Excel マクロ　－Visual Basic の文法－

Excel マクロは Visual Basic（以下 VBA という。）というプログラミング言語によって記述される。VBA は Excel とは異なるアプリケーションソフトであり、独自の世界を構成している。作成された Excel マクロを実行することは、Excel の操作を VBA に委ねることを意味し、本来ユーザーがすべき操作を VBA に代行させることを意味する。きめが細かく一般性の高い Excel マクロを作成するためには、自動記録によるマクロの作成だけでなく、VBA の文法を理解した上で、直接プログラム入力することでマクロを作成することが必要となる。以下では VBA の文法を簡単に説明する。

１．データ

VBA で扱うデータとしては、文字データ、数値データ、論理データがあり、それぞれを異なるものとして使用するので、違いを意識しておく必要がある。

（１）文字データ

文字データは、それぞれの文字がコード表に基づいてコード変換され、その変換されたコードがそのまま記憶される。文字データは、１つの文字でできていることもあるが、一般には複数の文字でできていることが多い。このため、文字データは文字列で構成されることになる。このとき、一まとまりになっている文字列を、一つのデータとして扱うことになり、非常に長い文字列を扱うときは、いくつかのデータに分けて扱うこともある。例えば、住所録を扱うときには、一人分の住所、氏名、電話番号を一つのデータとし、人数分のデータを用いたり、場合によってはそれぞれの住所、氏名、電話番号を別々のデータとして、人数分の３倍のデータを用いたりする。

文字データは

“NAGOYA CITY Univ.”

“名古屋市立大学”

などのように、文字列を ” ” で囲んだものであり、空白を含むこともできる。

（２）数値データ

数値データは、一般的な意味での計算（四則演算）に使われるデータであり、一つずつが別々のデータとなる。したがって、各数値はそれぞれに別の変数名が割り当てられ、適当なバイト数を用いて記憶される。

数値データには、いくつかの型があり、それぞれの型に応じて記憶に使用されるメモリ数が決っている。VBA で扱われる数値データには、バイト型、整数型、長整数型、単精度浮動小数点数型、倍精度浮動小数点数型、10 進型、通貨型、日付型、バリアント型があり、それぞれを利用者が使い分けることが必要である。

これらの型を使い分けるために型宣言文字があり、整数型の型宣言文字はパーセント記号（%）、長整数型長整数型はアンパサンド（&）、単精度浮動小数点数型はエクスクラメーションマーク（!）、倍精度浮動小数点数型はシャープ記号（#）であるが、10 進型実数型の宣言はできずバリアント型の内部処理形式としてのみ使われる。通貨型の型宣言文字はアットマーク（@）であり、日付型データはシャープ記号（#）で囲むことで区別する。バリアント型の型宣言文字はなく、型宣言文字を付けなければバリアント型になる。

3%は整数型データであり、3&は長整数型データとなる。その他も同様である。

① 整数型（バイト型、整数型、長整数型）

整数とは、小数点以下の存在しない数値であり、0、5、269、-95 などのことである。数値データがバイト型として記憶されるときは 1B、整数型のときは 2B、長整数型のときは 4B のメモリが使われる。整数型データと超整数型データでは負の値も使えるが、バイト型の場合使えるのは正の値のみである。バイト型データは 0～255、整数型データは-32,768～32,767、超整数型データは-2,147,483,648～2,147,483,647 の範囲の値となる。

② 実数型（単精度浮動小数点数型、倍精度浮動小数点数型、10 進型実数型）

実数型の数値データは小数点以下が存在する数値データであり、1.5、123.45、0.00987、-0.25 などのことである。このような数値を扱うとき、各データはいったん「指数表現形」に変換される。指数表現形とは、0.*****×10** となるものである。すなわち

$1.5 = 0.15\times10^{1}$, $123.45 = 0.12345\times10^{3}$

$0.00987 = 0.987\times10^{-2}$, $-0.25 = -0.25\times10^{0}$

と変換されることになる。このような変換を行う意味は、非常に大きな値から非常に小さな値までを同じように扱うことができるということである。例えば、

$123456789.0123 = 0.1234567890123\times10^{9}$

$0.00000000098765432 = 0.98765432\times10^{-9}$

となる。このように変換されると、実数型数値データは 0.***** で表される数値（「仮数部」と呼ばれる）と 10 のベキ乗を表す整数（「指数部」とよばれる）の２つの数値に分解されることになり、それぞれを記憶することで数値データの記憶が行われる。さらに、$\times10^{9}$は E9、$\times10^{-9}$は E-9 と表記することになっており、$0.1234567890123\times10^{9}$は 0.1234567890123E9、$0.98765432\times10^{-9}$は 0.98765432E-9 と表記される。

