



VBA エキスパート 公式テキスト

「Excel VBA スタンダード」演習問題 解答例

1章 VBA の基礎

解答 1-1

:=

VBA のオブジェクト式のうち、メソッドには、動作の詳細や動作後の挙動などを指示するオプションを指定できるものがあります。こうしたオプションを引数（ひきすう）と呼び、引数の名称と、引数に設定する値は「:=」という記号で結びます。

解答 1-2

①Dim ②As

変数の宣言は「Dim 変数名 As 型」が基本です。「Dim A, B, C As String」というコードでは、文字列型 (String) になるのは変数「C」だけです。変数「A」と変数「B」は、型の指定が省略されたとみなされて、どちらもバリエーション型 (Variant) となります。なお、本問題では【 ① 】に Static を指定しても、マクロは正常に動作しますが、問題文で「静的変数『buf』」と明記していないので、①は「Dim」を正解とします。

解答 1-3

Public

Sub プロシージャ内で宣言した変数は、宣言したプロシージャ内でしか使用できません。同一モジュールにある複数のプロシージャで使用できる変数は、モジュールの宣言セクションで宣言します。このとき、「Private」で宣言した変数は、そのモジュール内の全プロシージャで使用できますが、別のモジュールでは使用できません。複数のモジュールがあったとき、異なるモジュールでも同じ変数を使用するには、宣言セクションで、「Public」を使って宣言します。

なお、「Global」は隠しステートメントのため、VBA エキスパートでは推奨しないので、本問題では不正解とします。

解答 1-4

①Step ②Cells

For...Next ステートメントは、指定した回数だけ処理を繰り返すステートメントです。「カウンタ変数 = 」に続けて「初期値 To 終了値」のように繰り返しの回数を指定します。このとき、終了値に続けて「Step 間隔」を指定すると、カウンタ変数の増加間隔を指定できます。

任意のセル範囲をすべて操作するような処理では、操作対象のセル範囲を For...Next ステートメントで指定することが多いです。そのとき、Cells(行, 列)で複数のセルを指定する処理が一般的です。②には、ほかの書き方でセル範囲を指定することも可能ですが、本問題では Cells を正解とします。

解答 1-5

①If ②Else ③End If

If ステートメントの書式は次の通りです。

【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 1 Then  
    処理 1  
ElseIf 条件 2  
    処理 2  
End If
```

解答 1-6

①Name ②Delete

ワークシートの名前は Name プロパティで操作します。ワークシートを削除するときは Delete メソッドを実行します。

解答 1-7

ActiveSheet

新しいワークシートを挿入すると、必ず、挿入したワークシートがアクティブシートになります。これは、新規ブックを挿入したときも同じです。

2章 Visual Basic Editor (VBE) の操作

解答 2-1

Stop

マクロを任意の場所で一時停止するには、ブレークポイントを設定したり、ウォッチウィンドウを使う方法などがあります。また、Stop ステートメントを実行すると、マクロは一時停止してデバッグモードになります。Stop ステートメントを使うと、任意の条件でマクロを停止することができるので便利です。

解答 2-2

?Worksheets.Count

イミディエイトウィンドウで、プロパティの値を調べるときは、先頭に「?」を入力してから調べたいプロパティを入力します。イミディエイトウィンドウの中では、マクロを作成するときと同じ VBA の構文を使用できますので、ブックを指定しない「Worksheets.Count」はアクティブブックが対象となります。もちろん「?ActiveWorkbook.Worksheets.Count」でも正解です。また、イミディエイトウィンドウ内では、VBA の命令語を入力しても、自動的に大文字には変換されません。すべて小文字で入力してもかまいません。

解答 2-3

①Debug ②Print

イミディエイトウィンドウは Debug オブジェクトで操作できます。イミディエイトウィンドウに出力するには、Print メソッドを使います。「Debug.Print」以外で Debug オブジェクトを操作することはほとんどありませんので、「Debug.Print」だけをしっかり覚えてください。

3章 プロシージャ

解答 3-1

Call

プロシージャから、別のプロシージャを呼び出す（コールする）には、Call ステートメントに続けてプロシージャ名を記述します。Call ステートメントは省略することも可能ですが、別のプロシージャを呼び出していることを明示的にするためにも、省略しないことを推奨します。

解答 3-2

Sample4

Function プロシージャは、何らかの値を返すプロシージャです。Function プロシージャ内で値を返すときは、プロシージャ名に値を代入します。別の Sub プロシージャを呼び出すときは Call ステートメントを使いますが、Function プロシージャを呼び出すときには、Function プロシージャの返す値を変数に代入するなど、何らかの処理を行いますので、一般的に Call ステートメントをしません。

解答 3-3

東京

別のプロシージャに値を渡すときは、値を受け取る側のプロシージャで引数を定義します。そのとき「ByVal キーワード」を付けて定義した引数には、値だけが渡されます。本問題では、変数「buf」に格納した「東京」という文字列だけが渡されますので、「Sub Sample6」で受け取った引数を変更しても、もとの変数「buf」に影響はありません。

対して、受け取る引数に「ByRef キーワード」を付けて定義すると、引数への参照が渡されます。「Sub Sample6」の引数「msg」は、「Sub Sample5」で引数として渡した変数「buf」を参照している状態ですので、「Sub Sample6」側で引数を変更すると、もとの変数「buf」も変化します。引数の定義に「ByVal キーワード」「ByRef キーワード」のいずれも指定しなかったときは、「ByRef キーワード」を指定したものとみなされます。

解答 3-4

Optional

省略可能な引数を定義するときは、引数名の前に「Optional キーワード」を指定します。複数の引数を定義するとき、Optional キーワードを指定した引数より後ろの引数も、Optional キーワードをつけて省略可能にしなければなりません。たとえば、A,B,C という3つの引数を受け取る次のようなプロシージャで、

