



# Macro Programming

---

**Vignesh V**

Department of Computer Applications

**vigneshv@pes.edu**

---

# Macro Programming

---

## Experiential learning — Working with Excel objects, Arrays & Collections

**Vignesh V**

Department of Computer Applications



## Working with Object Variables

---

- Object variables store references to objects (Worksheet, Range, Workbook).
- Syntax example:

```
Dim ws As Worksheet
```

```
Set ws = ThisWorkbook.Worksheets("Sheet1")
```

```
ws.Range("A1").Value = "Hello"
```

### Steps:

1. Insert Module in VBE.
2. Paste code and run (F5) or call from Immediate window.
3. Verify cell A1 of Sheet1 contains "Hello".



# Macro Programming

## Working with Object Variables

---

**Explanation:** Using `ws` shortens code and improves readability and performance.



# Macro Programming

## Task: Create an Index of Worksheets

---

**Task:** Loop through all worksheets in the active workbook and write their names to a new sheet called "Index".



# Macro Programming

## Task: Create an Index of Worksheets

```
Sub CreateSheetIndex()  
Dim wb As Workbook  
Dim ws As Worksheet  
Dim idx As Worksheet  
Dim r As Long  
Set wb = ThisWorkbook  
On Error Resume Next  
Set idx = wb.Worksheets("Index")  
On Error GoTo 0  
If idx Is Nothing Then Set idx = wb.Worksheets.Add(Before:=wb.Worksheets(1)): idx.Name = "Index"  
idx.Cells.Clear  
r = 1  
For Each ws In wb.Worksheets  
If ws.Name <> "Index" Then  
idx.Cells(r, 1).Value = ws.Name  
r = r + 1  
End If  
Next ws  
End Sub
```



# Macro Programming

## Task: Create an Index of Worksheets

---

### Steps:

1. Insert the code into a Module.
2. Save workbook as macro-enabled.
3. Run `CreateSheetIndex`.
4. Open "Index" sheet to verify list of sheet names.



## Copying Ranges Efficiently (Value vs. Copy)

---

- Direct value assignment is faster than Copy/Paste because it avoids the Clipboard.

### Code

---

```
Sub CopyValuesFast()  
Dim src As Range, dst As Range  
Set src = Worksheets("Data").Range("A1:A100")  
Set dst = Worksheets("Output").Range("A1")  
dst.Resize(src.Rows.Count, src.Columns.Count).Value = src.Value  
End Sub
```



## Copying Ranges Efficiently (Value vs. Copy)

---

### Steps:

1. Ensure sheets "Data" and "Output" exist.
2. Populate A1:A100 on "Data" with sample numbers.
3. Run `CopyValuesFast` and check values on "Output".

**Explanation:** Assigning the `.Value` property transfers an array of values in one operation.



## Arrays

---

- An **array** stores multiple values in one variable.
- Types: static (fixed size) and dynamic (resizable).
- Arrays are much faster than writing to cells one-by-one.

### Example (static):

```
Dim nums(1 To 5) As Long
```

```
nums(1) = 10
```

**Explanation:** Arrays are memory-based structures ideal for bulk data operations.



## Range to Array

---

```
Sub ReadRangeToArray()  
Dim arr As Variant  
arr = Worksheets("Data").Range("A1:A100").Value  
' arr is now a 2D array (1 to 100, 1 to 1)  
Debug.Print arr(1, 1) ' first value  
End Sub
```



## Range to Array

---

### Steps:

1. Fill Data!A1:A100 with values.
2. Run `ReadRangeToArray`.
3. Open Immediate window to see output.

**Explanation:** Reading the range once into `arr` reduces many Range calls and speeds processing.



# Macro Programming

## Iterating a 2D Array

```
Sub Fill2DArray()
```

```
Dim arr(1 To 3, 1 To 4) As Integer
```

```
Dim i As Integer, j As Integer
```

```
' Fill array with row * col
```

```
For i = 1 To 3
```

```
    For j = 1 To 4
```

```
        arr(i, j) = i * j
```

```
    Next j
```

```
Next i
```

```
' Print array contents to Immediate Window
```

```
For i = 1 To 3
```

```
    For j = 1 To 4
```

```
        Debug.Print "Row " & i & ", Col " & j & " = " & arr(i, j)
```

```
    Next j
```

```
Next i
```

```
End Sub
```



## Dynamic Arrays

---

- A **Dynamic Array** can change its size at runtime.
- Declared **without dimensions**, then defined using **ReDim**.
- Can be **resized** or **cleared** using **ReDim** or **Erase**.
- Use when the number of elements is unknown in advance.

### Example:

```
Dim arr() As Variant 'no size yet
```

```
ReDim arr(1 To 5)
```



## Dynamic Arrays

---

Unlike static arrays (`Dim arr(1 To 10)`), dynamic arrays are flexible and memory-efficient.



## Declaring and Initializing

---

**Sub DynamicExample()**

**Dim numbers() As Integer**

**ReDim numbers(1 To 3)**

**numbers(1) = 10**

**numbers(2) = 20**

**numbers(3) = 30**

**End Sub**



## Declaring and Initializing

---

### Steps:

1. Declare `numbers()` without dimensions.
2. Use `ReDim` to assign size.
3. Assign and use values.

### Explanation:

`ReDim` allocates memory dynamically; array exists only after resizing.



## Built-in Collections

---

- A **collection** is a group of related objects.
- Built-in collections:
  - **Workbooks:** `Workbooks.Count` gives number of open workbooks.
  - **Worksheets:** `Worksheets("Sheet1").Activate`.
  - **Sheets:** Includes worksheets and chart sheets.

### Example:

For Each ws In Worksheets

    Debug.Print ws.Name

Next ws



## Built-in Collections

---

Collections simplify iterating through groups of objects.



# Macro Programming

## Custom Collections

---

**Sub CustomCollection()**

**Dim students As New Collection**

**students.Add "Alice"**

**students.Add "Bob"**

**students.Add "Charlie"**

**Dim s As Variant**

**For Each s In students**

**Debug.Print s**

**Next s**

**End Sub**



## Comparing Arrays vs Collections

---

**Arrays:** Fixed or dynamic size, index-based access, best for structured tabular data.

**Collections:** Flexible size, object-based, easier to add/remove items.

**When to use:** Arrays for data processing, Collections for lists of objects.



## Comparing Arrays vs Collections

Feature	Arrays	Collections
Structure	Fixed-size or dynamic table of values	Dynamic list of items
Data Types	Usually one type	Can mix different types
Indexing	Integer-based (1, 2, 3...)	Can use key names
Resizing	Must use <code>ReDim</code>	Automatic when adding items
Access Speed	Very fast (especially for numeric data)	Slightly slower (object-based)



## Experiential Task 1: Array Practice

---

**Task:** Create a 1D array to hold 10 student scores. Calculate the average using a loop and display in MsgBox.



## Experiential Task 2: 2D Array Practice

---

**Task:** Create a 5×5 multiplication table using a 2D array and write it into Excel (cells A1:E5).



## Experiential Task 3: Collections Practice

---

**Task:** Create a custom collection of product names. Loop through and write each to column A.



**THANK YOU**

---

**Vignesh V**

Department of Computer Applications

**vigneshv@pes.edu**