仮数部と指数部をあわせて、10 進型数値データは 8B、単精度浮動小数点数型数値データは 4B、倍精度浮動小数点数型数値データは 8B で記憶される。同じ 8B である 10 進型と倍精度浮動小数点数型の違いは、倍精度浮動小数点数型が仮数部と指数部の容量が固定されているのに対して、10 進型は指数部の記憶容量に応じて仮数部の記憶容量が変化するものである。単精度浮動小数点数型数値データは負の値は-3.402823E38～-1.401298E-45、正の値は 1.401298E-45～3.402823E38 の範囲の値、倍精度浮動小数点数型数値データは負の値は-1.79769313486231E308～-4.94065645841247E-324、正の値は 4.94065645841247E-324～1.79769313486232E308 の範囲の値となる。また、10 進型の小数点以下の桁数は、0～28 の範囲の値であり、小数点以下の桁数が 0 の場合（小数部分

を持たない場合)、絶対値の最大値は 79,228,162,514,264,337,593,543,950,335、小数点以下 28 桁の場合、絶対値の最大値は 7.9228162514264337593543950335、絶対値の最小値は（0 を除いた場合）0.0000000000000000000000000001 となる。

実数型数値データのとき、仮数部で記憶容量に格納しきれない値を切り捨てるため、実際の値よりも若干小さな値になる。1/3 を計算したとき、単精度浮動小数点数型は 0.333333343267441000、倍精度浮動小数点数型と 10 進型は 0.3333333333333333000 となる。このように切り捨てられることによる誤差は、「丸め誤差」と呼ばれ、小数点付きデータを扱うときに必ず発生する誤差である。しかし、丸め誤差は非常に小さいので実際には問題となることはほとんどない。

実数型は、非常に大きな値から非常に小さな値まで扱うことができるが、丸め誤差が常につきまとうのに対し、整数型は、扱える値の範囲がせまいが誤差の生じないものである。利用者は、この両型間の違いを理解し、うまく使い分けることが必要である。

③　通貨型

通貨型データは 8B で格納され、その値が 10,000 倍されて整数として記憶される。このことから、15 桁の整数部分と 4 桁の小数部分を持つ固定小数点数データとなる。すなわち、-922,337,203,685,477.5808～922,337,203,685,477.5807 の範囲の値をとる。

④　日付型

日付型データは、8B で格納される浮動小数点数の数値データである。西暦 100 年 1 月 1 日～西暦 9999 年 12 月 31 日の範囲の日付と、0:00:00～23:59:59 の範囲の時刻を表すことができる。日付型データは、#1993 January 1# または #93 Jan 1# のようにシャープ記号 (#) で囲む必要がある。

日付型データは、日付はコントロールパネルで設定されている「短い形式」に従って表示され、時刻はコントロールパネルで設定されている「時刻の形式」（12 時間制“h”か 24 時間制“H”）に従って表示される。

日付型以外の数値型の変数を日付型に変換すると、整数部の値は日付、小数部の値は時刻として表される。午前 0 時は 0、正午は 0.5 である。負の整数は 1899 年 12 月 30 日より前の日付を表す。

⑤　バリアント型

バリアント型は、文字列型とユーザー定義型を除く、あらゆる種類のデータを格納することができる。バリアント型は、-1.797693134862315E308～-4.94066E-324（負の値）、4.94066E-324～1.797693134862315E308（正の値）の範囲のすべての整数または実数の値をとることができる。

通常、数値データはバリアント型として格納される場合でも、その数値本来のデータ型で格納される。たとえば、整数型の値をバリアント型の変数に代入した場合、続く操作でその変数は整数型として処理される。ただし、バイト型 、整数型、長整数型 、単精度浮動小数点数型のいずれかの数値データを格納したバリアント型の変数を演算処理したとき、演算結果が元の数値本来のデータ型の範囲を超えた場合、バリアント型の内部処理形式として、その演算結果を格納できるデータ範囲を持つ、より大きなデータ型が割り当てられる。つまり、バイト型は整数型になり、整数型は長整数型になり、長整数型および単精度浮動小数点数型は倍精度浮動小数点数型になる。ただし、バリアント型に格納

