



## Week 7:

- Recursion
- Working with Outlook Object
- Writing new text with Word

---

---

---

---

---

---

---

---



## SinApprox.vbs

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!} = \sum_{n=0}^{\infty} (-1)^n T_n$$

- Modularization based on mathematical formula
  - Factorial and power are obvious subtasks
  - Ratio of them in term  $T$  is also
- What are the efficiency issues here?
- Check them out!

---

---

---

---

---

---

---

---



## Recursion

- Gaddis, ch 12
- Problem must be reducible to a series of identical problems.
- Always inefficient
- Can be easier to design and code.

---

---

---

---

---

---

---

---



## Fac.vbs

- Compare explicit and recursive factorial functions in *Fac.vbs*.

---

---

---

---

---

---

---

---



## Nonrecursive - Factorial

```
Sub Factorial(ByVal n, ByRef result)
    result = 1
    Do While ( n > 1 )
        result = result * n
        n = n - 1
    Loop
End Sub
```

---

---

---

---

---

---

---

---



## Recursion - Factorial

```
Function Fac( ByVal n)
    Dim localResult

    MsgBox "Doing Fac of " & n
    if n <= 1 Then
        localResult = 1    base case: stops recursion
    Else
        localResult = n * Fac(n-1)
    End If
    MsgBox "Returning from Fac of " & n & " localRslt is:" &
        localResult
    Fac = localResult
End Function
```

---

---

---

---

---

---

---

---



## Recursion - Factorial

- What are the inefficiency issues here?
- Does ease of coding and design outweigh inefficiency issues here?

---

---

---

---

---

---

---

---



## Pi.vbs

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

$$\pi = 4 \left( 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \dots \right)$$

$$\pi = 4 \sum_{n=1}^{\infty} (-1)^{n+1} \left( \frac{1}{2n-1} \right)$$

- Compare algorithms within Pi.vbs
- Can be done recursively or explicitly

---

---

---

---

---

---

---

---



## Nonrecursive - Pi

```
Function iPI(ByVal n)
  Dim sign, term, i
  sign = 4
  iPI = 0
  For I = 1 to n Step 1
    term = sign / ((2*I) - 1)
    iPI = iPI + term
    sign = sign * -1
  Next
End Function
```

---

---

---

---

---

---

---

---



## Recursion - Pi

```
Function rPI (ByVal n)
    Dim sign, myTerm
    If (n mod 2) = 0 Then
        sign = -1
    Else
        sign = 1
    End If

    If (n = 0) Then
        rPI = 0 base case: stops the recursion
    Else
        myTerm = (sign * 4 / ((2 * n) - 1))
        MsgBox "n=" & n & " MyTerm=" & myTerm
        rPI = myTerm + rPI(n-1)
    End If
End Function
```

---

---

---

---

---

---

---

---



## Recursion – Pi.vbs

- Compare Pi algorithms
- Does ease of coding and design outweigh inefficiency issues here?

---

---

---

---

---

---

---

---



## Recursion – array print

- Compare print array algorithms

---

---

---

---

---

---

---

---



## Nonrecursive - PrintArray

```

Sub PrintOneArray(ByRef myArray, ByRef label)
  Dim msg, index
  For index = 0 to UBound(myArray)
    ' loop to upper bound
    msg = msg & "Array(" & index & ") = " & _
          myArray(index) & vbCrLf
    ' get array values
  Next
  MsgBox msg, ,label
  ' display message box
End Sub

```

---

---

---

---

---

---

---

---



## Nonrecursive - PrintArray

```

Message(1)      Message(1)      Message(1)
                Message(2)      Message(2)
                Message(3)      Message(3)

                Message(1)
                Message(2)
                Message(3)
                Message(4)

```

Suggests recursion.  
At each step, you add the nth message to the (n-1) ones that have been added previously.

---

---

---

---

---

---

---

---



## Recursion – print array

```

Function RecursivePrintArray(ByRef anArray, ByVal anIndex)
  RecursivePrintArray = ""
  If (anIndex <= UBound(anArray)) Then recursive case
    Dim msg
    msg = "Index[" & anIndex & "] = " & anArray(anIndex) & vbCrLf
    anIndex = anIndex + 1
    RecursivePrintArray = msg & RecursivePrintArray(anArray, anIndex)
  End If base case falls through the If block
End Function

```

---

---

---

---

---

---

---

---



## Recursion – Array Print

- Compare algorithms
- Does ease of coding and design outweigh inefficiency issues here?

