生産情報処理実習テキスト

Excel VBA

高齢・障害・求職者雇用支援機構
北海道職業能力開発大学校

ver.201602

ＶＢＡ超簡単テキスト

このテキストは､Visual Basic の学習を目的に作られた ExcelVBA 学習テキストである。そのため、Excel が持つ表計算ソフトとしての機能よりも、GUI を活用したアプリケーション開発に重点をおいている｡このテキストを通して GUI アプリケーション開発の基本を理解してもらいたい。

目　次

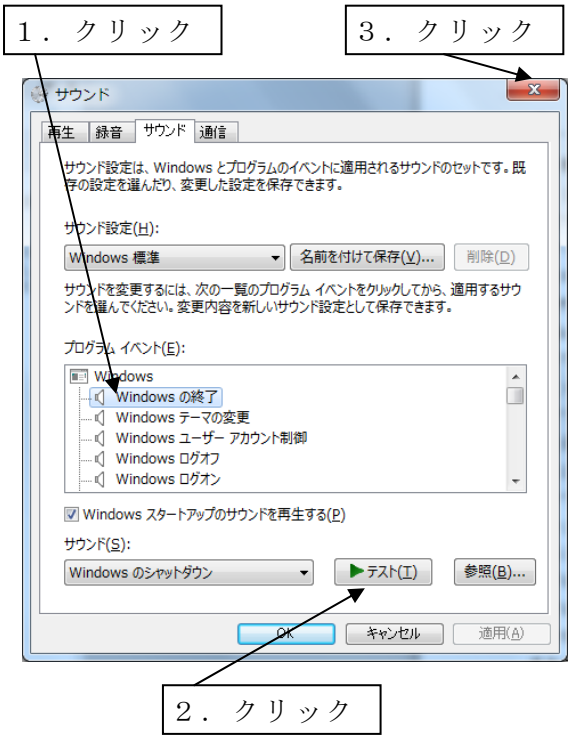
Ａ．はじめに

１．概要

１．１　Windows のプログラム

まず、初めに Windows のプログラムとはどのようなものかを見てみましょう。例えば、「マイコンピュータ」の中にある「コントロールパネル」の「サウンド」を開いてみて下さい。

「サウンドのプロパティ」のウインドウでは、Windows で使われる「音」をいろいろと決めることができます。



このウインドウには「サウンド設定は・・・」や「サウンド設定(H):」などの文字を表示する部分、音を選択する「プログラムイベント」、その音を再生する「テスト(T)」ボタンなどが配置されています。

マウスで「Windows の終了」をクリックし、「テスト(T)」ボタンをクリックすると、Windows を終了したときの音楽が再生されます。これは、「”Windows の終了“が選択され、テストボタンがクリックされたら、終了時の音楽を再生しなさい」という命令が組み込まれているからです。また、ウインドウ右上にある[×]印の「閉じるボタン」をクリックすると、開いていたウインドウが閉じます。これも「” 閉じるボタン“がクリックされたら、そのウインドウを閉じなさい」という命令が入っているからです。

このように Windows のプログラムには、操作するためのウインドウがあり、そこに文字を表示する部分、選択肢を選択する部分、動作を実行するボタンなどを配置します。さらに、「この部分がクリックされたらこの動作をする」といった命令がウインドウに組み込まれています。

従って、Windows のプログラムを作るためには、

1)ウインドウをデザインする部分

2)実際の動作を決める部分

の二つを作らなければなりません。

このように作られたウインドウがいくつか集まって（もちろんウインドウが一つしかないものもあります)、一つのアプリケーションソフトになります。

１．２　VBA による Windows プログラム

Windows のプログラムを作成するには VisualBasic や VisualC++などの専用のソフトウエアが必要になります。しかし、簡易的なものであれば一般的に使用されているワープロソフトや表計算ソフトなどに付属されているアプリケーション用プログラム言語を使っても Windows プログラムを組むことが出来ます。実行ファイル（exe ファイル）を作ることは出来ませんが、同じワープロソフトや表計算ソフトがあれば、容易にそのプログラムを実行することは可能です。

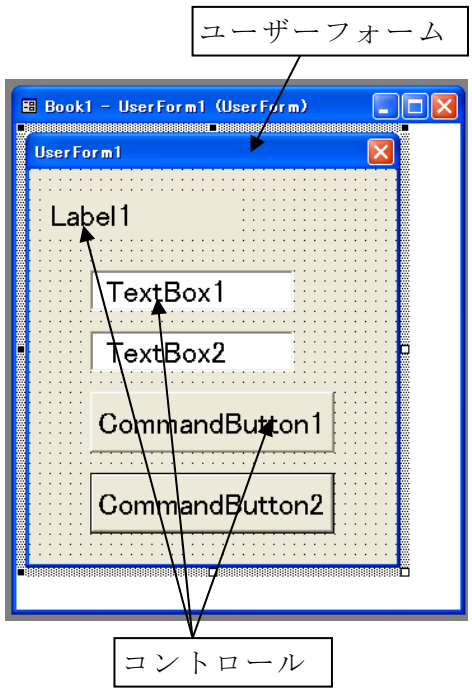
現在、広く使われているワープロソフト、表計算ソフトにマイクロソフト社の「Word」、「Excel」があります。これらのソフトウエアには VBA（Visual Basic for Application）と呼ばれる専用のプログラム言語が付属されています。VBA は Basic 言語であるため、初心者でも容易に覚えることができ、また、プログラム専用言語の VisualBasic にも類似しているため、VBA の技術を VisualBasic に発展させることも可能です。

表計算ソフト Excel にはプログラミング言語として「ExcelVBA」が用意されています。Excel そのものが表計算ソフトであるため、ExcelVBA にはデータ管理機能や専用の計算機能も持っています。データベースソフトの「Access」のようなデータベース専用の機能はありませんが、この ExcelVBA を上手に活用すれば、かなり広い範囲のデータ処理や事務処理が可能になります。

ExcelVBA ではデータ処理以外にも、計測器のデータ自動読み取りや機械の制御など様々なことをすることができます。このテキストでは、VBA の基本を習得し、データ管理、自動計測、機械制御などの各種アプリケーションの開発について説明します。

１．３　Windows プログラムの内容

ExcelVBA（以下、VBA と呼ぶ）で Windows のプログラムを作る場合でも、ウインドウをデザインする部分と実際の動作を決める部分の二つが必要となります。



VBA では、ウインドウのことを「ユーザーフォーム」と呼びます。ユーザーフォームの中には数字を入力する部分、文字を表示する部分、何らかの動作を実行するボタンなど、たくさんの部品が使われています。ユーザーフォームの内部に配置されているこれらの部品を「コントロール」といいます。また、ユーザーフォームとコントロールを合わせて「オブジェクト」といいます。一つのプログラムには一つ以上のユーザーフォームがあり、そのユーザーフォームにコントロールを配置してウインドウの外観を作っていきます。

それぞれのオブジェクトには大きさ、表示する文字列、色など、いろいろな性質を持っています。これらの性質を「プロパティ」といいます。このプロパティはあらかじめ設定したり、また、プログラム実行中に

変更したりします。

VBA を使って Windows のプログラムを作成するとき、第 1 の「ウインドウをデザインする部分」では

1)ユーザーフォームを用意する

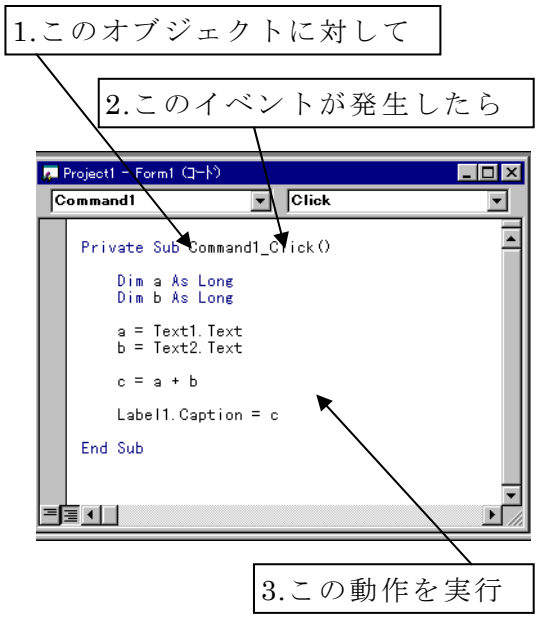
2)コントロールを配置する

3)プロパティを設定する

ことによって、ウインドウの外観をデザインします。

Windows のプログラムでは、プログラムを実行してもウインドウが開かれるだけで、普通は何の動作もしません。操作している人が「マウスをクリックする」などの何らかの働きかけをすることによって、初めてプログラムが動作します。

VBA で作られたユーザーフォームでも、オペレーターが「実行」ボタンなどのオブジェクトに対してクリックなどの働きかけを行ったり、もしくは Windows 自身が何らかの働きかけをすることによって、決められた動作をしていきます。このような外部からの働きかけのことを「イベント」といいます。このようなイベントによって何かを実行する、というようなプログラムを「イベントドリブン型」といいます。プログラムを組むときには、まず「どのオブジェクトに対し、どのようなイベントが発生したら」を決めていきます。

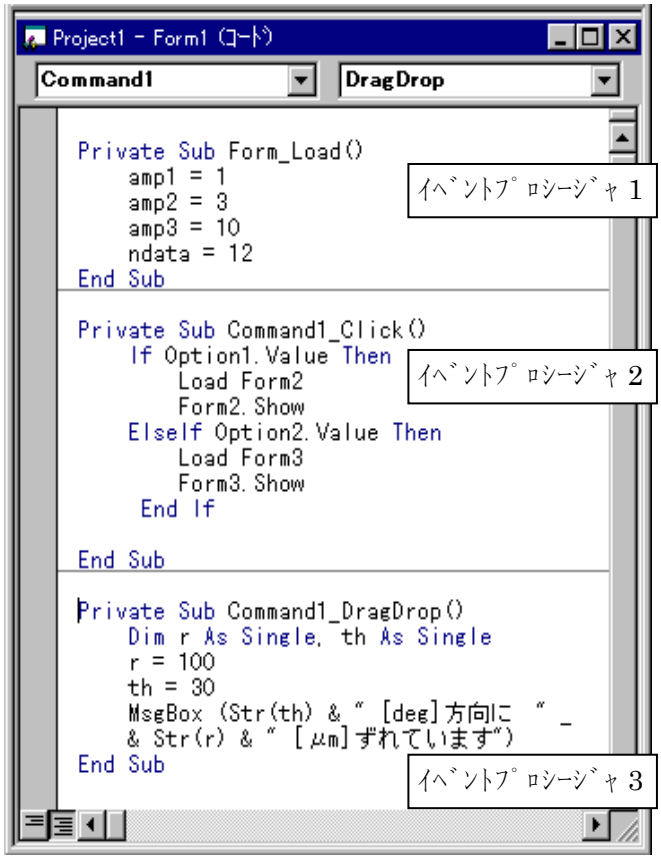


プログラムの動作はたくさんの命令語（コマンド）によって指示されます。これらの命令語には「オブジェクト」、「プロパティ」、「メソッド」、「ステートメント」、「イベント」、「関数」、「演算子」などがあります。これらの命令語を組み合わせたものを「コード」といい、コードによってプログラムの動作が決まります。

メソッドはオブジェクトの動作を指示する命令語、またステートメントはプログラムの流れを制御したり、変数の宣言などを行う命令語です。イベントはオブジェクトに対する働きかけのことです。オブジェクトはウインドウの外観を構成する要素、プロパティはそれぞれのオブジェクトの特性を決める要素です。これらの命令語の詳細は、VBA のヘルプファイルの「ランゲージリファレンス」に書かれています。

命令語を組み合わせてコードを作るとき、それぞれの動作に対応したコードをひとかたまりにすることによって、簡単にコードを組むことができます。このコードのブロックを「プロシージャ」と言います。プロシージャを使ってコードを組むことにより、コードがわかりやすくなります。また、同じ動作を何回もさせるとき、一つのプロシージャを共用することにより、コードを短くすることができます。

プロシージャには、「Sub プロシージャ」、「Function プロシージャ」、「Property プロシージャ」があります。Sub プロシージャはある処理をするプロシージャ、Function プロシージャはある処理をしたあとその結果を返すプロシージャです。この二つはよく使われます。



VBA では「このオブジェクトに対し、このようなイベントが発生したら、このように動作しなさい」という一連の動作を一つの Sub プロシージャとして指示します。このような Sub プロシージャを特に「イベントプロシージャ」といいます。いくつかのイベントプロシージャを組み合わせて、一つのコードを作ります。

説明が長くなりましたが、VBA で Windows のプログラムを作成するときの、第２の「実際の動作を決める部分」では

1)動作を引き起こすイベントを指定する

2)動作内容をコードで記述する

ことによって、プログラムの動作を決めていきます。

２．まず、やってみる

２．１　VBA の立ち上げ

それでは早速、VBA を立ち上げてみます。VBA は直接立ち上げるのではなく、まず始めに Excel を立ち上げます。以下、Excel2010 での立ち上げ方です。

Excel2010 が立ち上がったならば、リボンに「開発」タブがあるかを確認します。「開発」タブがないときは、以下の手順で「開発」タブを有効にします。

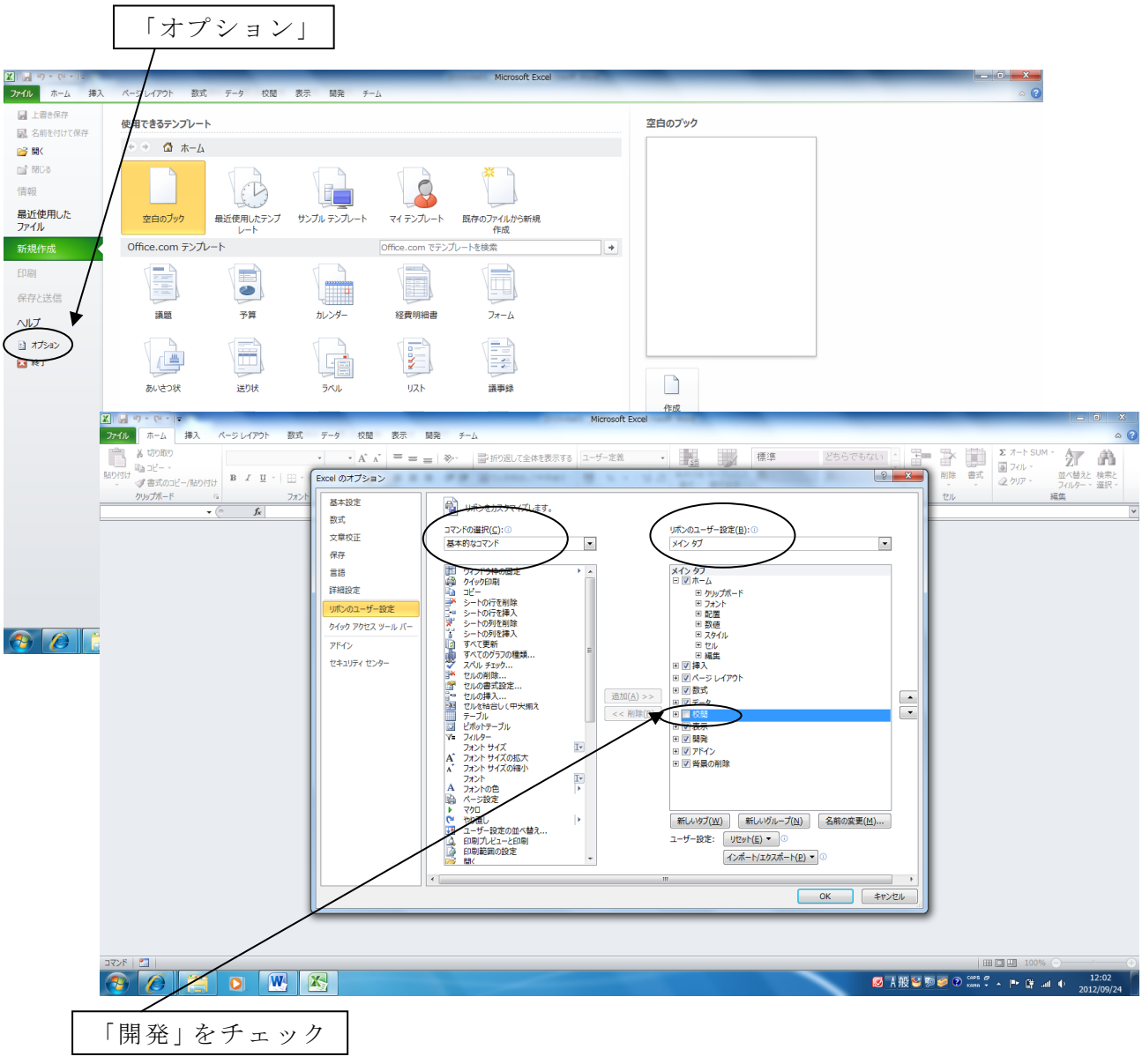
1)「ファイル」タブの「オプション」を選択して「Excel のオプション」ダイアログ ボックスを表示します。

2)ダイアログ ボックスの左側にある「リボンのユーザー設定」をクリックします。

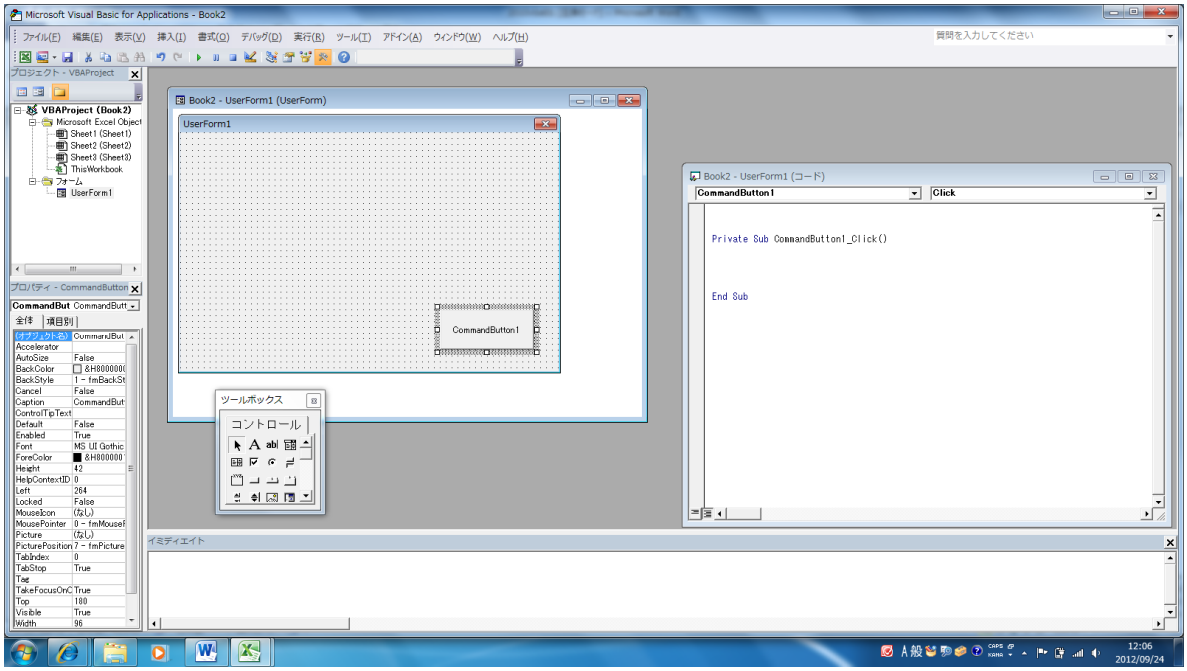
3)ダイアログ ボックスの左側にある「コマンドの選択」で、「基本的なコマンド」を選択します。

4)ダイアログ ボックスの右側にある「リボンのユーザー設定」で、「メイン タブ」を選択し、次に「開発」チェック ボックスをオンにします。

5)「OK」をクリックします。



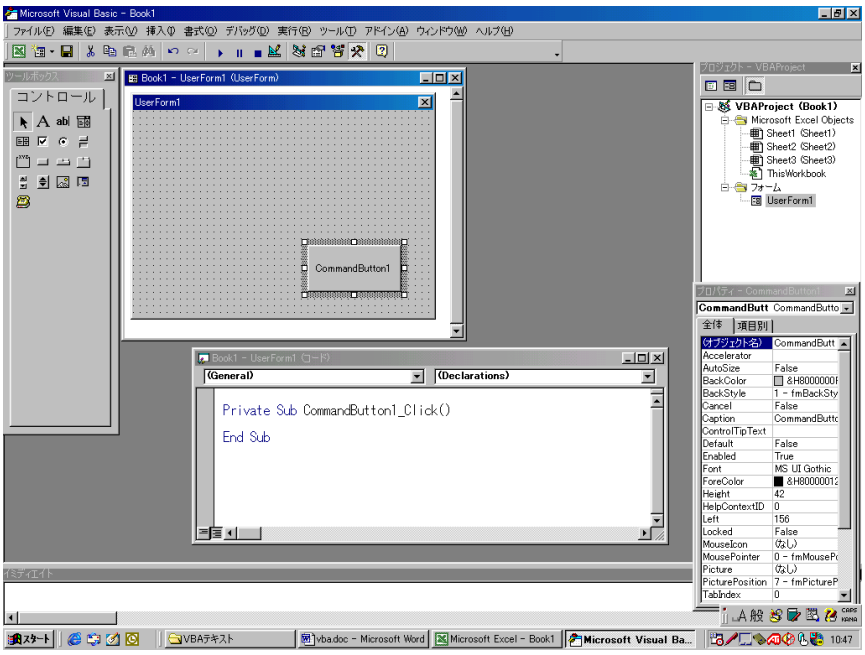
Excel のリボンに「開発」が表示されるので、「開発」→「Visual Basic」をクリックすると、VBA プログラムを組むための編集ウインドウが開かれます。この編集ウインドウを VBE（Visual Basic Editor）といいます。



Excel2007 の場合は、画面左上の「office ボタン」をクリックし、「Excel のオプション」→「基本設定」→「「開発」タブをリボンに表示する」にチェックを入れて「OK」をクリックします。すると「開発」リボンが表示されます。

Excel2007 以前のバージョンでは、メニューの「ツール」→「マクロ」→「Visual Basic Editor」をクリックすると、VBE が表示されます。

VBE では、Excel のバージョンの違いによる大きな変化はないので、ここから先は古いバージョンも交えて説明していきます。



VBE には、メニュー、ツールバー、その他いくつかの要素が組み込まれています。これらの要素によって VBA プログラムを容易に作成することができるようになります。これらの要素が必ずしも表示されているとは限りません。もし表示されていない要素があれば、メニューの「表示」から必要な要素を選択して下さい。

VBA プログラミングで必要となる主な要素を以下に示します（下図を参照）。

1)メニュー　：　画面一番上の「ファイル(F)」、「編集(E)」・・・・
　VBA の様々な機能を選択し、実行します

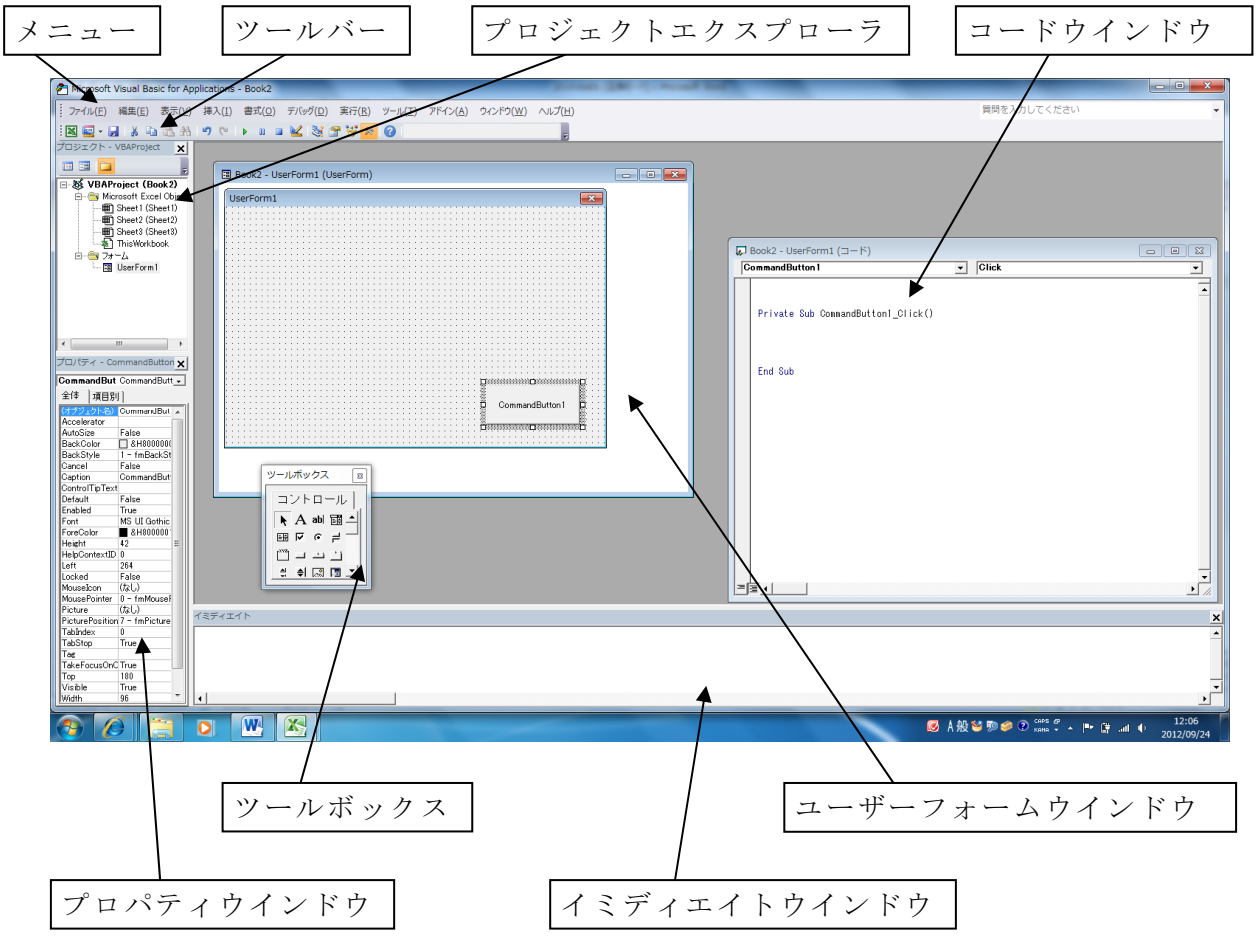
2)ツールバー　：　メニューの下にあるいくつかのアイコン
　VBA でよく使う機能をただちに実行します

