

Storing information with Variables

Let's dive into the world of Excel VBA variables.

Variables in VBA

Variables are used to store data that your VBA code can manipulate. Think of them as containers that hold information which can be referenced and manipulated throughout your code.

Declaring Variables

Why declare variables?

Improves Code Readability: Declaring variables makes your code easier to read and understand.

Error Checking: VBA can catch errors related to undeclared variables.

Memory Management: Helps manage memory usage efficiently.

How to declare variables?

Use the Dim statement:

```
Dim variableName As DataType
```

Example:

```
Dim total As Integer
```

```
Dim name As String
```

Determining Data Types

Choosing the right data type is crucial for efficient memory usage and performance. Here are some common data types:

Integer: Whole numbers.

Long: Larger whole numbers.

Double: Numbers with decimals.

String: Text.

Boolean: True or False.

Example:

```
Dim age As Integer
```

```
Dim salary As Double
```

```
Dim isActive As Boolean
```

Public vs Private Scope

Scope determines where a variable can be accessed from.

Procedure-Level (Local) Variables: Declared within a procedure using Dim. Accessible only within that procedure.

Module-Level (Private) Variables: Declared at the top of a module using Private. Accessible to all procedures within that module.

Public Variables: Declared at the top of a module using Public. Accessible from any module in the project.

Example:

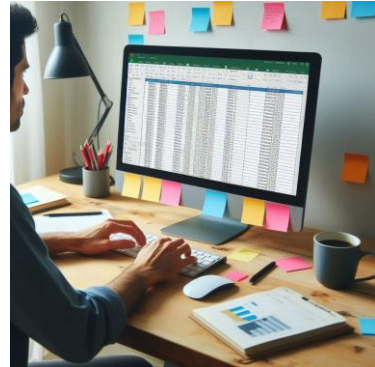
```
Private moduleVariable As Integer
```

```
Public globalVariable As String
```

```
Sub ExampleProcedure()
```

```
    Dim localVariable As Double
```

```
End Sub
```





Assigning a Value to a Variable

After declaring a variable, you can assign a value to it using the assignment operator =.

```
variableName = value
```

Examples

Example 1: Storing a Number

```
Sub StoreNumber()  
    Dim age As Integer  
    age = 25  
    MsgBox "Age: " & age  
End Sub
```

In this example, the variable age is declared as an Integer and assigned the value 25. The MsgBox function then displays the value of age.

Example 2: Storing Text

```
Sub StoreText()  
    Dim name As String  
    name = "Alice"  
    MsgBox "Name: " & name  
End Sub
```

Here, the variable name is declared as a String and assigned the value "Alice". The MsgBox function displays the value of name.

Example 3: Storing a Calculation Result

```
Sub StoreCalculation()  
    Dim price As Double  
    Dim quantity As Integer  
    Dim total As Double  
  
    price = 19.99  
    quantity = 5  
    total = price * quantity  
  
    MsgBox "Total cost: " & total  
End Sub
```

In this example, price and total are declared as Double, and quantity is declared as an Integer. The total cost is calculated by multiplying price and quantity, and the result is stored in the total variable.

Example 4: Storing a Boolean Value

```
Sub StoreBoolean()  
    Dim isActive As Boolean  
    isActive = True  
    MsgBox "Is Active: " & isActive  
End Sub
```

Here, the variable isActive is declared as a Boolean and assigned the value True. The MsgBox function displays the value of isActive.

[How do I sort in Excel VBA?](#)

Using Variables in Conditions

Variables can also be used in conditions to control the flow of your program.

Example 5: Using Variables in an If Statement

```
Sub CheckAge()
```

```
    Dim age As Integer  
    age = 18
```

```
    If age >= 18 Then  
        MsgBox "You are an adult."  
    Else  
        MsgBox "You are a minor."  
    End If
```

```
End Sub
```



Using Variables to Trap Errors

You can use variables to store error information and handle errors gracefully.

Example:

```
On Error GoTo ErrorHandler
```

```
Dim result As Double
```

```
result = 10 / 0 ' This will cause a division by zero error
```

```
Exit Sub
```

```
ErrorHandler:
```

```
    Dim errorMessage As String  
    errorMessage = "An error occurred: " & Err.Description  
    MsgBox errorMessage
```

```
End Sub
```



Using the Locals Window

The Locals Window in the VBA editor allows you to observe the values of variables while debugging.

Set a breakpoint in your code by clicking in the margin next to a line of code.

