【ExcelVBAエキスパート】書籍学習用

ノートブック:	220_調査/ナレッジ		
作成:	2015/12/26 19:18	**更新:**	2016/02/14 16:31
作成者:	Tsutsui tomoaki		
URL:	http://www.asahi-net.or.jp/~ef2o-inue/vba_k/sub04_050_06.html		

タイトル： ExcelVBAエキスパート

chapter 01 【VBAの基礎】

01.VBAの基本構文

02.変数

①Dim
宣言したプロシージャ内でしか使用できない
②Private
宣言したモジュール内のすべてのプロシージャで使用可能
③Public
すべてのモジュールで使用可能

03.セルの操作

◆セルをコピーするときはRangeオブジェクトのcopyメソッドを使用します
コピー元.Copy Destination:=コピー先
例)
```
Range("A1").Copy Destination:=Range("B3")
```
※Destinationは省略可能
```
Range("A1").Copy Range("B3")
```

04.ステートメント

Ifステートメント
【書式①】
```
If 条件 Then 処理1
```
【書式②】
```
If 条件 Then
   処理1
End If
```
【書式③】
```
If 条件 Then
   処理1
Else
   処理2
End If
```
【書式④】
```
If 条件 Then
   処理1
ElseIf 条件2 Then
   処理2
End If
```

Forステートメント
```
For 変数 = 初期値 To 終了値
   処理
Next 変数(任意)

For i = 1 To 5 Step 2
   処理
Next i

For i = 3 To 1 Step -1
   処理
Next i
```

05.シートとブックの操作

◆新しいワークシートを挿入するにはWorksheetsコレクションのAddメソッドを使用します。

○名前設定
```
Worksheets("Sheet1").Name = "合計"
```
○削除
```
Worksheets("Sheet1").Delete
```

※削除時の確認メッセージを表示させない方法
```
Application.DisplayAlerts = False
Worksheets("Sheet4").Delete
Application.DisplayAlerts = True
```

○コピー(シート跨ぎ)
```
Worksheets("Sheet1").Range("A1").Copy Worksheets("Sheet2").Range("B3")
```

○保存
```
Worksheets("Book.xlsx").Save
ActiveWorkbook.SaveAs "C:¥Sample1.xlsx"
```

○閉じる
```
ActiveWorkbook.Close
```

○アクティブではないワークブックを操作するとき
Range("A1") = Workbooks("Book2.xlsx").Worksheets("Sheet2").Range("B3")

```
    'sheet1の内容をsheet2にコピー
    'Worksheets("Sheet1").Range("A1").Copy Worksheets("Sheet2").Range("A1")
    'sheet2の内容をsheet1にコピー
    Worksheets("Sheet1").Range("A1") = Worksheets("Sheet2").Range("A1")
```

06.デバッグ

◆イミディエイトウィンドウを使用する
値を返す命令を実行するときは「?」を付ける
? Workbooks(1).Name

[F9キー]
ブレークポイントの設定
[F8キー]
ステップインの実行

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 02 【Visual Basic Editor(VBE)の操作】

01.イミディエイトウインドウ

[ctrl + G]
イミディエイトウィンドウ起動
◆特徴
・常に暗黙の変数宣言が許されている

・Debug.Print
任意のデータをイミディエイトウィンドウに出力することができる

02.ウォッチウィンドウ

・変数の値の変化をウォッチできる

03.呼び出し履歴

・[表示] → [呼び出し履歴の表示]

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 03 【プロシージャ】

01.他のプロシージャを呼び出す

◆Call命令
他のプロシージャを呼び出すときはCall命令を使用します。
Callは使用しなくても大丈夫ですが、ほかのプロシージャを呼び出していることを明確にするためにも、宣言しておいたほうが良い。
基本的には
Subプロシージャを呼ぶ時　　宣言
Functionプロシージャを呼ぶ時　　省略 で書く！！

02.Functionプロシージャ

◆特徴
・実行結果を呼び出し元に返す働きをする
Function プロシージャ名(受け取る値) As 返すデータ型
プロシージャ名の変数にデータを格納して返却する

03.プロシージャの引数

◆引数の設定
Subプロシージャ もFunctionプロシージャも引数を設定することができる
```
Sub Sample()
    Call SampleX("Excel VBA")
End Sub

Sub SampleX(msg As String)
   MsgBox msg
End Sub
```

◆値渡しと参照渡し
・値渡し
```
Sub SampleX(ByVal msg As String)
   MsgBox msg
End Sub
```

・参照渡し
```
Sub SampleX(ByRef msg As String)
   MsgBox msg
End Sub
```
※何も指定をしなかった場合はByRefが指定されたものとみなされ、参照渡しが成立する。

◆省略可能な引数を定義する
・引数の前に「Optional」というキーワードを定義する
```
Sub Sample(msg As String, Optional n As Long)
     Dim***
```