```
Sub Proc(A, B, C)
```

引数 B に Optional キーワードをつけて省略可能にする場合、後ろの引数 C にも Optional キーワードをつけて省略可能にしなければなりません。

```
Sub Proc(A, Optional B, Optional C)
```

また、プロシージャに渡す引数の数が決まっていない場合は、引数に「ParamArray キーワード」をつけて定義します。たとえば、ワークシート上で使用する SUM 関数では「SUM(A1, B1)」や「SUM(A1, B1, C1, D1)」のように指定する引数の数が可変です。このように、可変数の引数を受け取る場合は、引数の定義に ParamArray キーワードをつけて、動的配列として定義します。

```
Sub Proc(ParamArray myData())
```

「Sub Proc(ParamArray myData(3))」のように、要素を指定した配列に ParamArray キーワードをつけることはできません。また、ParamArray キーワードをつけて可変にする引数は、バリエーション型 (Variant) でなければなりません。

次の「Sample9」は、可変数の引数を受け取ります。

```
Sub Sample8()  
    Call Sample9(1, 2, 3)  
End Sub  
  
Sub Sample9(ParamArray myData())  
    Dim d, Ans As Long  
    For Each d In myData  
        Ans = Ans + d  
    Next d  
    MsgBox "合計は" & Ans & "です"  
End Sub
```

受け取った配列データの各要素を取り出すには、「Sample9」のように For Each...Next ステートメントを使うか、次のように For...Next ステートメントを使います。

```
Sub Sample10()  
    Call Sample11(26, 18, 35)  
End Sub  
  
Sub Sample11(ParamArray myData())  
    Dim i As Long, Ans As Long  
    For i = 0 To UBound(myData)  
        Ans = Ans + myData(i)  
    Next i  
    MsgBox "合計は" & Ans & "です"  
End Sub
```

UBound 関数は、配列に格納されているデータのうち、もっとも大きなインデックス番号を返します。VBA の一般的な配列は、インデックス番号が 0 から始まりますので、

26, 18, 35

という 3 つのデータを渡したとき、配列の要素は

myData(0) → 26

myData(1) → 18

myData(2) → 35

となります。

4 章 変数と配列

解答 4-1

1

配列の宣言は

Dim <配列変数名> (要素の下限(最小値) To 要素の上限(最大値))

とします。Dim Member (2 To 5) と宣言した配列「Member」は、要素の下限が 2、要素の上限が 5 ですから「Member (2)」から「Member (5)」まで 4 つの要素を使用できます。配列の宣言時には「要素の下限」を省略することができます。

本問題の「Dim Member (3)」は、要素の下限が省略されて、要素の上限として 3 が指定されています。標準の VBA では、配列の要素は 0 から始まりますので、「Dim Member (3)」は「Dim Member (0 To 3)」と同じ指定になります。したがって「Dim Member (3)」で使用できるのは、「Member (0)」から「Member (3)」までの 4 つの要素になります。

解答 4-2

ReDim

配列の宣言時に要素数を指定せず、マクロのコード中で要素数を再定義できるような配列を動的配列と呼びます。動的配列は、宣言時に要素数を指定せず、配列名の後ろに「() (空の括弧)」を付けます。動的配列として宣言された配列の要素を再定義するときは、ReDim ステートメントを使用します。配列の宣言時に要素数を指定した配列は、ReDim ステートメントで要素数を再定義することはできません。

解答 4-3

空欄（何も表示されない）

ReDim ステートメントで動的配列の要素を再定義すると、それまで配列に格納されていたすべてのデータはクリアされてしまいます。本問題では、3 行目で「Member (1)」に「秋山」を格納していますが、次行の「ReDim Member (2)」を実行すると、それまで配列内に格納されていたデータがクリアされ、5 行目で格納した文字列だけが残ります。したがって、「Member (1)」はクリアされて空なので、メッセージボックスには何も表示されません。

解答 4-4

Preserve

ReDim ステートメントを使って動的配列の要素数を再定義すると、それまで配列に格納されていたデータはすべてクリアされてしまいます。それまでのデータを保持したままで要素数を再定義したいときは、ReDim ステートメントに Preserve キーワードを付けます。本問題のコードでは「ReDim Preserve Member (4)」とすることで、それまで「Member (0)」～「Member (2)」に格納されていたデータはクリアされずに保持されます。

解答 4-5

Static

プロシージャの内部で宣言した変数は、そのプロシージャが実行されるたびに宣言されて初期化されます。したがって、プロシージャ内で変数に格納したデータは、次にプロシージャを実行するときには残っていません。そうではなく、プロシージャ内で宣言した変数であっても、前回格納したデータが残っているような変数を静的変数と呼びます。静的変数を宣言するときは、Dim ステートメントではなく、Static ステートメントを使います。また、静的変数はプロシージャ内でしか宣言できません。

解答 4-6

①Worksheet または Object ②Set

オブジェクトを格納するオブジェクト変数を宣言するときは、格納するオブジェクトの種類に応じたオブジェクト型を指定します。ワークシート (Worksheet オブジェクト) を格納するオブジェクト変数でしたら「As Worksheet」とします。また、すべてのオブジェクトを指す Object 型を指定することもできます。どちらでも間違いではありませんが、固有のオブジェクト型 (ここでは Worksheet) と、すべてのオブジェクトを表す Object 型の違いと意味を理解してください。また、すべての値を格納できる万能のバリエーション型 (Variant) にオブジェクトを格納することもできます。本問題では「Dim WS As Variant」としても正常に動作します。

オブジェクト型変数にオブジェクトを格納するときは「変数名 = オブジェクト」ではなく「Set 変数名 = オブジェクト」と Set ステートメントを使用します。オブジェクトを格納したオブジェクト型変数は、格納したオブジェクトと同等に操作できます。アクティブシート (Worksheet オブジェクト) を格納した変数「WS」には、Worksheet オブジェクトが持つ Name プロパティがあり、Name プロパティの値を変更すると、オブジェクト変数に格納したオブジェクト (ここではアクティブシート) の Name プロパティが変更されます。

解答 4-7

Nothing

VBA では、プロシージャ内で宣言したローカルレベル変数は、プロシージャが終了すると自動的に破棄されます。

オブジェクト変数のように他のメモリを参照するタイプの変数は、使用が終わったらコード内で明示的に解放します。前述のように、VBA ではプロシージャが終了するときに、オブジェクトの解放が自動的に行われるため、明示的な解放を行わなくても問題はありませんが、解放する方法は覚えておくべきでしょう。

オブジェクト変数を解放するには、変数に「Nothing」を代入します。このときも、Set ステートメントが必要です。オブジェクト変数に「Nothing」を代入すると、それまでオブジェクト変数内に格納されていた他メモリへの参照がクリアされます。こうしたオブジェクト変数の解放は、本問題のように Excel のオブジェクトを格納する場合には、それほど神経質になる必要はありません。しかし、CreateObject 関数などを使って、他のアプリケーションなどのインスタンスを利用するような場合、予期せぬ事態やバグなどによって、インスタンスがメモリ内に残ってしまう場合もあります。そのような、インスタンスを格納するオブジェクト変数では、プロシージャの最後でオブジェクト変数を解放するようにしましょう。

解答 4-8

①Type ②Prof

ユーザー定義型とは、ユーザーが独自に定義できる変数の型です。長整数型と文字列型など、異なる複数の型をひとつの変数に格納するようなときに便利です。ユーザー定義型の定義は、プロシージャなどを記述するコードウィンドウの「宣言セクション」に記述します。宣言セクションとは、コードウィンドウの、プロシージャよりも上の領域です。ユーザー定義型の定義は