Run your code. When it hits the breakpoint, the Locals Window will display the current values of all variables in the current scope.

More variable examples

Example 1: Simple Calculation

```
Sub CalculateTotal()
```

```
    Dim price As Double  
    Dim quantity As Integer  
    Dim total As Double
```

```
    price = 19.99  
    quantity = 5  
    total = price * quantity
```

```
    MsgBox "Total cost: " & total  
End Sub
```

Example 2: Using Public Variables

```
Public counter As Integer
```

```
Sub IncrementCounter()
```

```
    counter = counter + 1  
    MsgBox "Counter: " & counter  
End Sub
```

Example 3: Error Handling

```
Sub SafeDivision()
```

```
    On Error GoTo ErrorHandler  
    Dim numerator As Double  
    Dim denominator As Double  
    Dim result As Double
```

```
    numerator = 10  
    denominator = 0  
    result = numerator / denominator
```

```
    MsgBox "Result: " & result  
Exit Sub
```

```
ErrorHandler:
```

```
    MsgBox "Error: Division by zero!"  
End Sub
```

Creating functions

Creating your own functions in Excel VBA can greatly enhance your productivity by allowing you to perform custom calculations and operations. Let's go through the process of creating User Defined Functions (UDFs) with multiple examples.

Writing Your Own User Defined Functions (UDFs)

A UDF is a custom function that you can create using VBA. These functions can be used in Excel just like built-in functions.

Basic Structure of a UDF

To create a UDF, you need to define a function using the Function keyword, followed by the function name, any arguments, and the data type of the return value.

```
Function FunctionName(arguments) As DataType
```

```
    ' Function code
```

```
    FunctionName = result
```

```
End Function
```

Example 1: Simple Addition Function

```
Function AddNumbers(x As Double, y As Double) As Double
```

```
    AddNumbers = x + y
```

```
End Function
```



You can use this function in Excel by typing `=AddNumbers(5, 10)` in a cell, which will return 15.

Working with Multiple Arguments

You can create functions that take multiple arguments to perform more complex calculations.

Example 2: Calculating the Area of a Rectangle

```
Function RectangleArea(length As Double, width As Double) As Double
```

```
    RectangleArea = length * width
```

```
End Function
```

Use this function in Excel by typing `=RectangleArea(5, 10)` to get the area of a rectangle with length 5 and width 10.

Example 3: Calculating the Average of Three Numbers

```
Function AverageThreeNumbers(a As Double, b As Double, c As Double) As Double
```

```
    AverageThreeNumbers = (a + b + c) / 3
```

```
End Function
```

Use this function in Excel by typing `=AverageThreeNumbers(5, 10, 15)` to get the average of the three numbers.

Using Your Function in Excel

Once you've created a UDF, you can use it in Excel just like any other function. Here's how you can do it:

Open the Visual Basic for Applications (VBA) editor by pressing `Alt + F11`.

Insert a new module by clicking `Insert > Module`.

Copy and paste your function code into the module.

Close the VBA editor.

Use your function in Excel by typing `=FunctionName(arguments)` in a cell.

Advanced Examples

Example 4: Calculating Compound Interest

```
Function CompoundInterest(principal As Double, rate As Double, periods As Integer) As Double
```

```
    CompoundInterest = principal * (1 + rate) ^ periods
```

```
End Function
```

Use this function in Excel by typing `=CompoundInterest(1000, 0.05, 10)` to calculate the compound interest on a principal of 1000 at a 5% interest rate over 10 periods.

Creating UDFs can significantly extend the capabilities of Excel, allowing you to perform custom calculations tailored to your specific needs.

Message Boxes and Input Boxes

Let's go through some examples of how to create Message Boxes and Input Boxes in Excel VBA.

1. Displaying a Message

To display a simple message box, you can use the MsgBox function. Here's an example:

```
Sub DisplayMessage()  
    MsgBox "Hello, this is a simple message box!"  
End Sub
```

2. Adding a Yes/No User Choice

To create a message box with Yes and No buttons, you can use the vbYesNo constant. You can also capture the user's response:

```
Sub YesNoMessageBox()  
    Dim response As VbMsgBoxResult  
    response = MsgBox("Do you want to continue?", vbYesNo, "Yes/No Example")  
  
    If response = vbYes Then  
        MsgBox "You chose Yes!"  
    Else  
        MsgBox "You chose No!"  
    End If  
End Sub
```

3. Getting Feedback from the End User

To get input from the user, you can use the InputBox function. Here's an example:

```
Sub GetUserInput()  
    Dim userInput As String  
    userInput = InputBox("Please enter your name:", "User Input")  
  
    If userInput <> "" Then  
        MsgBox "Hello, " & userInput & "!"  
    Else  
        MsgBox "You didn't enter anything."  
    End If  
End Sub
```



These examples should help you get started with creating Message Boxes and Input Boxes in Excel VBA.