◆初期値を設定しておく
```
Sub Sample(msg As String, Optional n As Long = 3)
```
引数が省略された場合、初期値が有効になります。
初期値を設定せずに流すと値は何が入るか？
```
Sub main()

    Sample1 ("型検証START!!")

End Sub


Sub Sample1(msg As String, _
            Optional b As Boolean, _
            Optional str As String, _
            Optional inte As Integer, _
            Optional lon As Long, _
            Optional sin As Single, _
            Optional dou As Double, _
            Optional cur As Currency, _
            Optional byt As Byte, _
            Optional dat As Date, _
            Optional var As Variant _
            )

    Debug.Print "#########: " & msg
    Debug.Print "Boolean  : " & b
    Debug.Print "String   : " & str
    Debug.Print "Integer  : " & inte
    Debug.Print "Long     : " & lon
    Debug.Print "Single   : " & sin
    Debug.Print "Double   : " & dou
    Debug.Print "Currency : " & cur
    Debug.Print "Byte     : " & byt
    Debug.Print "Date     : " & dat
    On Error Resume Next
    Debug.Print "Variant  : " & var
    On Error GoTo 0
    Debug.Print "#########: 型検証END"

End Sub
```
―――――――――――――――　実行結果　―――――――――――――――
```
#########: 型検証START!!
Boolean  : False
String   :
Integer  : 0
Long     : 0
Single   : 0
Double   : 0
Currency : 0
Byte     : 0
Date     : 0:00:00
#########: 型検証END
```

◆省略されたことを調べる。
・IsMissing関数で判定
・IsMissing関数で判定する引数はバリアント型でないといけない

```
Sub Sample(msg As String, Optional n)
     Dim i As Long, buf As String
     If IsMissing(n) = True Then n = 3
```

```
## データ型の説明 ##
'*************************************************************************
'    データ型の説明
'
'    作成者:井上治　URL:http://www.ne.jp/asahi/excel/inoue/ [Excelでお仕事!]
'*************************************************************************
Option Explicit

' ※プロシージャレベルで説明しますが、モジュールレベルでも同じです。
'    これは一般的によく使うデータ型です。
Sub TEST()
    ' 文字列型
```

```
    Dim A As String       ' 0～2GB
    Dim B As String * 10 ' 10文字に限定

    ' 整数型
    Dim C As Integer  ' -32,768～32,767の範囲
    Dim D As Long     ' -2,147,483,648～2,147,483,647の範囲

    ' 実数型
    Dim E As Single   ' -3.402823E38～-1.401298E-45(負),
                      ' 1.401298E-45～3.402823E38(正)の範囲
    Dim F As Double   ' -1.79769313486232E308～-4.94065645841247E-324(負),
                      ' 4.94065645841247E-324～1.79769313486232E308(正)の範囲
    ' 通貨型
    Dim G As Currency ' -922,337,203,685,477.5808～922,337,203,685,477.5807

    ' バイト型
    Dim H As Byte     ' 0～255の範囲

    ' ブール型
    Dim I As Boolean  ' 真(True)又は偽(False)

    ' 日付型
    Dim J As Date     ' 西暦100年1月1日～西暦9999年12月31日

    ' バリアント型
    Dim K As Variant  ' データ型を明示しない時と同じ

End Sub
```

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 04 【変数と配列】

01.配列とは

・複数の値を同時に格納できる変数の事

・配列の宣言
Dim ＜配列変数名＞(要素の下限(最小値) To 要素の上限(最大値))
要素の下限(最も小さい部屋番号)は省略することも可能です。要素の下限を省略すると、「0」を指定したものとみなされます。

02.動的配列

・動的配列の宣言
Dim Member() As String
・マクロ内で配列要素を指定する場合は ReDim という命令を使用します。
ReDim Member(3)
・ReDimで要素数を変更すると、それまで格納されていた値が消えます
・既存の値を消さないで、要素数を変更するには、ReDim命令にPreserveというキーワードを付けます
ReDim Preserve Member(4)

redimで型変更可能？

```
Sub ReDimMain()

    Dim member() As String
    ReDim member(3) As Integer    ＃ここでエラーになる！！　配列要素の型を変えることはできません

    member(0) = ""

End Sub

Sub ReDimMain()
    Dim member() As Integer
    member(0) = ""                #配列要素の型が一致しません。 String配列に数値を入れることは可能
End Sub

Sub ReDimMain()
    Dim member() As String
    'ReDim member(3) As Integer
    member(0) = 0                 #配列の要素数定義前に値を格納しようとすると、インデックスが有効範囲にありません。
End Sub
```

03.静的変数

◆静的変数の宣言

・Dimではなく「Static」という命令を使用
```
Static Number As Long
......
```
※静的変数は、プロシージャの中で宣言しなければなりません。
モジュールレベル変数　や　パブリック変数を静的変数として宣言することはできません。

static変数が初期化されるタイミングを調査、確認
・Excelブックを閉じると初期化される。。。
・新しいプロシージャを作成すると初期化される。VBAが再コンパイルされるので！初期化されることを確認済み！！
・Endステートメントを呼び出して処理を終了すると初期化される！！
※Excelではセルで値を保持できるのでStatic変数を使用する場面はあまりないのではないかという意見。
サンプルソース
```
Option Explicit

Sub staticMain()

    Static st As Long
    Dim i As Long

    st = st + 1
    Debug.Print "OUTFOR   : " & st

    For i = 1 To 10
        Static forSt As Long
        forSt = forSt + 1
        Debug.Print " INFOR   : " & forSt
    Next

    If MsgBox("static値をクリアしますか？", vbYesNo) = vbYes Then
        End
    End If

End Sub
```

