

資料

心理学における Excel VBA の利用 その1 VBA の基本文法

久本 博行

Applications of Excel VBA for the Psychology (1)

VBA Programming

Hiroyuki HISAMOTO

Abstract

An experimental systems in psychology using Excel VBA(Visual Basic Applications) is proposed, and the basic grammar of VBA is introduced.

Key words: Excel VBA, Psychological Experiments, Programming

抄録

Excel VBA(Visual Basic Applications)によって、心理学のコンピュータ実験システムを作成することが提案され、その1としてVBAの基本文法が紹介された。

キーワード : Excel VBA, 心理学実験, プログラミング

はじめに

現在心理学の実験にはコンピュータが使われることが多くなってきている。これは、コンピュータによる刺激提示や提示時間の制御、反応時間の測定などが容易なためである。また、コンピュータ上で実験が行われ、データが収集されるとそのデータをすぐに分析することができる、という利点もある。

コンピュータによる心理学実験システムとしては、Cedrus 社の Super Lab が有名であり多くの実験で用いられている。しかし、Super Lab は高価で、システムは柔軟性に富むものであるがそれゆえに設定が複雑である。また、コンピュータによる心理学実験についての著作は数多く発表されている(田中 1975, 吉村・山上 1983, 市川・矢部 1985, 野沢 1986, 金子ら 1990, Dixon, M.R. & MacLin, O.H. 2003, 北村・坂本 2004, 水野りか 2004)が、この中で 1970 年代から 90 年代に発表されたものは BASIC で作られたものがほとんどで、Windows に対応していない。Dixon & MacLin のものは Visual Basic .Net が必要であること、北村・坂本のものは Visual Basic 6.0, HSP, Delphi といった各種のプログラミング言語を使っているため初心者には難しく、さらに Visual Basic 6.0 や Delphi の開発環境が必要である。水野は Web ブラウザ上で動く心理学実験システムを JAVA と Perl で作成しているが、これは開発環境はフリーで入手しやすいのであるが、JAVA や Perl を学習するのが難しいという難点がある。もちろん、北村・坂本や水野らのプログラムをそのまま利用するうえでは問題はないのであるが、同様なプログラムを開発しようとするとき上記のような障害がある。

したがって、開発環境が手軽に手に入り、プログラミングが容易な言語で実験のシステムを構築することが多くの実験者にとって有益であると考えられる。そこでフリーウェアではないが、多くの Windows システムにバンドルされており、プログラミングも比較的容易な Microsoft 社の Excel の VBA(Visual Basic for Applications)を使い、いくつかの心理学実験用ソフトやデータ分析用ソフトを作成した。これらは、心理学実験を行おうとするものが、そのままでも使えるが、さらに自分の実験向きにプログラムを改良したり、サンプルプログラムとして利用できるようなものを目指した。今回は「その 1」として自分で VBA のプログラミングが可能になるように、VBA の基本文法を紹介する。

また、今後は「錯視」、「Stroop 効果」などの実験ソフトウェアや「SD 法」、「一対比較法」などの刺激提示とデータ収集用ソフトウェア、「因子分析」などのデータ解析ソフトウェアを順次発表していきたい。

なおこれを読むにあたり、Excel の基本操作を理解していることを前提としている。

1. Excel VBA の操作

1. 1VBA とは

VBA とは、Visual Basic for Applications の略で Microsoft 社の Word, Excel, Access, Power Point といった Office 製品で使われるプログラミング言語です。VBA を利用すると Office を使ったアプリケーション・ソフトを開発することができます。ここでは、Excel のアプリケーションを開発していきますが、この知識は他の Office 製品のアプリケーションを作る時にも役立ちます。

1. 2マクロのセキュリティ

Excel でマクロを利用する場合には、はじめにマクロのセキュリティ設定をしなければなりません。マクロのセキュリティレベルを変えるには、[ツール]メニューをクリックし、次に[マクロ]をポイントし[セキュリティ] をクリックします。

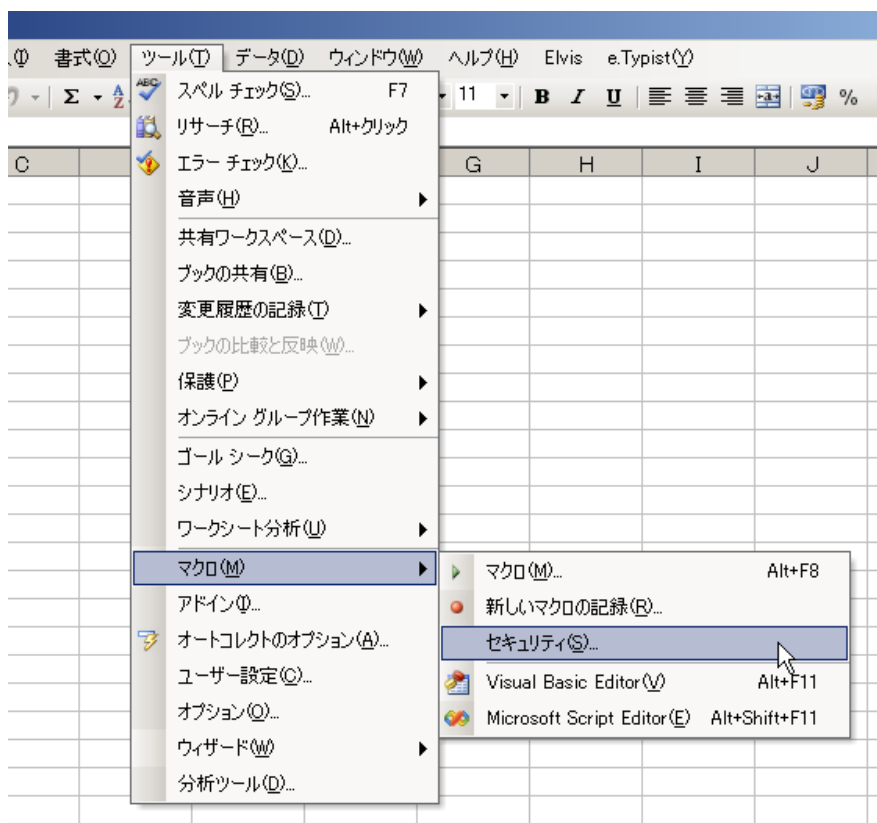


図 1.2.1

そうすると、次の図のようなセキュリティ設定のダイアログボックスが出てきます。通常、デフォルト(既定値)ではセキュリティレベルは「高(H)」に設定されており、自分で作成したマクロを実行することができません。そこで、セキュリティレベルを「中(M)」にします。そして、「OK」ボタンをクリックします。この状態ではまだセキュリティレベルは変わっていませんので、いったん Excel を終了し、もう

一度 Excel を起動しなおします。これで、セキュリティレベルの変更が完了しました。

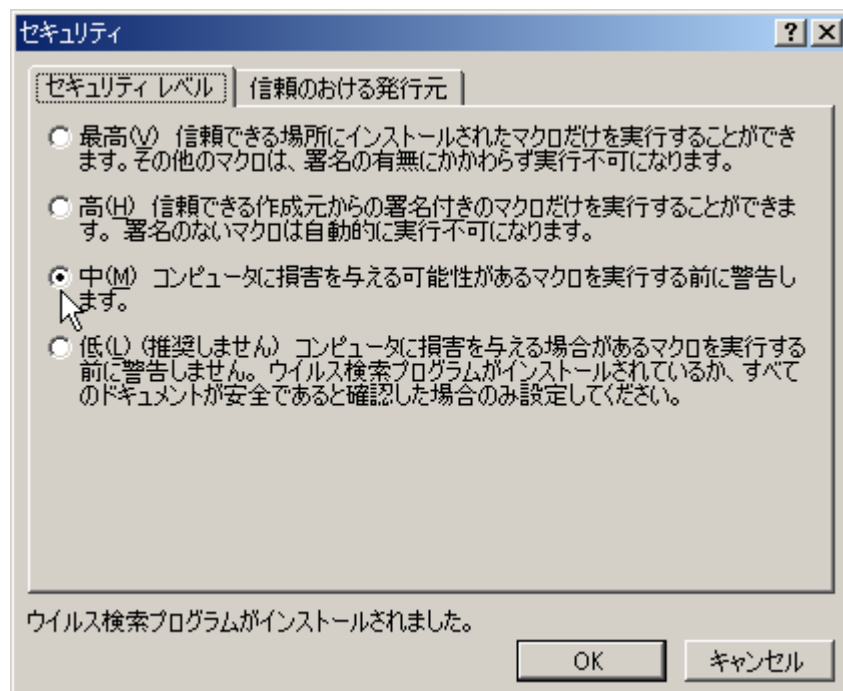


図 1.2.2

次にマクロを含む Excel のファイルを開こうとすると、下図のようなセキュリティ警告が出ます。そのときは、自分が作ったような信頼できるマクロであれば「マクロを有効にする(E)」をクリックしファイルを開きます。