した通貨型、10 進型、および倍精度浮動小数点数型の値が、演算の結果、有効範囲を超えるとエラーが発生する。特定のデータ型の代わりにバリアント型を用いることにより、より柔軟な方法でデータを処理できる。

（３）論理データ

論理データはブール型として 2B で格納され、真（True）と偽（False）のどちらかの値をとるデータである。論理データは VBA の判断文にはしばしば用いられるが、データとして独立に用いられることのめったにないものである。このことから、判断文の説明のところで詳しく説明することにし、ここでは説明を省略する。

なお、ブール型変数に論理データを格納する場合は、True または False を使う。また、ブール型の値を他の型に変換すると、真（True）は-1、偽（False）は 0 になる。

２．変数名

Excel で何らかの処理を行うためには、処理されるべきデータが存在しなければならない。また、処理に使用されるデータは、いったんはメモリのどこかに記憶されなければならない。このとき、記憶したデータが後で捜せないのでは、記憶した意味がないことになる。このため、データを記憶するときは、記憶場所に名前を付けることになる。

Excel の場合、データを保存する場所（セル）には A1 や B5 などのようにあらかじめ名前が付いている。しかし、VBA の場合は、データを保存する場所（「変数」という。）に名前が付いておらず、変数名（変数の名前）は、自分で決めなければならない。

変数名作成のルールは

① 変数名に使える文字は、英数字、漢字、ひらがな、カタカナとアンダスコア（_）である。

② 変数名の先頭の文字は、英字、漢字、ひらがな、カタカナのいずれかでなければならない。

③ 変数名の長さは、半角換算で 255 文字以内でなければならない。

④ 変数名の後ろに型宣言文字を付けることで、どのようなデータを記憶するための変数名であるかを区別する。

型宣言文字はデータの型の場合と同じであるが、文字列型データは「""」で囲むのに対し文字列型変数の場合の型宣言文字「$」を付けることになる。一般に、文字型変数以外には型宣言文字を付けずにバリアント型としている。

x = 5 は次に説明する“代入文”と呼ばれる VBA の命令であり、「変数名 x という名前の付いた変数（データ保存場所）にデータ 5 を保存しなさい。」という意味である。ただし、このとき変数名 x という名前の付いた変数がないときには、名前の付いていない変数に x という名前を付けて保存する。すなわち、データを保存する変数を記述すれば自動的に変数が作成されることになる。一方、変数 x が存在するときには、それまで保存されていたデータを消去して、指示されたデータを保存することになる。

変数名は、標準的には「ローカル変数」であり、その変数の使用される「SUB プロシージャ」内

でのみ共有され、異なる「SUB プロシージャ」で同一の変数名を使用しても、別のものとして扱われ共有はされない。

３．代入文

あるデータをある変数名の付いた記憶場所に記憶させることは、「変数名へのデータの代入」と呼ばれる。この表現は、厳密さを欠くかもしれないが、変数名の利用者にとっては当を得た表現であろう。ある変数へあるデータを代入させる命令は、「代入文」と呼ばれ、x =5 のように記述されるものである。

代入文の意味は、上で説明したものであるが、一般的には

変数名＝算術式

と記したものである。左辺には一つの変数名しか記述することができない。右辺の算術式とは、データないし数式のことであり、数式は Excel の数式と同様に記述したものである。ただし、Excel の数式では参照するセルの番地（A1 や B5 など）を記すことができるが、VBA の場合は変数名を記すことになる。また、関数の名前が Excel と VBA では異なることがあるので注意する必要がある。

A = 1.5

が代入文の例であり、この命令の意味は、『A という変数名の付いたバリアント型の変数に 1.5 という数値を記憶させよ。』というものである。

x = y + 2 * z – 5

の場合は、変数 y と z は、この命令が実施されるときに y と z に保存されているデータに置き換えられて右辺の算術式が計算され、計算結果が変数 x に保存される。y に 10、z に 3 が保存されているときには、11 が x に保存される。

x = x + 1　や　x = x + y

という代入文をしばしば用いるが、その意味は、この命令が実施される直前に x に保存されていたデータに 1 ないし変数 y に保存された値を加え、その結果を x に保存することである。したがって、この命令が実施されると、x に保存されたデータが 1 ないし変数 y の値だけ増えることになる。

文字データを代入するときは、スペース等もデータの一部となり得ることから、

D$=　ABCDEF

としたのでは、どこからどこまでがデータか分からなくなってしまう。そこで、文字列データを表すときは、引用符（"）で前後をくくらなければならない。すなわち

D$=" ABCDEF　"