04.オブジェクト変数

【オブジェクト型変数の宣言例】
```
Dim buf As Range
Dim buf As Worksheet
Dim buf As Workbook
```

・As Object 宣言
すべてのオブジェクトを格納できる
また、バリアント型の変数は、なんでも入る万能の型ですから、「As Object」で宣言した変数と同じようにすべてのオブジェクトを格納することが可能です。

Object型とValiant型の違いについて纏める
Object型　：オブジェクトの基底型。値を入れずに処理するとエラー
Valiant型　：格納された値に応じて柔軟に対応してくれる特殊な型。メモリの消費量は大きい。値を入れずに処理するとエラー。
・オブジェクトの格納
Set 変数名 = オブジェクト
オブジェクトを格納する際は、Set変数を使用します。
```
Dim buf As Range
Set buf = Range("A1")
```

例)
```
Sub Sample8()
     Dim buf As Range
     Set buf = Range("A1")
     buf.Font.ColorIndex = 3
End Sub
```

※オブジェクト変数を破棄する
オブジェクト変数をメモリ上から削除する為にも、使い終わったオブジェクト変数を明示的に破棄することができます
```
Set buf = Nothing
```

◆便利な使いどころ！！
・ブックを開いたり、新しいワークシートを挿入するなど、マクロ中に新しいオブジェクトが登場するようになった場合、オブジェクト変数を使用すると便利なケースがある。
例)
```
Sub Sample9()
     Dim WS1 As Worksheet, WS2 As Worksheet
     Set WS1 = ActiveSheet
     Set WS2 = Worksheets.add
     WS1.Activate
     WS2.Name = "合計"
End Sub
```

05.ユーザー定義型

◆ユーザー定義型とは
複数の異なる型をセットにして、ユーザーが独自の型を定義するもの
ユーザー定義型は宣言セクションで定義する
```
Type ProfileData
     Address As String
     Age As Long
End Type

Sub Sample10()
     Dim Member As ProfileData
     Member.Address = "東京"
     Member.Age = 25
     MsgBox Member.Address & vbCrLf & Member.Age
End Sub
```

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 05 【イベント】

イベント表形式に纏めたい
>>excel イベント纏め

01.イベントとは

◆イベントとは何か
Excel上で特定の操作が行われたとき、その操作が行われたことをExcelからVBAに通知する仕組み
◆イベント
ワークシートのイベント
Worksheetオブジェクト　Changeイベント

◆イベントの発生抑止方法
ApplicaitonオブジェクトのEnableEventsプロパティにFalseを設定
～イベント内～
Applicaiton.EnableEvents = False
＊＊処理＊＊
Application.EnableEvents = True
～イベント内～

※削除時の確認メッセージを表示させない方法
Application.DisplayAlerts = False

02.ブックのイベント

◆イベント
・WorkbookオブジェクトのBeforeCloseイベント
引数のCancelにtrueを入れることでその後のイベント処理をキャンセルできる

・WorkbookオブジェクトのBeforePrintイベント
・WorkbookオブジェクトのBeforeSaveイベント
・WorkbookオブジェクトのNewSheetイベント
新しく作成されたシートは引数shに格納されているので、
そのNameプロパティを編集
・WorkbookオブジェクトのOpenイベント

```
Private Sub Workbook_Open()
Dim i As Long
For i = 1 To Worksheets.Count
     If Sheets(i) .Name <> "Sheet1" Then
          Sheets(i).Visible = False
     End If
End Sub
```

03.シートのイベント

◆イベント
・Activate
・BeforeDoubleClick
・BeforeRightClick
・Change
・Deactive
ワークシートがアクティブでなくなった時に発生します。
・SelectionChange
ワークシート上で選択されているセルが変更されたときに発生します。

※すべてのワークシートで同じイベントを利用するには
Workbook_Sheet○○イベントを使用します！

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 06 【ステートメント】

01.Select Caseステートメント

◆書式
```
Select Case 値
     Case 条件１
          処理１
     Case 条件２
          処理２
     Case 条件３
          処理３
     [Case Else]
          [その他の処理]
End Select
```

【1または3または5】
```
Case 1, 3, 5
```

【10から20】
```
Case 10 To 20
```

・Caseの条件で値に指定した評価対象を比較するときはＩｓというキーワードを使用します。
```
Select Case Range("A1").Value
     Case Is < 10
          処理１
     Case Is >= 20
          処理２
End Select
```