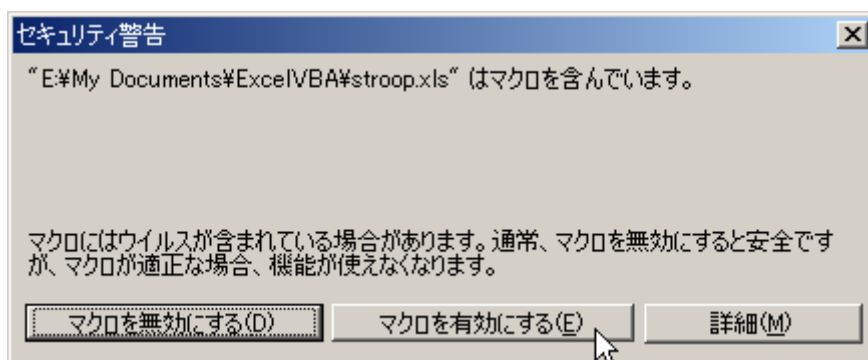


図 1.2.3

1. 3Visual Basic Editor の起動

マクロを作成するには、まずマクロを作成するツール Visual Basic Editor(以下 VBE と略)を起動します。[ツール]メニューをクリックし、[マクロ]→[Visual Basic Editor]の順でクリックします。(Alt キーを押しながら、F11 を押しても VBE を起動できます。)

マクロを作る他のやり方として、Excel では「マクロ記録」という方法があります。これは、マクロに行わせたい操作を実際に行い、その一連の操作手順を記録し、マクロとして利用するものです。これは簡単にマクロが作成できますが、使える機能が限られているのでここでは取り上げません。

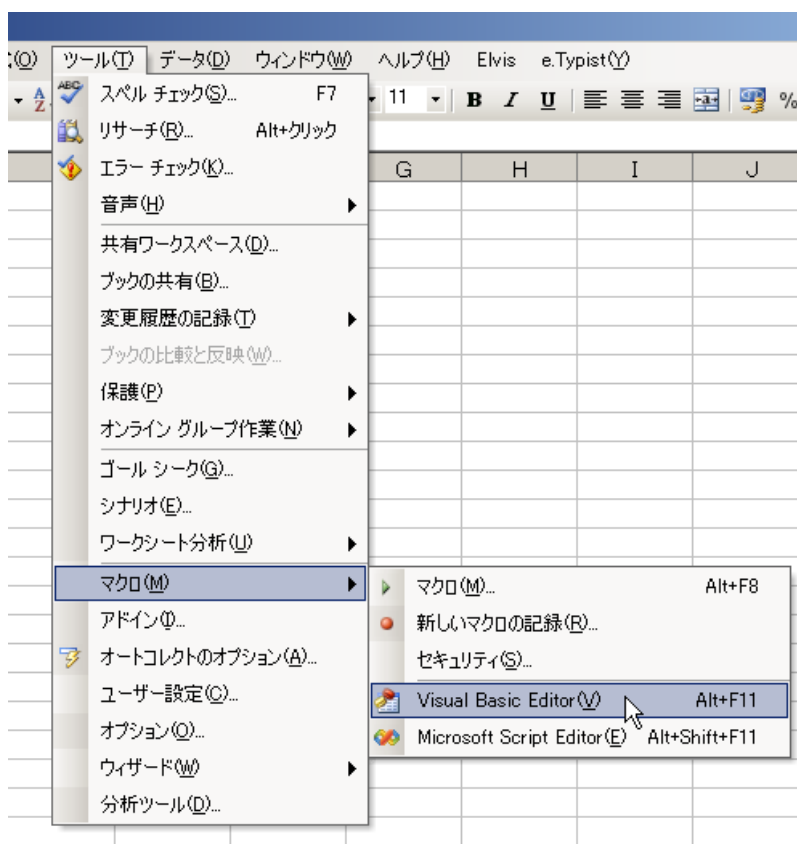


図 1.3.1

1. 4 やさしい入門

では、実際にプログラムを書いてみましょう。図 1.4.1 のような形のフォームを作り、「はじめてのプログラム」というボタンをクリックすると、「Hello World!」と表示されるようにします。



図 1.4.1

1. 4. 1 ユーザーフォームの作成

「挿入」ボタンから「ユーザーフォーム」を選び(図 1.4.2), ユーザーフォームを作ります。この操作は「挿入」メニューから「ユーザーフォーム」を選んででもできます。

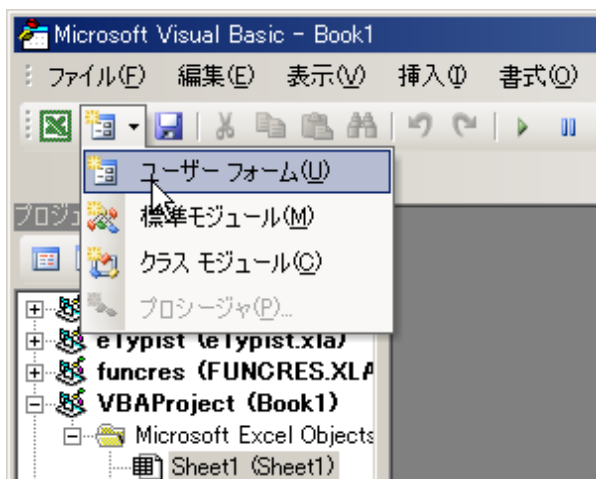


図 1.4.2

ここで少し、Visual Basic Editor の画面(図 1.4.3)の説明をしておきましょう。プロジェクトエクスプローラは、そのブックに含まれるオブジェクトやモジュールの階層構造を表したものです。

プロパティウィンドウは、選択されているオブジェクトのプロパティを見たり、編集したりするためのものです。フォームウィンドウは、フォームを編集するためのもので、ツールボックスはフォームに貼り

付けるいろいろなコントロールを 用意しているものです。ツールボックスが非表示の場合には、標準ツールバーのツールボックスボタンを押すと表示されます。作業ウィンドウはヘルプを表示します。