とする。このとき、この文字列データは 9 文字となる。このように、引用符（"）で前後を囲まれたたものは 1 つの文字列データを表し、代入文の右辺だけでなく、文字データをプログラム中に記述するときの表現法である。

４．算術式

一般に計算式と呼ばれているものは、VBA では「算術式」と呼ばれる。算術式を記述するための記号は、「算術演算子」と呼ばれ、次のものがある。

+	･･･ 加算記号	^	･･･ ベキ乗演算記号
-	･･･ 減算記号	¥	･･･ 整数の除算記号
*	･･･ 乗算記号	Mod	･･･ 剰余演算記号
/	･･･ 実数の除算記号		

これらの算術演算子以外に

(	･･･ 左かっこ	)	･･･右かっこ

が算術式で用いられるが、{、}、[、]、は使用することができない。

算術式とは、数値データをこれらの記号でつないだものであり、VBA に計算をさせるためのものである。例えば

2^5	･･･ 2 の 5 乗	10/3	･･･ 10 割る 3
1.05^20.5	･･･ 1.05 の 20.5 乗	23+8.5	･･･ 23 足す 8.5
5.5*6	･･･ 5.5 かける 6	13-0.8	･･･ 13 引く 0.8

などである。ベキ乗演算 x^a のとき、a が整数のときのみ x が負の値であってもかまわない。

¥と Mod は商と余りを計算するためのものであり、整数同士を演算することが基本であるが、実数（小数点付き数値）をデータとして用いてもよい。ただし、データとして実数を用いると、計算する前に整数へ四捨五入されてから計算される。すなわち、13.6¥4 は四捨五入されて 14¥4 となり、計算結果は 3 となる。また、算術演算子 MOD には必ず前後に一つずつのスペースを付けなければならず、25.68 MOD 6.9 のように記述する。なお、この算術式は四捨五入されて 26 MOD 7 となり、計算結果は 5 となる。

算術式は、上で述べたような一つの演算を行うものばかりでなく、多数の演算子を用いた複雑なものとなる場合もある。例えば

2 + 5 * 4 / 2 ^ 5 * 9 - 1

のようなものである。このとき、演算の順序によって、計算結果が全く異なったものになってしまうことに注意しなければならない。例えば

10 / 2 * 5

という算術式において、10/2 が先に計算され、その結果に 5 がかけられるとすると、結果は 25 となる。しかし、逆に 2*5 が先に計算され、その結果で 10 を割るとすると、結果は 1 となる。このような問題を解決するために、VBA では、演算子に優先順位が付けられている。優先度の高いものから順に並べると、次のようになる。

１．	( )	（内側が優先）
２．	^	（左にあるものが優先）
３．	* /	（左にあるものが優先）
４．	+ -	（左にあるものが優先）

すなわち、算術式にかっこでくくられた部分があるときは、かっこ内の演算が最優先される。また、同じかっこ内ないしかっこがないときには、ベキ乗演算が先に実施され、ついでかけ算・割り算が左にあるものから順に計算され、最後に足し算・引算が左にあるものから順に計算される。

この規則によって、先の例

2 + 5 * 4 / 2 ^ 5 * 9 - 1

を計算すると、かっこがないので、ベキ乗演算 2^5 が最初に計算され、32 となる。すなわち、元の算術式は

2 + 5 * 4 / 32 * 9 - 1

となる。他にベキ乗演算子が存在しないので、次にかけ算・割り算が左から順に行われる。

2 + 20 / 32 * 9 - 1 → 2 + 0.625 * 9 - 1 → 2 + 5.625 - 1

最後に、足し算・引算が左から順に行われ、

7.625 - 1 → 6.625

と計算される。もう一つの例、10/2*5 は、結果が 25 となることは明らかであろう。

これまで、演算に用いるデータとして直接数値のみを使用して説明してきたが、算術式に変数名を用いることもできる。すなわち、

x = a や a * x + 2 * b

というようなものである。このように、算術式内に変数名を用いると、その算術式が計算される段階で、それらの変数名に代入されているデータで置き換えられることになる。すなわち、上の最初の代入文は、変数名 a に現在代入されているデータそのものを変数名 x に代入せよということになる。この命令が実行されると、変数名 a の中身は変わらないが、変数名 x の中身は変数名 a の中身と同じになる。２番目の代入文を実施するとき、変数名 a、x、b にそれぞれ 5、3、6 が代入されていたとすれば、右辺の算術式は 5 * 3 + 2 * 6 という算術式と同じ意味になる。