・Format関数
```
Format(元の値, 書式)
Format(日付, "aaa")
```
曜日を返す　月　とか、、、

日付として認識する書き方を纏める！
http://officetanaka.net/excel/vba/tips/tips110.htm

・IsDate関数
```
IsDate(日付)
```
引数が日付であれば、Trueを返却する
```
2010/3/31
2010-3-31
? IsDate("2010/3-1")
? IsDate("2010 3 1")
? IsDate("2010-3/1")
```
これらも日付として扱われるので、
Trueを返す

```
##  サンプル  ##
Sub Sample()
    Dim Tmp As String
    Tmp = "2013-3-31"
    If IsDate(Tmp) Then
        MsgBox Tmp
    Else
        MsgBox False
    End If

End Sub
##  実行結果  ##
2013-3-31
```

02.Do...Loopステートメント

【繰り返しの前で条件を判定する】
```
Do 条件
     処理
Loop
```

【繰り返しの後で条件を判定する】
```
Do
     処理
Loop 条件
```

【条件が正しい場合のLoop処理】
```
Do While 条件
     処理
Loop
```

【条件が誤りの場合のLoop処理】
```
Do Until 条件
     処理
Loop
```

03.For Each...Nextステートメント

◆書式
```
For Each 変数名 In グループ名
     変数を使った操作
Next 変数名

Dim ws As Worksheet
For Each ws In Worksheets
     MsgBox ws.Name
Next ws
```

どんな変数が入るかわからないときはバリアント型で定義しておく！！
```
Dim ws [As Valiant]
For Each ws In Worksheets
     MsgBox ws.Name
Next ws
```
変数宣言は省略してもよい

・グループに配列を指定して、配列の中に格納されている値を１つずつ取り出すときは、
配列に格納されているデータ種類によらず、バリアント型の変数を指定します。

◆選択範囲の書式を変更するサンプル
```
Sub Sample4()
     Dim c
     For Each c In Selection
```

```
            c.Font.ColorIndex = 3
        Next c
End Sub
```

04.Exitステートメント

≫処理を中止するステートメント
◆種類
・Exit Sub
・Exit Function
・Exit For
・Exit Do

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 07 【関数】

01.配列を操作する関数

○LBound関数, UBound関数
【書式】
LBound(配列)　:　引数に指定した配列で使用できる最も小さなインデックス番号を返します
UBound(配列)　:　引数に指定した配列で使用できる最も大きなインデックス番号を返します
配列の大きさが決まっていないケースで便利

○Split関数
文字列データを区切り文字で区切り、各要素を配列形式で返します。
Split(文字列, 区切り文字)

```
Dim tmp As Valiant, i As Long
tmp = Split("tanaka/suzuki/yamada", "/")
For i = 0 To UBound(tmp)
    MsgBox tmp(i)
Next i
```

空文字入りの文字列をsplitした場合は？
空文字配列が生成される。エラーにはならない

```
Sub sp()
    Dim tmp As Variant
    Dim i As Long
    tmp = Split("tanaka/suzuki/yamada//////////////", "/")
    For i = 0 To UBound(tmp)
        Debug.Print tmp(i)
    Next i
End Sub
```

Splitの結果が、いくつの配列になるのかわからないので、Valiant型で宣言する(もしくは型を決めない)

02.データを判定する関数

○IsArray関数、IsDate関数、IsNumeric関数
IsArray関数　　：引数が配列の時にTrue
IsDate関数　　：引数が日付の時にTrue
IsNumeric関数　：引数が数値の時にTrue

03.日付を操作する関数

○DateSerial関数
引数で指定した年月日に該当する日付データ(シリアル値)を返します。

```
buf = InputBox("判定する年は？")
if Day(DateSerial(buf, 3, 1) - 1) = 29 Then
    MsgBox "閏年です"
Else
    MsgBox "平年です"
End if
```

引数の種類を調べる！年月日だけがどうか確認
構文
DateSerial(year,month,day)
引数yearは、年を表す0～9999の範囲の数値または数式を指定します。
引数monthは、月を表す1～12の範囲の数値または数式を指定します。
引数dayは、日を表す1～31の範囲の数値または数式を指定します。

解説
DateSerial関数は、3つの引数で指定された日付を意味するシリアル値を返します。
引数yearに0～29を指定すると、2000年～2029年と読み替えられます。また、30～99を指定すると、1939年～1999年と読み替えられます。

サンプル

次の例は、ユーザーが入力した年・月・日から日付シリアル値を生成し、和暦で表示します。

```
Sub Sample()
    Dim myYear As Integer, myMonth As Integer, myDay As Integer
    Dim myDate As Date
    myYear = InputBox("年を入力してください")
    myMonth = InputBox("月を入力してください")
    myDay = InputBox("日を入力してください")
    myDate = DateSerial(myYear, myMonth, myDay)
    MsgBox "入力された日付は、" & Format(myDate, "ggge年m月d日") & "です"
End Sub
```

```
? dateserial(2001,1,1)
2001/01/01
```

04. XXXXX

05. XXXXX

06. XXXXX

07. XXXXX

08. XXXXX

09. XXXXX

10. XXXXX

○ Coulmn:

chapter 08 【エラーへの対応】

01. エラー処理

～エラー種類～

1. 文法エラー
2. 論理エラー

○文法エラー
VBAの構文に違反した書式で命令を記述したときに発生します
○論理エラー
VBAの文法に誤りはないものの、論理的な矛盾によって発生するエラー
・コンパイルエラー
・実行時エラー