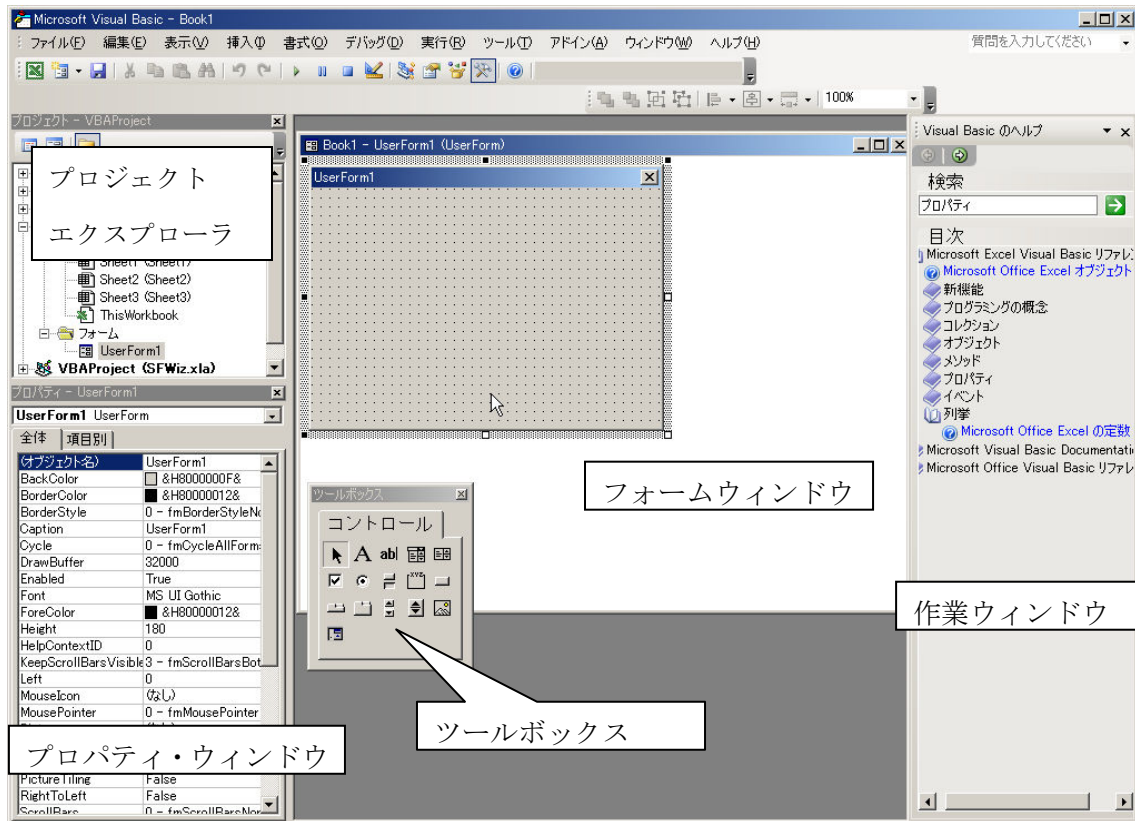


図 1.4.3

「Hello World!」と表示が出ている部分は、テキストボックスと呼ばれるものです。このテキストボックスをフォームに貼り付けます。ツールボックスのテキストボックスをクリックし(図 1.4.4), フォーム上で適当な大きさになるようにドラッグします。(図 1.4.5)

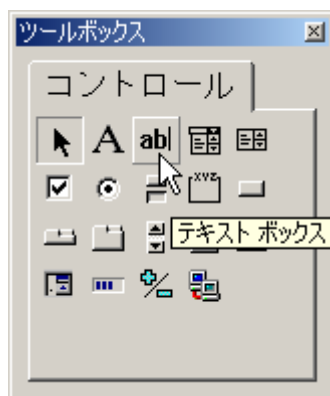


図 1.4.4

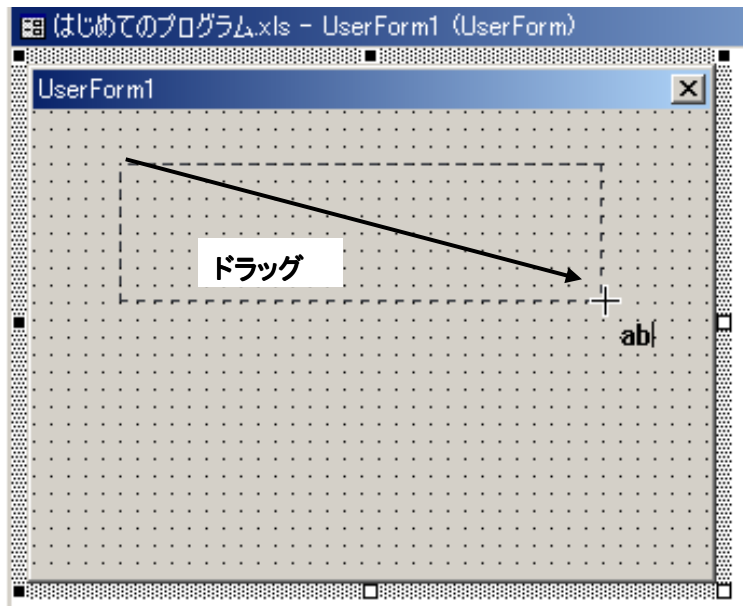


図 1.4.5

次に、「初めてのプログラム」というボタンを作ります。このボタンのことをコマンドボタンといいます
 が、これをテキストボックスの同じ要領で貼り付けます。ツールボックスのコマンドボタンをクリックし
 (図 1.4.6), フォーム上で適当な大きさになるようにドラッグします。(図 1.4.7)



図 1.4.6

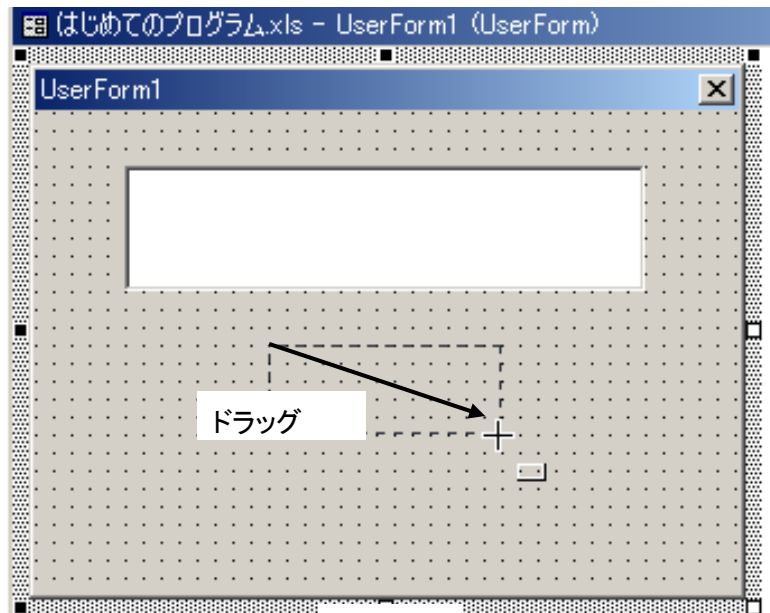


図 1.4.7

次に、フォームのタイトルバーに表示されている名前が「UserForm1」となっていますので、これを「やさしい入門」に、コマンドボタンには「CommandButton1」と表示されていますので、これを「初めてのプログラム」に変更します。

まず、フォーム上でテキストボックスやコマンドボタンのないところならどこでも構いませんからクリックし、フォームを選択状態にします。そして画面左下のプロパティ・ウィンドウの「Caption」欄を「やさしい入門」に書き換えます(図 1.4.8)。これでユーザーフォームのタイトルバーの表示が「UserForm1」から「やさしい入門」に変わりました。コマンドボタンについても同様です。コマンドボタンを一度クリックし、選択状態にするとコマンドボタンのプロパティがプロパティ・ウィンドウに表示されますので、「Caption」欄を「はじめてのプログラム」に書き換えます(図 1.4.9)。プロパティでは、この他に文字の色、サイズ、そのオブジェクトの幅、高さ、背景色などの属性が定義されています。プロパティはこのようにいろいろな属性を定義しているもので、あるオブジェクトを選択状態にすると、プロパティ・ウィンドウにそのプロパティが表示されるような仕組みになっています。なお、プロパティを変更するのは、今のようにプロパティ・ウィンドウでもできますし、プログラムからもできます。