Handling Errors

let's look at handling errors in Excel VBA with some examples!

Defining VBA's Error Trapping Options

VBA provides three main error trapping options:

On Error GoTo 0: Disables any error handling in the current procedure.

On Error Resume Next: Continues execution with the next line of code after an error occurs.

On Error GoTo [label]: Transfers control to a specified line label when an error occurs.

Capturing Errors with the On Error Statement

The On Error statement is used to define how VBA should handle errors. Here are the three main forms:

On Error GoTo 0: This is the default setting. It stops code execution and displays an error message.

On Error Resume Next: This tells VBA to ignore the error and continue with the next line of code.

On Error GoTo [label]: This directs VBA to jump to a specific line label when an error occurs.

Example:

```
Sub ExampleOnError()  
    On Error GoTo ErrorHandler  
    Dim x As Integer  
    x = 1 / 0 ' This will cause a division by zero error  
    Exit Sub  
ErrorHandler:  
    MsgBox "An error occurred: " & Err.Description  
End Sub
```

Determining the Err Object

The Err object contains information about the error that occurred. Key properties include:

Err.Number: The error number.

Err.Description: A description of the error.

Err.Source: The name of the object or application that caused the error.

Example:

```
Sub ExampleErrObject()  
    On Error Resume Next  
    Dim x As Integer  
    x = 1 / 0  
    If Err.Number <> 0 Then  
        MsgBox "Error " & Err.Number & ": " & Err.Description  
        Err.Clear ' Clear the error  
    End If  
End Sub
```

**Coding an Error-Handling Routine**

An error-handling routine is a section of code that executes when an error occurs. It typically includes:

Error handling code: To manage the error.

Cleanup code: To release resources or reset states.

Resume statement: To continue execution after handling the error.

Example:

```
Sub ExampleErrorHandlingRoutine()  
    On Error GoTo ErrorHandler  
    ' Code that may cause an error  
    Dim x As Integer  
    x = 1 / 0  
    Exit Sub  
ErrorHandler:  
    MsgBox "An error occurred: " & Err.Description  
    ' Cleanup code  
    Resume Next ' Continue with the next line of code  
End Sub
```

**Using Inline Error Handling**

Inline error handling allows you to handle errors directly where they occur, rather than jumping to a separate error handler.

Example:

```
Sub ExampleInlineErrorHandling()  
    On Error Resume Next  
    Dim x As Integer  
    x = 1 / 0  
    If Err.Number <> 0 Then  
        MsgBox "Error handled inline: " & Err.Description  
        Err.Clear  
    End If  
    On Error GoTo 0 ' Reset error handling  
End Sub
```

Creating custom dialogue boxes with UserForms

Creating custom dialogue boxes with UserForms in Excel VBA can greatly enhance the interactivity of your spreadsheets. Let's go through each step with examples:

1. Drawing UserForms

To create a UserForm:

Open the Visual Basic for Applications (VBA) editor by pressing Alt + F11.
Insert a new UserForm by selecting Insert > UserForm.

2. Setting UserForm Properties, Events, and Methods

You can set properties such as the form's name, caption, and size in the Properties window. For example:

Name: MyUserForm

Caption: Custom Dialog

To handle events like initializing the form, you can use the code window:

```
Private Sub UserForm_Initialize()  
    Me.Caption = "Welcome to My Custom Dialog"  
End Sub
```

3. Using Text Boxes, Command Buttons, Combo Boxes, and Other Controls

You can add controls by selecting them from the Toolbox and drawing them on the UserForm.

For example:

TextBox: For user input.

CommandButton: To trigger actions.

ComboBox: For dropdown selections.

