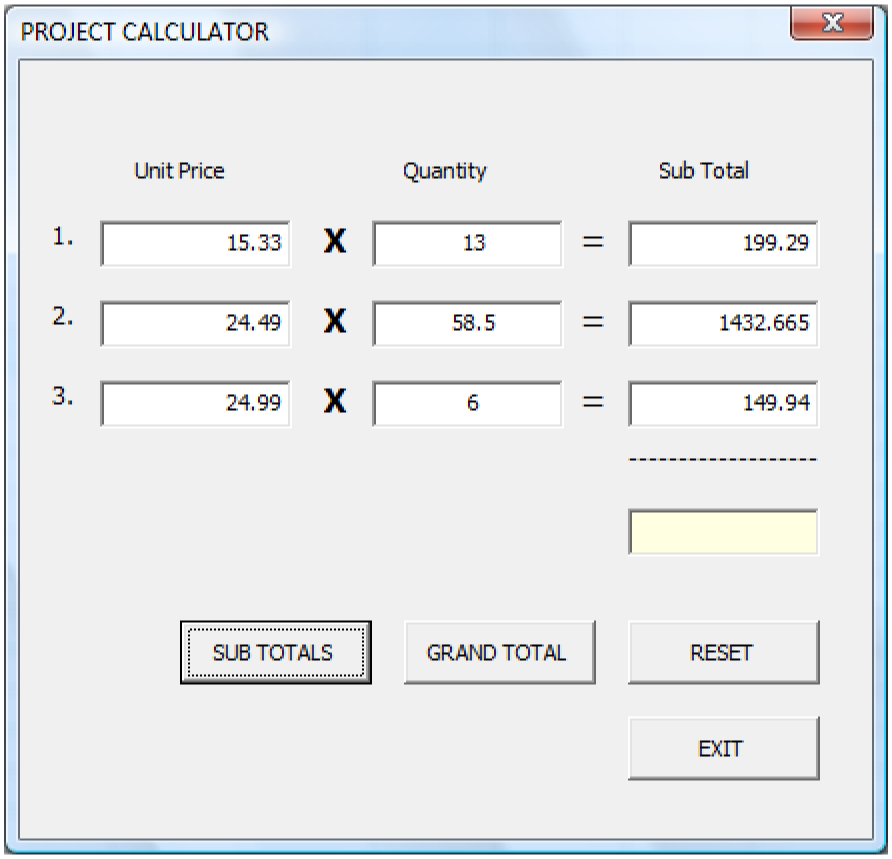
```
Dim curUnitPrice1, curSubtotal1 As Currency

Dim dblQuantity1 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice2, curSubtotal2 As Currency

Dim dblQuantity2 As Double

'declare variables to hold the input dat for line item 2
Dim curUnitPrice3, curSubtotal3 As Currency

Dim dblQuantity3 As Double


'assign variables with relevant textbox values

'line item 1

curUnitPrice1 = UserForm1.TextBox1.Value
dblQuantity1 = UserForm1.TextBox2.Value

'line item 2

curUnitPrice2 = UserForm1.TextBox4.Value
dblQuantity2 = UserForm1.TextBox5.Value

'line item 3

curUnitPrice3 = UserForm1.TextBox7.Value
dblQuantity3 = UserForm1.TextBox8.Value

'calculate subtotal for line item 1
curSubtotal1 = curUnitPrice1 * dblQuantity1
'display result to form
UserForm1.TextBox3.Value = curSubtotal1

'calculate subtotal for line item 2
curSubtotal2 = curUnitPrice2 * dblQuantity2
'display result to form
UserForm1.TextBox6.Value = curSubtotal2

'calculate subtotal for line item 3
curSubtotal3 = curUnitPrice3 * dblQuantity3
'display result to form
UserForm1.TextBox9.Value = curSubtotal3

'grand total
curTotal = curSubtotal1 + curSubtotal2 + curSubtotal3

'display result
UserForm1.TextBox10.Value = curTotal
End Sub

Private Sub CommandButton2_Click()
'this is the grand total button

'display the grand total in textbox 10

UserForm1.TextBox10.Value = curTotal


End Sub

Private Sub CommandButton3_Click()
```

Upon clicking **Reset**, the application reverts to design time.