3)ツールボックス　：　画面左端
　ユーザーフォームに配置するコントロールを選択します

4)プロジェクトエクスプローラ　：　画面左上
　プロジェクトに含まれるフォームやコードなどの表示や管理をします

5)プロパティウインドウ　：　画面左下
　フォームやコントロールのプロパティを設定します

6)イミディエイトウインドウ　：　画面下
　変数の値やオブジェクトのプロパティを確認したり、変更したりします



２．２　プログラムの作成

それでは早速、プログラムを作成してみます。

（１）ユーザーフォームウインドウの用意

1)Excel を立ち上げ、VBE を開きます。

2)VBE が開いたら、メニューの「挿入」→「ユーザーフォーム」をクリックし、新しいユーザーフォームを表示させます。

（２）コントロールの配置

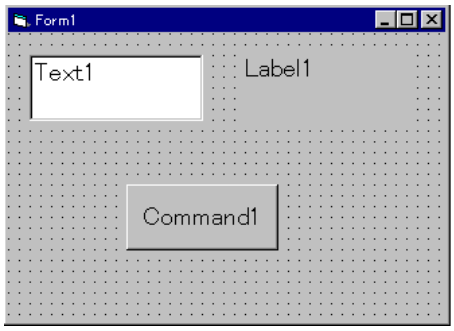
画面にユーザーフォームウインドウ(UserForm１)が表示されたならば、この UserForm１にいろいろなコントロールを配置していきます。

1)ツールボックスでコントロールを選択し、ユーザーフォーム上でドラックして配置します。よく使われるコントロールとして、「Label」、「TextBox」、「Frame」、「CommandButton」、「CheckBox」、「OptionButton」などがあります。

2)配置したコントロールを一度クリックすると、ハンドル(８個の小さい四角)が表示されます。この状態を「コントロールが選択された」といいます。

3)選択されたコントロールの上でマウスをドラックすると、コントロールが移動します。

4)ハンドルの上にマウスポインタを移動して細い矢印に変わったときにドラックすると、コントロールの大きさが変わります。



5)コントロールを選択し、「Delete」キーを押すと、選択されたコントロールが消えます。

6)それでは一旦、すべてのコントロールを消去し、「Label」、「TextBox」、「CommandButton」を一つずつ配置して下さい。

（３）ユーザーフォーム、コマンドボタンのプロパティの設定

次にコントロールのプロパティを設定してみます。

1)ユーザーフォーム上のコマンドボタンを選択し、プロパティウインドウが「CommandButton1」になったことを確認して下さい。

2)「Caption」プロパティを選択し、キーボードから「実行」と入力して下さい(「Caption」の項目に表示されている「CommandButton1」をダブルクリックするとカーソルが現れる)。

3)ユーザーフォーム上のテキストボックスを選択し、プロパティウインドウが「TextBox1」になったことを確認して下さい。

4)「Text」プロパティを選択し、キーボードから「テキスト」と入力して下さい。

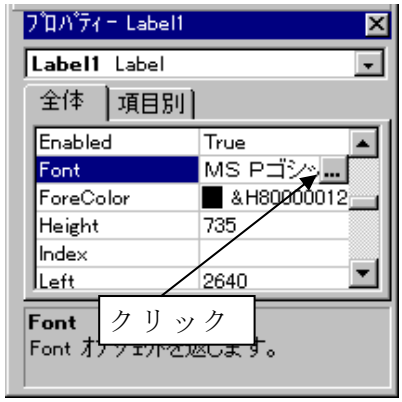
5)ユーザーフォームを選択し、プロパティウインドウが「UserForm1」になったことを確認して下さい。

6)「Caption」プロパティを選択しキーボードから「文字の入出力」と入力して下さい。

7)ユーザーフォーム上のラベルを選択し、プロパティウインドウが「Label1」になったことを確認して下さい。

8)「Font」プロパティを選択し、右側に現れたボタンをクリックして下さい。

9)「フォント」のダイアログボックスが表示されたら、適当にフォントを設定して下さい。(例えば、フォント名：MSP 明朝、スタイル：斜体、サイズ：16)。



（４）コードの作成

最後に、ユーザーフォームのデザインが終わったら、コードを使って実際の動作を決めていきます。以下の手順に従ってコードを作成していきます。

1)メニューの「表示」→「コード」をクリックしてコードウインドウを表示させます。

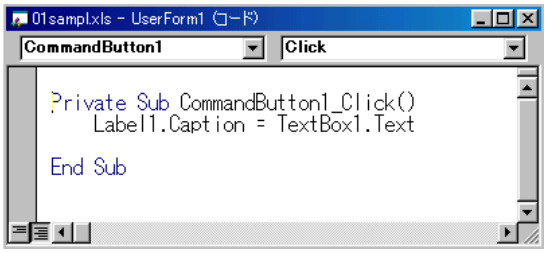
2)コードウインドウ左上でオブジェクトを指定し、右上でイベントを指定します。オブジェクトを「CommandButton1」に、イベントを「Click」にして下さい。(最初は「Click」が選択されています)。このとき、不要なイベントプロシージャも作られてしまうので、不要なものは削除してください。

3)コードウインドウに

```
Private Sub CommandButton1_Click()
    Label1.Caption = TextBox1.Text
End Sub
```

と、入力してください。

入力後、コードウインドウを閉じて下さい。



（５）動作確認、プログラム保存

それでは、出来上がったプログラムの動作確認をしてみます。

1)ユーザーフォームを選択して、メニューの「実行」→「Sub/ユーザーフォームの実行」、もしくはツールバーの右向き三角形をクリックして下さい。今作ったプログラムが実行します。

2)テキストボックスをクリックし、カーソルがでたら適当な文字を入力してコマンドボタンをクリックして下さい。

3)テキストボックスで入力した文字がラベルに表示されたことを確認して下さい。

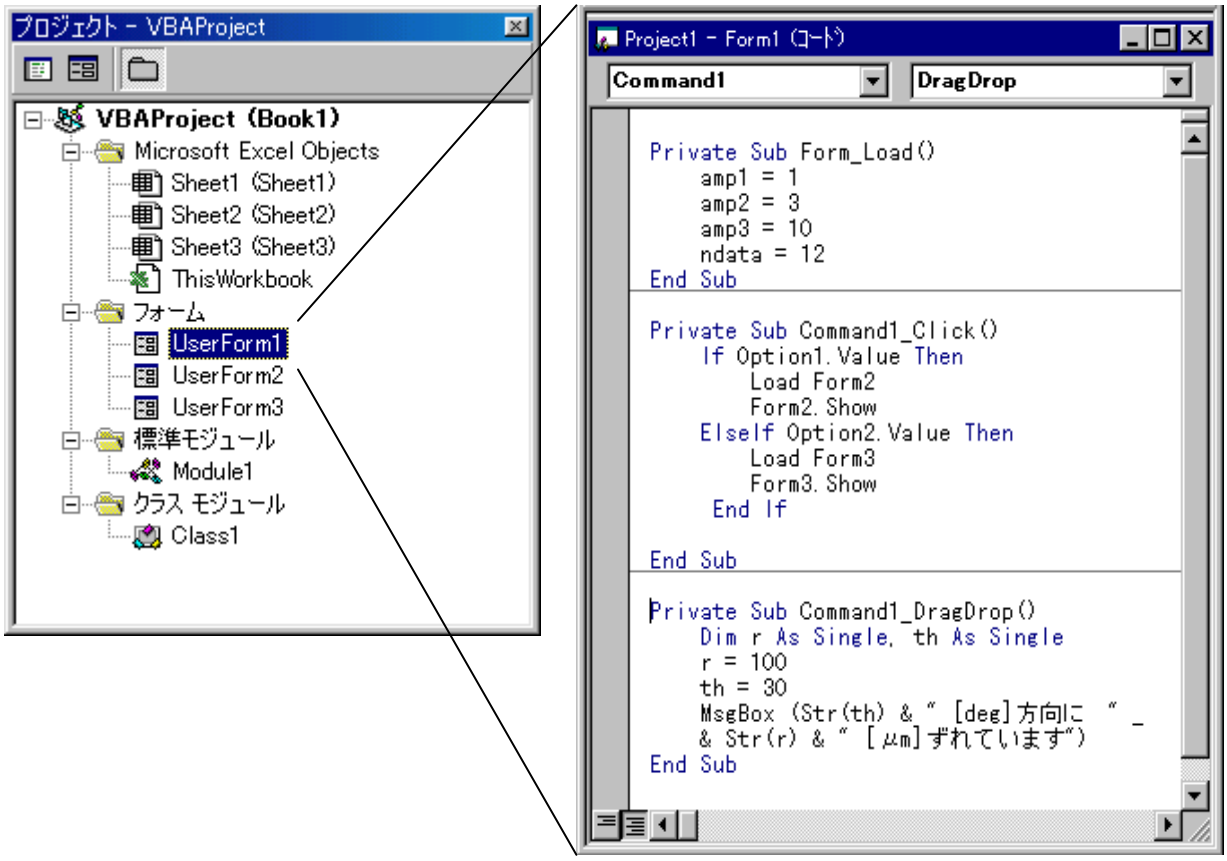
4)フォーム右上の[×]を押し、プログラムを終了させてください。

5)画面を Excel の画面に戻し、Excel の「ファイル」タブ→「名前を付けて保存」で「ファイルの種類」を「Excel マクロ有効ブック（*.xlsm)」にして、作ったプログラムを保存します。ファイル名は 01smpl.xlsm にして下さい。なお、このファイルを開くとセキュリティーの関係で警告メッセージが表示されますが、「コンテンツの有効化」をクリックすると、以後、VBA によるプログラム動作が有効になります。

２．３　VBA ファイル

Excel では、いくつかのデータが集まって一つの VBA プロジェクトとして管理しています。VBA プロジェクトには、そこで使用されているデータシートのデータのほかに、VBA で作成したユーザーフォーム、標準モジュール、クラスモジュールがあります。これらのものを一つにまとめて Excel ファイルとして保存します。

ユーザーフォームには、そのユーザーフォームの外観を決めるオブジェクトのデータやプログラムの動作を決めるコードが入っています。標準モジュールには、複数のフォームで共通して使用するようなコードや情報が入っています。クラスモジュールには、他のアプリケーションでも使用できるようにする情報が入っています。



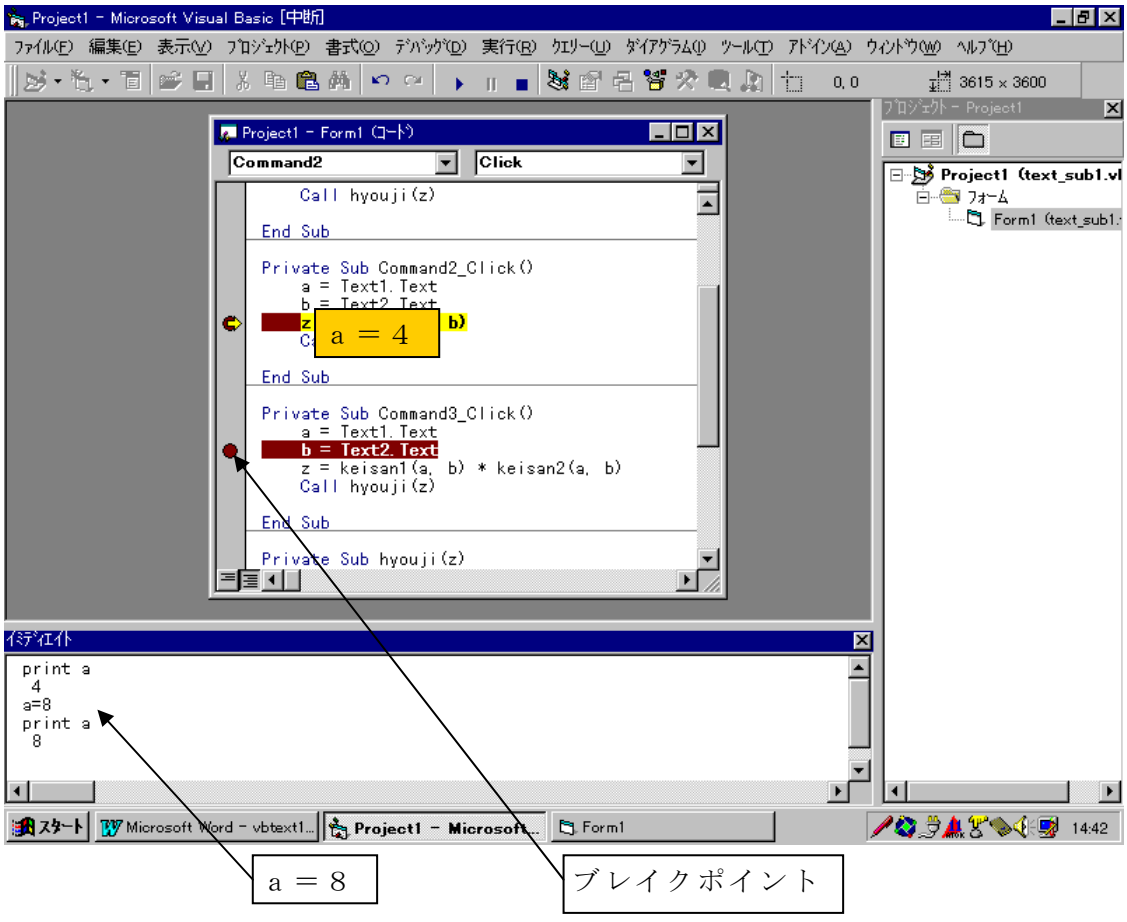
２．４　デバック

コード作成には間違いが付きものです。この間違いがある限り、プログラムは正しく動作しません。このプログラムの間違いを「バグ」といい、バグを取る作業を「デバッグ」と言います。本格的にコードの作成を行う前に、デバッグについて説明します。

デバッグを行うもっとも有効的な方法としてブレイクポイントを使う方法があります。ブレイクポイントを使うと、指定した場所で実行中のプログラムを一時停止させ、変数の値を調べることができます。以下にその手順を示します。

1) コードウインドウを開き、ブレイクポイントを設定します。
　設定はコードウインドウの左端をクリックすると設定され(茶色の丸が表示されます)、再度クリックすると解除されます。
2) プログラムを実行すると、ブレイクポイントの直前までを実行して一時停止します。
　このとき、コードウインドウが表示されます。
3) マウスポインタを変数の上に移動させると､その変数に入っている値が表示されます。
4) メニューの「実行」→「継続」をクリックすると、次の行から実行されます。
　また、「Ｆ８」を押すとブレイクポイントの行から１行ずつ実行することができます。

また、イミディエイトウインドウ内で　print　変数　（例えば　print a）を実行しても、その変数に入っているデータが表示されます。また　変数＝･･･（例えば　ａ＝８）を実行すると、ここで入力したデータを新たに変数の中に代入することができます。



Ｂ．ユーザーフォームをデザインする

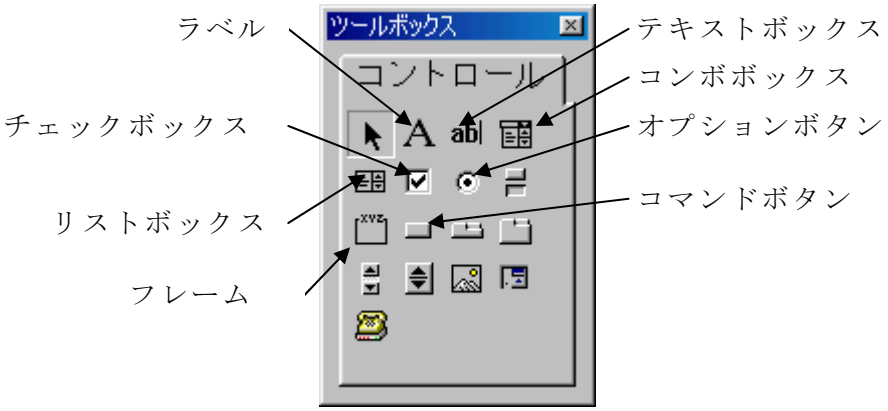
それでは、これからユーザーフォームのデザインについて詳しく説明します。

ユーザーフォームのデザインは VBE に表示されているユーザーフォームウインドウで行います。新たにユーザーフォームを作成するには、VBE のメニューの中の「挿入」→「ユーザーフォーム」をクリックして下さい。そうすると、ユーザーフォームウインドウが表示され、プロジェクトエクスプローラに「UserForm1」が追加されます。もしユーザーフォームが存在するのに、そのウインドウが閉じられていたならば、プロジェクトエクスプローラの「UserForm1」をダブルクリックして下さい。

３．コントロール

コントロールとは VBA のユーザーフォームに配置する部品のことです。ユーザーフォームウインドウがアクティブになると、VBE に右図のようなツールボックスが開かれます。この中に使うことに出来るコントロールが表示されます。もしも、ツールボックスが開かれていないときは、メニューの「表示」→「ツールボックス」を実行して下さい。

VBA には全部で数 10 個のコントロールが用意されています。その中でも代表的なものが一覧の中に入っています。その一部について説明します。



ラベル	文字を表示するときに使います
テキストボックス	文字や数字をキーボードから入力するときに使います
コマンドボタン	ある特定の動作を実行するときに使います
オプションボタン	複数の選択肢のなかから一つを選ぶときに使います
チェックボックス	二者択一の選択をするときに使います
フレーム	複数のコントロールをグループ化するときに使います
リストボックス	複数の項目を表示し、ユーザーに選択させるときに使います
コンボボックス	リストボックスとテキストボックスを組み合わせた機能を持っています

ユーザーフォームウインドウにこれらのコントロールを配置するには、ツールボックスで配置するコントロールをクリックし､ユーザーフォームウインドウ上でドラックします。一旦配置した後に、大きさや位置を修正することもできます。コントロールの中には、大きさの変わらないものや、実行したときに表示されない特別なコントロールもあります。

リストボックスやコンボボックスにデータを登録するには以下のように AddItem メソッドを使います。

```
ListBox1.AddItem "ABC"
ComboBox1.AddItem   "J204
```

まだ詳しく説明していませんが、ワークシートのセルに入力されているデータを登録するときは、以下のように行います

```
ListBox1.AddItem Cells( 3 , 5 ).Value
ComboBox1.AddItem Cells( i , 1 ).Value
```

セルの指定に変数を使い、繰り返しの命令を使うと、一瞬でセルのデータを項目に加えることができます。

４．プロパティ

プロパティとは、各オブジェクト(ユーザーフォームとコントロール)の特性のことをいいます。プロパティの例として、オブジェクトの位置や大きさ、文字列、色などがあります。使い方として「ラベル（オブジェクト）の色（プロパティ)」、「テキストボックス（オブジェクト）の幅（プロパティ)」のように使います。

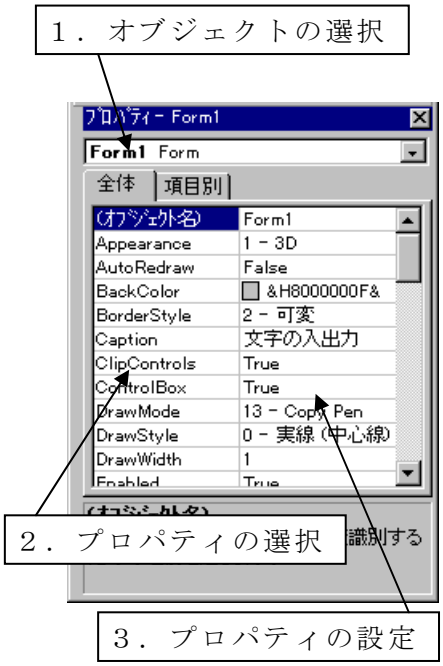
(ｵﾌﾞｼﾞｪｸﾄ名)	オブジェクトの名前を設定します(Name プロパティという)
Alignment	コントロール内の文字の配置を設定します
Autosize	文字サイズを自動的に調節するかどうかを設定します
AutoRedraw	描画した図を再度表示するかどうかを設定します
BackColor	コントロールの背景の色を設定します
Borderstyle	ユーザーフォームの種類を設定します
Caption	表示させる文字を設定します
Enabled	利用できるかどうかを設定します
Font	文字のフォントを設定します
ForeColor	文字の色などを設定します
Height	オブジェクトの高さを設定します
Left	オブジェクトの横方向の位置を設定します
MaxLenght	入力できる文字数の最大値を設定します
Moveable	ユーザーフォームが移動可能かどうかを設定します
Multiline	複数行の編集が可能かどうかを設定します
StartUpPosition	ユーザーフォームを最初に表示する位置を設定します
Style	コントロールの表示をユーザ設定で行うときに設定します
TabIndex	フォーカスの移動する順序を設定します
TabStop	「Tab」キーでフォーカスされるかどうかを設定します
Text	編集領域に表示させる文字を設定します
TextAlign	文字の配置を設定します
ToolTipText	ツールチップとして表示させる文字を設定します
Top	オブジェクトの縦方向の位置を設定します
Value	選択状態を設定、取得します
Visible	コントロールを表示するかどうかを設定します
Width	オブジェクトの幅を設定します
WindowState	実行時のユーザーフォームの表示状態を設定します

なお、それぞれのオブジェクトに含まれているプロパティには若干の違いがあり、ここで示したプロパティが全てのオブジェクトに含まれているものではありません。

ユーザーフォームをデザインするときにプロパティの初期設定をします。初期設定は、VBE に表示されるプロパティウインドウを使って行います。

設定を行うオブジェクトをユーザーフォームウインドウの中から選択するか、もしくは

プロパティウインドウの一番上にあるオブジェクト一覧でオブジェクトを選択すると、選択されたオブジェクトのプロパティが表示されます。左側のプロパティをクリックして選択し、右側の項目をクリックして設定します。設定の方法は、項目をクリックしてキーボードから直接入力するもの、項目の右側に出てきたボタンをクリックして設定するもの、などプロパティによって異なります。



また、プロパティはプログラム実行中に設定（変更）することもできます。実行中に設定するには、コードの中にプロパティを設定する命令を入れておきます。

[オブジェクト].[プロパティ] ＝ [設定値]

のようにすると、そのプロパティが設定されます。

例題　01smpl を以下のように作り替えて、動作確認をして下さい。

[UserForm]
　　01smpl.xlsm　に CommandButton2 を追加
　　CommandButton2.Caption　移動

[コード]
```
　　Private Sub CommandButton1_Click()    'CommandButton１がクリックされたら
　　    Label1.Top = 30                   'Label１の位置を上から 30
　　    Label1.Left = 150                 'Label２の位置を左から 150
　　    Label1.Caption = TextBox1.Text    'Label１のキャプションにテキスト１
　　End Sub                                              の文字を入れる

　　Private Sub CommandButton2_Click()    'CommandButton２がクリックされたら
　　    Label1.Top = 50                   'Label１の位置を上から 50
　　    Label1.Left = 170                 'Label１の位置を左から 170
　　End Sub
```

あらかじめ Label1 のプロパティで Label1.Top や、Label1.Left の初期値を調べておきます。

コマンドボタン２がクリックされたら Label1.Top、Label.Left は初期値に 20 を加えた値に再設定（変更）します。

コマンドボタン１がクリックされたら Label1.Top、Label.Left は初期値に戻し、さらにテキストボックスに入力した文字をラベルに表示します。

それでは実行して「実行」、「移動」を交互にクリックして見て下さい。クリックするごとにラベルの位置が動きます。これは、Label1.Top = ･･･、Label1.Left = ･･･でそれぞれ

のプロパティの設定値を変更しているためです。
　このほかにも、それぞれのオブジェクトにたくさんのプロパティが用意されています。いろいろと作り替えて確認して下さい。

　確認が終わったらファイル名を 02prop.xlsm に変更して保存します。VBE から Excel の画面に移動し、Excel のウインドウから、メニューの「ファイル」→「名前を付けて保存」でファイル名を 02prop.xlsm に変更してファイルの保存を行ってください。