```
Type ユーザー定義型の名前
    要素名 As 要素の型
    要素名 As 要素の型
    :
End Type
```

とします。定義したユーザー定義型を、変数の型として指定するときは、Type の後ろに指定した「ユーザー定義型の名前」を、宣言する変数の型として指定します。

5章 イベント

解答 5-1

Target

イベントとは、ユーザーが何かの操作をしたとき、あらかじめ作成しておいたマクロを自動的に実行する仕組みです。たとえば、ブックを開いたときに自動実行するマクロや、セルの値を変更したときに自動実行するマクロなどを作成できます。本問題は、セルの値が変更されたとき実行されるイベントプロシージャです。変更されたセルは、プロシージャの引数「Target」に格納されますので、引数「Target」を調べることで、どのセルが変更されたかを知ることができます。

解答 5-2

Cancel

イベントプロシージャの中には、ユーザーが行った操作を中止できるものがあります。たとえば本問題の「Workbook_BeforePrint」は、ユーザーが印刷の操作を行ったとき、印刷が実際に実行される前にイベントが発生します。イベントプロシージャ内では、まだ印刷が実行されていませんから、印刷を中止することも可能です。このように、ユーザーが行った操作を中止するときは、プロシージャの引数「Cancel」に「True」を代入します。

セルを右クリックしたときに発生するイベントプロシージャ「Worksheet_BeforeRightClick」や、セルをダブルクリックしたときに発生するイベントプロシージャ「Worksheet_BeforeDoubleClick」などにも、引数「Cancel」がありますので、右クリックやダブルクリックによる Excel の操作を中止することができます。

6章 ステートメント

解答 6-1

①Range("A1").Value ②End Select

Select Case ステートメントは、複数の条件判定を行うときに使用します。Select Case ステートメントでは、1行目「Select Case」の後ろに判定の対象を指定します。本問題では、セル A1 の値を判定しますので、セル A1 の値を表す「Range("A1").Value」または「Cells(1,1).Value」を指定します。Select Case ステートメントは、「Case」に続けて条件を記述し、最後に「End Select」を記述します。

解答 6-2

①1, 3, 5 ②10 To 20

Select Case ステートメントでは、「Case」に続けて条件を指定します。変数の値を判定するケースで、変数の値が「10 だったら」という条件は「Case 10」とします。「1 または 3 または 5」のように、1つの「Case」で複数の条件を判定するときは、それぞれの条件をカンマで区切って指定します。また「10 から 20 の間」のように、ある範囲を条件に指定するときは、条件の始まりと終わりを「To」で結びます。

解答 6-3

①Is ②Is

条件判定の対象自体を、「Case」の条件で使用するときには、Is キーワードを使用します。本問題では、条件判定の対象が「本日の日付」です。これは Now 関数と Day 関数で求めています。

「Day(Now())」は、1 日だったら「1」を返し、25 日なら「25」を返します。関数の返り値が「15 以下」か、あるいは「16 より大きい」か、を判定するには、比較演算子を用いた式を使います。このとき、式内で判定対象の値を「Is」で表します。なお、「Is」は式の左辺に記述しなければなりません。

解答 6-4

Case Else

本問題のように「どの条件にも該当しない」ときに処理を行うには「Case Else」という条件を最後に指定して、処理の内容を記述します。「Case Else」のない Select Case ステートメントでは、条件判定の対象が、どの「Case」にも該当しないときは、何も処理されません。なお、本問題に限っては①に「Case "土", "日」と記述しても同じ動作をしますが、本問題の主旨を理解していないとして不正解にします。

解答 6-5

Until

Do...Loop ステートメントの条件では、「While」または「Until」を指定します。「While」は「条件が正しい間」だけ処理が繰り返され、「Until」では「条件が正しくない間」だけ処理が繰り返されます。本問題は「Cells(cnt, 1).Value = ""」→「Cells(cnt, 1)が空欄である」という条件が「正しくない」→「空欄でない」間だけ繰り返されます。もし、条件が「Cells(cnt, 1) <> ""」だったときは、「Until」ではなく「While」を指定します。

解答 6-6

セル A1 に 11 が代入される

Do...Loop ステートメントの繰り返し条件は、「Do」の後ろまたは「Loop」の後ろに指定します。本問題の条件は「While Range("A1").Value < 5」ですから、セル A1 の値が 5 より小さい間だけ処理を繰り返します。マクロ実行前のセル A1 には 10 が入力されていて、この条件には該当しません。しかし、その条件判定は「Loop」の行で行われるため、無条件に、最低 1 回は Do...Loop ステートメント内の命令が実行されてしまいます。もし、