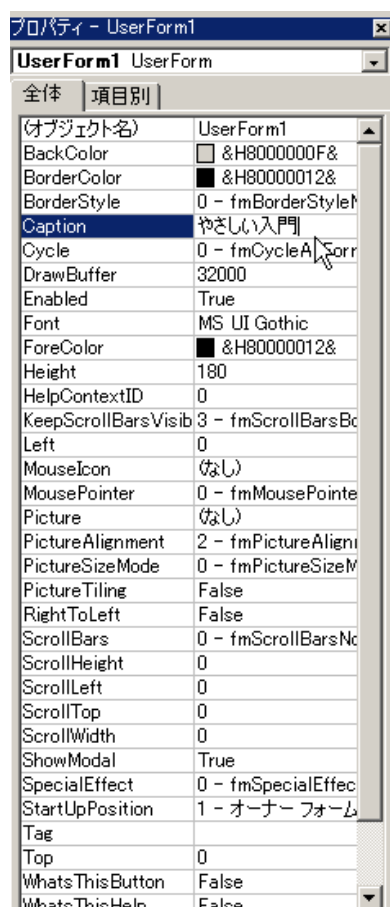


図 1.4.8

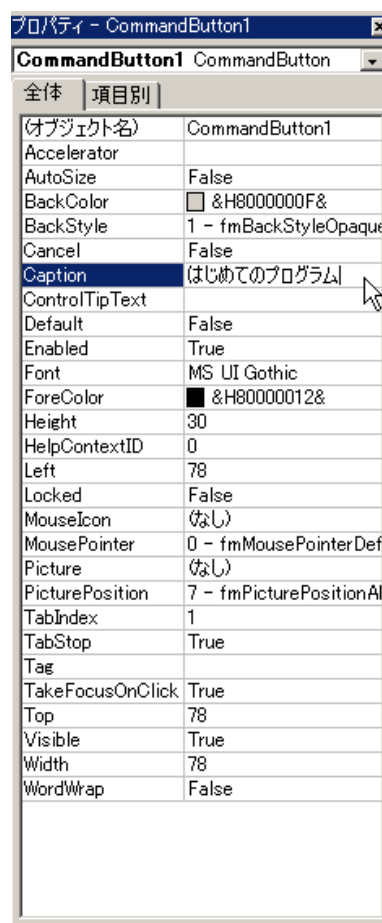


図 1.4.9

1.4.2 プログラムの作成

では、次にコマンドボタンを押すと「Hello World!」と表示されるようにプログラムを作ってみます。

先ほど作った「はじめてのプログラム」というコマンドボタンをダブルクリックします(図 1.4.10)。そうすると、図 1.4.11 にあるようなコードウィンドウが表示されます。このコードウィンドウでコマンドボタンが押されたときの動作を記述します。

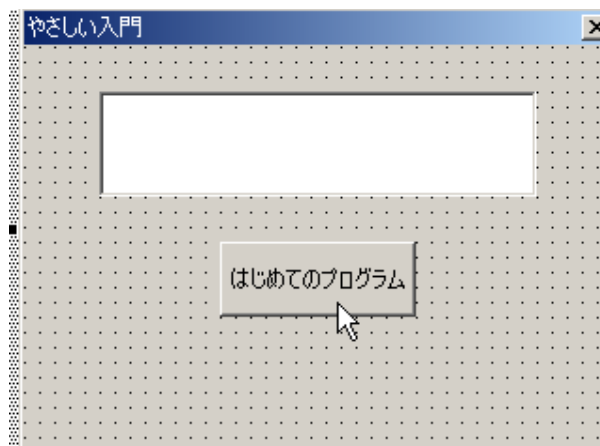


図 1.4.10

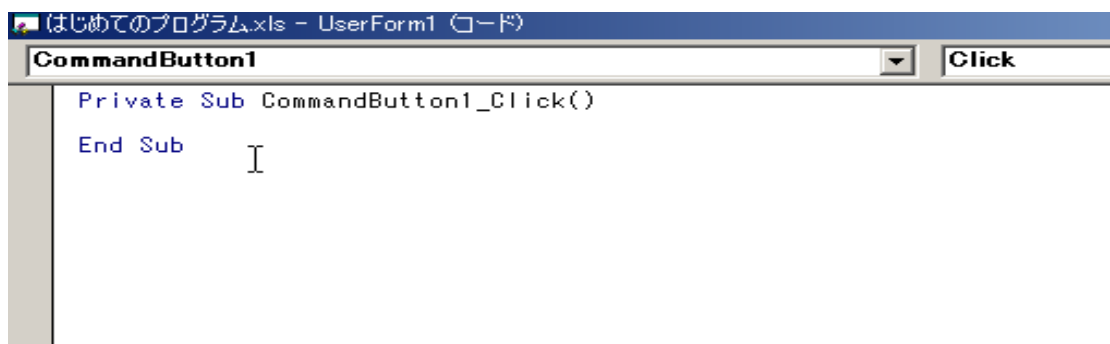


図 1.4.11

```
Private Sub CommandButton1_Click( )
```

と

```
End Sub
```

の 2 行の間に

```
TextBox1.Text = "Hello World!"
```

と入力します。これで出来上がりです。



図 1.4.12

プログラムの詳しい説明の前に、このプログラムを動かしてみましょ。ツールバーにある「Sub/ユーザーフォームの実行」ボタンをクリックしてください(図 1.4.13)。そして、「はじめてのプログラム」のボタンをクリックしましょ。「Hello World!」と表示されましたか(図 1.4.14)。

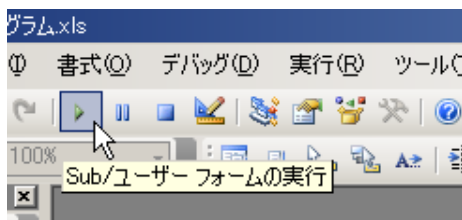


図 1.4.13



図 1.4.14

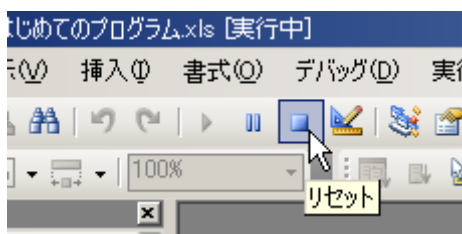


図 1.4.15

もし、うまく表示されなかつたり、エラーが表示された場合は、左記のプログラムとよく見比べてください。正しく実行できれば、リセットボタン(図 1.4.15)を押して停止します。

では、プログラムについて説明しておきましょう。プログラムの 1 行目は、このプログラムが CommandButton1 をクリックされたときの動作を定義しているプログラムであることを示しています。

```
Private Sub CommandButton1_Click()
    TextBox1.Text = "Hello World!"
End Sub
```

2 行目が実際にテキストボックスに「Hello World!」を表示している部分です。TextBox1.Text は TextBox1 というオブジェクトのなかの Text プロパティを示しており、=は代入の記号で、Text プロパティに""で囲まれた文字列の「Hello World!」を代入しなさい、ということになります。文字列を指定

するときには、””で囲むことになっており、””で囲まれた文字列はプログラムではなく、データとして取り扱われます。

3 行目はこのプログラムの終わりを示す部分です。

1 基本文法

ここでは、Excel Visual Basic for Applications (VBA と略) の基本的な文法を説明します。VBA や Visual Basic の文法をすでにご存知の方は、この章をとばしていただいて構いません。なお、文法の説明の中で [] で囲まれた部分は、省力可能であることを示しています。

2.1 変数

変数とはプログラム中で a, b などのように名前がつけられた、数値や文字列を収納しておく記憶領域です。変数の名前のつけ方には規則があります。

- 名前の先頭は数字以外の文字 (英字, 漢字, ひらがな, カタカナ) で始めます。
- 名前には空白や&, #などの記号 (アンダースコアを除く) を使うことはできません。
- 名前は 255 文字 (半角の場合) 以内でなければなりません。
- Visual Basic の関数, ステートメント, およびメソッドと同じ名前を使うことはできません。

正しい変数名の例

a

heikin

利息

悪い変数名の例

1gakki … 数字で始まっている

abc@def … @が使われている

right … Visual Basic に同じ関数名がある

2.2 データ型

VBA で取り扱う変数には、表 2.2 のようなデータ型があります。通常、取り扱うデータが整数の場合は、Integer 型か Long 型を、実数型の場合は Single 型か Double 型を使います。金銭の計算を行う場合は、Currency 型を使います。用途に合わせることで、その変数型で表現できる範囲を考え合わせてデータ型を選択します。