4. Formatting Controls

You can format controls by setting their properties. For example, to set the text box properties:

```
Private Sub UserForm_Initialize()  
    Me.TextBox1.Text = "Enter your name"  
    Me.TextBox1.Font.Size = 12  
    Me.TextBox1.Font.Bold = True  
End Sub
```

5. Applying Code to Controls

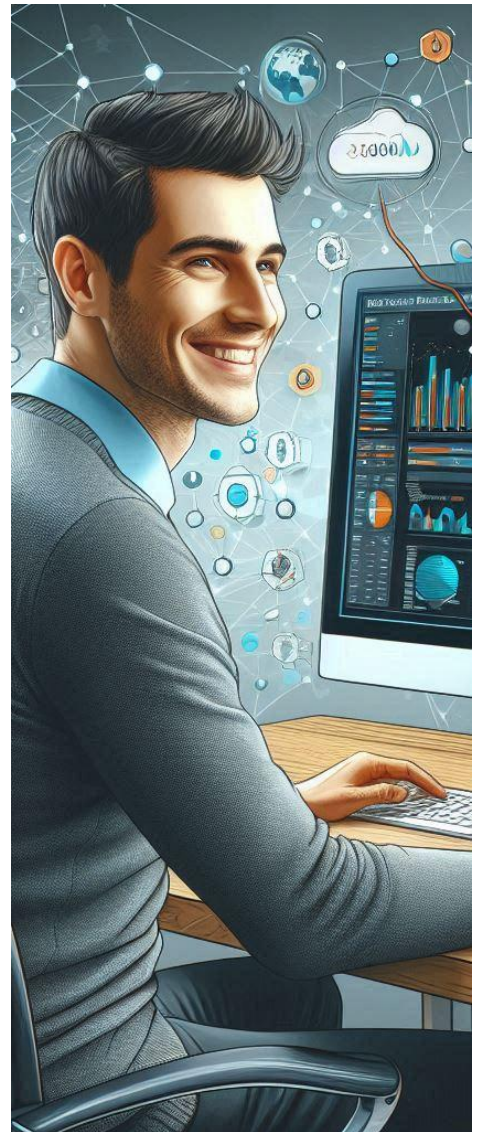
You can write VBA code to handle events for controls. For example, to handle a button click:

```
Private Sub CommandButton1_Click()  
    MsgBox "Hello, " & Me.TextBox1.Text  
End Sub
```

6. How to Launch a Form in Code

To show the UserForm, you can use the following code in a module:

```
Sub ShowMyUserForm()  
    MyUserForm.Show  
End Sub
```



Example 1

Here's a complete example that ties everything together:

Create a UserForm named MyUserForm.

Add a TextBox (TextBox1) and a CommandButton (CommandButton1).

Set the properties and write the following code:

```
Private Sub UserForm_Initialize()  
    Me.Caption = "Welcome to My Custom Dialog"  
    Me.TextBox1.Text = "Enter your name"  
    Me.TextBox1.Font.Size = 12  
    Me.TextBox1.Font.Bold = True  
End Sub
```

```
Private Sub CommandButton1_Click()  
    MsgBox "Hello, " & Me.TextBox1.Text  
End Sub
```

In a module, add the code to show the form:

```
Sub ShowMyUserForm()  
    MyUserForm.Show  
End Sub
```

Run ShowMyUserForm to display your custom dialog box.

Example 2

Let's create a more advanced UserForm example that includes multiple controls, validation, and dynamic updates.

We'll create a form for user registration with fields for name, email, and age, and a submit button that validates the input and displays a summary.

Step-by-Step Advanced Example

1. Drawing the UserForm

Open the VBA editor (Alt + F11).

Insert a new UserForm (Insert > UserForm).

Name the UserForm UserFormRegistration.

2. Adding Controls

Add the following controls to the UserForm:

Labels: For "Name", "Email", and "Age".

TextBoxes: For user input (TextBoxName, TextBoxEmail, TextBoxAge).

CommandButton: For submission (CommandButtonSubmit).

3. Setting Properties

Set properties for better user experience:

UserForm: Caption = "User Registration"

TextBoxName: Name = "TextBoxName"

TextBoxEmail: Name = "TextBoxEmail"

TextBoxAge: Name = "TextBoxAge"

CommandButtonSubmit: Caption = "Submit",

Name = "CommandButtonSubmit"