```
Sub Sample6 ()
    Do While Range("A1").Value < 5
        Range("A1").Value = Range("A1").Value + 1
    Loop
End Sub
```

だった場合、セル A1 に 10 が入力されていると、「Do」の行の条件がいきなり「該当せず」と判断されるため、Do...Loop ステートメント内の命令は 1 度も実行されません。

解答 6-7

Exit Do

Exit ステートメントは、引数に指定したステートメントから強制的に抜け出る働きをします。変数「cnt」を1ずつ増加させ、もし変数「cnt」が10を超えたとき、Do...Loop ステートメントから強制的に抜け出る（繰り返しを終了する）には「Exit Do」と記述します。もちろん「Exit Do」は、「Do」～「Loop」の中で使用しなければなりません。

Exit ステートメントは、ほかにも「Exit For」で For...Next ステートメントや For Each...Next ステートメントを抜け出たり、「Exit Sub」で Sub ステートメントを抜け出すことができます。Sub ステートメントは、Sub プロシージャを定義するステートメントで、一般的には現在実行されているプロシージャを指します。したがって、あるプロシージャの中で「Exit Sub」を実行することは、そのマクロ（プロシージャ）を強制的に終了させることになります。たとえば、次のコードは、セル A1 の値をセル B1 に代入しますが、もしセル A1 が空欄だった場合は、マクロを終了します。