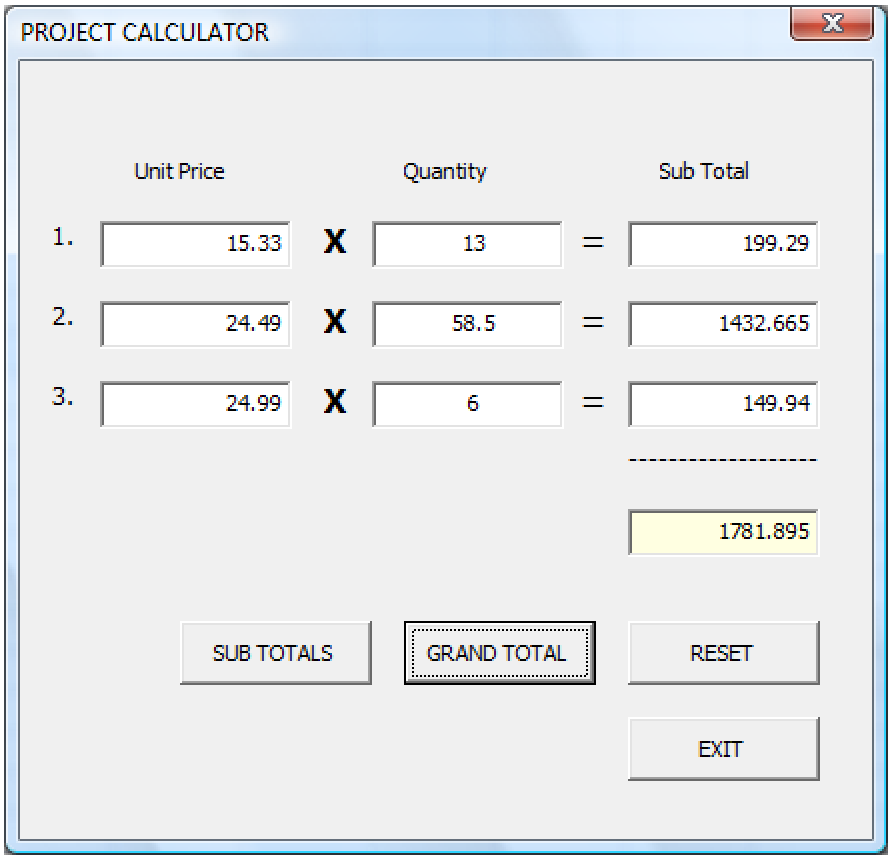
| | |
|---|---|
| (Name) | UserForm1 |
| BackColor | &H8000000F& |
| BorderColor | &H80000012& |
| BorderStyle | 0 - fmBorderStyleNc |
| Caption | PROJECT CALCULAT |
| Cycle | 0 - fmCycleAllForms |
| DrawBuffer | 32000 |
| Enabled | True |
| Font | Tahoma |
| ForeColor | &H80000012& |
| Height | 311.25 |
| HelpContextID | 0 |
| KeepScrollBarsVisible | 3 - fmScrollBarsBoth |
| Left | 0 |
| MouseIcon | (None) |
| MousePointer | 0 - fmMousePointerI |
| Picture | (None) |

Click **Run** to open the form
Enter the input values to test the application.
Click on the **Sub Totals** button.
Success.