表 2.2 データ型

データ型	サイズ	説明
Byte(バイト型)	1 バイト	0～255 の整数
Boolean (ブール型)	2 バイト	True あるいは False のみです。数値をブール型に変換すると 0 は False で 0 以外の値は True となります。また、ブール型の変数を数値に変換すると True は-1, False は 0 となります。
Integer(整数型)	2 バイト	-32,768～32,767
Long(長整数型)	4 バイト	-2,147,483,648～2,147,483,647
Single(短精度浮動小数点数型)	4 バイト	-3.402823E38～-1.401298E-45(負の値) 1.401298E-45～3.402823E38(正の値)
Double(倍精度浮動小数点数型)	8 バイト	-1.29769313486231E308～ -4.94065645841247E-324(負の値) 4.94065645841247E-324～ 1.29769313486231E308(正の値)
String(文字列型)	10 バイト+文字列の長さ	0～2GB
Currency(通貨型)	8 バイト	-922,337,203,685,477.5808 ～ 922,337,203,685,477.5807
Decimal(10進型)	14 バイト	小数部分を持たない数値の場合, -79,228,162,514,264,337,593,543,950,335 ～ 79,228,162,514,264,337,593,543,950,335 小数点以下 28 桁の数値の場合, -7.9228162514264337593543950335 ～

		7.9228162514264337593543950335
Date(日付型)	8 バイト	西暦 100 年 1 月 1 日～西暦 9999 年 12 月 31 日の範囲の日付と, 0:00:00 ～ 23:59:59 の範囲の時刻
Object(オブジェクト型)	4 バイト	オブジェクトを参照するためのアドレスです。
Variant(バリエーション型)		数値の場合は, 倍精度浮動小数点数型と同じで, 文字列の場合は, 文字列型と同じです。

2. 1. 2 変数の宣言

変数は使う前に必ず宣言してから使うようにしましょう。VBA では変数は宣言しなくても使えますが, 宣言してから使うようにした方が, 間違いを発見しやすくなります。

変数の宣言には, Dim 文を使います。Dim の構文は以下の通りです。

Dim 変数名 [As データ型]

□で囲まれた部分は省略可能です。

例

```
Dim Mean As Single
Dim i As Integer
Dim x As Double, y As Double, s As String
```

悪い例

```
Dim j, k As Integer 'この場合, jはInteger型ですがkはVariant型になります。
```

変数を使う場合に必ず宣言を必要とするように強制するには, モジュールの先頭で次のように記述します。

Option Explicit

この宣言をしておくと, 宣言をせずに変数を使うとエラーメッセージが表示されます(図 2.1.1)。例えば, タイプミスで変数の綴りを間違えた場合でも, Option Explicit の宣言をしておくと, タイプミスした変数は宣言されていないのでエラーになり, すぐに見つけることができます。

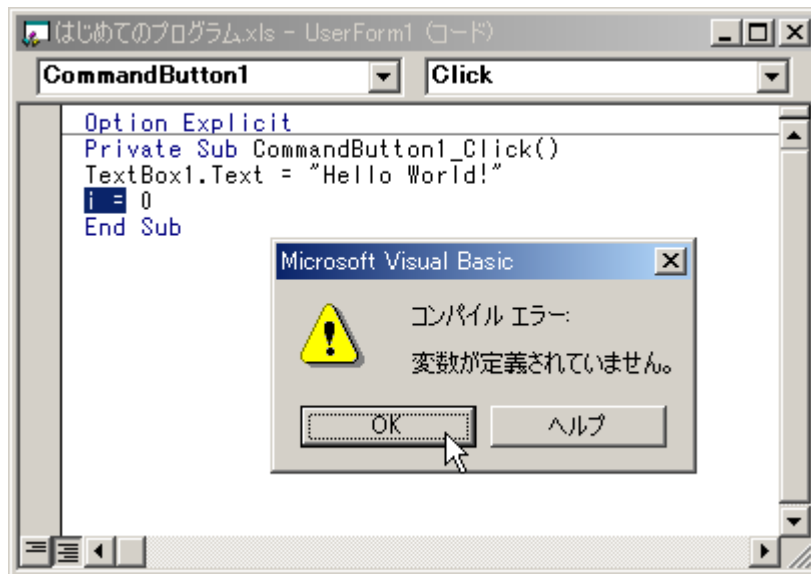


図 2.1.1

2.4 定数

一般的には、プログラム上に直接記述された数値や文字列のことを定数と呼びますが、ここではユーザーが定数を定義する方法について説明します。ユーザーが定数を宣言する場合は、Const キーワードを使います。

```
[Public/Private] Const 定数名 [As データ型] = 値
```

例

```
Const TaxRate As Single = 0.5
Const Morning As String = "おはようございます。"
```

VBA で既に定義されている定数もあります。非常に数多くの定数が定義されていますので、詳しくは Help を見てください。

例

```
'色の定数
vbBlack '黒色
vbRed '赤色
vbWhite '白色 など
'キーコードの定数
vbKeyA 'アルファベットのA
vbKey1 '数字の1
vbKeyF1 'ファンクションキーF1など
```

なお、比較的良好に使われる円周率については、PI()というワークシート関数を呼び出すことで 3.14159265358979 という値を得ることができます。

2.5 演算子

VBA で使われる演算子には、以下のようなものがあります。

表 2.5.1 演算子の一覧表

演算子	機能	使用例
算術演算子		a=10, b=4 とすると
+	2 つの数値の和を求める	a + b 14
-	2 つの数値の差を求める	a - b 6
*	2 つの数値の積を求める	a * b 40
/	2 つの数値の商を求める	a / b 2.5
¥	2 つの数値の商の整数部を求める	a ¥ b 2
Mod	2 つの数値の割り算の余りを求める	a Mod b 2
^	べき乗を求める	a ^ b (a を b 乗する) 10000
文字列連結演算子		a="Good ", b="Morning" とすると
&	2 つの文字列を連結する	a & b "Good Morning"
+		a + b "Good Morning"
比較演算子		a=5 b=8 とすると
=	等しい	a = b False
<>	等しくない	a <> b True
<	より小さい	a < b True
>	より大きい	a > b False
<=	以下	a <= b True
>=	以上	a >= b False
Like	文字列の比較を行う	a="Good", b="G*" の場合(*はメタ文字) a Like b は True
Is	オブジェクト変数の比較を行う	Dim aObject, bObject Set bObject=aObject の場合 a Is b True
論理演算子		a=2,b=3, c=4, d=5 とすると
Not	論理否定	Not a = b True
And	論理積	a <= b And c <> d True

Or	論理和	a = b Or c = d	False
Xor	排他的論理和	a <= b Xor c <= d	False
Eqv	論理等価演算	a <= b Eqv c <= d	True
Imp	論理包含演算	a <= b Imp c >= d	False

2.7 配列

2.7.1 1次元配列

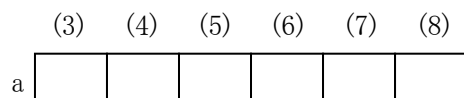
配列は、たくさんの変数を1つの変数名で利用できるようにしたもの、と考えることができます。線形代数のベクトルや行列を、思い浮かべてもらってもいいでしょう。

配列の宣言は、次のように行います。

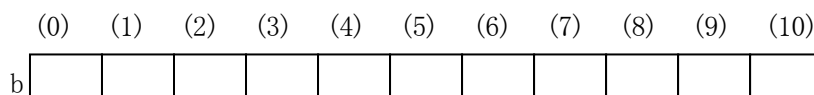
Dim 変数名(要素の初めの番号 To 要素の終わりの番号) As データ型

例

```
Dim a(3 To 8) As Single 'a(3) ~ a(8)まで6つのSingle型の要素を持つ配列を宣言したことになります。
```



```
Dim b(10) As Integer '要素の初めの番号とToを省略した場合、要素は0から始まるとみなされます。この場合はb(0)~b(10)までの11の要素を持つ配列を宣言したことになります。
```