---

---

---

---

---

---

---

---



## FindAllFiles.vbs

- Goal is to check every file beneath a chosen directory for particular file extension.
- Given a starting directory:

```
Set BeginFolder =
  FS.GetFolder(BeginFolderName)
CheckFolder FS, BeginFolder, ExtType,
  FoundResults
```

---

---

---

---

---

---

---

---



## FindAllFiles.vbs

- Each folder contains both files and subfolders.

```
Sub GetAllFilesAndSubFolders(ByRef thisFolder, ByRef
  allFiles, ByRef allSubFolders)
  '
  Set allSubFolders = thisFolder.SubFolders
  '
  Set allFiles = thisFolder.Files
  '
End Sub
```

---

---

---

---

---

---

---

---



## FindAllFiles.vbs

- The files can be searched directly.
- The subfolders can be searched for files and further subfolders (recursive) folder contains both files and subfolders.

---

---

---

---

---

---

---

---



## FindAllFiles.vbs

```
Sub CheckFolder(ByRef oFS, ByVal myFolder, ByVal extType, ByRef foundNames)
  Dim mySubFolders
  Dim myFiles

  GetAllFilesAndSubFolders myFolder, myFiles, mySubFolders
  ' passing in myFolder
  ' results comes back in: myFiles and mySubFolders

  Dim subFolderObj
  for each subFolderObj in mySubFolders
    '
    CheckFolder oFS, subFolderObj, extType, foundNames
  Next
```

---

---

---

---

---

---

---

---



## FindAllFiles.vbs

In each subfolder, after the recursive search returns, check the local files.  
Files are found from the bottom of the directory.

```
Dim subFolderObj
for each subFolderObj in mySubFolders
  MsgBox "Working with folder: " & subFolderObj.Name
  CheckFolder oFS, subFolderObj, extType, foundNames
Next

Dim fileObj
for each fileObj in myFiles
  if oFS.FileExists(fileObj.path) then
    CheckFileName oFS, fileObj, extType, foundNames
  end if
Next
```

---

---

---

---

---

---

---

---



## FindAllFiles.vbs

Examine the vbs file.

Does the ease of coding and design outweigh inefficiency issues?

---

---

---

---

---

---

---

---



## WorkWithMSOL.vbs

- Similar to other objects and object models we have worked with.
- References: Lomax Chapter 6
- On-line Help: [G](#)

---

---

---

---

---

---

---

---



## WorkWithMSOL.vbs

```
Dim WshShell, CurrentDirectory
Set WshShell = WScript.CreateObject("WScript.Shell")
' For getting our operating environment (e.g.
  Current Working Folder)
CurrentDirectory = WshShell.CurrentDirectory
```

```
Dim OLApp
Set OLApp = CreateObject("Outlook.Application")
' This, we have seen many times.
```

---

---

---

---

---

---

---

---



## WorkWithMSOL.vbs

```
Dim OLNS
' name space of Outlook
Set OLNS =
OLApp.GetNamespace("MAPI")
```

- See Page 115 of Lomax
- NameSpace is a top-level object under application.
- The folders we wish to work with exist beneath NameSpace.
- GetNamespace is method within Application.
- MAPI is the only such space used by Outlook

---

---

---

---

---

---

---

---

---

---



## PrintFolderNames OLNS

```
set allFolders = oINS.Folders 'collection object of all folders

msg = "Current User is: " & oINS.CurrentUserName (Table 6-11, Lomax)
msg = msg + "Total Number of Folders:" & allFolders.Count & vbCrLf
count = 1
For Each myFolder in allFolders
  allFolderNames = allFolderNames & " [" & count & "]:" &
myFolder.Name & vbCrLf
  count = count + 1
Next
msg = msg & allFolderNames
```