＊＊＊［プログラミングのテクニック］＊＊＊

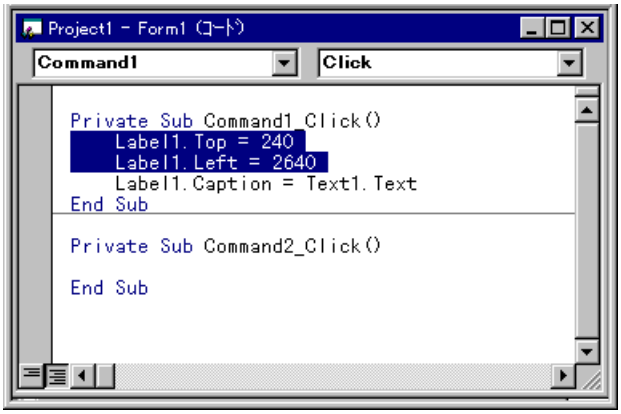
　プログラムを作るためのテクニックを紹介します。

（１）コントロールをコピーする方法（コピー　アンド　ペースト）

　1)元となるコントロールを選択
　2)メニューの「編集」→「コピー」
　3)ユーザーフォームを選択し、メニューの「編集」→「貼り付け」

（２）コードをコピーする方法

　1)元となるコードの行を選択
　2)メニューの「編集」→「コピー」
　3)挿入する部分へカーソルを移動
　4)メニューの「編集」→「貼り付け」



　このときの「編集」→「コピー」、「編集」→「貼り付け」は、ツールバーのコピーボタンや貼り付けボタン、ショートカットの「Ctrl」＋「C」、「Ctrl」＋「V」を使ってもかまいません。

（３）長いコードを複数行に分割する方法

　　スペースとアンダースコア（ _）を使って分割します。

```
Data1.RecordSource = _
"SELECT * FROM Titles, Publishers " _
& "WHERE Publishers.PubId = Titles.PubID" _
& "AND Publishers.State = 'CA'"
```

（４）１行に複数の文を入れる方法

　　複数の文をコロン（：）で区切ります

```
Label1.Top = 30 : Label1.Left = 150 : Label1.Caption = TextBox1.Text
```

（５）コードにコメントを入力する方法

　　コメントの前にアポストロフィー（'）を入れます。　例：'　メッセージの表示

```
TextBox1.Text = "Hi!"　　' テキスト ボックスにメッセージを表示します
```

C．実際の動作を決める

ここからは、実際の動作を決めるコードについて説明します。コードは「イベント」、「オブジェクト」、「プロパティ」、「メソッド」、「ステートメント」、「関数」、「演算子」などを組み合わせて作成します。

５．イベント

Windows のプログラムでは、操作者や Windows（OS）がオブジェクトやアプリケーションソフトに対して何らかの働きかけをすることによって、処理が実行します。この働きかけのことをイベントといいます。以下に、よく使われるイベントを示します。

Activate	ユーザーフォームがアクティブにされる
Click	オブジェクトがクリックされる
Dblclick	オブジェクトがダブルクリックされる
Initialize	ユーザーフォームが開く
Terminate	ユーザーフォームが閉じる
MouseUp	マウスの左右のボタンが離される
MouseDown	マウスの左右のボタンが押される
MouseMove	マウスがオブジェクト上で動く

コード作成におけるイベントの指定は、コードウインドウ左上でコントロールを選択し、右上でイベントを選択すると、イベントプロシージャが作られます。この時、それぞれを選択する毎にイベントプロシージャがつくられるので、不要なものは削除しておきます。

例題　02prop を以下のように作り替えて、動作確認をして下さい。

[UserForm]
　　02prop.xlsm　をそのまま使用

[コード]

```
Private Sub Label1_DblClick(ByVal ･･･) 'Label1 がダブルクリックされたら
    Label1.Top = 30
    Label1.Left = 150
    Label1.Caption = TextBox1.Text
End Sub

Private Sub UserForm_Layout()          'UserForm の位置が変わったら
    Label1.Top = 50
    Label1.Left = 170
End Sub
```

Label1 がダブルクリックされるとテキストボックスの文字が表示されます。また、ユーザーフォームの位置が変わると表示された文字が移動します。確認して下さい。

確認が終わったらファイル名を 03evnt.xlsm に変更して保存して下さい。

６．メソッドとステートメント

６．１　メソッド

Windows のプログラムでは、様々なオブジェクト（ユーザーフォーム、コントロール、ワークブック、ワークシートなど）に動作をさせることができます。このような命令をメソッドといいます。メソッドによるオブジェクトの動作は

[オブジェクト].[メソッド]

のように記述します。

ワークシートやワークブックなどに対するメソッドは多く用意されていますが、ユーザーフォーム､および配置するオブジェクトに対してのメソッドはあまり多くはありません。代表的なものを以下に示します。

Copy	選択されたものをコピーする
Cut	選択されたものを切り取る
Pest	コピーされたもの、または切り取られたものを貼り付ける
Move	
Setfocus	
Show	指定されたユーザーフォームを表示する
Hide	指定されたユーザーフォームを非表示にする

６．２　ステートメント

Windows のプログラムでオブジェクトに関係しないその他のコマンドをステートメントといいます。以下に良く使われるものを示します。その中でも特に重要なものは後ほど改めて説明します。

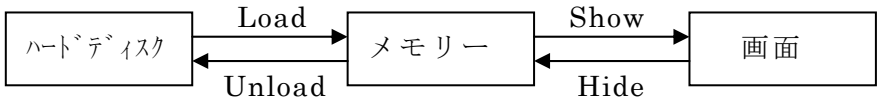
Option Explicit	すべての変数の宣言を強制する
Dim	変数を宣言する
Private	変数を宣言する
Const	定数を宣言する
If...Then...Else	条件に応じてを処理を分岐する
Select Case	条件に応じて複数の選択肢に処理を分岐する
For...Next	回数を指定し、指定した回数だけ処理を繰り返す
Do...Loop	条件を指定し、条件に応じて処理を繰り返す
On Error	エラーが発生したときに、処理を実行する
Exit	プロシージャまたはブロックを抜け出す
End	プロシージャまたはブロックを終了する
Sub	Sub プロシージャを作成する
Function	Function プロシージャを作成する
Call	Sub プロシージャや Function プロシージャなどを実行する
Load、UnLoad	フォームをメモリに読み込む
Date	現在のシステムの日付を設定します。
Time	システムの時刻を設定します。
Beep	ビープ音を鳴らします。

６．３　ユーザーフォームの表示

Windows のプログラムではいくつかのウインドウが集まって一つのアプリケーションになることがほとんどです。VBA で作られたアプリケーションでも同様で、一つのプロジェクトで複数のユーザーフォームを持つことができます。あるユーザーフォームから別のユーザーフォームを開いたり閉じたりするのに、以下のメソッドやプロパティを使います。

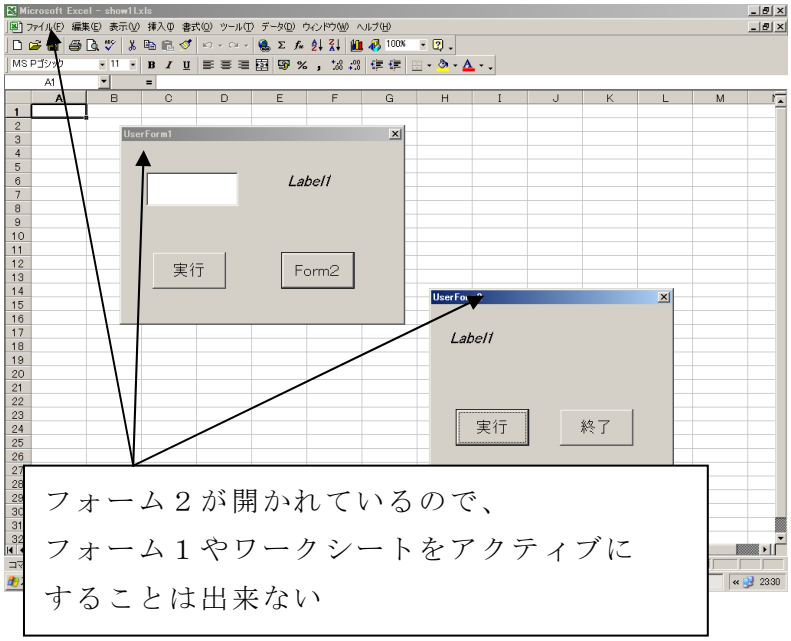
Load UserFormN	UserFormN のデータを読み込みます
Unload UserFormN	UserFormN のデータを消去します
UserFormN.Show	UserFormN を表示します
UserFormN.Hide	UserFormN を非表示にします

新しいユーザーフォームを開く場合、まず Load でユーザーフォームのデータをメモリーに読み込み、次に Show メソッドでそのユーザーフォームを画面に表示させます。ユーザーフォームを画面から消すときは、Hide メソッドを使って非表示にします。もちろん、Unload でメモリーに読み込んだユーザーフォームデータを消去しても、ユーザーフォームを画面から消すことが出来ます。



ここで、注意しなければならないことは、プログラムを終わらせるときは読み込んだユーザーフォームのデータをメモリーから消去しなければならないことです。Unload せず Hide で画面を非表示にした状態でプログラムを終了させると、見た目は終わっていても実際には終わっていない状態になります。最後は必ず Unload して下さい。

ExcelVBA のユーザーフォームは、その初期設定として、ユーザーフォームが一旦開かれると、そのユーザーフォームが閉じない限り、別のユーザーフォームやワークシートに移ることが出来ないようになっています。これは、ユーザーフォームを実行中にワークシートのデータを書き換えること出来ないようにするためです。



この設定を解除するには、Userform の ShowModal プロパティを False にすると、他のユーザーフォームに移ることが出来ます。

例題　次のプログラムを作り、その動作確認をして下さい。
ユーザーフォームを新たに追加するときは、メニューの「挿入」→「ユーザーフォーム」で新しいユーザーフォーム（UserForm2）がプロジェクトに追加されます。

```
[UserForm1]
      02prop.xlsm　を修正
      CommandButton1.caption     実行
      CommandButton2.caption     フォーム２
      ShowModal                  false
[UserForm2]
      Label1
      CommandButton1.caption     実行
      CommandButton2.caption     フォーム１
      ShowModal                  false

[コード:UserForm1]
      Private Sub CommandButton1_Click()
          Label1.Caption = TextBox1.Text
      End Sub

      Private Sub CommandButton2_Click()
          Load UserForm2
          UserForm2.Show
      End Sub

[コード：UserForm2]
      Private Sub CommandButton1_Click()
          Label1.Caption
                =UserForm1.TextBox1.Text
      End Sub

      Private Sub CommandButton2_Click()
          UserForm2.Hide
          Unload UserForm2
      End Sub
```

注：表示の都合上２行に分けています

UserForm1 の「実行」をクリックするとテキストボックスに入力した文字が表示されます。「フォーム２」をクリックすると、まず、Load で新しい UserForm2 をメモリーに読み込み、Show でそのユーザーフォームを表示します。

UserForm2 の「実行」をクリックすると UserForm1 のテキストボックスで入力した文字が表示されます。別のユーザーフォームで設定したオブジェクトのプロパティを使うときは、先にフォームを指定してから、オブジェクトのプロパティを指定します。「フォーム１」をクリックすると Hide で UserForm2 を画面から消し、Unload でメモリーからユーザーフォーム２のデータを消去します。

動作確認が終わったら、プログラムを保存します。ファイル名を 04show.xlsm にして保存して下さい。

７．変数と変数宣言

コンピュータで数値計算をさせるときは、変数と呼ばれる記号を使います。この変数に数値を入れて様々な計算をさせます。もちろん、変数を使わずに計算をすることもできますが、変数を使った方が、数値の変更や、コードの変更が生じたときなど容易に対応することができます。

７．１　変数

VBA で変数を使用するには、基本的にはその変数がどのようなものかを指定しなくてはなりません。この指定のことを「変数の宣言」と言います。

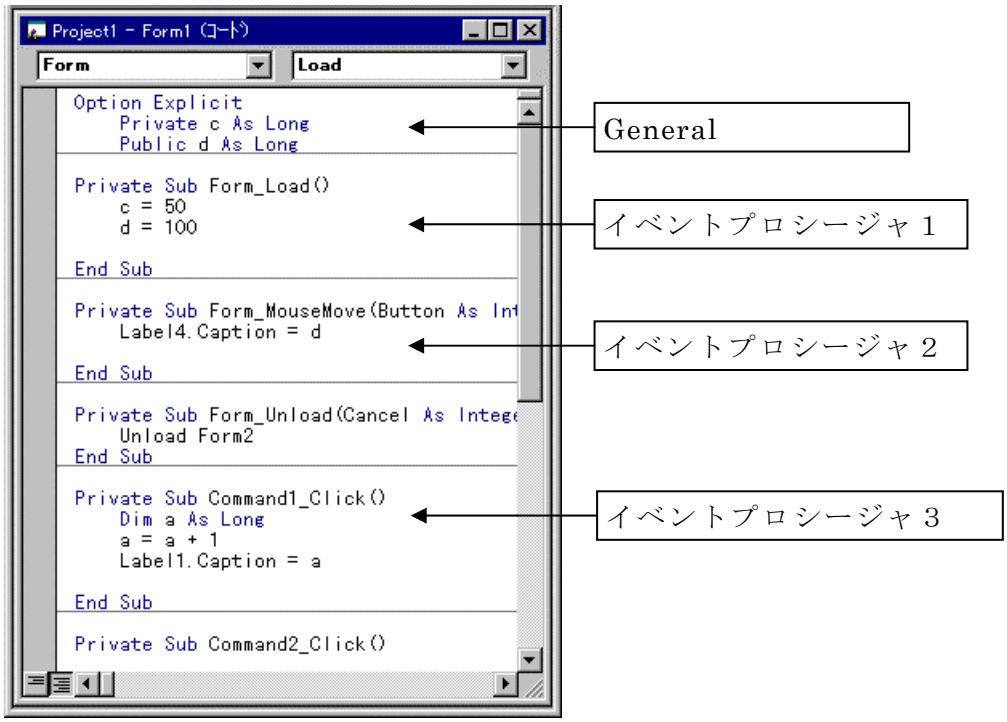
変数の宣言とは、その変数の型と変数の使用範囲（スコープ）を指定することです。変数にはいくつかの型(種類)があります。VBA をはじめ、多くのプログラム言語では、これらの変数を使用する前に、「その変数がどの型であるのか」、また同時に「その変数がプログラムのどの部分で使うのか」を宣言します。変数の宣言は以下に示すように行います。

［使用範囲］［変数名］ As ［型］

例：　Private　abc　As　Byte

変数 abc はバイト型としてフォーム全体で使用します

変数の宣言はコードの中で行います。宣言する場所は宣言領域（コードウインドウ先頭の（General）の部分）とプロシージャの先頭部分の二つがあります。どの部分で宣言するかによって、使用するコマンドや、使用範囲が異なります。



（１）変数の使用範囲の指定

変数の使用として、プロジェクト全体、宣言したユーザーフォーム全体、宣言したプロシージャ内があります。変数宣言のステートメントとして以下のものを使います。

Public	プロジェクト全体で変数を使用する
Private	宣言したユーザーフォーム全体で変数が使用できる
Dim	宣言したユーザーフォーム全体、もしくは宣言したプロシージャ内で変数が使用できる。
Static	宣言したプロシージャ内で変数が使用できる。

1)プロジェクト全体で使うことのできる変数宣言

Public を使い、標準モジュール、またはユーザーフォームの宣言領域で宣言します。標準モジュールで宣言した変数 a は、どのユーザーフォームでも変数名が a のままで使うことができるます。一方、UserFormN の宣言領域で宣言した変数 a は、ほかのユーザーフォームでその変数を使うときは、変数名を UserFormN.a にして使います。

2)ユーザーフォーム全体で使うことのできる変数宣言

Dim または Private を使い、ユーザーフォームの宣言領域で宣言します。宣言したフォーム内で使用できます。このとき、別のユーザーフォームに移っても変数はクリアされません。

3)イベントプロシージャで使うことの出来る変数宣言

Dim または Static を使い、イベントプロシージャの中で宣言します。宣言したイベントプロシージャでのみ使用できます。Dim で宣言した変数はイベントプロシージャから抜けると変数はクリアされます。Static で宣言した変数はイベントプロシージャから出ても、変数はクリアされません。

（２）変数のデータ型の指定

変数の型として様々なものがありますが、ここでは良く使われるものを以下に示します。

Boolean(ブーリアン型)	True、False
Byte(バイト型)	0～255
Integer(整数型)	-32,768～32,767
Long(長整数型)	-2,147,483,648～2,147,483,647
Single(単精度浮動小数点数型)	負の値　-3.402823E38～-1.401298E-45
	正の値　1.401298E-45～3.402823E38
String(文字列型)	最大約２０億文字(半角)
Date(日付型)	年月日　時分秒　****/**/**　**：**：**
Variant（バリアント型）	VBA が自動的にいずれかの型で扱う

＊バリアント型は、その変数がどの型で宣言されているのをユーザーは判断することができない。そのため、思わぬエラーが出たり処理速度が遅くなったりするので、極力避けるようにする。

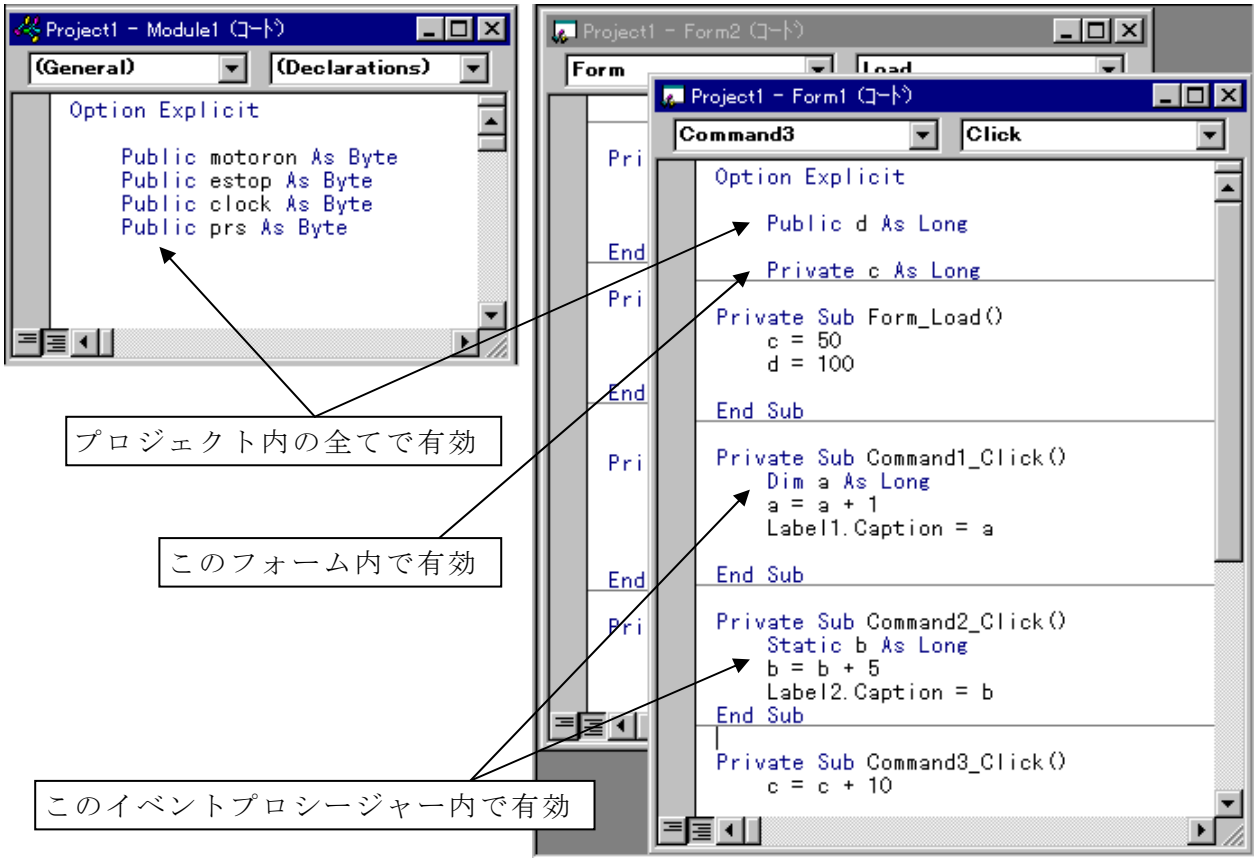
（３）複数の変数の宣言

複数の変数の宣言は、以下のようにコンマで区切ることにより１行で行なえます。

Dim a As Integer , b As Long , c As String

なお、以下のように宣言すると、最後の変数だけが指定した型で宣言されたことになり、それ以外は自動的にバリアント型として扱われます。

Dim a , b , c As String



変数を使うことだけを考えるのならば、「Public」を使ってプロジェクト全体で使えるように宣言すると簡単にはなります。しかし、処理速度などを考えると必要最小限度の範囲で宣言をした方が望ましいでしょう。

VBA の設定によっては宣言をしなくても変数を使うことはできますが、この場合、自動的に Variant 型で扱われるので、宣言はしておいた方が望ましいでしょう。

コードの先頭に「Option Explict」を入れると、宣言していない変数は使えないようになります。もしくは、メニューの「ツール」→「オプション」→「編集」→「変数の宣言を強制する」のチェックを入れることにより、宣言しなければ変数が使えないようになります（自動的に Option Explict　が入ります)。

例題　01smpl.xlsm を使い、次のプログラムを作り、変数の型の動作確認をして下さい。

[UserForm1]

01smpl.xlsm　をそのまま使用

[コード]

```
Dim a As _____        ←変数の型を変えて、データと表示の関係を確認する
a = TextBox1.Text
Label1.Caption = a
```

変数 a の宣言を Integer、Long、Single、Double、Boolean、Date、String などで行い、入力したデータと表示されるデータを比較してください。

a を Integer で宣言し、テキストボックスに小数（例えば 123.456）を入力しても、変数 a にはその整数部分が入力されます。これは、ブレイクポイントを　Label1.Caption = a　のところで設定し、プログラムを一時停止させると、変数に格納されたデータを確認することができます。また、テキストボックスに型の範囲を超える大きな数値を入力すると（例えば 12345678)、「オーバーフロー」のエラーが出ます。またテキストボックスに型が一致しないデータを入力すると（例えばＡＢＣ、1964/04/11　など)、「型が一致しません」のエラーが出ます。

変数宣言の型と、代入するデータの関係を調べてみます。本来であれば、型は完全に一致しなければエラーとなりますが、VBA はエラーとなる組み合わせでも、使えるものがあります。

変数の型として「整数」、「小数」、「ブール」、「日付」、「文字列」があり、代入するデータとして「整数(123)」、「小数(123.456)」、「ブール(True、False)」、「日付(2012/10/10)」、「文字列(ABC)」があります。これらの組み合わせで調べてみると、以下の表のようになりました。

データ＼宣言	「整数」	「小数」	「ブール」	「日付」	「文字列」
「整数」	可 or Over	可 or Over	可(*)	日にち	可(***)
「小数」	整数 or Over	可 or Over	可(*)	日にち、時間	可(***)
「ブール」	型不一致	型不一致	可	型不一致	可(***)
「日付」	型不一致	型不一致	型不一致	可	可(***)
「文字列」	型不一致	型不一致	型不一致	一部可(**)	可(***)

「Over」はオーバーフロー　　Long や Double で対応

*　　False のときは「0」、True のときは「111･･･1」（Intger のときは、補数で-1)

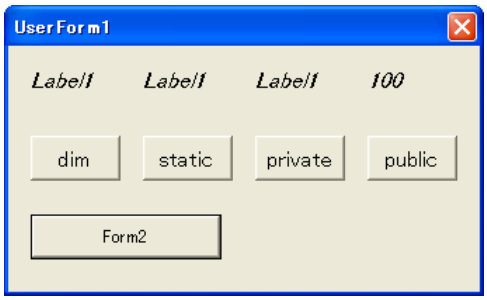
**　　「○○年○月○日」　は可

***　ただし、全て文字として認識

例題　次のプログラムを作り、変数の使用範囲の動作確認をして下さい。

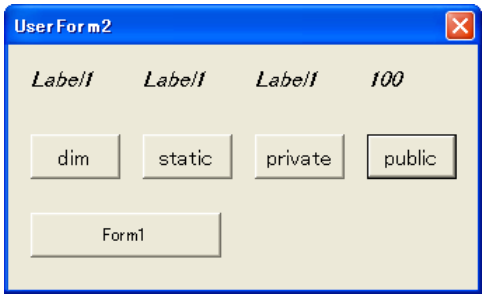
[UserForm1]

Label1	
Label2	
Label3	
Label4	
CommandButton1.Caption	dim
CommandButton2.Caption	static
CommandButton3.Caption	private
CommandButton4.Caption	public
CommandButton5.Caption	form2



[UserForm2]

Label1	
Label2	
Label3	
Label4	
CommandButton1.Caption	dim
CommandButton2.Caption	static
CommandButton3.Caption	private
CommandButton4.Caption	public
CommandButton5.Caption	form1