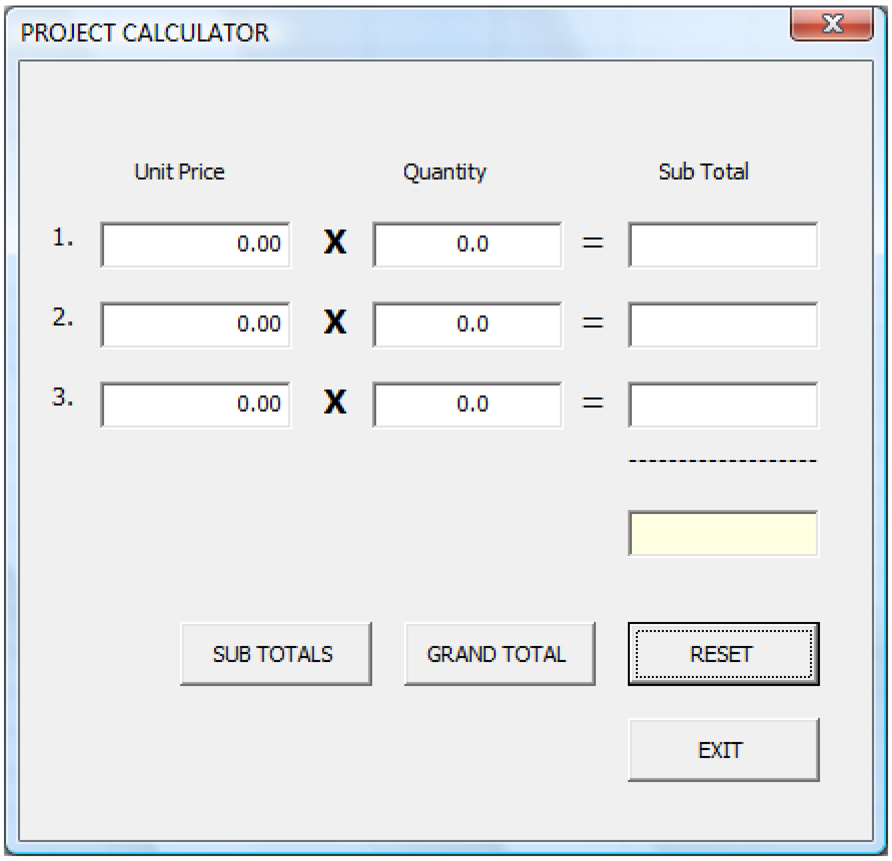
| | Unit Price | | Quantity | | Sub Total |
|---|---|---|---|---|---|
| 1. | 15.33 | X | 13 | = | 199.29 |
| 2. | 24.49 | X | 58.5 | = | 1432.665 |
| 3. | 24.99 | X | 6 | = | 149.94 |

Click on the **Grand Total** button
Success.


PROJECT CALCULATOR

|  | Unit Price |  | Quantity |  | Sub Total |
|---|---|---|---|---|---|
| 1. | 15.33 | X | 13 | = | 199.29 |
| 2. | 24.49 | X | 58.5 | = | 1432.665 |
| 3. | 24.99 | X | 6 | = | 149.94 |
|  |  |  |  |  | 1781.895 |

Click on **Reset**.
Success. The data entries clear as expected.



Click on **Exit** to exit the application and close the form.
The test has been a success.

## 4. CONCLUSION

This course has presented a broad overview of fundamental concepts and principles of computer programming, and presented them in situations encountered by practicing engineers and scientists. All codes were developed using the *Excel Visual Basic for Applications* (VBA) programming language.

This course, the first of a four-part series, covered an introduction to computers and computer programming languages, the Excel VBA programming environment, and the concept of variables and how they are applied in VBA.

This course has enabled participants to identify situations where programming is relevant and will be of advantage to the professional. Practitioners are strongly encouraged to look out for situations in their domains of expertise where programming solutions are applicable and will be of benefit to their work and their organization.

Computer programming requires a careful and meticulous approach, and can only be mastered and retained by practice and repetition.

Good Luck and Happy Programming.

## REFERENCES

Bradley, J. C., & Millspaugh, A. C. (1999). *Programming in Visual Basic 6.0*. Irwin McGraw-Hill.

FunctionX Inc. (2013). *VBA for Microsoft Office Excel 2007*. Retrieved December 21, 2013, from FunctionX Tutorials: www.functionx.com/

Microsoft. (2013). *Excel 2013 developer reference*. Retrieved October 15, 2013, from Office Dev Center: http://msdn.microsoft.com/en-us/library/office/ee861528.aspx

Images were all drawn/ prepared by K. Ofosu