```
Sub Sample8()
    If Range("A1").Value = "" Then
        Exit Sub
    Else
        Range("B1").Value = Range("A1").Value
        MsgBox "コピーしました"
    End If
End Sub
```

解答 6-8

c

For Each...Next ステートメントの構文は、

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

です。グループ（コレクション）の構成要素を1つずつ取り出して、変数に格納します。For Each...Next ステートメントの内部では、この変数に対して処理を行います。

解答 6-9

①Worksheets ②For

すべてのワークシートを調べるには、ブックに存在するすべてのワークシートの集合（コレクション）である「Worksheets」を対象にします。「Worksheet」ではなく「Worksheets」です。本問題では「ワークシート「合計」が存在するかどうか」を調べるのですから、もし、ワークシート「合計」が見つかった場合は、それ以降のワークシートを調べる必要がありません。そこで、ワークシート「合計」が見つかったときは For... Each ステートメントを抜け出るために Exit ステートメントを使用しています。本問題では、For... Each ステートメントを終了することと、Sub プロシージャを終了することは同じ動作ですので「Exit Sub」でも正解とします。

7章 関数

解答 7-1

①Split ②UBound

Split 関数は、文字列を任意の文字で区切って、区切った各文字列を要素とする配列を返します。配列の、最も大きなインデックス番号は UBound 関数で調べることができます。なお、配列の、最も小さなインデックス番号は LBound 関数で取得できます。

解答 7-2

IsDate

IsDate 関数は、引数に指定した値が、日付として有効な場合に True を返す関数です。

解答 7-3

IsNumeric

IsNumeric 関数は、引数に指定した値が、数値として有効な場合に True を返す関数です。

解答 7-4

①IsNumeric ②IsDate ③DateSerial

DateSerial 関数は、引数に指定した年月日で表される日付を返す関数です。

8章 エラーへの対応

解答 8-1

アクティブブックのワークシートが2枚以下しか存在しないとき

本問題のプロシージャは、構文的には間違っていない。しかし、For... Next ステートメントで指定した繰り返しの回数は1から3であり、1枚目から3枚目までのワークシートを操作しようとしています。もし、このブックにワークシートが2枚しかなかったら、3枚目のワークシート名を調べようとしたところでエラーが発生します。

解答 8-2

①GoTo ②Exit Sub

マクロ実行中にエラーが発生した場合、一般的にはマクロが停止してデバッグモードになります。第三者が使用するようなマクロでは、そのユーザーがマクロに精通しているとは限りませんので、デバッグモードで停止するようなマクロは適切ではありません。

エラーが発生したとき、任意の処理を実行するには「On Error GoTo ラベル名」という命令を使います。このとき、エラーが発生しなかった場合の想定を忘れないでください。ラベルは、GoTo によるジャンプ先ですが、ラベルの前でマクロの流れが自動的に停止するわけではありません。エラーが発生しなかったときは、エラー用のメッセージを表示しないのですから、ラベルの前でプロシージャを強制終了させなければなりません。プロシージャを強制的に終了させるには「Exit Sub」を使います。

なお、End ステートメントもマクロを強制終了させる働きがありますが、End ステートメントは実行中のすべてのマクロを終了させる点を正しく理解してください。「Exit Sub」は「実行中の Sub プロシージャ」を終了させます。もし、別のプロシージャから本問題の「Sample2」プロシージャが呼び出されていた場合、Exit Sub は「Sample2」プロシージャを終了させて、呼び出し元プロシージャに制御が戻りますが、End ステートメントは呼び出し元プロシージャを含めてすべてのマクロを終了させます。次の2つの例で、動作の違いを確認してください。

【Exit Sub ステートメントを使った例】

```
Sub Test1 ()
    MsgBox "Test2 を呼び出します"
    Call Test2
    MsgBox "処理が終了しました"
End Sub
Sub Test2 ()
    MsgBox "Exit Sub を実行します"
    Exit Sub
    MsgBox "この行は実行されません"
End Sub
```

【End ステートメントを使った例】

```
Sub Test3()
    MsgBox "Test4 を呼び出します"
    Call Test4
    MsgBox "処理が終了しました"
End Sub
Sub Test4()
    MsgBox "End を実行します"
    End
    MsgBox "この行は実行されません"
End Sub
```

解答 8-3

①Err ②Number

マクロ実行中に発生したエラーは、Err オブジェクトに格納されます。Err オブジェクトの Number プロパティは、発生したエラー番号を表します。Err オブジェクトの標準プロパティは Number プロパティですから、「Err.Number」は「Err」のようにプロパティを省略することも可能です。しかし、省略した場合は Number プロパティを取得しているという点を正確に理解してください。なお、Err オブジェクトの Description プロパティには、エラー発生時に表示される文字列が格納されます。合わせて覚えておきましょう。

解答 8-4

Resume Next

エラーが発生しても、マクロを停止せず無視するには「On Error Resume Next」という命令を使います。On Error Resume Next を多用して、やみくもにエラーを無視すると、プログラムの誤りなどによって「本来はエラーで止まって欲しいところ」でもエラーが発生しなくなり、かえってデバッグ作業を困難にすることもあります。ただし、本問題のように、どうしてもエラーの発生を避けられないようなケースでは有効な手段です。On Error Resume Next を実行すると、その後のコードでエラーが発生しても中断しませんが、発生したエラーの情報は Err オブジェクトに格納されます。本問題では、エラーが発生したかどうかを「ブックが保存されたかどうか」で判定していますが、次のように Err.Number の値で判定することも可能です。

```
Sub Sample4()
    On Error Resume Next
    ActiveWorkbook.SaveAs "Book1.xlsm"
    If Err.Number = 0 Then
        MsgBox "保存されました"
    Else
        MsgBox "保存されていません"
    End If
End Sub
```

9章 UserForm

解答 9-1

Show

作成した UserForm を表示するには、Show メソッドを実行します。

解答 9-2

①Caption ②Text

ラベル (Label コントロール) に表示する文字列は Caption プロパティで操作します。テキストボックス (TextBox コントロール) に入力されている文字列は Text プロパティで操作します。どちらのプロパティも、それぞれのコントロールの標準プロパティですから、次のようにプロパティを省略することも可能です。

```
Private Sub CommandButton1_Click()
    Label1 = TextBox1
End Sub
```

しかし、標準のプロパティを暗黙的に利用するときでも、実際には何のプロパティを操作しているのかを意識することは重要です。決して、“知らなくてもいい”ことではありません。

解答 9-3

Unload

UserForm を閉じるには、Unload ステートメントを実行します。「Me」は自分自身を表すキーワードで、ここでは、このコードが記述されている UserForm を表します。Unload ステートメントには、閉じる UserForm の名前を引数で指定しますので、「Unload UserForm1」や「Unload UserForm2」のように、UserForm の名前を明記することもできますが、UserForm で“自分自身を閉じる”ときには、一般的に Me キーワードを使うことが多いです。また、Unload ステートメントは「UserForm を閉じる」命令であり、実行されているマクロを終了させる End ステートメントとは挙動が異なるので注意が必要です。次の 2 つの例で、動作の違いを確認してください。

【標準モジュールに書いた UserForm を開くマクロ】