[コード：UserForm1]

```
Option Explicit             '変数の宣言をしないと変数が使えない設定
    Private b As Long       'c は長整数型として UserForm1 内で有効
    Public c As Long        'd は長整数型としてプロジェクト全体で有効

Private Sub UserForm_Initialize()       'UserForm が立ち上がると
    b = 50
    c = 100
End Sub

Private Sub UserForm_Terminate()        'UserForm が終了すると
    Unload UserForm2                    'UserForm2 も終了させる
End Sub


Private Sub UserForm_MouseMove(ByVal Button As Integer, ByVal Shift As
Integer, ByVal X As Single, ByVal Y As Single)
                                        'UserForm 上でマウスが動くと
    Label4.Caption = c
End Sub

Private Sub CommandButton1_Click()
    Dim a As Long               'a は長整数型としてこのﾌﾟﾛｼｰｼﾞｬ内で有効
    a = a + 1
    Label1.Caption = a
End Sub

Private Sub CommandButton2_Click()
    Static a As Long            'b は長整数型としてこのﾌﾟﾛｼｰｼﾞｬ内で有効
    a = a + 5
    Label2.Caption = a
End Sub
```

```
Private Sub CommandButton3_Click()
    b = b + 10
    Label3.Caption = b
End Sub

Private Sub UserForm_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    b = b + 10
    Label3.Caption = b
End Sub


Private Sub CommandButton4_Click()
    c = c + 20
    Label4.Caption = c
End Sub

Private Sub CommandButton5_Click()
    Load UserForm2
    UserForm2.Show
End Sub
```

[コード：UserForm2]

```
Option Explicit
    Private b As Long           ’c は長整数型として UserForm2 内で有効

Private Sub UserForm_MouseMove(ByVal Button As Integer, ByVal Shift As
Integer, ByVal X As Single, ByVal Y As Single)
                                ’UserForm 上でマウスが動くと
    Label4.Caption = UserForm1.d  ’Label4 の Caption に UserForm1 の d を代入
End Sub

Private Sub CommandButton1_Click()
    Dim a As Long               ’a は長整数型としてこのﾌﾟﾛｼｰｼﾞｬ内で有効
    a = a + 1
    Label1.Caption = a
End Sub

Private Sub CommandButton2_Click()
    Static a As Long            ’b は長整数型としてこのﾌﾟﾛｼｰｼﾞｬ内で有効
    a = a + 5
    Label2.Caption = a
End Sub

Private Sub CommandButton3_Click()
    b = b + 10
    Label3.Caption = b
End Sub

Private Sub UserForm_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    b = b + 10
    Label3.Caption = b

End Sub

Private Sub CommandButton4_Click()
    UserForm1.c = UserForm1.c + 20
    Label4.Caption = UserForm1.c
End Sub
```

```
Private Sub CommandButton5_Click()
    UserForm1.Show
End Sub
```

「dim」をクリックすると、a=a+1 を実行して、label1 には 1 が表示されます。しかし、「dim」を何回クリックしても数値は 1 以上にはなりません。これは、変数 a はイベントプロシージャの中で Dim を使って宣言されており、イベントプロシージャから抜けるとデータはクリアされて 0 になるためです。

「static」をクリックすると、a=a+5 を実行して、label2 には 5 が表示されます。これは、変数 a はイベントプロシージャの中で宣言されており、他のイベントプロシージャで宣言された変数 a とは区別されているからです。また、「static」をクリックするごとに数値は 5 ずつ増えていきます。これは、変数 a は static で宣言されており、イベントプロシージャから抜けてもデータはクリアされず、元の数値に 5 を加えていくためです。

「private」をクリックすると、b=b+10 を実行して、label3 には 60 が表示されます。これは、フォームが呼び出されたとき（Private Sub UserForm_Initialize()）c の初期値を b=50 にしているためです。また、「private」をクリックするごとに数値は 10 ずつ増えていきます。さらにフォームをダブルクリックしても、同様に c は 10 ずつ増えます。これは、c は UserForm1 の宣言領域で Private を使って宣言されおり、UserForm1 内のプロシージャであればどこででも使用できるからです。

「public」をクリックすると、c=c+20 を実行して、label4 には 120 が表示されます。これは、フォームが呼び出されたとき（Private Sub UserForm_Initialize()）d の初期値を c=100 にしているためです。また、「public」をクリックするごとに数値は 20 ずつ増えていきます。

「form2」をクリックすると別のユーザーフォーム、UserForm2 が表示されます（もしくはアクティブになります)。UserForm2 での「dim」、「static」、「private」をクリックすると、UserForm1 でのそれらと同じように動作します。変数 c は UserForm1 で public で宣言されているので、UserForm2 では宣言しなくても c を使うことができます。このときの変数名は UserForm1.c になります。

マウスポインタが UserForm2 の上を動くと、Lavel4 には UserForm1 での c の値が表示されます（Private Sub UserForm_MouseMove(・・・・・)）。

「public」をクリックすると、UserForm1.c=UserForm1.c+20 を実行して、label4 にはそれまでの c に 20 を加えた値が表示されます。アクティブなフォームを UserForm2 から UserForm1 へ移すと、UserForm2 で変更した c が UserForm1 の c に表示されます。

UserForm1 を閉じると（Private Sub UserForm_ Terminate (・・・・・)）UserForm2 も閉じられます。

確認が終わったらファイル名を 05dim1.xlsm にして保存して下さい。

＊＊＊［プログラミングのテクニック］＊＊＊

例題にあった　a ＝ a ＋ 1　の式ですが、数学ではこの様な式はありませんが、プログラムではよく見かけます。これは変数 a に入れる数値を 1 だけ増やす式となります。 a に入っている変数に 1 を加え、新たに a に入れ直すものです。

７．２　配列

たくさんのデータを扱う場合、そのデータ数だけ変数が必要になってきます。変数が多くなるとプログラムが長く、複雑になります。仮に変数が 100 個あれば、宣言文も 100 行必要になります。これを簡単に出来るようにしたものに配列があります。これは一つの変数名にインデックス（添字）をつけたものです。例えば７つの変数 a、b、c、d、e、f、g は、配列を使うと a(0)、a(1)、a(2)、a(3)、a(4)、a(5)、a(6)として扱うことができます。

配列の宣言は変数の宣言とほとんど同じで、変数にインデックスがついただけです。

［使用範囲］［変数名](インデックス)　As　［型］

＊インデックスは使う配列の数を表します。配列の番号は a(0)～a(配列の数-1)になります。

例：　Dim a(100) as long

long 型で a(0)～a(99)までの 100 個を使います

例えば、100 個の変数に同じ処理をする場合、100 個の命令をしなければなりません。ところが、配列を使った場合、一つの命令を 100 回数繰り返すだけで済みます。

以下にそのサンプルを示します。変数 a、b は 100 個ずつあり、a に 100 を加えた値を b にする命令です。配列を使わなければ、同じような行が 100 行必要ですが、配列を使うと３行で終わります。

変数：a0、a1、a2、a3、a4・・・・を使う場合

```
b0=a0+100
b1=a1+100
b2=a2+100
b3=a3+100
  ・
  ・
b99=a99+100
```

配列：a(0)、a(1)、a(2)、a(3)・・・・を使う場合

```
For i=0 to 99    ←同じ命令を 100 回
    b(i)=a(i)+100          繰り返す
Next
```

＊この繰り返し命令は後で説明します

配列のインデックスは複数個持つことも出来ます。この場合の宣言方法は以下のように行います。

Dim a(100,50) as long

a(0,0) ・・・・ a(99,0)
a(0,1) ・・・・ a(99,1)
:　　:
a(0,49)・・・・ a(99,49)

上の場合、5000 個の変数の宣言が、１行で済みます。

７．３　文字列

VBA では文字変数(String)として宣言した変数に文字列を代入することができます。

```
Dim a As String
```

例題　01smpl を以下ように作り替えて、動作確認をして下さい。

[UserForm]

01smpl.xlsm　をそのまま使用

[コード]

```
Private Sub CommandButton1_Click()
    Dim a As String
    Dim b As String

    a = TextBox1.Text
    b = TextBox1.Text
    Label1.Caption = a & b      ’文字列 a と b を結合する
End Sub
```

まず a、b を文字列として宣言します。a、b にはそれぞれテキストボックスに入力した文字列が入ります。Label1. Caption には文字列 a と文字列 b を連結したものが入ります。

確認が終わったら、ファイル名を 07str.xlsm に変更して保存して下さい。

文字列に対する演算として結合があります。二つの文字列を結合するには、「＆」もしくは「＋」を使います。また、文字列の関数を使うと様々な処理を施すことが出来ます。

数字を入れる変数を扱う場合、変数の宣言をしないで変数を使うと、数字が文字として扱われることがあります。

```
a=3
b=4
c=a+b
```

これを実行したときに、変数 c の中に「３４」が入ることがあります。これは、変数 a の中に「３」という文字が、変数 b の中に「４」という文字が入り、変数 c の中にこれらの文字を結合したものが入ったためです。変数をあらかじめ宣言することによって、このようなトラブルを防ぐことができます。

8　演算

VBA で行うことの出来る演算には算術演算、比較演算、論理演算があります。算術演算とは通常の加減乗除などの演算、比較演算とは二つの数値の大小を比較する演算、論理演算とは二つの状態（真／偽、Tule／False、０／１など）を扱う演算です。

算術演算のなかでもよく使われるもの、およびその演算子を以下に示します。

代入	＝	変数に値や文字を代入	例： a= 12　a に 12 を代入
加算	＋	二つの数字の足し算	例： 3 + 5　→ 8
減算	－	二つの数字の引き算	例： 8 - 2　→ 6
乗算	＊	二つの数字のかけ算	例： 4 * 3　→ 12
除算	／	二つの数字のわり算	例： 9 / 4　→ 1.25
	￥	除算の整数部（商）	例： 9 ￥ 4　→ 2
	Mod	除算の余り	例： 9 Mod 1　→ 1
累乗	^	べき乗の計算(ｍのｎ乗)	例： 2 ^ 3　→ 8

次に、比較演算の中でよく使われるもの、およびその演算子を以下に示します。

より小さい	＜	例：	ａ＜ｂ	ａはｂより小さい
以下	＜＝	例：	ａ＜＝ｂ	ａはｂ以下
より大きい	＞	例：	ａ＞ｂ	ａはｂより大きい
以上	＞＝	例：	ａ＞＝ｂ	ａはｂ以上
等しい	＝	例：	ａ＝ｂ	ａはｂと等しい
等しくない	＜＞	例：	ａ＜＞ｂ	ａはｂと等しくない

次に、文字列演算およびその演算子を以下に示します

結合　＆　若しくは　＋　文字列の結合　　例：″文字の″ & ″結合″

次に、論理演算の中でよく使われるもの、およびその演算子と真理値表を以示します。

否定	NOT	例：	C=NOT A
論理積	AND	例：	C=A AND B
論理和	OR	例：	C=A OR B
排他的論理和	XOR	例：	C=A XOR B

A	B	C			
		NOT A	AND	OR	XOR
False	False	True	False	False	False
False	True	True	False	True	True
True	False	False	False	True	True
True	True	False	True	True	False

また、論理演算は条件式の True、False に対して演算を行うものもあり、判断、分岐のところでよく使います。上のＡ、Ｂには条件式が入りその条件式が満たされているか(True)満たされていないか（False）でＣの値（結果）が決まります。

パソコンの内部では、全て 0/1 の信号（２値信号。厳密に言うと、電圧の H/L 信号）だけで様々な処理がなされています。「100」という数字や「ABC」という文字は、この 0/1 信号をいくつか集めたかたまりで表されています。この 0/1 信号の一つを「ビット（bit)」という単位で数えます。そして、８ビットを「１バイト（byte)」、16 ビットを「１ワード（word)」と呼びます。足し算まどのデータ処理は、この 0/1 信号を別の信号に作り変えています。

論理演算は数値データにも行われ、1 bit データの場合、右のようになります。

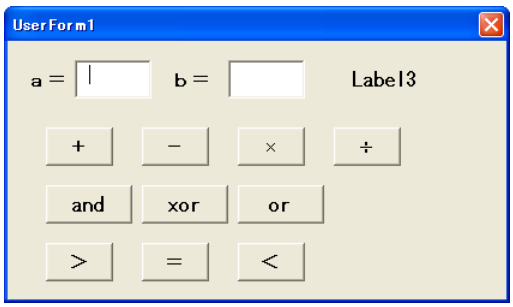
またデータが複数ビットの場合、入力したデータを２進数に変換したものの、各ビットの演算を行うものです。

A	B	C NOT A	AND	OR	XOR
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

例題　次のプログラムを作り、動作確認をして下さい。

[UserForm]

```
TextBox1
TextBox2
Label1
Label2.Caption          a=
Label3.Caption          b=
CommandButton1.Caption  +
CommandButton2.Caption  -
CommandButton3.Caption  *
CommandButton4.Caption  /
CommandButton5.Caption  and
CommandButton6.Caption  xor
CommandButton7.Caption  or
CommandButton8.Caption  >
CommandButton9.Caption  =
CommandButton10.Caption <
```



[コード]

```
Option Explicit
    Dim a As Integer
    Dim b As Integer
    Dim c As Single
    Dim d As Integer
    Dim e As Boolean

Private Sub CommandButton1_Click()
    a = TextBox1.Text
    b = TextBox2.Text
    c = a + b
    Label1.Caption = c
End Sub
```

```
Private Sub CommandButton2_Click()
    a = TextBox1.Text
    b = TextBox2.Text
    c = a - b
    Label1.Caption = c
End Sub

Private Sub CommandButton3_Click()
    a = TextBox1.Text
    b = TextBox2.Text
    c = a * b
    Label1.Caption = c
End Sub
```

```
Private Sub CommandButton4_Click()
    a = TextBox1.Text
    b = TextBox2.Text
    c = a / b
    Label1.Caption = c
End Sub

Private Sub CommandButton5_Click()
    a = TextBox1.Text
    b = TextBox2.Text
    d = a And b
    Label1.Caption = d
End Sub

Private Sub CommandButton6_Click()
    a = TextBox1.Text
    c = a Xor b
    Label1.Caption = d
End Sub

Private Sub CommandButton7_Click()
    a = TextBox1.Text
    b = TextBox2.Text
    c = a Or b
    Label1.Caption = d
End Sub
```

```
Private Sub CommandButton8_Click()
    a = TextBox1.Text
    b = TextBox2.Text
    e = a > b
    Label1.Caption = e
End Sub

Private Sub CommandButton9_Click()
    a = TextBox1.Text
    b = TextBox2.Text
    e =  a = b
    Label1.Caption = e
End Sub

Private Sub CommandButton10_
                              Click()
    a = TextBox1.Text
    b = TextBox2.Text
    e =  a < b
    Label1.Caption = e
End Sub
```

二つのテキストボックスに数値（整数）を入力してａ、ｂそれぞれの値を設定します。四則演算のコマンドボタン（＋、－、＊、／）をクリックすると、それぞれの演算を行い、その結果を表示します。演算結果を格納する変数ｃは Single で宣言しています。これは除算の結果が小数となることがあるためです。もしこれを Integer で宣言すると、どのような結果が表示されるのかを確認しておいてください。

今はａ、ｂとも Integer で宣言しています。a=1.5、b=1,6 にして和を求めてください。その結果から、パソコン内部でどのような処理がなされているかを考えてください。

小数も扱えるようにするためには、ａ、ｂを Single で宣言します。そうすると a=1.5、b=1,6 のときの和が 3.1 になります。

除算を実行すると小数点以下の桁数が何桁にもなることがあります。小数点以下の桁数を指定するには Format 関数を使います。桁合わせの文字として"0"と"#"を使います。0 のときは値がなければ 0 を、#のときは値がなければ空欄を表示します。

```
Label1.Caption = Format(c, "0.##")
```

論理演算のコマンドボタン（and、xor、or）をクリックすると、入力した数値の And、Or、Xor の結果を１０進数で表示します。

比較演算のコマンドボタン（＞、＝、＜）をクリックすると、ｂに対してａが大きいのか、小さいのか、それとも等しいのかを判断し、その結果を表示します。結果が「真」のときは「True」を返し、「偽」のときは「False」を返します。演算結果を格納する変数 e は Boolean で宣言されていますが、これを String や Integer で宣言すると、どのような結果が表示されるのかを確認しておいてください。

確認が終わったらファイル名を 06enzan.xlsm にして保存して下さい。

９．関数

プログラムを組む場合、文字や数値のデータを別の形に変換しなければならないことがあります。この様な変換の働きのものを関数と言います。

VBA では約 200 個の関数が用意されています。これらの関数の大部分はワークシートの関数と共通になっていますので、VBA で関数を使う場合、ワークシートの関数を参考にすると良いでしょう。

９．１　数学関数、数値の関数

VBA で数値計算をさせる場合、単なる四則計算のほかに様々な数学関数が必要になります。数学関数、数値関数の中の主なもの以下に示します。

a=Abs(数値)	指定した数値の絶対値を返す
a=Sgn(数値)	指定した数値の符号（プラス、マイナス）を返す
a=Sin(数値)	指定した数値の正弦を返す
a=Cos(数値)	指定した数値の余弦を返す
a=Tan(数値)	指定した数値の正接を返す
a=Atn(数値)	指定した数値の逆正接を返す
a=Log(数値)	指定した数値の常用対数を返す
a=Ln(数値)	指定した数値の自然対数を返す
a=Exp(数値)	指定した数値の e を底とするべき乗を返す
a=Sqr(数値)	指定した数値の平方根を返す
a=Int(数値)	指定した数値を超えない整数を返す　Int(-3.6) → -4
a=Fix(数値)	指定した数値の整数部分を返す　Fix(-3.6) → -3
a=Rnd(初期値)	１未満の乱数を返す（乱数を発生させる場合 Randomize()ステートメントを実行しないと同じ値が繰り返されることがある）
a=Round(数値,桁数 n)	指定した数値の小数点以下 n－1 桁目をを四捨五入したものを返す
a=Format(数値,"#.##")	指定した数値の表示桁をあわせたものを返す

９．２　日にち・時間の関数

日にちや時間に関する関数の中の主なものを以下に示します。

a=Date	実行したときの日にちを返す（整数）
a=Time	実行したときの時間を返す（小数、0～24h→0～1）
a=Now	実行したときの日にちと時間を返す（Date と Time の合計）
b=Timer	0:00:00 からの経過秒数を返す
b=Year(日にち)	指定した日にちの年の部分を返す
b =Month(日にち)	指定した日にちの月の部分を返す
b =Day(日にち)	指定した日にちの日の部分を返す
b =Hour(時間)	指定した時間の時の部分を返す
b =Minute(時間)	指定した時間の分の部分を返す
b =Second(時間)	指定した時間の秒の部分を返す

a=Weekday(日にち)　指定した日にちの曜日を返す
a=WeekdayMane(Weekday,Abbreviate,Firstdayofweek)

変数の宣言は、上記の a（Date、Time、Now）については日付型（Date）で宣言したときには日時で、文字型（String）で宣言したときは日時が文字として表示されます。また数値型（Long、Intger など）で宣言したときは 1900 年 1 月 0 日 0 時 0 分 0 秒を基準にしたときの経過日時が数値で表示されます（実際には 1 月 0 日はありません）。

上記の b（Year、Month など）は数値型（Long、Intger など）で宣言すると数値で表示されます。Date のデータは整数で、Time のデータは 1 未満の小数で、Now のデータはこの二つをあわせた数値で表示されます。Intger や Sigle で宣言すると、桁数が足りないために正確なデータが表示されないことがあります。

９．３　文字列の関数

文字列に関する関数の中で主なものを以下に示します。

a=Len(文字列)　指定した文字列の長さ（文字数）を返す
a=Left(文字列,Length)　指定した文字列の左から指定した数の文字列を返す
a=Mid(文字列,Start,Length)　指定した文字列の指定した文字数から指定した文字数の文字列を返す
a=Right(文字列,Length)　指定した文字列の右から指定した数の文字列を返す
a=Chr(コード番号)　コード番号で指定した文字を返す
a=LTrim(文字列)　指定した文字列の先頭の空白を取る
a=RTrim(文字列)　指定した文字列の末尾の空白を取る
a=Trim(文字列)　指定した文字列の前後の空白を取る
a=Substitute(文字列, 検索文字列, 置換文字列, 置換対象)　指定位置から指定された数の文字列に置換する
a=Replace(文字列,"文字列１","文字列２")　指定された文字列内の文字列１を文字列２に置換する
a=Replace(文字列," ","")　指定した文字列内の空白を取る
a=Upper(文字列)　文字列の中の英字をすべて大文字に変換する
a=Lower(文字列)　文字列の中の英字をすべて小文字に変換する
a=Asc(文字列)　全角の英数カナ文字を半角カナ文字に変換する
a=Jis(文字列)　半角の整数カナ文字を全角文字に変換する
a=Val(文字列)　指定した文字列を数値に変換する
a=Str(数値)　指定した数値を文字列に変換する

文字の関数は、ファイルの読み出しや保存で使用することがあります。また、計測器などの装置とデータ通信を行うときに、文字列の関数が必要になります。

例題　01smpl を以下ように作り替えて、動作確認をして下さい。

[UserForm]
　　01smpl.xlsm　をそのまま使用

[コード]

```
Private Sub CommandButton1_Click()
    Dim a As Single
    Dim b As Single

    a = TextBox1.Text
    b = Sin(a)
    Label1.Caption = b
End Sub
```

この関数を実行すると、指定した数値の正接（Sin）がラベルに表示されます。角度の単位はラジアンなので、a として 3.14 や 1.57 などを指定して確認してください。

終わったら、他の関数についても同様に確認しておいてください 。また、変数を宣言する型の違いによる表示も合わせて確認してください。

```
    Dim a As _____
    Dim b As _____

    a = TextBox1.Text
    b = __________
    Label1.Caption = b
```

確認が終わったらファイル名を 08enzan.xlsm にして保存して下さい。

Ｄ．プログラムの流れ

１０．制御構造

コンピュータを使うと様々なことが簡単に自動でできます。たとえば、切符の自動販売機、おいしいご飯を炊く電子炊飯器、安全に走行する自動車など、全てコンピュータが使われています。これらのことができるのは、コンピュータのプログラム言語に「判断」機能と「ジャンプ」機能があるからです。

プログラムは、それぞれのイベントプロシージャの中で、記述されたコードを左から右へ、上から下へ順番に実行していきます。しかし、実際にコンピュータに処理をさせる場合、「この場合はこの処理を、そうでなければ別の処理を実行したい」、「条件によってはここだけを実行したい」、「ここを５回繰り返したい」、「この条件が満たすまで繰り返したい」といったことが起こります。そのような動作をさせるためには、プログラムの処理の流れを上手に指示しなければなりません。

VBA などのプログラム言語ではプログラムの流れを制御する命令として、「条件分岐」と「繰り返し」などがあります。ここではプログラムの流れを制御する代表的なステートメントについて説明します。

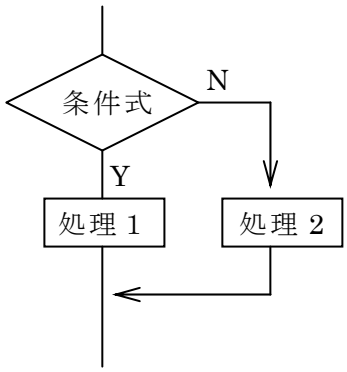
１０．１　ＩＦステートメント

IF ステートメントは、あらかじめ条件を定めておき、その条件に応じてプログラムの流れを制御するものです。

以下に代表的な流れのパターンとそのフローチャート、コードを示します。

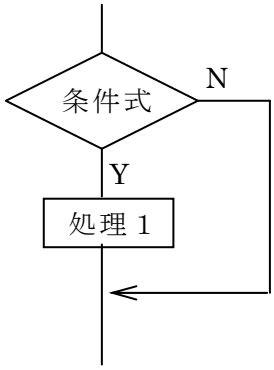
1)条件を満たせば処理１、満たさなければ処理２を実行

```
If ［条件式］ Then
    ［処理１］
Else
    ［処理２］
End If
```



このプログラムを実行すると、まず、If ステートメントの条件式を満たしているか否かを判断します。条件式が満たされたときは処理１を実行し、満たされなかったときは処理２を実行します。

2)条件を満たすと処理１を実行し、満たさなければ何もしない

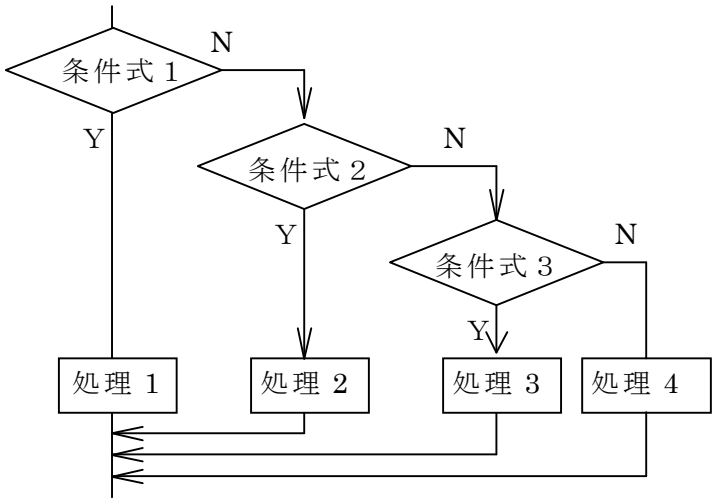



```
If ［条件式］ Then
    ［処理１］
End If
```