要素の初めの番号を1からはじめたい場合には、モジュールの先頭で次のように書きます。

Option Base 1

この宣言で配列要素の先頭は、(1)から始まるようになります。この指定がなければ(0)から始まります。指定できるのは1か0しかありません。

2.7.2 多次元配列

2次元以上の配列も作成することができます。

例

`Dim x(4, 5) As Long` '下記の5行6列の表のような配列を宣言したことになります。

	(0)	(1)	(2)	(3)	(4)	(5)
(0)	x(0,0)	x(0,1)	x(0,2)	x(0,3)	x(0,4)	x(0,5)
(1)	x(1,0)	x(1,1)	x(1,2)	x(1,3)	x(1,4)	x(1,5)
(2)	x(2,0)	x(2,1)	x(2,2)	x(2,3)	x(2,4)	x(2,5)
(3)	x(3,0)	x(3,1)	x(3,2)	x(3,3)	x(3,4)	x(3,5)
(4)	x(4,0)	x(4,1)	x(4,2)	x(4,3)	x(4,4)	x(4,5)

この表からも分かるように、2次元配列の宣言では

`Dim` 変数名(最後の行番号,最後の列番号)

という形になります。

また3次元以上の配列も利用することができます。次元数の最大値は60です。

```
Dim x(3, 5, 7) As Double '3次元の配列
Dim y(2, 4, 6, 8) As String '4次元の配列
```

例

配列の要素数の最大値は、使用しているコンピュータのメモリの容量によって決まります。上限を超えて大きな配列を宣言すると「メモリが不足しています。」(図 2.7.2)というエラーが出ます。エラーが出なくても、不必要に大きな配列を宣言すると実行速度が極端に低下しますので注意しましょう。

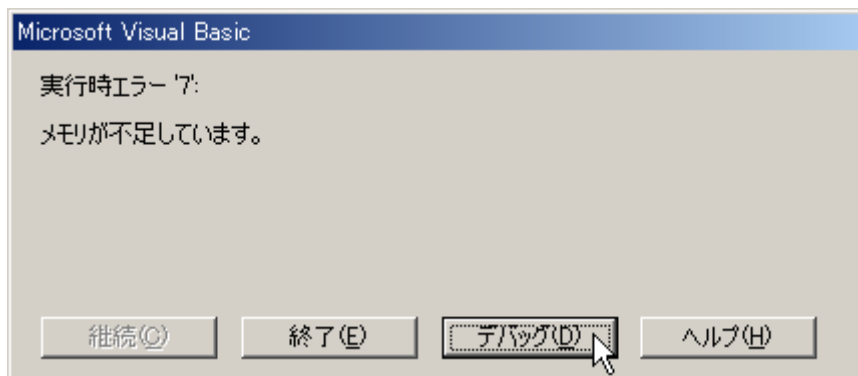


図 2.7.2

2. 7. 3 動的配列

必要な配列の大きさが、プログラムの実行時にならないと分からない場合にはどうすればいいのでしょうか。そのような場合には、動的配列を使うことができます。

動的配列は、最初に要素数を指定しないで配列を宣言しておき、実行時に要素数が確定したときに、その配列を再宣言します。

例

```
Dim z() As Double      '要素数を指定しないで配列を宣言しておく
.
.
.
ReDim z(n, m) As Double '要素数が決まった時に、もう一度宣言しなおす。
```

ReDim 文の構文は、下記のとおりです。この文は配列の大きさや次元数を変えるために何度でも使うことができます。

ReDim [Preserve] 変数名(要素数) [As データ型]

ReDim 文は Dim 文とほぼ同じですが、違うのは Preserve というオプションを指定できることです。Preserve をつけると、既に配列に格納されているデータを保持したまま配列の大きさや次元数を変えることができます。Preserve がない場合は、配列は初期化されます。

2. 8 コメント

コメントというのは、プログラムを読みやすくするために記述する、解説文のことです。コメント文は、プログラムの実行には何も影響を与えません。プログラムを作成している時点では、よく分かっているつもりでも、後で読み返すとどのような処理をしているのかさっぱり分からない、というようなことがあります。そのようなことを防ぐために、きちんと分かりやすいコメントを書く習慣を付けましょう。

「Rem」あるいは「' (シングルクォーテーション)」に続く文字列は、コメントとみなされ、緑色の文字で表されます。

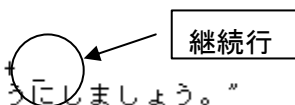
例

```
Rem Rem1に続く文字列はコメント文です。
' コメント文はプログラムを分かりやすくする解説です。
a = b + c ' 文の途中からでもコメントを入れることができます
```

2. 9 継続行

プログラムの1行が非常に長くなると、画面をスクロールして見なければならず、見づらくなってしまいます。そういう場合は、1行のプログラムを2行以上に書くことができます。「_(半角の空白とアンダースコア)」を行の最後にすると、次の行と続いていて同一の行とみなされます。

```
string1 = "長い行は見づらくなります。"
          "  適当なところで継続行を使うようにしましょう。"
```



例

2. 10 条件判断

条件判断とは、変数の値などによってプログラム中の処理を変えるような操作をいいます。コンピュータが他の機械と大きく異なるのは、この条件判断ができるところにあるのです。

2. 10. 1 If 文

If 文の構文は以下のような2通りのものがあります。

```
If 条件式 Then 実行文 [ Else 実行文 EndIf ]
```

```
If 条件式 Then 実行文 [ ElseIf 条件式 Then 実行文 [ Else 実行文 ] EndIf ]
```

If のあとの条件式には、比較演算子や論理演算子を使って答えが真(True)か偽(False)であるような式か、あるいは Boolean 型の変数が入ります。条件式の答えが真であれば、Then の後の実行文を実行し、偽であれば何もしないか、Else の後の実行文あるいは ElseIf のあとの条件式の評価を行います。

例

```
If a = 10 Then b = 10      '変数aの値が10と等しい場合変数bにも10を代入する
```

```
If a = 10 Then           ' 変数aの値が10と等しい場合
  b = 10                 ' 変数bに10を代入する
Else                     ' そうでなければ
  b = 5                 ' 変数bに5を代入する
End If
```

```
a = b = 10              '変数bが10と等しい場合aにはTrueが、bが10等しくな
                        ' ければaにはFalseが入る
```

```

Dim a As Boolean      '変数aをBooleanとして宣言
:
If a Then c = 5      ' Boolean型の変数の場合は、その変数が
                    ' TrueかFalseかでIfの判断が行われ、
                    ' ここではaがTrueであればc = 5 が実行されます。

```

次は `ElseIf` の例ですが、`ElseIf` を使った場合、注意しなければいけないのは `If` や `ElseIf` の条件式のうち一つでも真の場合があれば、それ以降の条件式については判定を行わないことです。下記の例は `tokuten` が 80 以上なら `seiseki` に“優”を、70 以上 80 未満の場合は“良”、60 以上 70 未満は“可”、60 未満は“不可”を代入するプログラムです。

ElseIf の正しい例

```

If tokuten >= 80 Then      '変数tokutenの値が80以上であるなら
    seiseki = "優"        '変数seisekiに"優"を代入する
ElseIf tokuten >= 70 Then  '80以上でなく、70以上であるなら
    seiseki = "良"        '変数seisekiに"良"を代入する
ElseIf tokuten >= 60 Then  '80以上でも70以上でもなく60以上なら
    seiseki = "可"        '変数seisekiに"可"を代入する
Else                        '上記の条件のいずれにも偽である場合
    seiseki = "不可"      '変数seisekiに"不可"を代入する
End If

```

下記の誤った例では、最初に `tokuten` が 60 以上かどうか判定され、60 以上の場合は全て “可”なり、60 未満のものだけが“不可”となってしまいます。それは、`ElseIf` ではいずれかの条件式が真(True)となると、それ以降に記述されている `ElseIf` の条件式は、評価されなくなるからです。

ElseIf の誤った例

```

If tokuten >= 60 Then
    seiseki = "可"
ElseIf tokuten >= 70 Then
    seiseki = "良"
ElseIf tokuten >= 80 Then
    seiseki = "優"
Else
    seiseki = "不可"
End If