```
Sub Sample1()
    UserForm1.Show
    MsgBox "終了しました"
End Sub
```

【UserForm に書いた UserForm を閉じるマクロ】

```
Private Sub CommandButton1_Click()
    Unload Me
End Sub
```

【標準モジュールに書いた UserForm を開くマクロ】

```
Sub Sample1 ()
    UserForm1.Show
    MsgBox "終了しました"
End Sub
```

【UserForm に書いた UserForm を閉じるマクロ】

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

解答 9-4

MultiLine プロパティ

テキストボックス (TextBox コントロール) は、標準では「1 行しか入力できない」状態です。複数行の入力や表示を可能にするには、MultiLine プロパティに True を設定します。あらかじめ、複数行の入力をするとわかっている場合は、VBE のプロパティウィンドウで設定しますが、マクロのコードで設定を変更することも可能です。

解答 9-5

AddItem

リストボックス (ListBox コントロール) にデータを追加するには、AddItem メソッドを実行します。また、ワークシート上に入力されているデータをリストボックス (ListBox コントロール) に登録するには、RowSource プロパティを使う方法もあります。たとえば、ワークシート「Sheet3」のセル範囲 A1:A4 に入力されているデータを登録するには、UserForm の設計時に、プロパティウィンドウの [RowSource] に「Sheet3!A1:A4」と設定します。また、マクロのコードで設定するには、次のようにします。

```
Private Sub CommandButton1_Click ()
    ListBox1.RowSource = "Sheet3!A1:A4"
End Sub
```

いずれにしても「Worksheets ("Sheet3").Range ("A1:A4")」ではなく、ワークシート上の関数で使用するような「Sheet3!A1:A4」といった形式である点に留意してください。RowSource プロパティによるリストの登録は、ワークシート上のセルを手軽にリンクできるというメリットがありますが、RowSource プロパティを設定したリストでは、AddItem メソッドによるデータの登録や、RemoveItem メソッドによるデータの削除ができなくなるという制約もあります。

解答 9-6

①ListIndex ②Text

リストボックス (ListBox コントロール) に登録されているデータのうち、上から何番目のデータが選択されているかは、ListIndex プロパティで調べられます。また、ListIndex プロパティに任意の数値を設定することで、選択されているデータを変更することも可能です。ただし、ListIndex プロパティでは、リストボックス (ListBox コントロール) に登録されているデータの先頭が「0」から始まることに留意してください。次のコードは、リストボックス (ListBox1) の上から3番目のデータを選択状態にします。3番目は「3」ではなく「2」を指定します。

```
Private Sub CommandButton1_Click()
    ListBox1.ListIndex = 2
End Sub
```

リストボックス (ListBox コントロール) で選択されているデータを取得するには Text プロパティを使います。Text プロパティは、現在選択されているデータを返しますが、Text プロパティを変更することで、選択されているデータを変更することも可能です。

```
Private Sub CommandButton1_Click()
    ListBox1.Text = "平沢"
End Sub
```

上記のマクロを実行すると、リストボックス (ListBox1) で「平沢」が選択された状態になります。ただし、リストボックス (ListBox1) には「平沢」というデータが登録されていなければなりません。登録されていないデータを Text プロパティに設定するとエラーになります。

なお、本問題の②は「List(ListBox1.ListIndex)」でも同じ結果が得られるので正解とします。List プロパティは、リストボックス (ListBox コントロール) に登録されているデータを配列形式で操作するプロパティです。

また、リストボックスの Value プロパティも選択されているデータを取得することができますが、リストボックスで何もデータが選択されていないとき、Text プロパティは空欄を返しますが、Value プロパティはエラーが発生します。したがって、Value プロパティで選択データを取得するときは、エラーに対する処理が必須です。何のエラー対策もしていない本問題で、Value プロパティを使ってデータを取得するのは相応しくないので、Value プロパティは不正解とします。

解答 9-7

Controls

UserForm 上に「CheckBox1」「CheckBox2」「CheckBox3」の3つのチェックボックス（CheckBox コントロール）が配置されているとき、それぞれの CheckBox のオン/オフを調べるには次のように考えられます。

```
Private Sub CommandButton1_Click()
    If CheckBox1.Value = True Then
        MsgBox "CheckBox1 はオンです"
    End If
    If CheckBox2.Value = True Then
        MsgBox "CheckBox2 はオンです"
    End If
    If CheckBox3.Value = True Then
        MsgBox "CheckBox3 はオンです"
    End If
End Sub
```

しかし、これではコントロールの数が増えると、判定のためのコードが煩雑になります。そこで、チェックボックス（CheckBox コントロール）やオプションボタン（OptionButton コントロール）など、UserForm 上に複数配置されるようなコントロールを操作するときは、UserForm 上に配置されている「すべてのコントロール」を返す Controls コレクションを使います。

解答 9-8

LoadPicture

イメージ（Image コントロール）に表示する画像は、Image コントロールの Picture プロパティで操作できます。画像を表示するには、Picture プロパティに画像ファイルを直接指定するのではなく、LoadPicture 関数の戻り値を代入します。Image コントロールに表示されている画像を消すには、LoadPicture 関数の引数に空の文字列（""）を指定して実行した結果を代入します。