このプログラムを実行すると、まず If ステートメントの条件式を満たしているか否かを判断します。条件式が満たされれば処理１を実行し、満たされなければそのまま If ステートメントを抜けます。

3)条件１を満たせば処理１を実行、満たさなければ次を判断
条件１を満たさず条件２を満たせば処理２を実行、満たさなければ次を判断。
条件１、２を満たさず条件３を満たせば処理３を実行、満たさなければ処理４を実行

```
If ［条件式１］ Then
    ［処理１］
 Else If ［条件式２］ Then
    ［処理２］
 Else If ［条件式 3］ Then
    ［処理 3］
 Else
    ［処理 4］
 End If
```



このプログラムを実行すると、最初に条件式１を満たしているか否かを判断し、満たしていれば処理１を実行して IF ステートメントを抜けます。満たしてなければ次に条件式２を満たしているか否かを判断し、満たしていれば処理２を実行して IF ステートメントを抜けます。満たしていなければ条件式３を判断します。満たしていれば処理３を実行し、満たしていなければ処理４を実行して IF ステートメントを抜けます。

このように、IF ステートメントでは条件式を満たしているか否かで実行する内容を変えていきます。このときの条件式には比較演算子を使い、数値の等式や不等式が入ります。

a=5	a は５
b<>3	b は３でない
c>=30	c は３０以上

条件式には文字列の式も入れることができます。

wdy="金曜日"　　wdy は「金曜日」

name<>"上杉謙信"　name は「上杉謙信」でない

kamoku=""　　kamoku は空欄（データが入っていない）

kamoku<>""　　kamoku は空欄でない（何らかのデータが入っている）

また条件が複数あるときは、論理演算子を使い条件設定します。

c>3 And c<=5　　（c は 3 より大きく、かつ、5 以下）

wdy="土曜日" Or wdy="日曜日"　（wdy は「土曜日」または「日曜日」）

(mth="１月" Or mth="２月") And (wdy="月曜日" Or wdy="水曜日")

（mth は「1 月」または「2 月」、かつ、wdy は「月曜日」または水曜日」）

例題　01smpl を以下のように作り替えて、動作確認をして下さい。

[UserForm]

01smpl.xlsm　をそのまま使用

[コード]

```
Private Sub CommandButton1_Click()

    Dim a As Integer
    a = TextBox1.Text

    If a >= 10 Then
        Label1.Caption = ″a は１０以上″
    Else
        Label1.Caption = ″a は１０未満″
    End If

End Sub
```

テキストボックスに数値を入れて、その数値が１０以上であれば「a は１０以上」と表示されます。数値が１０未満のときは「a は１０未満」と表示されます。このようにＩＦステートメントで a が１０以上か否かを判断し、a の値によって実行する処理を変えていることがわかります。

確認が終わったらファイル名を 11if.xlsm に変更して保存して下さい。

１０．２　Select ステートメント

IF ステートメントで条件が連続してたくさんある場合､IF ステートメントの代わりに、Select Case ステートメントを使うことが出来ます。条件式によっては、Select Case ステートメントを使った方がプログラムを簡単に組めることがあります。

変数に入るデータがあらかじめ分かっており､そのデータによって条件分岐を行う場合は以下のように用います。

```
Select  Case  変数                ’変数を指定
    Case  1               ’変数が１のとき
        [処理 1]
    Case  2               ’変数が２のとき
        [処理 2]
    Case  3               ’変数が３のとき
        [処理 3]
    Case  Else            ’変数がそれ以外のとき
        [処理 4]
End  Select
```

Select Case ステートメントを使う場合、条件を満足するものを一つだけ実行し、ほかに条件を満たすものがあっても、それらは実行されません
また、条件分岐の変数の条件を以下のようにすることもできます。

```
Select  Case  a                   ’変数 a を指定
    Case  1  ･･･                  ’a が１のとき
        [処理 1]
    Case  2  ･･･                  ’a が２のとき
        [処理 2]
    Case  3  to  6  ･･･           ’a が３から６のとき
        [処理 3]
    Case  7 , 9 , 11  ･･･         ’a が７または９または１１のとき
        [処理 4]
    Case  Is <-3  ･･･             ’a がー３未満のとき
        [処理 5]
    Case  Else  ･･･               ’a がそれ以外のとき
        [処理 6]
End  Select
```

以下に IF ステートメントと、Select Case ステートメントの比較を示します。

```
If  a=1  Then  ･･･                ’a が１のとき
       [処理 1]
ElseIf  a=2  Then  ･･･            ’a が２のとき
       [処理 2]
ElseIf  a>3  And  a<6  Then  ･･･  'a が３から６のとき
       [処理 3]
ElseIf  a=7  Or  a=9  Or  a=11  Then  ･･･   'a が７または９または１１のとき
       [処理 4]
ElseIf  a<-3  Then  ･･･           'a がー３未満のとき
       [処理 5]
Else  ･･･                         'a がそれ以外のとき
       [処理 6]
Enf  If
```

応用的な使い方になりますが、変数の部分に「True」を入れると、Case の後の条件式が満たしたときに、その処理を実行します。

```
Select  Case  True          ’条件式が満たされたとき
    Case  a=1               ’a ＝１のとき
        [処理 1]
    Case  a=2               ’a ＝２のとき
        [処理 2]
    Case  a>=3 And a<=6     ’３≦a ≦６のとき
        [処理 3]
    Case  a=7 Or a=9 Or a=11    ’a ＝７または９または１１のとき
        [処理 4]
    Case  Else              ’a がそれ以外のとき
        [処理 5]
End  Select
```

さらにオプションボタンを使う場合、以下の様な処理が可能になります。

```
Select  Case  True              ’条件式が満たされたとき
    Case OptionButton1.Value    ’オプションボタン１が選択されたとき
        [処理 1]
    Case OptionButton2.Value    ’オプションボタン２が選択されたとき
        [処理 2]
    Case  Else                  ’それ以外のとき
        [処理 3]
End  Select
```

例題　３個のオプションボタン（１，２，３）、ラベル１、コマンドボタン１を配置。

オプションボタン１（以下 OPB１）を選択して CB1 をクリックするとラベルに「１」と表示

OPB2 を選択して CB1 をクリックすると「２」と表示

OPB３を選択して CB1 をクリックすると「３」と表示

[UserForm]

```
OptionButton1
OptionButton2
OptionButton3
Label1
CommandButton1
```

[コード]

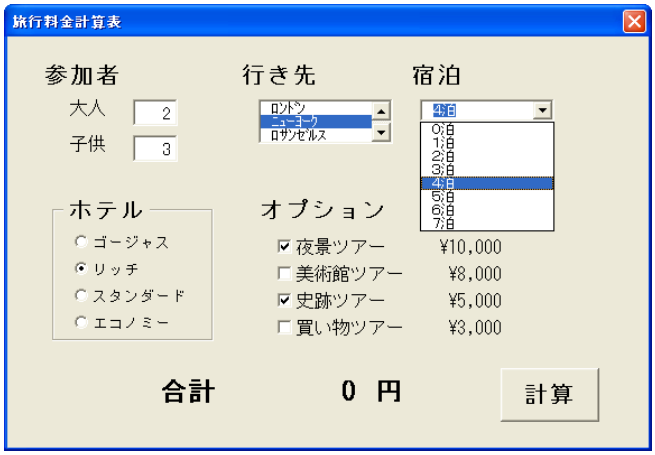
```
Private Sub CommandButton1_Click()
    Dim a As String

    Select Case True
        Case OptionButton1.Value
            a = ″１が選択された″
        Case OptionButton2.Value
            a = ″２が選択された″
        Case OptionButton3.Value
            a = ″３が選択された″
        Case Else
            a = ″何か選択してください″
     End Select

    Label1.Caption = a

End Sub
```

例題　次のプログラムを作り、その動作確認をして下さい。

（下のプログラムには間違いがあります。間違いを見つけて修正してください。）

[UserForm]

CommandButton1.Caption	計算	
Label1.Caption	参加者	
Label2.Caption	大人	
Label3.Caption	子供	
Label4.Caption	行き先	
Label5.Caption	宿泊	
Label6.Caption	ｵﾌﾟｼｮﾝ	
Label7.Caption	合計	
Label8.Caption	0	
TextBox1.Text	1	
TextBox2.Text	0	
ListBox1		
ComboBox1		
Frame１.Caption	ホテル	
OptionButton1.Caption	ゴージャス	
OptionButton2.Caption	リッチ	
OptionButton3.Caption	スタンダード	
OptionButton4.Caption	エコノミー	
CheckBox1.Caption	夜景ツアー	¥10,000
CheckBox2.Caption	美術館ツアー	¥8,000
CheckBox3.Caption	史跡ツアー	¥5,000
CheckBox4.Caption	買い物ツアー	¥3,000



図は「宿泊」を選択している状態

[コード]

```
Option Explicit

Private Sub UserForm_Initiarize()
    ListBox1.AddItem "ﾛﾝﾄﾞﾝ"                ’リストボックスのデータ設定
    ListBox1.AddItem "ﾆｭｰﾖｰｸ"
    ListBox1.AddItem "ﾛｻﾝｾﾞﾙｽ"
    ListBox1.AddItem "ﾊﾜｲ"
    ListBox1.AddItem "ｼﾄﾞﾆｰ"
    ListBox1.AddItem "香港"
    ListBox1.AddItem "小樽"
    ComboBox1.AddItem "０泊"                ’コンボボックスのデータ設定
    ComboBox1.AddItem "１泊"
    ComboBox1.AddItem "２泊"
    ComboBox1.AddItem "３泊"
    ComboBox1.AddItem "４泊"
    ComboBox1.AddItem "５泊"
    ComboBox1.AddItem "６泊"
    ComboBox1.AddItem "７泊"
End Sub

Private Sub CommandButton1_Click()
    Dim otona As Integer                    ’変数の宣言
    Dim kodomo As Integer
    Dim ninnzuu As Single
    Dim hikouki As Long
    Dim syukuhaku As Integer
    Dim hotel As Long
    Dim opsyon As Integer
    Dim kinngaku As Long
```

```
    otona = 0                                   '変数の初期値設定
    kodomo = 0
    ninnzuu = 0
    hikouki = 0
    syukuhaku = 0
    hotel = 0
    opsyon = 0
    kinngaku = 0

    otona = TextBox1.Text                       '参加人数の設定
    kodomo = TextBox2.Text
    ninnzuu = otona + kodomo / 2

    Select Case ListBox1.ListIndex              '目的地までの飛行機料金の設定
        Case 0:
            hikouki = 120000
        Case 1:
            hikouki = 110000
        Case 2:
            hikouki = 100000
        Case 3:
            hikouki = 80000
        Case 4:
            hikouki = 60000
        Case 5:
            hikouki = 40000
        Case 6:
            hikouki = 20000
    End Select

    syukuhaku = ComboBox1.ListIndex             '宿泊数の設定

    If OptionButton1.Value = True Then          'ホテル料金の設定
        hotel = 20000
    ElseIf OptionButton2.Value = True Then
        hotel = 15000
    ElseIf OptionButton3.Value = True Then
        hotel = 10000
    ElseIf OptionButton4.Value = True Then
        hotel = 6000
    End If

    If Check1.Value = 1 Then                    'オプショナルツアー料金
        opsyon = opsyon + 10000
    End If
    If Check2.Value = 1 Then
        opsyon = opsyon + 8000
    End If
    If Check3.Value = 1 Then
        opsyon = opsyon + 5000
    End If
    If Check4.Value = 1 Then
        opsyon = opsyon + 3000
    End If

    kinngaku = (hikouki _
    + hotel + syukuhaku + opsyon) * ninnzuu      '合計金額の計算

    Label3.Caption = kinngaku

End Sub
```

このプログラムは、IF ステートメント、Select Case ステートメントを使った、旅行予算算出を行うプログラムです。いろいろな組み合わせのコースの設定をすると、そのコースの大人一人あたりの単価をもとめ、その単価に人数を乗じています。子供料金は全て大人料金の半額としています。

コースの設定は、行き先、宿泊数、ホテルのグレード、オプショナルツアー、でそれぞれを選択します。行き先、宿泊、ホテルは一つだけを選択しますが、オプショナルツアーは複数選択できるようなっています。選択する項目それぞれに金額が決まっており、その総和が大人一人あたりの単価になります。

行き先はリストボックスで選択するようになっています。n 番目の項目が選択されると、リストボックスのプロパティ、List1.ListIndex に n が入力されます。 n には 0、1、2、3･･･が入ります。

宿泊数はコンボボックスで選択します。リストボックス同様、Combo1.ListIndex に選択した項目の番号が入ります。

ホテルはフレームとオプションボタンを使います。最初にフレームを配置し、そのフレームの中にオプションボタンを配置します。フレームの中にオプションボタンを配置することにより、そのフレームの中から一つを選択するようになります。選択されたオプションボタンは、そのプロパティ、Option1.Value が True になります。

オプショナルツアーの選択はチェックボックスを使います。チェックボックスを使うと複数個の選択ができます。選択されたチェックボックスは、そのプロパティ、Check1.Value が 1 になります。

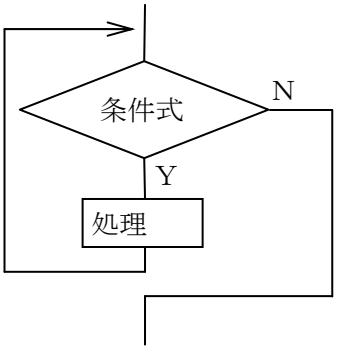
これらの設定を行った後にコマンドボタンをクリックすると、選択された項目に対応して金額を与えられ、その総和に人数を乗じることにより、合計金額を求めます。

確認が終わったらファイル名を 12ryokou.xlsm にして保存して下さい。

１０．３　Ｄｏループ

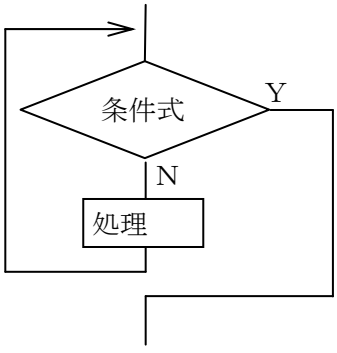
Do ループは条件を満たしているかどうかを判断して、同じ動作を繰り返し実行するステートメントです。Do ループには、ある条件が満たされている間（繰り返し条件）、もしくは満たしていない間（終了条件）だけ処理を繰り返します。満たしている間に繰り返す時は While 文を使い、満たしていない間に（満たされるまで）繰り返す時は Untl 文を使います。

以下にそのフローチャートと使用例を示します（前判定ループ文）。



条件が満たされている間（繰り返し条件）

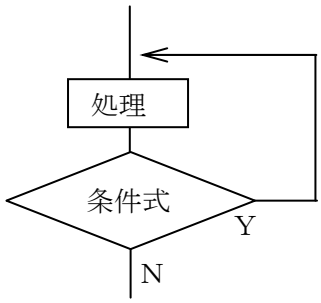
```
Do  While ［条件］
    ［処理］
Loop
```



条件が満たされるまで（終了条件）

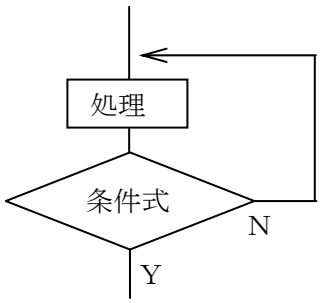
```
Do  Until ［条件］
    ［処理］
Loop
```

Do While（Do Until）では条件によっては処理が一度も実行されない場合が生じます。必ず一度は処理を実行したいときは、Loop の後に While 文や Until 文を持ってきます（後判定ループ文）。



条件が満たされている間（繰り返し条件）

Do
　[処理]
Loop While　[条件]



条件が満たされるまで（終了条件）

Do
　[処理]
Loop Until　[条件]

例題　ワークシートのサンプルの表を用意する（例えば「製造業マップ」の”A1”～”E100”）。ユーザーフォームのコマンドボタンをクリックすると１行目からデータの有無を確認し、行にデータが入っていなければ、その行の１列目に”ＡＡＡ”、２列目に”ＢＢＢ”、３列目に”ＣＣＣ”と入力する。

[UserForm]
　CommandButton1.Caption　　実行

[コード]

```
Option Explicit

Private Sub CommandButton1_Click()

    Dim n As Integer

    n = 1
    Do While (Cells(n, 1).Value   <>   "")
        n = n + 1
    Loop

    Cells(n, 1).Value = "ＡＡＡ"
    Cells(n, 2).Value = "ＢＢＢ"
    Cells(n, 3).Value = "ＣＣＣ"

End Sub
```

変数ｎを確認する行番号にします。データの有無確認を行う行を１行目（n=1）からにし、ｎ行１列目のデータの有無を確認します。データがあれば(条件を満たせば)次の行の確認を行い(ｎ=n+1)、データがなければその行の１列目に「ＡＡＡ」を、２列目に「ＢＢＢ」を、３列目に「ＣＣＣ」を入れます。

確認が終わったらファイル名を 13Do.xlsm に変更して保存して下さい。

１０．４　For ループ

For ループは同じ処理を繰り返し行うときに使います。For ループを使うときは、インデックス（変数）を用意し、インデックスの開始値、終了値、その間隔を指定します。

以下に、使用例を示します。

```
For  I=[開始値]  To  [終了値]  Step  [間隔]
    [処理]
Next
```

「間隔」に負の数を入れるとインデックス（上の例ではＩ）を減らすことが出来ます。また「間隔」を省略すると、そのときの間隔は１になります。

例題　01smpl を以下のように作り替えて、動作確認をして下さい。

[UserForm]

01smpl.xlsm　をそのまま使用

[コード]

```
Private Sub CommandButton1_Click()
    Dim a As Integer
    Dim b As Long
    Dim I As Long

    a = TextBox1.Text
    b=0
    For I=a to a+5
        b=b+I
    next
        Label1.Caption = b
End Sub
```

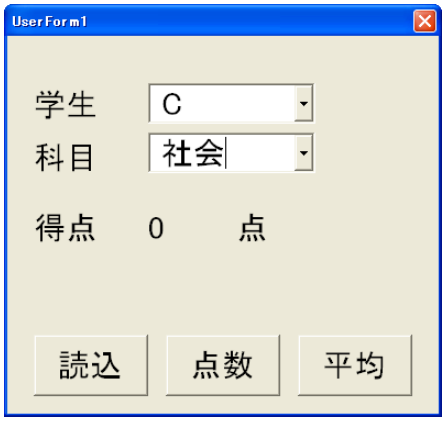
これを実行すると、まず a の初期値をテキストボックスから入力します。次に For ステートメントを使い、Ｉの値を a、a+1、a+2、a+3、a+4、a+5 と変えます。このときのＩを次々と b に加え、以下のように計算していきます。

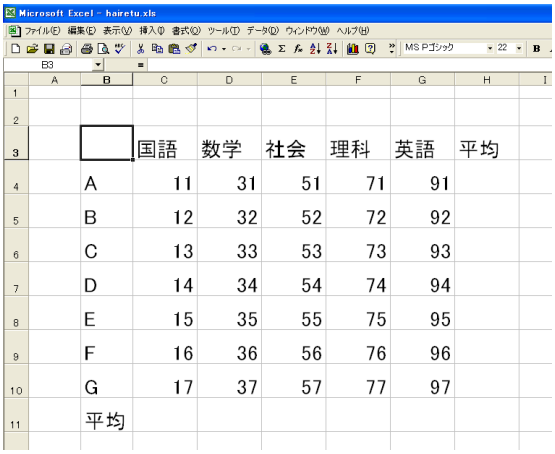
```
b=0+a
b=(0+a)+a+1
b=(0+a+a+1)+a+2
b=(0+a+a+1+a+2)+a+3
                  ・
                  ・
b=6*a+15
```

となり、最後に、計算結果ｂを表示させます。

確認が終わったらファイル名を 14for.xlsm に変更して保存して下さい。

例題　下のようなワークシート、およびユーザーフォームを作成する。





「読込」がクリックされたら、ワークシートのデータを配列ａ(ｉ,ｊ) に代入する

「点数」がクリックされたら、指定された学生、科目の得点をラベルに表示する。

「平均」がクリックされたら、個人の平均点、科目の平均点を各セルに表示する。

なお、学生の選択、科目の選択はコンボボックスとし、ユーザーフォームが立ち上がったときにリストを代入するようにすること

［コード］

```
Option Explicit

Dim a(100, 100) As Integer

Private Sub UserForm_Initialize()
    Dim i As Integer
    Dim j As Integer

    For j = 1 To 7
        ComboBox1.AddItem Cells(j + 3, 2)
    Next
    For i = 1 To 5
        ComboBox2.AddItem Cells(3, i + 2)
    Next

End Sub

Private Sub CommandButton1_Click()
    Dim i As Integer
    Dim j As Integer

    For i = 1 To 5
        For j = 1 To 7
            a(i, j) = Cells(j + 3,
                            i +2).Value
        Next
    Next

End Sub
```

```
Private Sub CommandButton2_Click()
    Dim m As Integer
    Dim n As Integer

    m = ComboBox1.ListIndex + 1
    n = ComboBox2.ListIndex + 1

    Label4.Caption = a(n, m)

End Sub

Private Sub CommandButton3_Click()
    Dim i As Integer
    Dim j As Integer
    Dim sump(100) As Integer
    Dim sums(100) As Integer

    For j = 1 To 7
        For i = 1 To 5
            sump(j) = sump(j) + a(i, j)
        Next
        Cells(j + 3, 8).Value = sump(j) / 5
    Next

    For i = 1 To 5
        For j = 1 To 7
            sums(i) = sums(i) + a(i, j)
        Next
        Cells(11, i + 2).Value = sums(i) / 7
    Next

End Sub
```

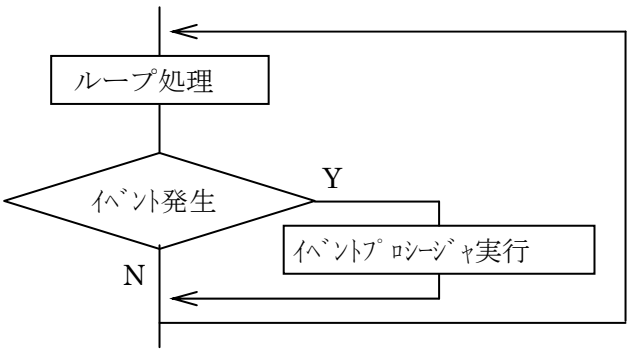
１０．５　Exit ステートメント

Exit ステートメントは、For ループや Do ループを使って繰り返し処理を行っているときに、条件を満たす前にループから抜けるときに使います。If ステートメントと組み合わせて使います。

Exit	Do	Do ループを終了する。
Exit	For	For ループを終了する。
Exit	Function	Function プロシージャを終了する。
Exit	Sub	Sub プロシージャを終了する。

１０．６　Do　Events ステートメント

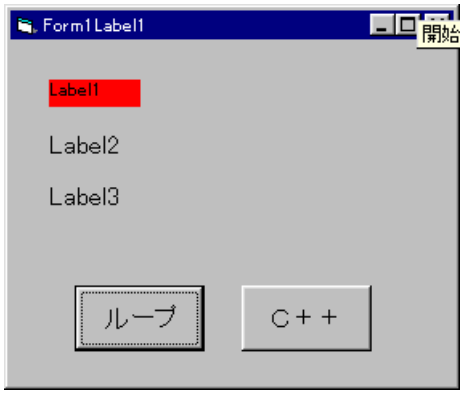
VBA では、ループ処理を実行中は OS はほかの処理（表示、描画、イベントの受付けなど）を実行しません。ループ処理実行中でも OS がこのような処理を実行するようにするためには、DoEvents ステートメントを使い、ループ処理中にこのステートメントを実行します。



これにより、ループ実行中であっても、イベントが発生したときは、Do Events で一旦ループを抜け、発生したイベントに対するイベントプロシージャを実行した後、もしくは文字表示や描画の処理を実行した後、またループに戻り、処理を繰り返します。

例題　次のプログラムを作り、その動作確認をして下さい。

```
[UserForm]
     Label1.BackColor = &h000000FF&
     Label2
     Label3
     CommandButton1.Caption        ループ
     CommandButton2.Caption        C＋＋

[コード]
     Option Explicit
         Dim c As Long

     Private Sub CommandButton1_Click()
         Dim I As Integer
         I = 0
         c = 0

         Do While (c < 10)                              ’ループ開始
             I = I + 1
             If I > 200 Then
                 I = 0
             End If

             Label1.Width = I                           ’ループ実行中表示
             Label1.Caption = I                         ’ループ実行中表示
             Label2.Caption = ″ループ実行中″
```



```
        Label3.Caption = c                      ' c の値表示

        DoEvents                                'イベントの受け付け
    Loop                                        'ループ終了

    Label2.Caption = "ループから抜けた"
End Sub

Private Sub CommandButton2_Click()
    c = c + 1
End Sub
```