```

2. 10. 2 Select Case 文

Select Case 文は, Elseif と同様に多方向への分岐を行う制御文です。構文は以下のとおりです。

```
Select Case 式 1
    Case 式 2
        実行文
        :
        実行文
    Case 式 3
        実行文
        :
        実行文
    Case 式 n
        :
        :
        :
    Case Else
        :
        :
        :
EndSelect
```

Select Case 文の式 1 には, 式や変数を記述します。この式 1 と式 2~式 n までで等しい式の Case に続く文を実行します。Case Else は必要がなければ, 記述しなくてもかまいませんが, 想定外の値が出た場合 Case Else に跳ぶようにしておくことと予期せぬエラーを防ぐことができます。

Else If で用いたものと同じ例を, Select Case で記述してみましょう。下記の例は tokuten が 80 以上なら seiseki に“優”を, 70 以上 80 未満の場合は“良”, 60 以上 70 未満は“可”, 60 未満は“不可”を代入するプログラムです。例 1 のプログラムは数値の大きいものから判定しています。それに対して, 例 2 のものは数値の小さいものから判定しています。例 3 は, 値の範囲を指定して分岐するものです。

ここで **Is**, **To** というキーワードが使われているのに注意して下さい。**Is** は比較式を使って値を指定したい場合に使います。**To** は最小値と最大値を指定し、値の範囲を示したい場合に使います。

例 4 はもっとも一般的に良く使われる **Select Case** 文の使い方です。

例 1

```
Select Case tokuten
  Case Is >= 80
    seiseki = "優"
  Case Is >= 70
    seiseki = "良"
  Case Is >= 60
    seiseki = "可"
  Case Is < 60
    seiseki = "不可"
  Case Else
    seiseki = "入力エラー"
EndSelect
```

例 2

```
Select Case tokuten
  Case Is < 60
    seiseki = "不可"
  Case Is < 70
    seiseki = "可"
  Case Is < 80
    seiseki = "良"
  Case Is >= 80
    seiseki = "優"
  Case Else
    seiseki = "入力エラー"
EndSelect
```

例 3

```
Select Case tokuten
  Case 80 To 100
    seiseki = "優"
  Case 70 To 79
    seiseki = "良"
  Case 60 To 69
    seiseki = "可"
  Case 0 To 60
    seiseki = "不可"
  Case Else
    seiseki = "入力エラー"
EndSelect
```

例 4

```
Select Case i
  Case 1
    days = 31
  Case 2
    days = 28
  Case 3
    days = 31
  Case 4
    days = 30
  Case 5
    days = 31
  Case 6
    days = 30
  Case 7
    days = 31
  Case 8
    days = 31
  Case 9
    days = 30
  Case 10
    days = 31
  Case 11
    days = 30
  Case 12
    days = 31
EndSelect
```

2. 11 繰り返し

条件判断がコンピュータのコンピュータらしいところなのですが、繰り返しはコンピュータが機械であることを思い出させてくれるところで、同じ処理を何度も何度も繰り返し実行していくところです。

2. 11. 1 For ... Next

For ... Next は反復回数が、事前に分かっている場合に使われる繰り返しの方法です。

For ... Next は次のような構文です。

For 変数 = 初期値 To 終値 [Step 増分]
実行文
...
実行文
Next [変数]

例 1

アクティブなシート上に下図のようなデータがある場合、これらのデータを 1 次元配列の変数に代入するプログラムを作ってみましょう。

	A	B	C	D	E	F	G	H	I	J
1	1	2	3	4	5	6	7	8	9	10

図 2.11.1 データの表

```
Dim a(1 To 10) As Integer
Dim i As Integer

For i = 1 To 10
    a(i) = Cells(1, i)
Next i
```

例 2

次にシート上に下図のような 2 次元のデータがある場合、このデータを 2 次元配列に代入するプログラムを作ってみましょう。この場合は、For...Next の繰り返しを二重に使用します。

	A	B
1	1	2
2	2	3
3	3	4

図 2.11.2 データの表

```
Dim a(1 To 3, 1 To 2) As Integer
Dim j As Integer, i As Integer

For j = 1 To 2
    For i = 1 To 3
        a(i, j) = Cells(i, j)
    Next i
Next j
```

上のプログラムでは、外側の繰り返しで初めにjが1となり、次に内側の繰り返しでiが1, 2, 3, 4, 5と変わっていき、iが5までくると内側の繰り返しが終了し、外側の繰り返しでjが2となります。a(i,j)は表 2.11.1 のように変化していきます。

表 2.11.1 繰り返しの値の変化

繰り返しの回数	j	i	a(i,j)の値
1	1	1	1
2	1	2	2
3	1	3	3
4	2	1	2
5	2	2	3
6	2	3	4

なお、例 1、例 2 では「Step 増分」の部分が省略されています。省略された場合の増分は、既定値 1 となります。次のように増分を 2 として、1 つ飛ばしに指定したり、負の値を指定し大きいものから、小さいものへと繰り返すこともできます。

例 3 増分が 2 刻みの場合

```
For i = 2 To 10 Step 2
```

...

```
Next i
```

例 4 増分が負の値

```
For i = 10 To 1 Step -1
```

...

```
Next i
```

ここで、例 1 と例 2 で出てきた **Cells** というキーワードについて説明しておきます。**Cells** とは、ワークシート上のセルの属性(プロパティ)を示すもので、そのセルの値、大きさ、色、フォント、フォントサイズといったプロパティがあるところです。

シート上の特定のセルのデータを取得するには、**Cells** プロパティを使います。**Cells** プロパティの構文は、下記のとおりです。

[オブジェクト名].Cells(行番号,列番号)

オブジェクト名は、通常はシート名が入りますが、何も指定がなければアクティブなシートが指定されたときみなされます。また、列番号は左から順に数えた場合の列番号を指定します。ですから、

列 A は列番号 1, 列 B は 2, 列 C は 3 というように指定します。

例

`Worksheets("Sheet1").Cells(4, 5) = 10` ' Sheet1のセルE4に10を代入しています。

2.11.2 Do ... Loop

`For ... Next` 文は, 繰り返しの回数が分かっている場合に使用しますが, 繰り返しの回数が分からないが, 定められた条件式が真の間繰り返したり, 条件式が真になるまで繰り返したりする場合, `Do ... Loop` を使用します。`Do ... Loop` には, 条件式が真の間処理を繰り返す `While` 型と, 条件式が真になるまで処理を繰り返す `Until` 型があり, それぞれに繰り返しの最初で条件判断を行うものと最後で条件判断を行うものの計 4 つのタイプがあります。繰り返しの最初で条件判断を行う場合は, 条件によってはその繰り返しが 1 回も実行されない場合があるのに対して, 条件判断を最後に行う場合は, 1 回目の繰り返しは無条件に実行されるという違いがある。

表 2.11.3 Do ... Loop の 4 つのタイプ

	While 型 条件式が真の間反復	Until 型 条件式が真になるまで反復
最初に条件判断	Do While 条件式 実行文 ... 実行文 Loop	Do Until 条件式 実行文 ... 実行文 Loop
最後に条件判断	Do 実行文 ... 実行文 Loop While 条件式	Do 実行文 ... 実行文 Loop Until 条件式

例 1

アクティブなシート上に下図のようなデータがある場合、これらのデータを 1 次元配列の変数に代入するプログラムを作ってみましょう。その際、データがいくつあるか分からないが、データの無

	A	B	C	D	E	F	G	H	I	J	K
1	1	2	3	4	5	6	7	8	9	10	

図2. 11. 2

いセルがあれば、そこでデータは終わりにします。

While による前判定

```
Sub t()  
  Dim i As Integer  
  Dim j As Integer  
  Dim a(1 To 20) As Integer  
  
  i = 1  
  j = 1  
  
  Do While Cells(1, j) <> ""  
    a(i) = Cells(1, j)  
    j = j + 1  
    i = i + 1  
  Loop  
  
End Sub
```