02. エラーへの対処

◆エラーが発生したら別の処理にジャンプする
・OnErrorステートメント
On Error GoTo ジャンプ先のラベル名

＊＊ここでエラーが発生すると＊＊
＊＊ラベルの箇所に遷移する　＊＊

ラベル名：

◆どんなエラーが発生したか調べる
・Errorオブジェクト
-Numberプロパティ　　：エラー番号
-Descriptionプロパティ　：エラーが発生した時に表示されるエラーメッセージ
-Clearメソッド　　：同一プロシージャ内で複数のOn Errorステートメントを使用するときには、Clearメソッドで情報をクリアしておく

エラーオブジェクトのメソッド、プロパティを調査する
基本的に使用するメソッド及びプロパティは上記③つ
http://officetanaka.net/excel/vba/tips/tips104.htm
http://excelvba.pc-users.net/fol6/6_8.html

オブジェクト名	プロパティ	内容
Err	Number	エラー番号
	Description	エラー番号に対応する文字列(エラー内容)
	Source	現在のプロジェクト名
	HelpFile	ヘルプファイル名
	HelpContext Number	ヘルプファイルに対応するコンテキスト番号

◆発生したエラーを無視する
・On Error Resume Next

```
Sub OnErrorTest()
    On Error GoTo ErrorTrap

    Dim i As Integer
```

```
    i = "test"              '←ココで構文エラーとなる。

    Debug.Print "終了します。"

    Exit Sub

ErrorTrap:
    Debug.Print "エラー番号       :" & Err.Number
    Debug.Print "エラー内容       :" & Err.Description
    Debug.Print "ヘルプファイル名 :" & Err.HelpContext
    Debug.Print "プロジェクト名   :" & Err.Source
    Resume Next
End Sub
```

```
## 実行結果 ##
エラー番号          :13
エラー内容          :型が一致しません。
ヘルプファイル名  :1000013
プロジェクト名     :VBAProject
終了します。
```

03.エラー対策のポイント

エラーが発生する前にチェックする処理を設ける

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 09 【UserForm】

01.UserFormとは

VBAで自由に設計できるダイアログボックス
下記の3つの作業が必要
・設計
・コーディング
・表示

○Showメソッドで表示
ユーザーフォーム名.Show
showメソッドの引数の種類を調べる

○Unload Me で閉じる
unloadの引数、オブジェクトについて調べる

```
Option Explicit


Sub 表示()
    UserForm1.Show vbModal
    UserForm1.Hide
    UserForm1.Show vbModeless
    Unload   UserForm1
End Sub
```

02.よく使うコントロール

○ラベル
○コマンドボタン
○テキストボックス
○リストボックス
○コンボボックス
○チェックボックス
○オプションボタン：オンとオフを切り替えるコントロール
○イメージ
Image1.Picture = LoadPicture("C:¥Work¥****")

コントロールのメソッド、プロパティについてまとめておく！

	ラベル	コマンドボタン	テキストボックス	リストボックス	コンボボックス	チェックボックス	オプションボタ
文字列表示	Caption	Caption	Text,Value		Text	Caption	Caption
複数行表示			MultiLine				
ｽｸﾛｰﾙﾊﾞｰ表示			ScrollBars				
ｱｲﾃﾑ追加				AddItem	AddItem		
何番目を選択するか				ListIndex			
選択されているデータ				Text			
リストに登録されているデータ件数				ListCount			
リストに登録されているデータ全体				List() 配列形式			
オンオフ						Value	Value
表示したい画像							
画像拡大縮小							
マクロ中画像を設定する							

03.UserFormのイベント

○Initializeイベント
ユーザーフォーム初期化時に発生するイベント
○QueryCloseイベント
ユーザーフォームを閉じようとしたときに発生するイベント
実行時の引数の値によって、ユーザーがどのうような方法で画面を閉じようとしているのかを判定する。

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
     if CloseMode <> 1 Then
          MsgBox "[ 閉じる ]ボタンをクリックしてください"
          Cancel = True
     End If
End Sub
```

数値	意味
0	ユーザーフォームの[×]ボタンがクリックされた
1	コードからUnloadステートメントが実行された
2	現在のWindowsのオペレーティングシステム環境のセッションが終了した
3	Windowsのタスクマネージャによってアプリケーションが閉じられた

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 10 【メニューの操作】

全体的に復習と、深堀！！

01.コンテキストメニューとは

右クリックメニュー
VBAの標準操作でコンテキストメニューも編集可能。

○CommandBarオブジェクト
コンテキストメニューを表すオブジェクト
CommandBarオブジェクトの集合がCommandBarsコレクション
Workbookオブジェクトと Workbooksコレクションの関係です。

・コンテキストメニューに登録されているコマンドの数を調べる

```
MsgBox CommandBars("Cell").Controls.Count
```

・コンテキストメニューから４番目に登録されているコマンドメニューを調べる

```
MsgBox CommandBars("Cell").Controls(4).Caption
```

02.コマンドを登録する

Controlsコレクションの Addメソッドを使用する
【書式】