「ループ」をクリックすると c の初期値（c=0）が表示され、Do ループに入ります。Do ループに入っている間、Label1 の赤い棒は伸び縮みを繰り返します。「C＋＋」をクリックすると、DoEvents によって一旦ループから抜け、発生したイベントに対するイベントプロシージャを実行ます。ここでは c=c+1 を実行し、その値を表示します。そこでの処理が終わると、元のループに戻ります。ここでは赤い棒の伸び縮みが続いていることにより、ループの中を実行していることが確認できます。c の値が 10 になると、Do ステートメントの条件を満たさなくなるので「ループから抜けた」と表示され、ループから抜け、赤い棒の伸び縮みが止まります。

確認が終わったらファイル名を 15doevents.xlsm にして保存して下さい。

このコードから DoEvents を削除して、Do ループに入ると、以後のイベントは一切受け付けず、無限ループに入ります。プログラムの処理速度は早くなりますが、イベントを受け付けないため、このプログラムを終わらせることさえも出来ません。「Ctrl」＋「Alt」＋「Del」で強制的に終了させることは出来ますが、これを実行すると、Excel 自体が終了してしまいます。この動作確認を行う前に、必ずファイルの保存を実行して下さい。

上記例題の Do ステートメントを以下のように変えても同様な動作をします。

Do Until c > 9

このように書き換えると c が 9 より大きくなるまでループ内を実行します。

このプログラムでは、Do ループの条件を満たすまでループから抜けることはできません。そこで、Exit Do というステートメントを使い、ループの条件を満たさなくても、強制的にループから抜けらるようにしてみます。

新たに flg という変数を使い、ユーザーフォームをクリックしてこの flg が 0 になったらループから抜けるようにします。

例題　先程のプログラムを、ユーザーフォームをクリックしたらループから抜けるように作り替えて、動作確認をして下さい。

```
Option Explicit
    Dim c As Long
    Dim flg As Long                     ←追加

Private Sub CommandButton1_Click()
    Dim I As Integer
    flg = 1                             ←追加
    c = 0
    I = 0
    Label3.Caption = c

    Do While c < 10
    DoEvents
    I = I + 1
        If flg = 0 Then
            Exit Do                     ←追加
        End If
        If I > 200 Then
            I = 0
        End If
        Label1.Width = I
        Label2.Caption = "ループ実行中"
    Loop
    Label2.Caption = "ループから抜けた"
End Sub

Private Sub CommandButton2_Click()
    c = c + 1
    Label3.Caption = c
End Sub

Private Sub UserForm_Click()            ←追加
    flg = 0
End Sub
```

変数 flg はユーザーフォーム全体で使用するので、宣言領域で Dim を使って宣言します。「ループ」をクリックすると、flg＝１にして Do ループに入ります。ループ内には DoEvents ステートメントがあるので、イベントを受け付けるので、「Ｃ＋＋」をクリックすると、一旦ループを抜けてから c=c+1 を実行し、再びループに戻ります。また、ユーザーフォームをクリックすると、そのイベントプロシージャを実行し flg＝０にし、ループに戻ります。ループに戻ると、次の IF ステートメントで、flg＝０になっていることから、Exit Do により、ループから抜けます。

確認が終わったらファイル名を 16exitdo.xlsm にして保存して下さい。

また、Do ループを以下のようにしても、同様な動作をします。

```
Do While c < 10 and flg=1
    Label2.Caption = c
    DoEvents
Loop
```

ここでは、ｃ＜１０、かつ flg=1 ならばループを繰り返します。ｃ≧１０、または flg=0 になったならばループから抜けます。

１１．プロシージャ

複雑な処理をさせるためのプログラムを組んでいくと、同じ処理を何度もさせることがあります。その度に同じコードを組んでいくとプログラムが長くなってしまいます。また、変更が生じた場合、何箇所も同じ修正をしなければなりません。このように同じ処理を何度も実行するようなときは、プロシージャと呼ばれる決められた処理を実行するコードのブロックを作っておき、必要なときにこのプロシージャを呼び出して、処理を実行します。プロシージャを使うことにより、プログラムが短くしたり、デバッグ作業やプログラムの修正も容易になります。また、各処理をブロックごとに分けることから、わかりやすいプログラムを容易に組むことが出来るようになります。

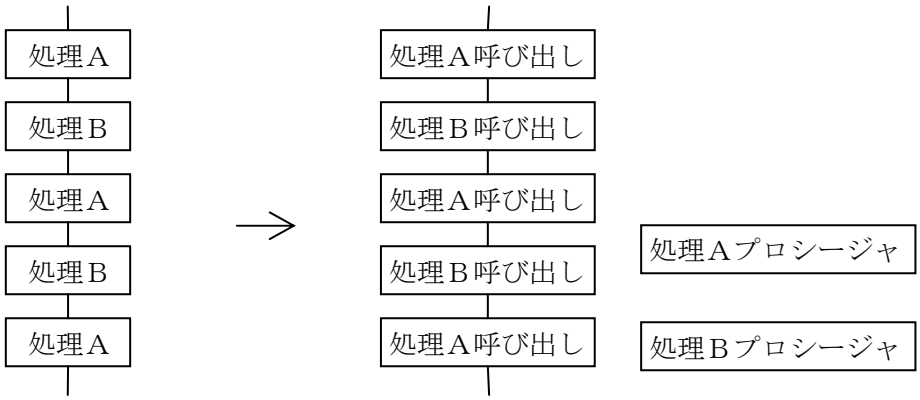
下図のような流れのプログラムを組む場合、同じ内容の処理Ａコードが３個、同じ内容の処理Ｂコードが二つ必要になります。

処理Ａ→処理Ｂ→処理Ａ→･･･

ここで、処理Ａコードを一つの処理Ａプロシージャとして作っておきます。同様に処理Ｂについても処理Ｂプロシージャを作ります。メインのコードを

処理Ａﾌﾟﾛｼｰｼﾞｬ呼び出し→処理Ｂﾌﾟﾛｼｰｼﾞｬ呼び出し→処理Ａﾌﾟﾛｼｰｼﾞｬ呼び出し→･･

としておくと、同じ動作をさせることが出来ます。

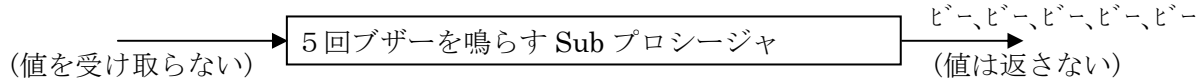


プロシージャには、Sub プロシージャと Function プロシージャの二つがあります。Sub プロシージャは、処理を実行するだけのプロシージャです。Function プロシージャは単に処理を実行するほかに、処理を実行した後にその結果を返すこともできます。Function プロシージャは実行後の結果を返すことから関数とも呼ばれています。

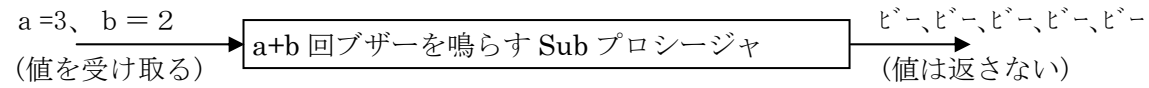
数値等のデータをプロシージャに送って処理を実行させる場合、その送るデータのことを引数といいます。Sub プロシージャ、Function プロシージャとも、引数がある場合とない場合があります。

また、Function プロシージャから返ってくる値のことを戻り値といいます。あたかもプロシージャ自体が値を持ち、その値を変数に代入したり、その値を演算処理に使ったりします。

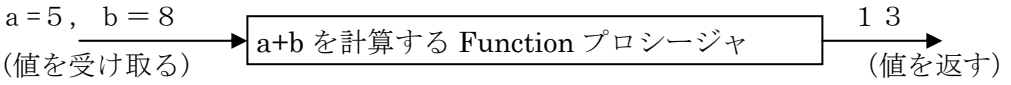
Sub プロシージャの場合　（引数なし）



Sub プロシージャの場合　（引数あり）



Function プロシージャの場合　（引数あり　、戻り値あり）



１１．１　プロシージャの宣言

Sub プロシージャ、および Function プロシージャは、「ここからここまでが、何々という名前のプロシージャです」と宣言して作ります。宣言では、そのプロシージャが利用できる範囲(適用範囲)、プロシージャの種類、プロシージャの名前、引数を受け取る変数を宣言します。

［適用範囲］［種類］［プロシージャ名］_
　　(変数１ As データ型, 変数２ As データ型・・・) As データ型
　　［処理］　　↑
EndSub（もしくは EndFunction）　　Function プロシージャの場合

プロシージャの適用範囲の指定

Private	宣言したフォーム内から
Public	全てのフォームから（標準モジュールで宣言）

プロシージャの種類

Function	戻り値を持つプロシージャ
Sub	戻り値を持たないプロシージャ

プロシージャ名、変数名は自由に決めることができます。また引数がある場合はその変数の宣言も行い、戻り値がある場合はそのデータの型も宣言します。

プロシージャの中の処理は、通常の処理と同じようにプログラムが組まれます。引数、戻り数のある Function プロシージャの例を示します。

```
                .
                .
                .
d = functest(3,-5,6)
                .
                .
                .
Private Function functest( a As Integer, b As Integer, c As Integer ) As Single
    functest = b^2 - 4 * a * c
    Label1.Caption = (-1*b+sqr(functrest))/2/a
    Label2.Caption = (-1*b-sqr(functrest))/2/a
End Function
```

１１．２　プロシージャの呼び出し

プロシージャの呼び出す方法は、戻り値を必要とする場合と、必要としない場合で異なります。

Sub プロシージャ、および戻り値を必要としない Function プロシージャの呼び出し

Call　[プロシージャ名](引数１、引数２，引数３・・・)

このようにプロシージャを呼び出し、そこでの処理をさせて終わります。Function プロシージャを Call で呼び出したとき、その戻り値は破棄されてしまいます。

Funcion プロシージャを呼び出してその戻り値を必要とするときは､そのプロシージャ自体があたかも変数のように値（戻り値）を持っているので、代入式や演算式などで戻り値を使います。

変数　＝　[プロシージャ名](引数１，引数２，引数３・・・)

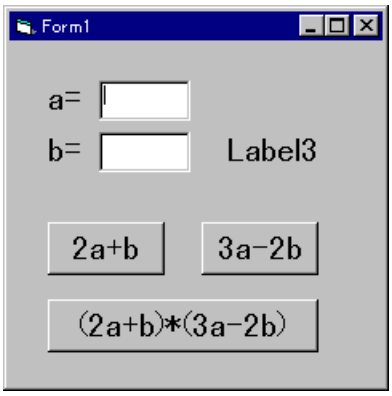
変数　＝　[プロシージャ名](引数１，引数２，引数３・・・)＊３＋７

If　[プロシージャ名](引数１，引数２，引数３・・・)　＞10　Then　･･･

例題　以下のプログラムを作り、その動作確認をして下さい。

[UserForm]

Label1.Caption　　a ＝
Label2.Caption　　b ＝
Label3
TextBox1
TextBox2
CommandButton1.Caption　　2a+b
CommandButton2.Caption　　3a-2b
CommandButton3.Caption　　(2a+b)＊(3a-2b)



[コード]

```
Option Explicit

Private Sub CommandButton1_Click()
    Dim a1 As Integer
    Dim b1 As Integer
    a1 = TextBox1.Text          '数値の入力
    b1 = TextBox2.Text
    Call keisan1(a1, b1)        'keisan1 プロシージャの呼び出し
End Sub

Private Sub CommandButton2_Click()
    Dim a2 As Integer
    Dim b2 As Integer
    a2 = TextBox1.Text          '数値の入力
    b2 = TextBox2.Text
    Call keisan2(a2, b2)        'keisan2 プロシージャの呼び出し
End Sub

Private Sub CommandButton3_Click()
    Dim a3 As Integer
    Dim b3 As Integer
    a3 = TextBox1.Text          '数値の入力
    b3 = TextBox2.Text
    Call keisan3(a3, b3)        'keisan3 プロシージャの呼び出し
```

```
Private Sub keisan1(ak1 As Integer, bk1 As Integer)     'keisan1プロシージャ
    Dim ck1 As Integer
    ck1 = 2 * ak1 + bk1
    Label3.Caption = ck1
End Sub

Private Sub keisan2(ak2 As Integer, bk2 As Integer)     'keisan2プロシージャ
    Dim ck2 As Integer
    ck2 = 3 * ak2 - 2 * bk2
    Label3.Caption = ck2
End Sub

Private Sub keisan3(ak3 As Integer, bk3 As Integer)     'keisan3プロシージャ
    Dim ck3 As Integer
    ck3 = (2 * ak3 + bk3) * (3 * ak3 - 2 * bk3)
    Label3.Caption = ck3
End Sub
```

このプログラムでは、同じ計算を計算を実行する Sub プロシージャを作り（keisan1、keisan2、keisan3)、それぞれの Sub プロシージャに数値データを引数として引き渡して計算を行い、計算結果をラベルに表示しています。

イベントプロシージャで使用している全ての変数は、イベントプロシージャ内で「Dim」によって宣言されており、そのイベントプロシージャ内でしか利用できません。Sub プロシージャや Function プロシージャには変数に格納されている「データ」を引数として引き渡しています。

引数を受け取る側のプロシージャでは、別の変数名で引数を受け取ることができます（勿論、同じ変数名でも構いませんが、あくまでもそれは別の変数となります)。受け取った引数を格納する変数の宣言を、プロシージャ名のすぐ後ろで行います。このとき、受け取るデータの数や型が違うとエラーになります。

コマンドボタン１をクリックすると、a1、b1 にテキストボックスの数値が入力されます。次に計算を行う Sub プロシージャ(keisan1)を呼び出し、a1、b1 のデータを引数として引き渡します。keisan1 では引数を受け取り、計算をして、その計算結果を表示させています。

他の計算も同じように行っています。

確認が終わったらファイル名を 21sub.xlsm にして保存して下さい。

例題　先程のプログラムを次のように書き直し、その動作確認をして下さい。

［コード］

```
Option Explicit

Private Sub CommandButton1_Click()
    Dim a1 As Integer
    Dim b1 As Integer
    Dim z1 As Integer
    a1 = nyuuryokua()
    b1 = nyuuryokub()
    z1 = keisan1(a1, b1)
    Call hyouzi(z1)
End Sub
```

```
Private Sub CommandButton2_Click()
    Dim a2 As Integer
    Dim b2 As Integer
    Dim z2 As Integer
    a2 = nyuuryokua()
    b2 = nyuuryokub()
    z2 = keisan2(a2, b2)
    Call hyouzi(z2)
End Sub

Private Sub CommandButton3_Click()
    Dim a3 As Integer
    Dim b3 As Integer
    Dim z3 As Integer
    a3 = nyuuryokua()
    b3 = nyuuryokub()
    z3 = keisan1(a3, b3) * keisan2(a3, b3)
    Call hyouzi(z3)
End Sub

Private Function nyuuryokua() As Integer    '数値入力プロシージャ
    nyuuryokua = TextBox1.Text
End Function

Private Function nyuuryokub() As Integer    '数値入力プロシージャ
    nyuuryokub = TextBox2.Text
End Function

Private Function keisan1(a1 As Integer, b1 As Integer) As Integer
                                            '数値計算プロシージャ
    keisan1 = 2 * a1 + b1
End Function

Private Function keisan2(a2 As Integer, b2 As Integer) As Integer
                                            '数値計算プロシージャ
    keisan2 = 3 * a2 - 2 * b2
End Function

Private Sub hyouzi(z As Integer)            '結果表示プロシージャ
    Label3.Caption = z
End Sub
```

コマンドボタン１をクリックすると、まず始めに変数の宣言を行います。次にデータを入力する Function プロシージャを呼び出します。このプロシージャには引数がなく、戻り値は

三つのコマンドボタンのうちのいずれかをクリックすると、テキストボックスの数値をａ、ｂに入力する nyuuryoku プロシージャを呼び出します。ここでは引数を必要としないので、引数は送りません。次に計算を行う Function プロシージャを呼び出し、引数ａ、ｂを送ります。計算結果が戻り数として戻ってきます。最後に、計算結果を表示させる hyouzi プロシージャを呼び出し、引数ｚを送り、計算結果を表示させます。

確認が終わったらファイル名を **22sub1.xlsm** にして保存して下さい。

E．表の操作

１２．ブック・ワークシート・セルの操作

ExcelVBA を使うと、本来の表計算でのワークシートやセル、さらにはそのファイルデータであるブックまでも操作することが出来ます。ここからはしばらくユーザーフォームから離れ、Excel の表を操作する方法について説明します。

１２．１　マクロ言語

Excel の高度な使い方を学ぶ場合､通常は VBA を始める前にマクロの登録および実行から入ります。マクロ機能とは、ワークシート上で人が操作する内容をマクロ言語でプログラム化して記録し、このプログラムを実行することにより同じ内容を自動で実行する機能のことをいいます｡この動作の内容を記録したものをマクロプログラムと呼びます｡マクロプログラムは、自動で作成することも出来ますし、また人があらかじめ作ることも出来ます。

Excel の自動処理として、最初はマクロ言語が導入され、その後、新たにプログラム言語の VBA が導入されました｡マクロ言語と VBA は類似していますが、厳密にいうとこの二つは異なります｡使用するウインドウの部品も異なり、VBA では「コントロール」を使いますが、マクロ言語では「フォーム」を使います。これらも似ていますが、全く異なるものです。

マクロ言語と VBA には若干の違いがありますが、大部分が類似しているため、マクロプログラムの自動作成は VBA プログラムの作成に大いに役立ちます。ここからしばらくは、VBA の補助的なものとしてマクロ言語の活用方法について説明します。

世の中の書籍にはマクロ言語と VBA が入り乱れており、どちらを学習するのかにより、選定する図書も異なります。このテキストでは最終的にはプログラム言語の Visual Basic を目標としているため、マクロ言語については補助的なものにしています。

マクロ言語は、計算表を操作するのに優れており、言語を覚えていなくても､「マクロの記録」という機能でマクロプログラムを自動で作成することができます。

早速、マクロプログラムを自動で作成してみます。以下の手順でマクロを記録して下さい。

1)「開発」タブ→「新しいマクロの記録」→マクロ名”Macro1”→「OK」

2)カーソルを"C4"にポイントし、「マクロのテスト」と入力

3)「ホーム」タブ→、フォントサイズを「20」に、文字を「太字」に、文字の色を「赤」に、セルの色を「黄色」にする。

4)「開発」タブ→「記録終了」

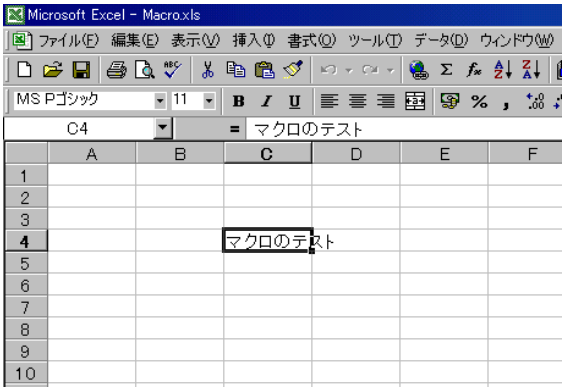
それでは登録したマクロを実行してみます。

1)カーソルを"C4"にポイントし、「マクロのテスト」を削除し、フォントサイズ、文字種、文字の色、背景の色を戻しておく

2)「開発」タブ→「マクロ」→マクロ名”Macro1”→「実行」

このマクロを実行すると自動で"C4"に｢マクロのテスト｣と入力され、フォントサイズ、文字の種類、文字の色、背景の色が自動で変わります。これは、人が操作した、内容を Excel が記録していたからです。

この記録した内容は、どこに、どの様に記録されたのかを調べてみます。VBA の VBE を開き、プロジェクトウインドウの中にある「標準モジュール」の「Module1」をクリックして下さい。そうすると、このブック全体で共用している標準モジュールのコードが表示されます。このコードの中に「Sub Macro1()」と書かれたプロシージャのコードが出てきます。先程、「新しいマクロの記録」～「記録終了」までの間に操作した内容がここに記録されています。

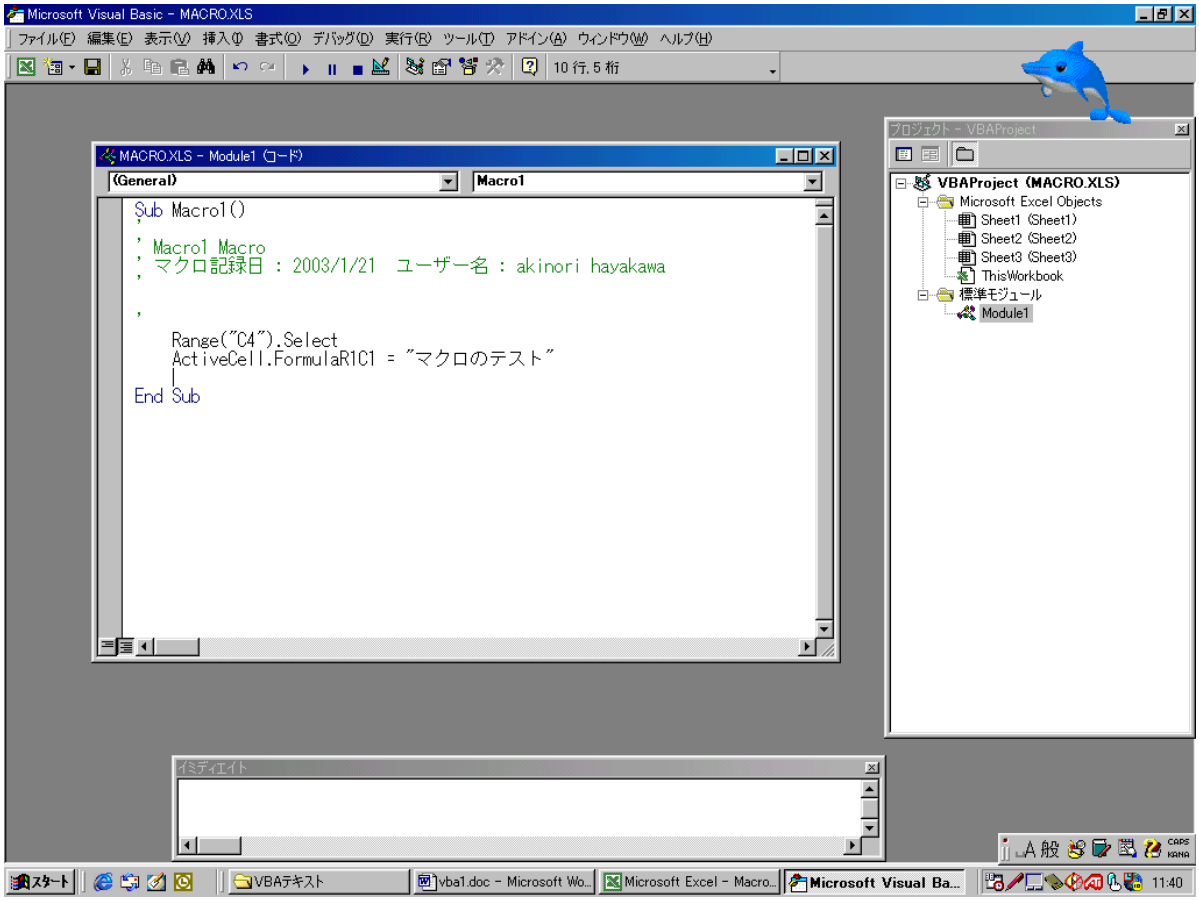


このプロシージャの中には

Range("C4").Select　　→C4 セルをアクティブにする

ActiveCell.FormulaR1C1 = "マクロのテスト"　　→アクティブセルに R1C1 形式で”マクロのテスト”と入力する

と書かれています。これらのコードについては後ほど詳しく説明します。



この先、VBA を使ってセルやワークシート、ブックなどに様々な操作をさせるときに、そのコマンドの一つ一つを覚えていなくても、この「新しいマクロの記録」を使うと必要となるなるコマンドを知ることができます。ただし、この場合、不要なコードも記述されるので、その中から必要としているコードを見つけ出さなければなりません。

今までのVBAによるプログラミングでは､ユーザーフォームにコマンドボタンなどのコントロールを配置し､何らかのイベントが発生したときにその処理が実行されていました｡マクロプログラムでは､｢標準モジュール｣という部分にコードが書かれ､そこの｢Macro｣を実行することにより､自動で記述されたコードどおりに処理していきます｡このMacroを実行する方法として､表にボタンを貼り付けて､このボタンをクリックすることにより実行することもできます｡しかし､このボタンは｢フォーム｣で配置されたボタンであり､そのフォームで実行するマクロをしていることにより､クリックしたときに､指定されたマクロが実行されるに過ぎません｡

このあたりが､VBAとの違いであり､似たような動作であっても､使用する部品､使用する言語が全く異なっています｡

１２．２　セルの操作

マクロの機能を使ってセルに様々な操作をしてみます｡

（１）セルの指定、選択

先程作ったマクロのコードをみて見ます｡

```
Sub Macro1()
’ Macro1 Macro
’ マクロ記録日 : 2003/1/21  ユーザー名 : akinori hayakawa
’
    Range("C4").Select
    ActiveCell.FormulaR1C1 = "マクロのテスト"

End Sub
```