```
Private Sub CommandButton1_Click()
    Image1.Picture = LoadPicture("")
End Sub
```

10章 メニューの操作

解答 10-1

①CommandBars ②Controls

Excel のメニューバー、ツールバー、コンテキストメニューなどは、すべて CommandBar オブジェクトで操作できます。CommandBar オブジェクトの集合体は CommandBars コレクションであり、セルのコンテキストメニューを表す CommandBar オブジェクトの名称は「Cell」です。したがって、セルのコンテキストメニューは CommandBars("Cell") で表されます。メニュー内のコマンドや、ツールバー内のボタンは、CommandBarControl オブジェクトですが、CommandBarControl オブジェクトを操作するときは Controls コレクションを使います。

解答 10-2

①Set ②Caption ③OnAction

メニューにコマンドを追加するには、Controls コレクションの Add メソッドを実行します。その後、登録したコマンドに表示する文字列や、クリックされたときに実行するプロシージャなどを設定しますので、Add メソッドで登録したコマンドをオブジェクト変数に格納しておくのが便利です。オブジェクト変数（ここでは「NewCommand」）に登録したコマンドを格納するには、Set ステートメントを使います。コマンドに表示される文字列は Caption プロパティ、クリックしたとき実行されるプロシージャは OnAction プロパティに設定します。なお、次のように、オブジェクト変数を使わず、With ステートメントを使用することもできます。

```
Sub Sample2()
    With CommandBars("Cell").Controls.Add(Type:=msoControlButton)
        .Caption = "処理 1"
        .OnAction = "myMacro"
    End With
End Sub
```

解答 10-3

①BeginGroup ②FaceId

コマンドの上部に区切り線を引くときは、そのコマンドの BeginGroup プロパティに True を設定します。区切り線が引かれていないコマンドの BeginGroup プロパティは False です。区切り線は“ここから新しいグループが始まる”という意味の BeginGroup プロパティを設定することで引かれますので、コマンドの下部に区切り線を引くことはできません。

コマンドにアイコンを表示するときは FaceId プロパティに、アイコンを表す数値を指定します。何番が何のアイコンを示すかといった公式の資料は公開されていませんが、インターネットなどでは、ユーザーが独自に調査した情報が公開されています。

解答 10-4

①Tag ②Delete

メニューのコマンドを削除するには Delete メソッドを使います。しかし、Excel の標準コマンドや、他のプログラムが追加したコマンドを削除してしまうと、その後の操作に支障が出るかもしれません。メニューは、どのプログラムからも操作できる、いわば“共有の場所”ですから、自分で追加したコマンドを削除するときは、そのコマンドが本当に自分で追加したものであるかどうかを確認する慎重さが必要です。それには、コマンドを追加するときに印を付けておき、削除するときにその印を確認するようにします。コマンドの印は Tag プロパティを使うといいでしょう。Tag プロパティはメニューコマンドなどのオブジェクトに自由な文字列を設定できるプロパティです。

11章 Windows の機能を利用する

解答 11-1

①Set ②CreateObject

OLE オートメーションは、VBA から別のアプリケーションを制御して操作できる仕組みです。OLE オートメーションを使うと、Internet Explorer を制御してネットのデータを取得するなど、Excel VBA だけでは不可能な機能を実現できます。しかし、どんなアプリケーションでも OLE オートメーションで制御できるとは限りませんし、他のアプリケーションを思い通りに制御するには、制御するアプリケーションの仕様やコマンドなどを学習しなければなりません。VBA の中でも、OLE オートメーションは高度な技術に属しますが、Excel VBA では実現不可能な操作を可能にするテクニックですので、簡単な使い方や概念はしっかり学習してください。

OLE オートメーションを利用するには、VBE の参照設定で利用したクラス名を登録する「事前バインディング」と、マクロのコード中で参照を作成する「実行時バインディング」の 2 種類の方法があります。実行時バインディングで作成する他アプリケーションへの参照を「インスタンス」と呼びます。実行時バインディングでインスタンスを作成するには、CreateObject 関数を使います。CreateObject 関数の戻り値をオブジェクト変数に格納するときは、Set ステートメントを使用します。

OLE オートメーションと同じように、Excel VBA では実現できない機能を実現する仕組みに API (Application Programming Interface) があります。API は、Windows などが公開している関数などの機能を、VBA などの別プログラムから利用する仕組みです。VBA から API を利用するには、利用したい API の宣言文を宣言セクションに記述し、利用したい関数の書式に合わせて適切なデータを指定します。API の使用には、VBA 以外のプログラミング知識なども要求されますので、誰もが安易に使用できる仕組みではありません。ここでは、VBA から API を利用できるという概念と、簡単な使い方だけ学習してください。

12章 レジストリの操作

解答 12-1

SaveSetting

レジストリは、Windows が管理する特別な領域です。レジストリには、Windows やアプリケーションなどの動作に必要な情報や設定項目などを記録できます。レジストリのデータを破壊すると、Windows が起動できなくなるなど致命的なトラブルが生じます。使用には慎重さが求められるレジストリですが、VBA で操作できるレジストリの領域は限定されていて、Windows が使用する重要な領域にはアクセスできません。VBA で用意されているレジストリ操作のコマンドを使えば、比較的安全にレジストリを活用できます。複数ブックで共有するデータを記録したり、前回の操作結果を記録するなど、Excel マクロからレジストリを操作できると便利なケースは少なくありません。慎重さを忘れずに、しかし積極的にレジストリを活用してください。

解答 12-2

GetSetting

レジストリのデータを取得するには GetSetting 関数を使います。GetSetting 関数の書式は次の通りです。

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

レジストリを活用するマクロでは、取得しようとしたレジストリデータが存在しなかった場合を考慮してください。GetSetting 関数は、指定したセクションやキーが存在しないと空欄 (“”) を返しますので、次のように判定することもできます。