```
Controls.Add.Type, Id, Parameter, Before, Temporary
```

・Type
msoControlButton ：クリックするとマクロが実行
msoControlPopup　：クリックするとサブメニュー表示
・Id、Parameter
Excelの組み込みコマンドを追加するときに使用します。本書は割愛
・Before　　：コマンドを挿入する位置を指定します。３を指定すると、３番目にあるコマンドの直前に新しいコマンドが追加されます。
・Temporary　　：コンテキストメニューでは使用しない

＊Addメソッドでコマンドを追加するが、追加しただけではからのコマンドで押してもなんの反応もしないので、
・Captionプロパティ　　：コマンドに表示する文字列
・OnActionプロパティ　　：実行するマクロ名
　の設定をする

書き方としては２つ存在する
○Addメソッドで追加したからのコマンドを変数に格納して、その変数に対して設定を行う方法。からのコマンドを格納する変数はObject型またはVariant型で宣言し、格納するときはSetステートメントを使います
```
Sub Sample()
     Dim NewCommand As Object[Variant]
     Set NewCommand = CommandBars("Cell").Controls.Add(Type:=msoControlButton)
     NewCommand.Caption = "処理１"
     NewCommand.OnAction = "myMacro"
End Sub
```
○変数に格納せず、Withステートメントを使用します。
```
Sub Sample()
     With CommandBars("Cell").Controls.Add(Type:=msoControlButton)
          .Caption = "処理１"
          .OnAction = "myMacro"
     End With
End Sub
```

◆区切り線をつける
BeginGroupプロパティをTrueにする
```
Sub Sample()
     With CommandBars("Cell").Controls.Add(Type:=msoControlButton)
          .Caption = "処理２"
          .OnAction = "myMacro"
          .BeginGroup = False
     End With
     With CommandBars("Cell").Controls.Add(Type:=msoControlButton)
          .Caption = "処理３"
          .OnAction = "myMacro"
          .BeginGroup = True
     End With
End Sub
```
Trueにしたコマンドの上に区切り線が挿入できる！！

◆追加する位置を指定する
コマンドが追加される位置を指定するには、Addメソッドの引数「Before」に数値を指定します。
```
     With CommandBars("Cell").Controls.Add(Type:=msoControlButton, Before:=3)
          .Caption = "処理4"
          .OnAction = "myMacro"
     End With
```
存在しないマクロをアクションに定義すると？

◆コマンドにアイコンを表示する
コマンドの左側に表示されるアイコンは、コマンドのFaceIdプロパティにアイコンの番号を指定します。
```
     With CommandBars("Cell").Controls.Add(Type:=msoControlButton)
          .Caption = "処理１"
          .OnAction = "myMacro"
          .FaceId = 23
     End With
```

◆コマンドにチェックマークを付ける
コマンドにチェックマークを付けるには、コマンドのStateプロパティにTrueを設定します
```
Sub Sample()
     With CommandBars("Cell").Controls.Add(Type:=msoControlButton)
          .Caption = "処理６"
          .OnAction = "Checked"
          .State = True
     End With
End Sub

Sub Checked()
     With CommandBars("Cell").Controls("処理６")
          .State = Not .State
     End With
End Sub
```

◆コマンドに説明を付ける
Tagプロパティを使用すると、コマンドに任意の説明を追加できます。
設定した文字列は画面には表示されず、特に決まった使い方もないので、ユーザーが任意に設定できます。
コマンドを削除するとき、Tagプロパティに独自の文字列を設定しておくことで安全に削除することができます。
```
     With CommandBars("Cell").Controls.Add(Type:=msoControlButton)
          .Caption = "処理１"
          .OnAction = "myMacro"
          .Tag = "VBAテスト"
     End With
```
コマンド削除時のループについて纏めておく！

03.コマンドを削除する

コンテキストメニューのコマンドを削除するには、ControlsコレクションのDeleteメソッドを使用します。
```
CommandBar("Cell").Controls(2).Delete
CommandBar("Cell").Controls("処理１").Delete
```

◆存在しないコマンドは削除できない
≫対処
・コマンドの存在確認を行う
```
For Each c In CommandBars("Cell").Controls
     If c.Caption = "処理１" Then
          flag = True
          Exit For
     End If
Next c
```
・コマンドが存在しない場合、削除を実行するとエラーになるが、エラーを無視して正常終了させる
```
On Error Resume Next
CommandBars("Cell").Controls("処理１").Delete
```

◆そのコマンドを削除してもいいか
Excelにデフォルトで組み込まれているコマンドを削除する可能性があるので、
自分で作成したコマンドである担保が必要
Tagプロパティに特定の文字列を記載しておき、その文字列がどうかで削除を決めることで解消

◆初期状態にリセット
CommandBarオブジェクトには、メニューを初期状態にリセットするResetメソッドがあります。
Resetメソッドを実行すると、ユーザーが追加したコマンドはすべて削除され、Excelをインストールした初めの状態になります。
```
Sub Sample()
     CommandBars("Cell").Reset
```

End Sub

CommandBars(" Cell ")controls.add
以外のコンテキストを確認しておきたい！！

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 11 【Windowsの機能を利用する】

01.OLEオートメーションとは

OLE(Object Liking and Embedding) アプリケーションから別のアプリケーションを操作したり、Word文書内にExcelの表を埋め込むなど、オブジェクトをやり取りするための規格

◆OLEオートメーションの使い方
・事前バインディング：利用したいオブジェクトをVBAの参照設定であらかじめ参照する