特定のセルを指定する方法として、Range（セル）と Cells（行、列）の２通りがあります。二つの違いとして、Range（セル）はあるセルの範囲を指定するオブジェクトです。( ) の中には｢"C4"｣の様に一つのセル、｢"C4:E6"｣の様にセルの範囲が入ります。一方、Cells（行、列）は一つのセルを指定するオブジェクトです。( ) の中には｢4,3｣のように行番号、列番号を数値が入ります。

セルを選択するにするメソッドに､ActivateとSelectがあります｡Activateは単一のセルを選択するときに使い、Select はセルの範囲を選択するときに使います。実際には混同して使用しても正しく動作することもありますが､思わぬところでエラーとなることもありえますので､使い分けをしたほうがよいでしょう｡以下に代表的な使い方を示します｡

Range("C4").Activate	セル"C4"を選択
Range("C4:E6").Select	セル"C4"～"E6"を選択
Cells (4,3) .Activate	4行3列 (C4) を選択
Range(Cells(5, 2), Cells(8, 3)).Select	5行2列 (B5) ～8行3列 (C8) を選択

（２）セルへのデータ入力

セルにデータを入れるには、入れるセルとデータの型の両方を指定してから、データを入れます。

セルの指定はRengeやCellsを使います。アクティブセルにデータを入れるときはActiveCellを使います。どの方法で指定しても、結果は同じになります。

データの型を指定するプロパティとしてはValue、Formula、FormulaR1C1があります。

Value	数値や文字列を入れる
Formula	数式を入れる（セルを移動しても数式は変化しない）
FormulaR1C1	数値、文字列、数式を入れる（セルを移動すると数式も変化する）

FormulaR1C1 で計算式で使われるセルがアクティブ位置を基準にして相対的に決まるのであればで指定します。アクティブセルを基準にして列の位置が左に３列（－３）、行の位置が下に２列（２）の位置にあるセルは R[-3]C[2]と表します。

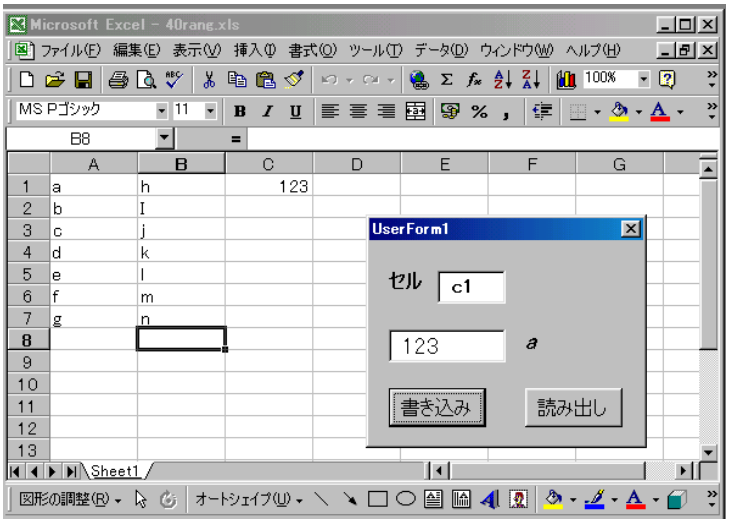
以下にその例を示します。

```
Cells(3,6).Value=123                指定したセルに数値を入れる
Range("F4").Value="ＡＢＣＤ"        指定したセルに文字を入れる
ActiveCell.Value="ＸＹＺ"           アクティブセルに文字を入れる
Cells(5,5).Value=  5 + 7            指定したセルに計算結果を入れる
Cells(5, 5).Value = Cells(5, 3) + Cells(5, 4)  指定したセルに計算結果を入れる
Cells(5,5).Formula="=C5+D5"         指定したセルに数式を入れる
                                      （データを変えると計算結果も変わる）
ActiveCell.FormulaR1C1="=RC[-2]+RC[-1]"
```

実際には、Value、Formula、FormulaR1C1 を混同して使用しても正しい結果が出ますが、思わぬエラーが出ることがあるので、なるべくであれば正確に使い分けたほうが良いでしょう。

```
Cells(5, 5).Value = "=c5+d5"        指定したセルに数式を入れる
Cells(5,5).Formula=  5 + 7          指定したセルに計算結果を入れる
Cells(5, 5).Formula = Cells(5, 3) + Cells(5, 4)  指定したセルに計算結果を入れる
                                      （データを代えても計算結果は変わらない）
```

例題　右のようなユーザーフォームを作成し、「セル」のテキストボックスでセルを指定し、テキストボックスにデータを入力して「書き込み」をクリックすると、指定したセルにデータが書き込まれ、セルを指定して「読み出し」をクリックするとラベルにそのデータが書き込まれるようなプログラムを作り、その動作を確認してください。



[コード]

```
Option Explicit
Private Sub CommandButton1_Click()
    Range(TextBox2.Text).Value = TextBox1.Text
End Sub
```

```
Private Sub CommandButton2_Click()
    Label1.Caption = Range(TextBox2.Text).Value
End Sub
```

例題　ユーザーフォームから、セルの行番号と列番号を指定し、上と同じような動作をするプログラムを作り、その動作確認をしてください。

[コード]

```
Option Explicit

Dim r As Integer
Dim c As Integer

Private Sub CommandButton1_Click()
  r = TextBox2.Text
  c = TextBox3.Text
  Cells(r, c).Value = TextBox1.Text
End Sub
```

```
Private Sub CommandButton2_Click()
  r = TextBox2.Text
  c = TextBox3.Text
  Label1.Caption = Cells(r, c).Value
End Sub
```

（３）セルのプロパティの設定

フォームのオブジェクトと同じように、ワークシートのセルにも、色、大きさ、データなど様々なプロパティがあります。これらのプロパティの設定には

（指定したセル）.(プロパティ)＝＊＊＊

で設定します。また、アクティブセルに対してプロパティを設定するときは

ActiveCell.(プロパティ)＝＊＊＊

で設定します。

例題　マクロを実行すると、"A1"に 123、"B1"に"ＡＢＣ"、"A2"に 456、"A3"のセルの色を赤にして"A1"と"A2"の和、を表示するマクロを作り、その動作確認をしてください。

Excel で自動処理のプログラムを作成する場合、VBA のコードをマニュアルで調べる方法もありますが、「マクロの記録」を使って自動でコードを作成し、そのコードの編集をする方法もあります。この例題を「マクロの記録」を使って記録すると、以下のようなコードが記述されます。

このマクロに With～End With ステートメントが使われています。これは同じオブジェクト（ここでは"A3"セル）に対して何種類かのプロパティを設定するとき、その同じオブジェクト名を省略して記述するときのステートメントです。セルのプロパティを設定する以外にも、ワークシートのプロパティの設定などにも有効です。

セルに入力する文字やセルの色などのプロパティを変える場合、手作業では毎回そのセルを選択してプロパティを変えたりしますが、VBA で自動処理を行う場合、セルの選択をしなくてもプロパティを変えることができます。入力するデータの型も、マクロの場合は全て FormulaR1C1 になりますが、特にその必要がなければ Valeu で処理する事もできます。

```
Sub Macro2()
'
' Macro2 Macro
' マクロ記録日 : 2003/2/12  ユーザー名 : akinori hayakawa
'
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "123"
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "ABC"
    Range("A2").Select
    ActiveCell.FormulaR1C1 = "456"
    Range("A3").Select
    With Selection.Interior
        .ColorIndex = 3
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
    End With
    ActiveCell.FormulaR1C1 = "=R[-2]C+R[-1]C"
    Range("A4").Select

End Sub
```

このコードからその意味を調べ、不要な部分を削除すると以下のように短くすることができます。

```
Sub Macro2()
'
' Macro2 Macro
' マクロ記録日 : 2003/2/12  ユーザー名 : akinori hayakawa
'
    Range("A1").Value = "123"
    Range("B1").Value = "ABC"
    Range("A2").Value = "456"
    Range("A3").Interior.ColorIndex = 3
    Range("A3").FormulaR1C1 = "=R[-2]C+R[-1]C"

End Sub
```

セルの指定方法として、Range を使わず Cells を使って Range("A1")を Cells(1,1)にしても構いません。また、最終行の計算式を Formula 形式で「 Range("A3").Formula="=A1+A2" 」などにしても同じ結果になります。

（４）セルの表示形式

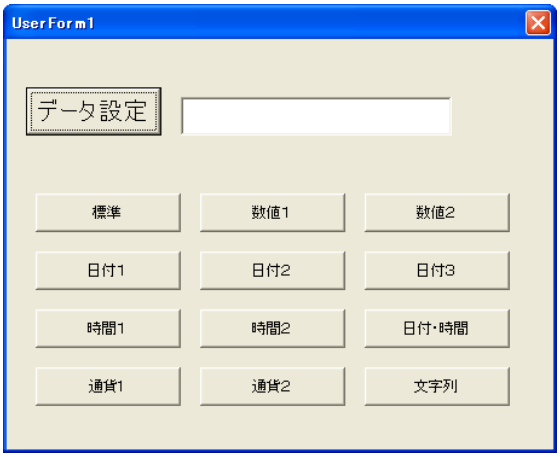
セルに入っている数値データを、表示形式を変えることにより様々な形に表現することが出来ます。その主なものとして、数値形式、通貨形式、日付形式、時間形式などがあり、これはセルのプロパティを使って決めることが出来ます。

```
Cells(1, 1).NumberFormat = "0"                  データを整数で表す
Cells(1, 1).NumberFormat = "0.000"              データを小数点以下３桁で表す
Cells(1, 1).NumberFormat = "yyyy/mm/dd"         年月日を表す
Cells(1, 1).NumberFormat = "hh:mm:ss"           時分秒を表す
Cells(1, 1).NumberFormat = "yyyy/mm/dd  " + "hh:mm:ss"      年月日時分秒を表す
Cells(1, 1).NumberFormat = "#,##0"              通貨を表す
Cells(1, 1).NumberFormat = "¥#,##0;¥-#,##0"    ¥マーク付き通貨を表す
```

```
Cells(1, 1).NumberFormat = "@"                    文字列を表す
```

このほかにも様々な表示方法がありますので、「マクロの記録」や次の例題などを参考にしてください。

例題　ユーザーフォームから、"A1"セルに数値を入力し、そのデータの表示形式を変えるプログラムを作り、その動作確認をして下さい。



[コード]

```
Private Sub CommandButton1_Click()
    Cells(1, 1).Value = TextBox1.Text
End Sub

Private Sub CommandButton2_Click()
    Cells(1, 1).NumberFormatLocal. _
     NumberFormat = "General""
End Sub

Private Sub CommandButton3_Click()
   Cells(1, 1).NumberFormat = "0"
End Sub

Private Sub CommandButton4_Click()
   Cells(1, 1).NumberFormat = "0.000"
End Sub

Private Sub CommandButton5_Click()
    Cells(1, 1).NumberFormat = "yyyy""年""m""月""d""日"""
End Sub

Private Sub CommandButton6_Click()
    Cells(1, 1).NumberFormat = "yyyy/mm/dd"
End Sub

Private Sub CommandButton7_Click()
    Cells(1, 1).NumberFormat = "yyyymmdd"
End Sub

Private Sub CommandButton8_Click()
    Cells(1, 1).NumberFormat = "hh""時""mm""分""ss""秒"""
End Sub

Private Sub CommandButton9_Click()
    Cells(1, 1).NumberFormat = "hh:mm:ss"
End Sub

Private Sub CommandButton10_Click()
    Cells(1, 1).NumberFormat = "yyyy/mm/dd  " + "hh:mm:ss"
End Sub

Private Sub CommandButton11_Click()
    Cells(1, 1).NumberFormat = "#,##0"
End Sub

Private Sub CommandButton12_Click()
    Cells(1, 1).NumberFormat = "¥#,##0;¥-#,##0"
```

```
End Sub

Private Sub CommandButton13_Click()
    Cells(1, 1).NumberFormat = "@"
End Sub
```

CommandButton4～10 をクリックすると、小数の数値データが様々な日付や時間で表示されます。データの整数部が日付を表し、データの「１」が「1900 年 1 月 1 日」となり、データが１増すことにより日付が１日多くなっておきます。パソコンの機種によって違いがあるかも知れませんが数値の上限は 2958465（9999 年 12 月 31 日）で、これを超えるとエラーとなります。

時間はデータの小数部によって表されます。データの「０」が「0 時 0 分 0 秒」を表し、「１」が「24 時 0 分 0 秒（即ち翌日）」となります。１日=86400 秒で換算すると、１秒が 0.0000115 になります。0.0000115 が 0 時 0 分 1 秒になり、逆に、0.9999885 が 23 時 59 分 59 秒になります（実際にはパソコン内の処理の関係で、必ずしもそのようにはなりません）。

日時の表示については、データの型を日付の型（Date）で宣言すると、同様に日付で表示することが出来ます。参考として、以下のプログラムで確認してください。

```
Private Sub UserForm_Click()
    Dim a As Date
    Dim c As Integer

    a = TextBox1.Text
    Cells(2, 1).Value = a
    UserForm1.Caption = a
    c = Cells(1, 1).Value
    Cells(3, 1).Value = c
End Sub
```

変数 a を Date 型で宣言し、その a に TextBox で設定したデータを代入して、Cells (2,2)に表示させます。すると CommandButton10 をクリックしたときのような表示になります。これは、Date 型の変数に数値データを入力すると自動的に日付データに変換され、そのデータをセルに表示すると、そのセルの表示形式が標準のままであれば自動的に表示形式を日付形式に変換して、日付データとして表示します。但し、セルの表示形式が標準でなければ、仮にデータが Date 型であってもそのときの表示形式に従います。

プログラムでは、変則的ではありますが、DATE 型の変数を UserForm1 の Caption に表示させています。TextBox で入力された数値データが日付データに変換され、その年月日日分秒が表示されています。

（６）データのコピー

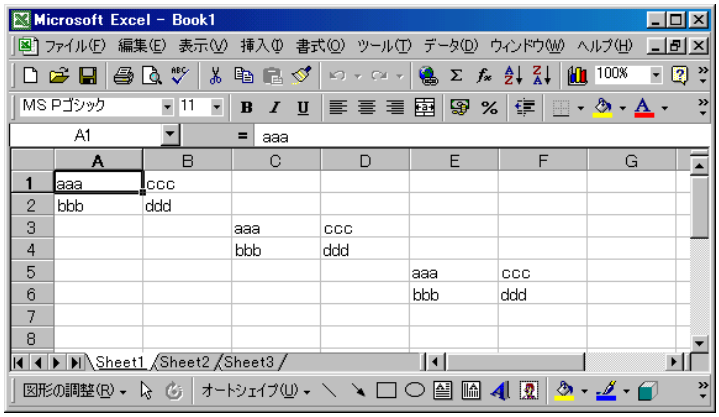
セルのデータをコピーするには、コピーするセルを選択し、アクティブになったセルに対して copy メソッドを実行し、コピー先のセルを選択して、paste メソッドを実行します。

```
Range("B3:D6").Select
Selection.copy
```

```
Range("F8").Select
ActiveSheet.Paste
```

カットやコピーについては、その範囲を選択せず、Range("B3:D6").copy　としても同じ動作をしますが、ペーストについては、Range("F8").Paste　とするとエラーが起こります。また、ペーストしたままだと、カットやコピーの対象となった範囲が点線で選択された状態になっているので、同じ範囲を何度もペーストすることができます。ペースト作業が終了した後、Application.CutCopyMode = False　とすると、カットやコピーの状態が解除されます。

例題　マクロを実行すると、右図のように A1～B2 の範囲を C3～D4、E5～F6 にコピーするようなマクロを作り、その動作確認をして下さい。



[コード]

```
Sub Macro1()
'
' Macro1 Macro
'
    Range("A1:B2").Copy
    Range("C3").Select
    ActiveSheet.Paste
    Range("E5").Select
    ActiveSheet.Paste

    Application.CutCopyMode = False
    Range("A1").Activate

End Sub
```

ここでは、マクロが終わると E5～F6 が選択された状態で終わってしまい見栄えが良くないので、A1 セルを選択してマクロを終わらせている。

（6）行番号、列番号の取り出し

指定されたセルの行番号（横方向）、列番号（縦方向）を取り出すのに、Row 、Column があります。アクティブセルの行、列を取り出すのには Activecell.Row 、 Activecell.Column を使い、その値を取り出します。

```
x=Activecell.Row          アクティブセルの行番号を x に代入
```

（7）行全体、列全体の指定、選択

行全体、列全体を選択する方法として、Row(行番号).Select、Colum(列番号).Select を使います。

１２．３　ワークシートの操作

マクロを使ってワークシートに何らかの操作をすることも出来ます。その代表的なものを以下に示します。

```
Worksheets("シート名").Select          ：ワークシートをアクティブにする
Worksheets.Add                         ：新規ワークシートを加える
Activesheet.Name="シート名"            ：アクティブシートの名前を設定する
Worksheets("シート名").Delete          ：ワークシートを削除する
```

ワークシートをアクティブにするメソッドは、セルをアクティブにするのと同様、Activate メソッドもあります。この他にもいくつかのワークシートに対する操作があります。必要に応じて「マクロの記録」を使って調べて下さい。

（１）シート間のセルのコピー

セルに対して何らかの操作を行う場合、今まではアクティブな一枚のワークシート上での説明でした。ワークシートをまたいで処理を行う場合（例えば、非アクティブな"Sheet2"の"B2"のデータをアクティブな"Sheet1"の"A1"に貼り付ける、など)、どのワークシートに対して操作を行うのか指示しなければなりません。

一つ目の方法として、操作するワークシートをアクティブにしてセルを操作する方法があります。

```
Sheets("Sheet2").Select          "Sheet2"をアクティブにする
Range("B2"). Select              "B2"を選択する
Selection.Copy                   選択されている箇所をコピーする
Sheets("Sheet1").Select          "Sheet1"をアクティブにする
Range("A1").Select               "A1"をアクティブにする
ActiveSheet.Paste                セルを貼り付ける
```

二つ目の方法として、操作するオブジェクトにワークシートを指定する方法があります。Sheet1 がアクティブのときの例を以下に示します。

```
Sheets("Sheet2").Range("B2").copy    "Sheet2"の"B2"をコピーする
Range("A1").Select                   "A1"をアクティブにする
ActiveSheet.Paste                    セルを貼り付ける
```

この方法だと、ワークシートの表示を切り替えることなく、非アクティブな Sheet2 のデータをアクティブな Sheet1 にコピーすることが出来ます。

セルの選択方法として、ワークシートと範囲を同時に選択しようとするとエラーが生じます。

```
Worksheets("Sheet2").Range("B2").Select          エラーが発生
```

複数のシート間でセルのコピーを行う場合、プログラムの長さや表示画面の切り替えによる処理速度の低下のため、二つ目の方法が良く利用されています。

（２）シートの追加と削除

新しいワークシートを挿入するには Add メソッドを使います。挿入する場所を指定するには、before や after を使います。

```
Worksheets.Add After:=Worksheets(Worksheets.Count)
```

更に、挿入したワークシートに名前をつけるには、挿入した直後はそのワークシートがアクティブになっているので、Name プロパティを使ってで名前を設定します。

```
Activesheet.Name="シート名"
```

ワークシートを削除する場合は、シート名を指定して、Delete メソッドでワークシートを削除します。

```
Worksheets("シート名").Delete
```

例題　マクロ１を実行すると、新しいシート（シート名”シートテスト”）が追加され、sheet1 の A1、A2、A3、A4 セルのデータを追加したシートの A1、A2、A3、A4 セルにコピーされる。またマクロ２を実行すると追加したシートが削除される。このようなマクロをつくり、動作確認をして下さい。

［コード］

```
Sub Macro1()
    Worksheets.Add
    ActiveSheet.Name = "シートテスト"
    Worksheets("Sheet1").Range("A1:A4").Copy
    Worksheets("シートテスト").Select
    Range("A1").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
    Range("A5").Activate
End Sub

Sub Macro2()
    On Error Resume Next
    Worksheets("シートテスト").Delete
End Sub
```

マクロ１では、最初に既存のワークシートの一番最後に新たにワークシートを追加し、そのワークシートに「シートテスト」という名前を付けています。シート１をアクティブにし、"A1"～"A4"を選択してコピーし、シートテストの"A1"を選択して、コピーしたセルを貼り付けています。最後にコピー状態を解除し、見栄えを良くするためにセル"A1"を選択しておきます。

マクロ２では、Delete メソッドを使って追加した「シートテスト」を削除しています。また、「シートテスト」が無いときにマクロ２を実行するとエラーが発生するので、「On Error Resume Next」を使ってエラーが発生した行を無視して、次の行から処理を続けます。

１２．４　ブックの操作

マクロを使ってブック（エクセルのファイル）を操作をすることも出来ます。その代表的なものを以下に示します。

（1）ブックを開く

ブックを新たに作成するには Add メソッドを使います。新規ブックが開くと、そのブックがアクティブになります。新規ブックの場合、名前をつけて保存するまでブックに名前がありません。

```
Workbooks.Add                                    新規ブックを開く
```

既存のブックを開くときは Open メソッドを使い、ファイル名を指定してブックを開きます。フォルダを指定せずカレントフォルダから開くか、もしくはフォルダを指定して開きます。

```
Workbooks.Open FileName:=”C:¥･･･.xls”          ファイル名を指定してブックを開く
```

（2）ブックを選択する

ブックを選択するときは Activate メソッドを使い、開いた順番を指定するか、開いたブックのファイル名を指定します。開いたブックについては両方とも使用できますが、新規ブックについては名前がないため“Workbooks(n).Activate”のように「ｎ番目に開いたブック」で指定しなければなりません。

```
Workbooks(n).Activate                            ｎ番目に開いたブックを選択する
Workbooks(“････.xls”).Activate                   指定したブックを選択する
```

（3）ブックの保存

ブックを保存するときは Save メソッドを使います。保存するブックによって、開いたブックを上書き保存する場合、開いたブックがアクティブになっていて保存する場合、新規ブックや開いた既存のブックを名前を変えて保存する場合、などがあります。

```
Workbooks(“････.xls”).Save              開いたブックを上書き保存する
ActiveWorkbook.Save                              アクティブなブックを保存する
ActiveWorkbook.SaveAs FileName:=”C:¥***¥･･･.xls”
                        フォルダ、ファイル名を指定してアクティブなブックを保存する
```

名前を付けて保存するときは、先に保存するブックをアクティブにしてから保存します。フォルダの指定をしなければカレントフォルダに保存されます。

新規に開いたブックには名前がついていないため、ブックをまたいで作業をする場合、開いた順番の値でブックを指定しなければなりません。この方法だと、開く順番が一定でない場合にはブックを指定する事が出来なくなります。そのため、新規にブックを開いたときは、開いた直後に新しいアクティブワークブックに名前を付けて保存し、ブックの名前を設定しておきます。こうすることによって、ブックの指定が確実に出来るようになります。

（4）ブックを閉じる

ブックを閉じる時はCloseメソッドを使います。閉じるブックによって以下に示すような閉じ方があります。

Workbooks.Close	全てのブックを閉じる
ActiveWorkbook.Close	アクティブなブックを閉じる
Workbooks(“・・・・.xls”).Close	ファイル名を指定してブックを閉じる
Workbooks(n).Close	ｎ番目に開いたブックを閉じる

ブックを閉じる場合、開いたブックを保存するかどうか確認するメッセージボックスが開かれます。保存する必要がない場合は、以下のようにすると保存の確認をせずに閉じることができます。

```
Workbooks(“・・・・.xls”).Close SaveChanges:=False
```

例題　マクロ１を実行すると新しいブックが開かれ、元のワークブックの Sheet1 の A1 セルのデータを新しいブックの Sheet1 の A1 へコピーする。
マクロ２を実行すると新しいブックを"Btest.xls"というファイル名で保存する。
マクロ３を実行すると"Btest.xls"を閉じる。
このような動作のするプログラムをつくり、動作確認をして下さい。なお、あらかじめこのファイルを 17wbook.xlsm と保存してからプログラムを組んでください。

```
Sub Macro1()
    Workbooks.Add
    ActiveWorkbook.SaveAs Filename:="Btest.xls"
    Workbooks("17wbook.xlsm").Worksheets("Sheet1").Range("A1").Copy
    Workbooks("Btest.xls").Activate
    Range("A1").Select
    ActiveSheet.Paste
End Sub

Sub Macro2()
    Workbooks("Btest.xls").Save
End Sub

Sub Macro3()
    Workbooks("Btest.xls").Close
End Sub
```

例題　マクロ１を実行すると"Btest.xls"が開かれ、元のワークブックのアクティブセルに"Btest.xls"の Sheet1 の A1 セルのデータをコピーして、"Btest.xls"が閉じる。このような動作をするプログラムをつくり、動作確認をしなさい。

［コード］

```
Sub Macro1()
    Workbooks.Open Filename:="Btest.xls"
    Worksheets("Sheet1").Range("A1").Copy
    Workbooks("wbook2.xls").Activate
    ActiveSheet.Paste
    Workbooks("Btest.xls").Close
End Sub
```

１２．５　マクロの作成

ここまでは、マクロ機能をを使ってセルやワークシート、ブックを操作する方法を説明しました。マクロではこのほかにも、今まで説明した IF ステートメントや、For ステートメント、Do ステートメントなどを使うこともできます。逆に、これらのセルなどに対する操作は VBA のコードでも勿論使用できます。