4. Writing Code for Initialization

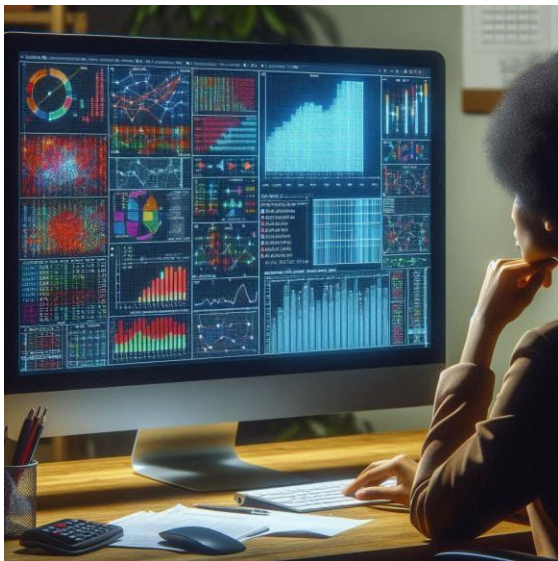
Initialize the form with default values and formatting:

```
Private Sub UserForm_Initialize()  
    Me.Caption = "User Registration"  
    Me.TextBoxName.Text = ""  
    Me.TextBoxEmail.Text = ""  
    Me.TextBoxAge.Text = ""  
End Sub
```

5. Adding Validation and Submission Code

Validate user input and display a summary:

```
Private Sub CommandButtonSubmit_Click()  
    Dim userName As String  
    Dim userEmail As String  
    Dim userAge As Integer  
  
    userName = Me.TextBoxName.Text  
    userEmail = Me.TextBoxEmail.Text  
  
    ' Validate Name  
    If userName = "" Then  
        MsgBox "Please enter your name.", vbExclamation  
        Exit Sub  
    End If
```



[How to: Excel VBA clear clipboard](#)

[Excel VBA Uppercase, Lowercase and more](#)

```
' Validate Email
If userEmail = "" Or InStr(1, userEmail, "@") = 0 Then
    MsgBox "Please enter a valid email address.", vbExclamation
Exit Sub
End If
```

```
' Validate Age
If IsNumeric(Me.TextBoxAge.Text) Then
    userAge = CInt(Me.TextBoxAge.Text)
    If userAge <= 0 Or userAge > 120 Then
        MsgBox "Please enter a valid age.", vbExclamation
    Exit Sub
End If
Else
    MsgBox "Please enter a numeric age.", vbExclamation
Exit Sub
End If
```

```
' Display Summary
MsgBox "Registration Successful!" & vbCrLf & _
    "Name: " & userName & vbCrLf & _
    "Email: " & userEmail & vbCrLf & _
    "Age: " & userAge, vbInformation
End Sub
```

6. Launching the UserForm

Add a module to launch the UserForm:

```
Sub ShowUserFormRegistration()
    UserFormRegistration.Show
End Sub
```

Complete Example

Here's the complete code for the UserForm and module:

UserForm Code:

```
Private Sub UserForm_Initialize()
    Me.Caption = "User Registration"
    Me.TextBoxName.Text = ""
    Me.TextBoxEmail.Text = ""
    Me.TextBoxAge.Text = ""
End Sub
```

```
Private Sub CommandButtonSubmit_Click()
    Dim userName As String
    Dim userEmail As String
    Dim userAge As Integer
```

```
    userName = Me.TextBoxName.Text
    userEmail = Me.TextBoxEmail.Text
```

```
' Validate Name
If userName = "" Then
    MsgBox "Please enter your name.", vbExclamation
Exit Sub
End If
```

```
' Validate Email
If userEmail = "" Or InStr(1, userEmail, "@") = 0 Then
    MsgBox "Please enter a valid email address.", vbExclamation
Exit Sub
End If
```

```
' Validate Age
If IsNumeric(Me.TextBoxAge.Text) Then
    userAge = CInt(Me.TextBoxAge.Text)
    If userAge <= 0 Or userAge > 120 Then
        MsgBox "Please enter a valid age.", vbExclamation
    Exit Sub
End If
Else
    MsgBox "Please enter a numeric age.", vbExclamation
Exit Sub
End If
```

```
' Display Summary
MsgBox "Registration Successful!" & vbCrLf & _
    "Name: " & userName & vbCrLf & _
    "Email: " & userEmail & vbCrLf & _
    "Age: " & userAge, vbInformation
End Sub
```

Module Code:

```
Sub ShowUserFormRegistration()
    UserFormRegistration.Show
End Sub
```

This example demonstrates how to create a more complex UserForm with multiple controls, input validation, and dynamic updates

**Ask questions on our
post course learning
support forum**

Log in using your email
and your post course
email when you
completed the feedback