```
Sub Sample()
      Dim IE As New InternetExplore
      IE.Visible = True
      IE.Navigate "http://vbae.odyssey-com.co.jp/"
      MsgBox "ページを表示しました"
      IE.Quit
      Set IE = Nothing
End Sub
```

・実行時バインディング：CreateObject関数を使ってオブジェクトを参照する方法

```
Sub Sample()
      Dim IE As Object
      Set IE = CreateObject("InternetExplorer.Application")
      IE.Visible = True
      IE.Navigate "http://vbae.odyssey-com.co.jp/"
      MsgBox "ページを表示しました"
      IE.Quit
      Set IE = Nothing
End Sub
```

--CreateObject関数
CreateObject(クラス名)
→オブジェクトを作成します。

◆OLEオートメーションの注意
クラス名、メソッド、プロパティは調べて使用しなければならない。

◆事例１：InternetExplorerを操作する
クラス名　InternetExplorer.Application
Webページからテキストを読み込む

```
Sub Sample()
      Dim IE As Object
      Set IE = CreateObject("InternetExplorer.Application")
      IE.Visible = True
      IE.Navigate "http://vbae.odyssey-com.co.jp/"
      '####ページが完全に開かれるまで待っている処理####
      Do While IE.Busy = True
            DoEvents
      Loop
      Do While IE.Document.ReadyState <> "complete"
            DoEvents
      Loop
      '################################################
      MsgBox "ページを表示しました"
      buf = IE.Document.Body.InnerText
      IE.Quit
      MsgBox buf
      Set IE = Nothing
End Sub
```

◆Windows Script Hostを利用する
Windows Script Host(WSH)とは、テキスト形式で作成した「スクリプト」と呼ばれるプログラムをWuindows上で実行する仕組みです。WSHはWindowsに対して様々な命令を実行するいわば汎用的なエンジンに相当します。VBAからOLEオートメーションを使ってWSHを利用すると、VBAでは実現が難しい機能を実行できます
[WScript.Shell]

```
Sub Sample()
      Dim WSH As Object
      Set WSH  = CreateObject("WScript.Shell")
      WSH.Popup "5秒後、自動的に閉じます", 5, "テスト", vbInfomation
```

```
    Set WSH = Nothing
End Sub
```

主要なオートメーションについて纏めておく！

02.APIとは

◆API(Applicaiton Programing Interface)の概念
◆APIの使い方
どのライブラリのどの関数を使うかをあらかじめ宣言しなければなりません。
この宣言にはDeclareステートメントを使い、宣言は宣言セクションで行います。
ここでは例として、ミリ秒単位で時間を計測できるWindowsのAPIである「GetTickCount関数」を使用

```
Option Explicit

Declare Function GetTickCount Lib "Kernel32" () As Long
```

基本的にAPI宣言はインターネットの技術情報や、API宣言を検索できるツールなどからコピーして使用します。

◆事例１：ゴミ箱に移動する

主要なAPIについて纏めておく！

03.XXXXX

04.XXXXX

05.XXXXX

06.XXXXX

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 12 【レジストリの操作】

レジストリを操作できるメソッドについて纏めておく！！

01.レジストリの概念

02.操作できる場所

ExcelVBAの命令で操作できるのは、

HKEY_CURRENT_USER¥Software¥VB and VBA Program Settings

の配下に限定されます。

03.レジストリに登録する

レジストリにデータを登録するには、SaveSettingステートメントを使用します。

SaveSetting アプリケーション名, セクション名, キー名, データ

引数「アプリケーション名」には、一般的にプログラム名や企業名などを指定します。
引数「セクション名」はデータを保存するフォルダ名に該当します
引数「キー名」はファイル名に相当し、その中に記録されている内容が、
引数「データ」になります

```
Sub Sample()
    SaveSetting "MyMacro", "Main", "Data", "VBAエキスパート"
End Sub
```

04.レジストリを取得する

GetSetting関数：レジストリに登録されているデータを取得する。

GetSetting (アプリケーション名, セクション名, キー名, 規定値)

引数はSaveSetting関数と同じで、登録されているデータを取得します。
キー名が存在しない場合は、規定値に設定されたデータを返り値として返します。

◆セクションに登録されている複数のデータを一気に取得するときは、GetAllSettings関数を使用します。

GetAllSettings(アプリケーション名, セクション名)

引数「セクション名」で指定したセクションに登録されているすべてのキー名とデータを二次元配列で返します。

```
Sub Sample()
    Dim tmp As Variant, i As Long
    SaveSetting "MyMacro", "Sample", "Data1", "Excel"
    SaveSetting "MyMacro", "Sample", "Data2", "VBA"
    tmp = GetAllSettings("MyMacro", "Sample")
```