標準モジュールに記録したマクロ名は、自動記録機能を使うと自動で番号が割り付けられますが、自分で作成する場合はマクロ名を自由に設定することができます（例えば“Sub セルのコピー()”など)。また自動で割り付けられたマクロ名も自由に変更することができます。

標準モジュールに書かれたマクロを実行する方法として、ワークシートにコマンドボタンを貼り付けて、そのコマンドボタンに実行するマクロを登録する、という方法があります。

1)エクセルのメニューから「表示」→「ツールバー」→「フォーム」でフォームツールバーを表示
　これはマクロ用のボタンで、VBA 用のコントロールツールボックスとは異なります
2)「ボタン」を選択し、ワークシート上でドラッグして、ボタンを作成
3)自動で「マクロの登録」ウインドウが開かれるので、登録するマクロを選んで「ＯＫ」
4)ボタンをクリックすると、登録したマクロが実行される

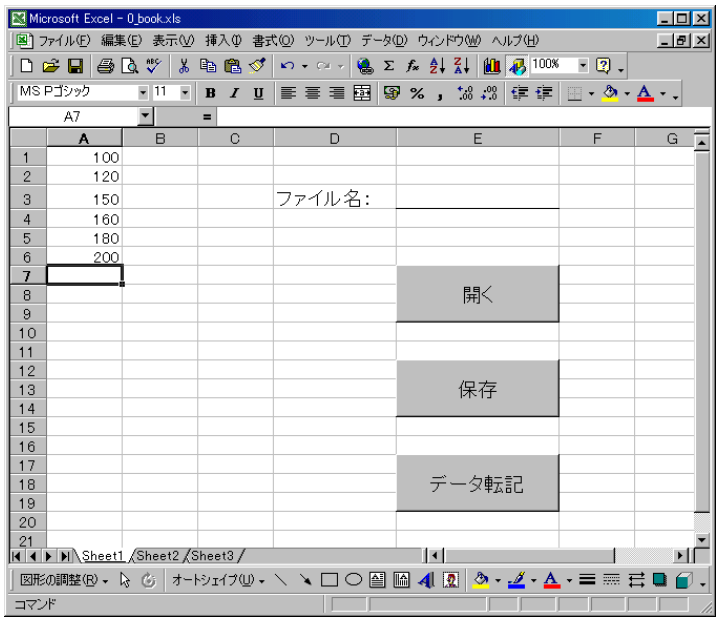
このようにマクロ言語を使うと、VBA を使わなくても簡単な自動処理を行うことができます。

例題　ワークシートにテキストボックス、コマンドボタン１、コマンドボタン２、コマンドボタン３を貼り付け、以下のような動作をするプログラムを作り、動作確認をして下さい。

1)テキストボックスに文字列を入力し、コマンドボタン１をクリックすると、Ｃドライブからその文字列と同じファイル名のエクセルブックを開く。もしそのファイル名のブックが存在しなければ、新規作成する。

2)コマンドボタン２をクリックすると、開いた、もしくは新規作成したブックを、テキストボックスに入力した文字列をファイル名として保存する。

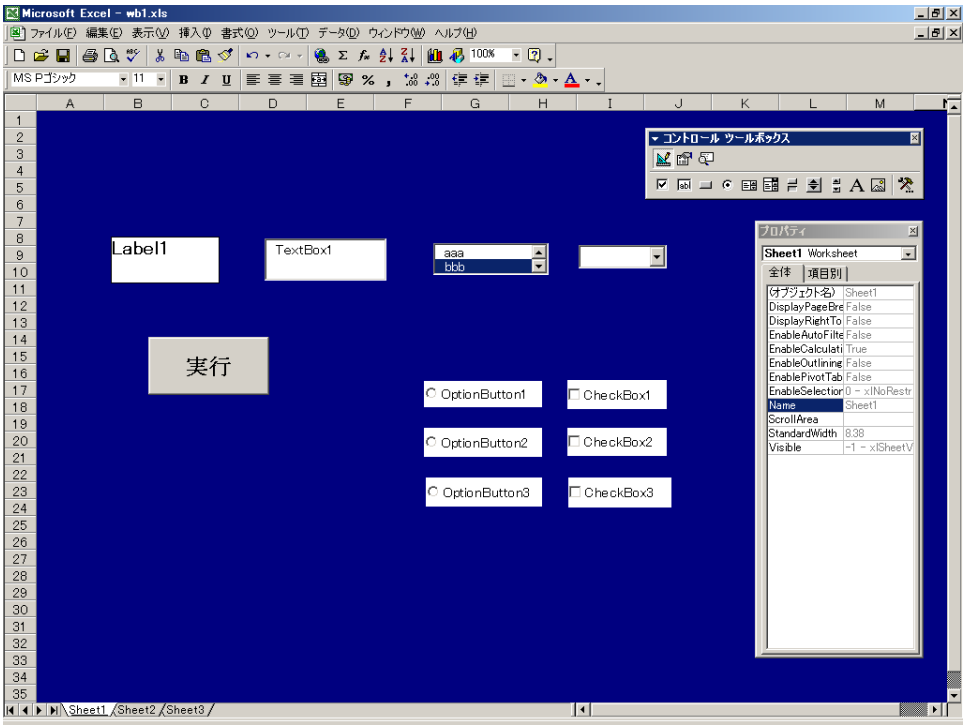
3)コマンドボタン３をクリックすると、コマンドボタンの貼り付けられているシートのセル"A1"から A 列最後までのデータを、新たに開いたブックのワークシート１のセル"A1"から列方向にコピーする



１３．ワークシートを使った自動処理

１２．５では標準モジュールにコードを組み、ワークシートに「フォームのボタン」を配置して、このボタンで自動処理を実行する方法を説明しました。ここではワークシートに「コントロールツールボックスのコマンドボタン」を使った自動処理について説明します。

まず、ワークシートにコントロールを配置する方法について説明します。今まではユーザーフォームにラベル、テキストボックス、コマンドボタンなどのコントロールを貼り付けてきましたが、同じようにワークシートにも同様なコントロールを貼り付けることができます。ワークシートに配置するコントロールは、Excel のメニューから「表示」→「ツールバー」→「コントロールツールボックス」で、コントロールツールボックスウインドウを開き、あとはユーザーフォームにコントロールを配置するのと同じ要領でワークシートに配置していきます。ここが、標準モジュールに組み込んだコードを実行する方法との違いです（標準モジュールのコードの実行には「表示」→「ツールバー」→「フォーム」でした）。なお、ここでは見やすくするために、全てのセルに色を付けています。



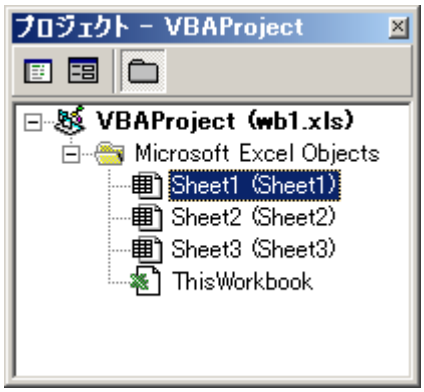
コントロールツールボックスウインドウ内の上方に「デザインモード」、「プロパティ」、「コード」の三つのボタンがあります。

「デザインモード」をＯＮにすると、コントロールを自由に設定することができます。コントロールの設定が終わったならばデザインモードをＯＦＦにします。すると、コントロールのが確定され、実際に様々な動作をさせることができるようになります。

「プロパティ」をクリックすると、ワークシート上のコントロールのプロパティを設定することができます。

「コード」をクリックすると VBA の VBE が立ち上がり、各イベントに対する動作内容をコードで記述していきます。

デザインモードでワークシート上でのコントロールの配置が終わったならば､次に動作内容を決めるコードを記述していきます。コードの記述はユーザーフォームでコードを記述したの同じ要領で、例えば、コマンドボタンをダブルクリックすると、自動的にコマンドボタンが配置されたワークシートのコード記述ウインドウが開かれます。あとは、各イベントに対応する部分（コマンドボタン１を選んだなら Private Sub CommandButton1_Click()）に具体的な処理を記述していきます。

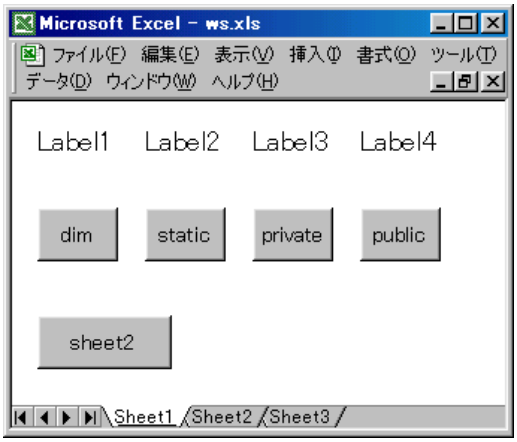


このとき、VBA の VBE のコードウインドウ、もしくはプロジェクトウインドウを見ると、これから書こうとするコードがコントロールを配置したワークシートに書かれるということがわかります。

例題　7.1 の「変数の宣言」での例題を、２枚のワークシート（Sheet1、Sheet2）を使って動作確認をしてください。但し、イベントに関してはユーザーフォームとワークシートで違いがあるので、ワークシートでのイベントを用いて下さい。

Excel のブックはメニューの「ツール」→「オプション」から、不要なものは非表示にしました。ツールバーは次回立ち上げる時は元に戻っていないので、終了する時に元に戻しておかなければなりません。メニューもオリジナルなものに作り変えることも出来るが、内容が高度になるためここでは省略します。

これらの事が出来るようになると、ワークシートだけでアプリケーションを作る事が出来るようになります。



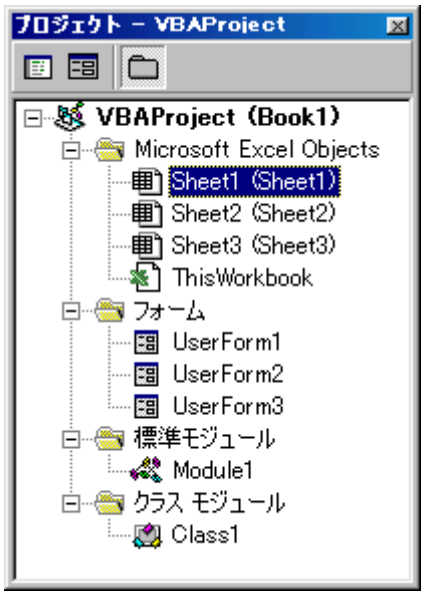
Ｆ．アプリケーションの開発

１４．ウインドウの選定とコードの記述場所

ここまでは、ユーザーフォームを使った自動処理、そしてマクロを使った自動処理、ワークシートを使った自動処理について説明してきました。ここでは実際に Excel を使ったアプリケーションの開発について説明します。

ExcelVBA を使って自動処理をさせるためにユーザーフォームにコードを記述したり、標準モジュールの Module にコードを記述してきました。ここでもう一度、VBA プロジェクトについてまとめみます。

最初にも書きましたが、Excel のプログラムは「VBAProject」として管理されています。このプロジェクトには、大きく見て「Microsoft Excel Objects」、「フォーム」、「標準モジュール」、「クラスモジュール」に分けられます。Microsoft Excel Objects にはエクセルの表のデータやこのブックに関する情報が入っています。フォームにはユーザーフォームの外観やその動作のデータが、標準モジュールにはマクロやプロジェクト全体で共用するデータが入っています。クラスモジュールはもう一段階上のレベルの話で、Excel というアプリケーションそのものに対してのデータが入っています（このテキストでは一切触れません）。これらのものが集まって一つのアプリケーションとして働きます。



Excel を使ったアプリケーションの開発（厳密な言い方をすれば、「プロジェクトの開発」）では、操作者が操作するための操作画面（ウインドウ）と、動作内容を記述したコードを作らなければなりません。

少し話が逸れますが、Excel の移り変わりを見てみると、表計算での関数機能、マクロ機能の追加、ワークシートでのプログラミング機能の追加・・・となっています。自動処理については、現在でも全て共通で使用できる標準モジュールにコードを記述する自動処理が主流です。しかし最近の VBA は、ユーザーフォームで使う自動処理はユーザーフォームにコードを記述し、Sheet1 で使う自動処理は Sheet1 にコードを記述する、とプロジェクト内の役割分担が出来るようになりました。市販の図書はマクロ機能としてのプログラミングについて書かれているものがほとんどで、ワークシートでのプログラミング機能についてかかれているものはほとんどありません。ExcelVBA のプログラマーでも、マクロのプログラミング機能だけでプログラムを組む人もいます。その方法でも十分なプログラムを組む事ができますが、ExcelVBA の開発の流れを考えると、今後はワークシートやユーザーフォームなどの各オブジェクトにコードを記述する方法に移ると筆者は思います。

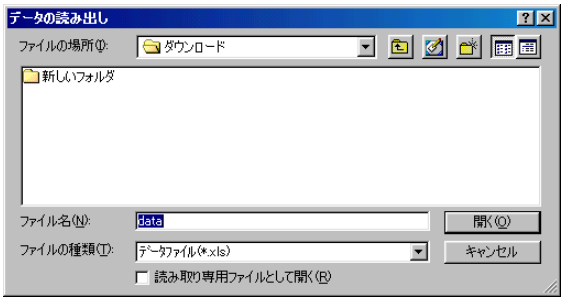
現時点ではワークシートでの自動処理は、まだその機能が開発途上のため、従来の使えた機能が使えなかったり、思わぬところでエラーが出たりするのですが、これからのことを考えると、ユーザーフォームやワークシートなどのオブジェクトを使って、操作画面（ウインドウ）を作成したり、そこにコードを記述するようなアプリケーション（プロジェクト）を開発するのがよいでしょう。

１５．ファイル操作

１５．１　コモンダイアログコントロールによるファイル操作

ここでは Excel などのデータを保存したり、開いたりする方法について説明します。ExcelVBA でデータの読み出しや書き込みを行う場合、コモンダイアログコントロールと Save メソッドや Open メソッドを使うと容易に行うことができます。

コモンダイアログコントロールは Windows で共通で使われるダイアログボックスで、「ファイルを開く」、「ファイル名をつけて保存」、「フォントの指定」、「色の指定」、「印刷」のダイアログボックスがあります。



コモンダイアログコントロールを使うには、ツールボックスの中にあるこのコントロールをユーザーフォームに配置し、プロパティを設定します。ツールボックスにコモンダイアログコントロールがない場合は、メニューの[ツール] → [その他のコントロール] をクリックし、コントロールの一覧の中から「Microsoft Common Dialog Control 」を選択し、[OK] をクリックすると、ツールボックスに新たに CommonDialog が追加されます。

コモンダイアログコントロールの表示は Show メソッドを使います。

ShowOpen	「ファイルを開く」を表示
ShowSave	「ファイル名をつけて保存」を表示
ShowColor	「フォントの指定」を表示

ここでは、「ファイルを開く」、「ファイル名をつけて保存」のダイアログボックスについて説明します。ShowOpen や ShowSave によって表示されたダイアログボックスではファイル名やフォルダを指定するプロパティが良く使われます。ファイルの指定などは、テキストボックスを使って行うこともできますが、ダイアログボックスを使うとフォルダの指定を簡単にすることができます。よく使われるものを以下に示します。

CommonDialog1.DialogTitle	ウインドウのタイトルを設定
CommonDialog1.InitDir	アクセスするフォルダを設定
CommonDialog1.Filter	ファイルの種類（拡張子）を指定
CommonDialog1.FileName	ファイル名、およびフォルダを設定
CommonDialog1.FileTitle	ファイル名のみを設定

InitDir プロパティは、ダイアログボックスを開く前に、ファイルのフォルダを指定するプロパティです。これを使い初期フォルダを指定し、ShowOpen、もしくは ShowSave を実行すると、指定されたフォルダにアクセスします。そのフォルダがないときは、直前でアクセスしたフォルダが開かれます。このデータは「開く」ボタンや「キャンセル」ボタンをクリックした後でも変化しません。次回ダイアログボックスを開いたときに前回と同じフォルダにアクセスさせるのであれば、その都度、データを設定しなおす必要があります。

FileName プロパティは、使用する場所で内容が異なります。ShowOpen を実行する前では、アクセスするファイル名を指定することができます。開くファイル名が既知のときはこれでファイル名を指定します。次に ShowOpen を実行して「開く」ボタンをクリックした後では、ここにはアクセスするフォルダ名とファイル名が一緒になって設定されます（ソフトウエアの開発ミスでしょうか。今後、変更になる可能性があります)。「キャンセル」ボタンがクリックされたときは、データは変化せず、前のデータがそのまま残ります。

FileTitle プロパティは、ダイアログボックスを開き、「開く」ボタンをクリックした後に、その開いたファイル名が設定されます。「キャンセル」ボタンがクリックされたときは、FileName プロパティ同様、前のデータがそのまま残ります。

ダイアログボックスが開かれても、そこにある「開く」ボタンや「保存」ボタン、「キャンセル」ボタンをクリックしても、何の動作もしません。クリックしたときにどのような動作をさせるかは、別にコードを使って指示します。「開く」をクリックしたときは Open メソッドを使って、「保存」をクリックしたときは Save メソッドを使います。

ファイルを開く場合は、ファイル名を指定してワークブックを開きます。ファイル名はダイアログボックスの FileName プロパティを使って指定します。「開く」をクリックした後では、指定したフォルダとファイル名が FileName プロパティに設定されるので、

```
Workbooks.Open Filename:=CommonDialog1.Filename
```

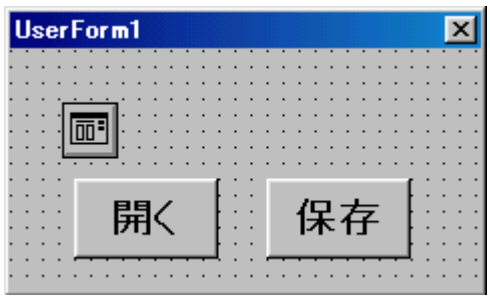
を実行して、指定したフォルダから指定したファイルを開きます。

ファイルを保存する場合は、保存するワークブックを指定し、さらにファイル名を指定して保存します。ワークブックの指定は ActiveWorkbook や Workbooks(″ファイル名″)などを使い、また保存するファイル名の指定はは FileName プロパティを使って指定します。下はアクティブワークブックに名前をつけて保存するときの例です。

```
ActiveWorkbook.SaveAs FileName:=CommonDialog1.Filename
```

キャンセルをする場合、「キャンセル」ボタンをクリックしても、ダイアログコントロールを表示した次の行から実行してしまい、キャンセルされません。「キャンセル」ボタンでキャンセルされるようにするには、On Error GoTo ～　を使います。コモンダイアログコントロールの CancelError プロパティを True にすると、「キャンセル」ボタンをクリックするとエラーが発生します。エラーが発生したら On Error GoTo ～　を使ってエラー処理が実行されるようなプログラムにしておきます。

例題　次のプログラムを作り、その動作確認をして下さい。
（先に、C ドライブに「ダウンロード」というフォルダを作っておく）



[UserForm]
CommandButton1.Caption = "開く"
CommandButton2.Caption = "保存"
CommonDialogControl

[コード]

```
Private Sub CommandButton1_Click()
        On Error GoTo cancel
        CommonDialog1.CancelError = True
        CommonDialog1.DialogTitle = "データの読み出し"
        CommonDialog1.InitDir = "c:¥ダウンロード¥"
        CommonDialog1.FileName = "data.xls"
        CommonDialog1.Filter = "ﾃﾞｰﾀﾌｧｲﾙ(*.xls)|*.xls|全てのﾌｧｲﾙ(*.*)|*.*"
        CommonDialog1.ShowOpen
        ExitSub
cancel:
End Sub

Private Sub CommandButton2_Click()
        On Error GoTo cancel
        CommonDialog1.CancelError = True
        CommonDialog1.DialogTitle = "データの書き込み"
        CommonDialog1.InitDir = "c:¥ダウンロード¥"
        CommonDialog1.FileName = "data.xls"
        CommonDialog1.Filter = "ﾃﾞｰﾀﾌｧｲﾙ(*.xls)|*.xls|全てのﾌｧｲﾙ(*.*)|*.*"
        CommonDialog1.ShowSave
        ActiveWorkbook.SaveAs FileName:=CommonDialog1.Filename
        ExitSub
cancel:
End Sub
```

「開く」をクリックすると「ファイルを開く」ダイアログボックスが開きます。最初のフォルダ指定とファイル名指定は InitDir プロパティと FileName プロパティを使います。FileName プロパティだけでフォルダとファイル名を指定する場合、指定したフォルダがないときはその全てがファイル名となってしまいます。InitDir でフォルダを指定すると、指定したフォルダが無い場合は、現在のフォルダが開かれます。Filter プロパティで拡張子を指定します。「開く」をクリックすると次のコードが実行され、CommonDialog1.FileName で指定されたファイルが開かれ、ExitSub でこのサブプロシージャ-から抜けます。「キャンセル」をクリックするとエラーが発生し、エラー処理コードを実行します。ここでは何もせずに終わるようにしています。
「保存」をクリックしたときは、「開く」をクリックしたときと同じように動作します。

このプログラムだと、「開く」ボタンを押したとき、毎回同じフォルダの同じファイルを開こうとします。実際のアプリケーションでは、前回開いたファイルを開くこともあります。このような動作をさせるためには、開いたフォルダやファイル名を覚えておく必要があります。

```
Option Explicit

Dim fn As String
Dim id As String

Private Sub UserForm_Initialize()
    id = "c:¥ダウンロード¥"
    fn = "data.xls"
End Sub

Private Sub CommandButton1_Click()
    On Error GoTo Cancel
    CommonDialog1.CancelError = True
    CommonDialog1.DialogTitle = "データの読み出し"
    CommonDialog1.InitDir = id
    CommonDialog1.Filename = fn
    CommonDialog1.Filter = "ﾃﾞｰﾀﾌｧｲﾙ(*.xls)|*.xls|全てのﾌｧｲﾙ(*.*)|*.*"
    CommonDialog1.ShowOpen    '「開く」コモンダイアログボックスの表示
    Workbooks.Open Filename:=CommonDialog1.Filename
    id = Left(CommonDialog1.Filename, Len(CommonDialog1.Filename) -
          -Len(CommonDialog1.FileTitle))
    fn = CommonDialog1.FileTitle
Cancel:
End Sub
```

１６．Windows プロシージャの組み込み

Windows や Excel では計算や機械の制御などいろいろなことができますが、なかにはできない処理もあります。どうしてもその処理をしなければならないとき、Windows や Excel の代わりにその処理をしてくれるプログラムを組み込むことによって、それまでできなかったことができるようになります。このようなプログラムを Windows プロシージャといいます。Windows プロシージャには Sub プロシージャや Function プロシージャと同様に、引数、戻り値があります。

Windows プロシージャは dll ファイルというファイルの中に入っています。Windows のシステムにはたくさんの dll ファイルが使われています。これらの dll ファイルにはさらにいくつかのプロシージャが入っています。このプロシージャーを組み合わせて、一つのアプリケーションソフトを作っていきます。

Excel でアプリケーションソフトを作る場合でも、これらの dll ファイルの中にあるプロシージャーを使うことがあります。プロシージャを使うためには、そのプロシージャの入っている dll ファイルと、その中のプロシージャ名を指定して、事前にブックに組み込まなければなければなりません。プロシージャを組み込むには、Declare ステートメントを使って宣言します。

```
[呼び出し範囲] Declare [種類] [プロシージャ名] Lib "[dll ファイル名] _
Alias "[実プロシージャ名]" (ByVal 引数 1 As 型,ByVal 引数 2 As 型) As 型
```

プロシージャが呼び出される範囲の指定

Private	宣言したフォーム内から（フォームの宣言領域で宣言）
Public	全てのフォームから（標準モジュールで宣言）

プロシージャの種類

Function	戻り値を持つプロシージャ
Sub	戻り値を持たないプロシージャ

プロシージャ名

VBA で使うときのプロシージャ名の指定

dll ファイル名

読み出すプロシージャが入っている dll ファイル名

実プロシージャ名

dll ファイルのなかに入っている実際のプロシージャの名前

宣言したプロシージャを実行するには、Sub プロシージャや Function プロシージャと同様、戻り値がない、もしくは必要としないときは「Call プロシージャ名」で呼び出し、戻り値が必要なときは式の中でプロシージャ名をそのまま使って呼び出します。

例題　コマンドボタンをクリックしたら「電卓」が実行されるプログラムを作っ下さい。
Windows のプログラムを実行するプロシージャは、¥windows¥system¥kernel32.dll のなかに Winexec として入っています。

[UserForm]

CommandButton1.caption＝"電卓"

[コード]

```
Option Explicit
Private Declare Function exec Lib "kernel32" Alias "WinExec" _
(ByVal fil As String, ByVal md As Long) As Long

Private Sub CommandButton1_Click()
   Dim retval As Long
   Call exec("c:¥Windows¥calc.exe", 1)
End Sub
```

ここでは、「kernel32」という dll ファイルの中にある「WinExec」というプロシージャを「exec」という名前で組み込んでいます。引数は「fil」という文字変数と、「md」という数字の二つを使っています。このプロシージャは戻り値を持っていますが、この戻り値は使わないので、Call で呼び出しています。

（　終了　）