---

---

---

---

---

---

---

---

---

---



## Function OpenPSTFile

Function OpenPSTFile(ByRef oINS, ByVal dir, ByRef folderName)

```
Set allFolders = oINS.Folders 'all folders collection object

Do While (found = False) And (count < allFolders.Count)
  Set myFolder = allFolders.Item(count) (see object model)
  oINS.RemoveStore myFolder (removes myFolder from Outlook
  directory)
  oINS.AddStore pstFileName (adds folderName.pst to Outlook
  directory)
Set OpenPSTFile = allFolders.Item(folderName) ' opens it
```

---

---

---

---

---

---

---

---

---

---



## Sub PrintFolderContent

Sub PrintFolderContent(ByRef aFolder)

```

Set allItems = aFolder.Items (collection object of messages)

Set allSubFolders = aFolder.Folders
' all the sub folders (MAPI Folders)

msg = "Folder: [" & aFolder.Name & "] has: " & vbCrLf
msg = msg & allItems.Count & " email messages and " & vbCrLf
msg = msg & allSubFolders.Count & " sub folders"

Call DisplayMessageInfo (aFolder.Name, allItems)

For Each subfolder In allSubFolders
Call DisplayMessageInfo (subfolder.Name, subfolder.Items)

```

---

---

---

---

---

---

---

---

---

---



## Sub DisplayMessageInfo

Sub DisplayMessageInfo (ByVal folderName, ByRef allMailItems)

```

Set allItems = allMailItems

shouldDisplay = MsgBox("There are " & allMailItems.Count & _
" items in Folder or subfolder " & folderName & _
& vbCrLf & " Do you wish to display them? ", vbYesNo)

If (shouldDisplay = vbYes) Then
For each aMail in allItems (in a folder)
senderInfo = senderInfo & "Sender:[" & aMail.SenderName & _
& "] SentDate:[" & aMail.ReceivedTime & "]" & _
& "Subject: " & aMail.Subject & vbCrLf
(see on-line for description of the item that is a mail message
Look at the mailItem... This is the item for this type of item object.

```

---

---

---

---

---

---

---

---

---

---



## ParseOLEmails.vbs

Sub ParseFolderContent(ByRef aFolder)

```

Set allItems = aFolder.Items
MsgBox "There are: " & allItems.Count & " emails in this folder."

For each aMail in allItems
senderInfo = aMail.SenderName
receiveTime = aMail.ReceivedTime
msg = "Sender is: [" & senderInfo & "]" & " eMailAddr=[" &
aMail.SenderEmailAddress & "]" & vbCrLf & "Email Says Receive Time is:
[" & receiveTime & "]" & vbCrLf
msg = msg + " Year Send is : " & Year(receiveTime) & vbCrLf
msg = msg + " Month Send is : " & Month(receiveTime) & vbCrLf
msg = msg + " Day Send is : " & Day(receiveTime) & vbCrLf
msg = msg + " Hour Send is : " & Hour(receiveTime) & vbCrLf
msg = msg + " Minutes is : " & Minute(receiveTime) & vbCrLf
msg = msg + " Second is : " & Second(receiveTime) & vbCrLf

```

---

---

---

---

---

---

---

---

---

---



## WritingWithWord.vbs

```

Dim WdAppl
Set WdAppl = CreateObject("Word.Application")

WdAppl.Application.Visible = True
'Add new document

Dim newWdDoc
Set newWdDoc = wdAppl.Documents.Add the collection and
' sets it as the active document

```

---

---

---

---

---

---

---

---



## WritingWithWord.vbs

```

Set sel = wdAppl.Application.Selection
' We now have available the set of methods and properties for
selection objects
With sel
.Style = newWdDoc.Styles("Heading 1") 'picks one of the
Styles collection styles belongs to the active document
.TypeText "Heading of Document" ' sample heading line
.TypeParagraph ' new line equivalent to ENTER
.TypeParagraph
End With

```

---

---

---

---

---

---

---

---



## WritingWithWord.vbs

```

With sel
.Style = newWdDoc.Styles("Normal")
.TypeText "This is the first line of the body."
.TypeParagraph
.TypeText "This is the second line - it is a long line." _
& "It is continued sufficiently to be run over at least two " _
& "lines of output text in the Word document. That is, this " _
& " should have exceeded one line."
.TypeParagraph
End With
' Save this file with a proper file name
newWdDoc.SaveAs("NewTextFile")

```

---

---

---

---

---

---

---

---