```
    For i = 0 To 1
        MsgBox tmp(i, 0) & vbCrLf & tmp(i, 1)
    Next i
End sub
```

05.レジストリを削除する

レジストリのデータを削除するにはDeleteSettingステートメントを使用します。

DeleteSetting アプリケーション名, セクション名, キー名

```
-- セクション「Sample」内のキー「Data1」を削除
DeleteSetting "MyMacro", "Sample", "Data1"
-- セクション「Sample」を削除
DeleteSetting "MyMacro", "Sample"
-- フォルダ「MyMacro」を削除
DeleteSetting "MyMacro"
```

06.レジストリを利用したマクロ例

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

chapter 13 【ファイルの操作】

01.テキストファイルの操作

◆ファイルを開くという意味
「開く」は、テキストファイルをExcel上に開くのではなく、テキストファイルに対して読み書きを行う許可をWindowsに承認してもらう手続きを指します。

◆テキストファイルを開く
Openステートメント
これからファイルに対して読み書きを行うという承認をWindowsに対して行う命令

Open ファイルパス For 操作モード As ファイル番号

```
Open "C:¥Sample.txt"
```

モード	操作
Append	ファイルに追記する
Binary	ファイルをバイナリモードで開く
Input	ファイルから読み込む
Output	ファイルに書き込む
Random	ファイルをランダムアクセスモードで開く。固定長データを使用するとき！

◆テキストファイルを閉じる
Closeステートメント

Close ファイル番号

```
Sub Sample()
    Open "C:¥Sample.txt" For Input As #1
    Close #1
End Sub
```

※ファイル番号
ファイルを操作する命令では、ファイルの番号を「#1」「#2」のように#を付けて指定します。

◆テキストファイルから１行読み込む

Line Input ファイル番号, 変数

```
EOF関数 ：ファイルの最後まで読み込んだかどうかの判定...
Sub Sample()
    Dim buf As String
    Open "C:¥Sample.txt" For Input As #1
        Do Until EOF(1)
            Line Input #1, buf
            MsgBox buf
        Loop
    Close #1
End Sub
```

◆ファイルに書き込む

Print ファイル番号, データ

◆ファイルをすべて読み込む
Getステートメント

Get #ファイル番号, 読み込み開始位置, データを格納する変数名

テキストファイルのデータを一括で取得するときには「読み込み開始位置」を省略します。
変数名のサイズと、ファイルのサイズを等しくしなければなりません。
Space関数とFileLen関数を用いて実現します。

```
Sub Sample()
```

```
    Dim tmp As String

    tmp = Space(FileLen("C:¥Sample.txt"))
    Open "C:¥Sample.txt" For Binary As #1
        Get #1,  , tmp
    Close #1
    MsgBox tmp
End Sub
```

テキストファイル以外をバイナリで読み込んだ時の挙動を確認！

02.ファイルのコピーと移動

FileCopyステートメントを使用します。

FileCopy コピー元ファイル, コピー先ファイル

移動するときはどうしたらいいのでしょう？
Nameステートメントを使用します。

Name 旧ファイル名, 新ファイル名

フォルダを変更して実行すると、ファイルの移動を実現できる。

03.ファイルの削除

Killステートメント

Kill ファイル名

ファイルが存在しないとエラーになる
削除するときに、
「削除しますか？」
等の確認メッセージは表示されずに完全に削除される。

04.ファイルの存在確認

Dir関数を使います。

Dir（ファイル名）

```
Sub Sample()
    if Dir("C:¥Book1.xlsx") = "" Then
        MsgBox "存在しません"
    Else
        MsgBox "存在します"
    End If
End Sub
```

Dir関数は返り値にファイル名を返すっぽい
正規表現が使用できる。アスタで複数指定
DIRでの存在確認方法は掴んでおく！

05.カレントフォルダ

現在のカレントフォルダを調べる方法
  CurDir関数
引数無しで記載すれば使用可能

```
  MsgBox CurDir
```

カレントフォルダを移動するには
ChDirステートメントを使用します

```
  ChDir パス

  ChDir "C:¥Sample"
```

カレントフォルダで移動できるのは現在のドライブのカレントフォルダなので、
ドライブを変更したいときは、ChDriveステートメントを使用しなければならない

```
  ChDrive ドライブ名

  ChDrive D
```

→Dドライブに変更

06.フォルダの操作

MkDirステートメント
新しいフォルダを作成する！

```
  MkDir パス


  MkDir "C:¥Sample"
```

※注意：MkDirステートメントで作成できるのは引数で指定したパスのうち最後のフォルダのみ！！

フォルダを削除
RmDirステートメントを使用します。
※存在しないフォルダを指定するとエラーになる。
RmDirステートメントで削除するフォルダにファイルが保存されていたり、サブフォルダが作成されているとエラーになる。

フォルダごと削除する方法があるかないかどうか調査！あればコマンドを記載する！！

07.XXXXX

08.XXXXX

09.XXXXX

10.XXXXX

○ Coulmn:

※その他テスト対策
・呼び出し履歴ダイアログボックスは、マクロが一時停止してデバッグモードになった時だけ表示できる