```
Sub Sample2()
    Dim buf As String
    buf = GetSetting("MyMacro", "Main", "Data")
    If buf = "" Then
        MsgBox "存在しません"
    Else
        MsgBox buf
    End If
End Sub
```

もちろん、この考え方も間違いはありませんが、GetSetting 関数の引数「既定値」を指定すると、取得できなかったときの処理が簡素化できます。

解答 12-3

DeleteSetting

レジストリを削除するには、DeleteSetting ステートメントを使います。DeleteSetting ステートメントの書式は次の通りです。

DeleteSetting アプリケーション名, セクション名, キー名

13章 ファイルの操作

解答 13-1

①Open ②Input ③Close

VBA にはファイルを操作するコマンドが数多く用意されています。あまり使い道のないコマンドだと感じる人がいるかもしれませんが、Excel を操作するマクロでは、こうしたファイル操作が必要になるケースも珍しくありません。たとえば、「C:\¥Work¥Book1.xlsx」を開くマクロでは、確かに「C:\¥Work¥Book1.xlsx」が存在するかどうかを事前に確認しなければなりません。あるいは、現在作業中のブックを別名で保存するとき、すでに存在する同名ファイルをリネームや削除することもあります。Excel のマクロであっても、マクロの完成度を高めたり、システムの規模が大きくなると、ファイルの操作は必須の技術となります。

ファイルからデータを読み込む手順は次の通りです。

- (1) ファイルを開く
- (2) ファイルから読み込む
- (3) ファイルを閉じる

ファイルを開くには Open ステートメントを使います。このとき「ファイルからデータを読み込む」ためのモードである「Input」を指定します。ファイルから 1 行分のデータを読み込むには、Line Input ステートメントを使います。読み込みが完了したら、Close ステートメントでファイルを閉じます。

テキストファイルを 1 行ずつ読み込んで処理するとき、このコードはいわば“定石”です。MsgBox の代わりに、セルへ書き込んだり、変数に格納するなど、さまざまな処理に応用できます。暗記するほど学習してください。

解答 13-2

①Open ②Output ③Close

ファイルに書き込むときも考え方は同じです。

- (1) ファイルを開く
- (2) ファイルに書き込む
- (3) ファイルを閉じる

の3ステップで行います。ファイルを開くとき指定する「ファイルにデータを書き込む」ためのモードは「Output」です。

解答 13-3

Append

「Output」で開いたファイルに書き込むと、それまでファイルに記録されていたデータは消えてしまいます。そうではなく、既存データの末尾に新しいデータを追記するには、「Append」を指定して開きます。

解答 13-4

FileCopy

ファイルをコピーするには、FileCopy ステートメント使います。FileCopy ステートメントの引数には、コピー元ファイルとコピー先ファイルを指定します。コピー先ファイルの名前を指定すると、別名としてコピーすることも可能です。

解答 13-5

Name

ファイルの名前を変更（リネーム）するには、Name ステートメントを使います。ファイルをコピーする FileCopy ステートメントと似ていますが、引数の指定に留意してください。FileCopy ステートメントでは2つの引数を「, (カンマ)」で区切りますが、Name ステートメントは「As」を使って指定します。

解答 13-6

Kill

ファイルを削除するには Kill ステートメントを使います。Kill ステートメントで削除したファイルは、Windows のゴミ箱には入らず、完全に削除されます。Windows Vista では、Kill ステートメントによるファイルの削除や、Open ステートメントによる新しいファイルの作成を、C ドライブのルート (C:\) に対して行おうとすると、警告が出たり、実行できないことがあります。Windows Vista の環境でファイル操作を行う場合は、Windows Vista の仕様を十分に把握してください。

解答 13-7

①Dir ②Sample.xlsx

ファイルの存在を調べるには、Dir 関数を使います。Dir 関数は、ファイル操作系コマンドの中でも、最も多く利用されるコマンドのひとつです。Dir 関数は、引数に指定したファイルが存在しないとき空欄 (“”) を返します。また、引数に指定したファイルが存在したときは、そのファイル名を返します。このとき、ファイルのパスは含まれません。「C:¥Work¥Sample.xlsx」が存在するとき、Dir (“C:¥Work¥Sample.xlsx”) は「Sample.xlsx」を返します。