While による後判定

```
Sub t()  
  Dim i As Integer  
  Dim j As Integer  
  Dim a(1 To 20) As Integer  
  
  i = 1  
  j = 1  
  
  Do  
    a(i) = Cells(1, j)  
    j = j + 1  
    i = i + 1  
  Loop While Cells(1, j) <> ""  
  
End Sub
```

Until による前判定

```
Sub t()  
  Dim i As Integer  
  Dim j As Integer  
  Dim a(1 To 20) As Integer  
  
  i = 1  
  j = 1  
  
  Do Until Cells(1, j) = ""  
    a(i) = Cells(1, j)  
    j = j + 1  
    i = i + 1  
  Loop  
  
End Sub
```

Until による後判定

```
Sub t()  
  Dim i As Integer  
  Dim j As Integer  
  Dim a(1 To 20) As Integer  
  
  i = 1  
  j = 1  
  
  Do  
    a(i) = Cells(1, j)  
    j = j + 1  
    i = i + 1  
  Loop Until Cells(1, j) = ""  
  
End Sub
```

2. 12 その他の制御文

2. 12. 1 GoTo 文

GoTo 文は無条件に分岐する文です。構文は次のような形式で、GoTo 文のあとに記述されたラベルと同じラベルの行まで制御が移ります。

```
GoTo ラベル
...
ラベル:
```

例 1

下記の例では Cell の値が負のとき Negative というラベルの行まで跳びます。ですから、Cell の値が負の場合は、それ以降の For ... Next は実行されなくなります。

```
Sub t()
Dim a(1 To 10) As Integer
Dim i As Integer

For i = 1 To 10
    If Cells(1, i) < 0 Then GoTo Negative
    a(i) = Cells(1, i)
Next i
Negative:

End Sub
```

2. 12. 2 Exit 文

For ... Next, Do ... Loop などの繰り返しから途中で抜け出したり、Sub プロシージャや関数プロシージャから抜け出す文です。構文としては、以下の 5 種類があります。

表 2. 12. 2 各種の Exit 文

文	説明
Exit For*	For ... Next の繰り返しから抜け、Next 文の次に制御が移ります。
Exit Do*	Do ... Loop の繰り返しから抜け、Loop 文の次に制御が移ります。
Exit Sub	Exit 文を含む Sub プロシージャから抜け、その Sub プロシージャを呼び出した文の次の文へ制御が移ります。
Exit Function	Exit 文を含む Function プロシージャから抜け、その Function プロシージャを呼び出した文の次の文へ制御が移ります。
Exit Property	Exit 文を含む Property プロシージャから抜け、その Property プロシージャを呼び出した文の次の文へ制御が移ります。

*上記の Exit 文のうち Exit For や Exit Do の場合、二重以上の繰り返しの中で Exit 文を使うとその Exit 文を含む最も内側の繰り返しから抜け出します。

例1 下記の例は二重の繰り返しの中でExit文が使われていますが、この場合Exit文が実行されると、次に実行されるのはNext jの文です。外側の繰り返しまでは抜け出すことはありません。

```
Sub t()
  Dim a(1 To 3, 1 To 2) As Integer
  Dim j As Integer, i As Integer

  For j = 1 To 2
    For i = 1 To 3
      If Cells(i, j) < 0 Then Exit For
      a(i, j) = Cells(i, j)
    Next i
  Next j
End Sub
```

ここへ抜け出す

2.13 Sub プロシージャ

Sub文の構文は下記のとおりです。Sub文はSubプロシージャを作るときに使用します。

```
Sub 名前([ 引数 1,引数 2,...,引数 n ])
    ...
    ...
End Sub
```

Sub文では、Subプロシージャの名前と名前の後の()が必要です。引数リストは、引数がなければ必要ありませんが、()はその場合でも必要です。

Subプロシージャの呼び出し方法には、2通りの方法があります。1つはCall文を使う方法と使わない方法です。

例

下記のようなExample1というSubプロシージャを呼び出します。

```
Sub Example1(a As Integer)
  a = a ^ 2
End Sub
```

Call文を使った呼び出しでは、引数を()で括る必要があります。

```
Call Example1(a)      ' Call文を使った呼び出し
Example1 a            ' Call文を使わない呼び出し
```

VBAのSubプロシージャは、自分自身を呼び出す再帰呼び出しが可能です。再帰呼び出しの例は、次のFunction文で示します。

2. 14 Function 文

Function 文の構文は下記のとおりです。Function 文は Function プロシージャを作るときに使用します。Function プロシージャと Sub プロシージャの違いは、戻り値がありそれによって呼び出したプロシージャに値を戻すことができるという点です。もちろん、Sub プロシージャでも引数を使うことによって値を戻すことはできますが、Function プロシージャの場合はそれ以外に戻り値という形で呼び出したプロシージャに値を戻すことができるのです。

```
Function 関数名([引数 1,引数 2,...,引数 n]) [As 型]
    ...
    ...
    [関数名 = 戻り値]
    ...
End Function
```

例 1 階乗の計算を行う関数です。

```
Function Factorial(n As Integer) As Long
    Dim t As Long
    Dim i As Integer

    t = 1
    For i = 2 To n
        t = t * i
    Next i

    Factorial = t '計算結果を戻り値として戻している
End Function
```

Function プロシージャの呼び出しは、下の例のように関数名の次に()で括って引数を記述します。引数がない場合でも()は必要です。

例 2 関数の呼び出し方

例 1 の関数を呼び出しています。

```
a = Factorial(5)
```

Function プロシージャも Sub プロシージャと同様に、再帰呼び出しができます。再帰呼び出しは、自分自身を呼び出すものです。次の例 3 は、例 1 の階乗を行う関数を再帰呼び出しの形に書き換えたものです。

```

Function Factorial(n As Long) As Long
' 再帰呼び出しによる階乗の計算

    If n = 0 Then
        Factorial = 1
    Else
        Factorial = n * Factorial(n - 1)
    End If

End Function

```

再帰呼び出しは、プログラムの実行効率から考えると必ずしも良くありませんが、再帰呼び出しを使ったほうが、よりわかりやすいプログラムになるアルゴリズムも多いので、うまく使い分ける必要があります。

自分で関数を作成する場合は、**Function** 文を使いますが、多くの便利な関数が VBA で用意されています。また、ワークシートで使われる関数の一部は VBA の中で利用することもできます。ワークシートで使われる関数をワークシート関数と呼びますが、その使い方は標準の VBA 関数と異なります。

ワークシート関数は次のような形で呼び出します。

Application.WorksheetFunction.関数名([引数])

例

```
mean = Application.WorksheetFunction.Average(Range("B2:B51"))
```

上記の例では、セル B2～B51 までの数値の平均を、ワークシート関数を使って求めています。このときに **Range("B2:B51")** は **Range** オブジェクトといい、セル範囲を指定するときに使います。ワークシート関数を使う場合は、必要になることが多いのでよく覚えておいてください。

参考文献

- アスキー書籍編集部 編 1999 *Excel VBA 2000 リファレンス* アスキー 東京
- Dixon, M.R. & MacLin, O.H. 2003 *Visual Basic for Behavioral Psychologists*. Context Press Reno
- 市川伸一, 矢部富美枝編 1985 *パーソナル・コンピュータによる心理学実験入門* ブレーン出版
東京
- 金子秀彬, 吉田俊郎, 伊田政司, 森山哲美 1990 *心理学に必要なコンピュータ技術* 北樹出版
東京
- 北村英哉, 坂本正浩編 2004 *パーソナル・コンピュータによる心理学実験入門 : 誰でもすぐにできるコンピュータ実験* ナカニシヤ出版 京都
- 水野りか 2004 *Web を介してできる基礎・認知心理学実験演習* ナカニシヤ出版 京都
- 野沢晨 1986 *パソコン BASIC 心理学実験* 東海大学出版会 東京
- 田中良久 1975 *BASIC 入門 行動科学のためのコンピュータ・プログラミング入門* 東京大学出版
会 東京
- 吉村浩一, 山上暁 1983 *BASIC 入門* ナカニシヤ出版 京都