算術式に変数名が使用できることは、複雑な計算を行うときに非常に便利なことである。例えば、複雑で長い計算式を計算するようなときには、その計算式をいくつかの部分に分解し、部分毎に計算した結果を適当な変数名に代入しておき、最後に、それらの変数名を用いて全体の計算を行う算術式を実施すればよいことになる。

代入文を使用するとき、右辺の算術式中で使用される変数名の中身は、計算時に参照されるのみであって、その代入文が実行されても不変であるが、左辺の変数名の中身は、その代入文の実行前にいかようなものであったとしても、右辺の計算結果によって置き換えられることに注意すべきである。

代入文の中には、初心者にとって奇異に見えるものも存在する。すなわち

k = k + 1

のように、代入文の左辺の変数名が右辺の算術式中にも存在する場合である。まず、「=」という記号であるが、代入文においては「等号」を意味するのではなく、ただ単なる代入文を記述するための記号と考えるべきである。（なお、言語によっては代入文の左辺と右辺をつなぐ記号として「：＝」なるものを使用するものもある。）また、この代入文の実行結果がどのようになるかについても疑問

が持たれるであろう。しかし、VBA は一時に一つのことしかできないということを考慮すれば、結果は自ずから明らかであろう。すなわち、このような命令を VBA が受けたときには、まず右辺の算術式の計算を行うことになる。そして、計算結果が得られた後にデータの代入を行うことになる。したがって、右辺を計算する段階においては、変数名 k の中身は変わっておらず、右辺の計算が終ってからはじめて変数名 k の中身が書き換えられる。この代入文の実行前の変数名 k の中身が 5 であったとすれば、実行後には中身が 6 となる。

数値データ同士の演算はバリアント型を使用する限り型についてさほど意識する必要がないが、数値データ以外のデータを使用する場合は注意する必要がある。数値データ同士の演算ではない場合で、意味のあるもののみについて説明する。

(1) 文字列同士の加算（+）

文字列同士の加算をした場合は、文字列が連結される。

“NAGOYA CITY Univ.” + “　“ + “名古屋市立大学”

としたときの結果は

“NAGOYA CITY Univ.　名古屋市立大学”

となり、文字列の連結が行われる。

(2) 文字列型のバリアント型同士の加算（+）

文字列が連結される。

(3) 文字列型とバリアント型の加算（+）

文字列連結となる。バリアント型のデータが数値であった場合は、数値が文字列に変換されて連結される。

(4) 文字列型のバリアント型と数値データ型のバリアント型の加算（+）

内部処理形式が文字列型のバリアント型のデータが数値を表す文字として解釈できる場合のみ、数値としての加算が実施される。

(5) 日付型と日付型の加算（+）ないし日付型と任意のデータ型の加算（+）

数値としての加算が実施され、日付型として格納される。

(6) 日付型と任意のデータ型の減算（-）

数値としての減算が実施され、日付型として格納される。

５．関数

一般に、関数としては 2 次関数、3 次関数、指数関数、対数関数、三角関数など多数の関数が用いられているが、VBA においてもこれらと全く同様に関数を用いることができる。関数とは、f(x)のように記述されるものであり、x の値が与えられると関数の値が一意に決定されるものである。また、x の値と関数の値とがどのような関係になるかは、関数の形 f によって定まっているものである。

VBA で使用される関数は、大きく 2 種類に分類することができる。その一つは、「組み込み関数」

と呼ばれ、VBA 自身がその計算公式を含んでいるものであり、利用者がその計算公式を知らなくても簡単に使用できるものである。もう一つは、「利用者定義関数」と呼ばれるものであり、利用者自身が VBA（マクロ）の中でその計算公式を定義して使用するものである。

（1）関数の使用法

VBA において関数を使用するためには、「関数名」と「引数」についての概念を理解していなければならない。「関数名」とは、変数名と同様の規則で作られた名前であり、関数それぞれに付けられた固有の名前である。一方、「引数」とは、関数の値を求めるために与えられるデータであり、関数によっては引数のないもの、複数の引数を必要とするものなどもある。

引数の存在しないものを除き、VBA 中で関数を使用するときは、関数名に続けてかっこを記述し、かっこ内に引数を記述しなければならない。引数としては、データそのもの・変数名・算術式等が使用可能である。例として、関数名が Fgh であり、引き数が一つである場合を考えると

```
Fgh(x)
```

となる。また、引数が二つの場合は

```
Fgh(x, y)
```

のようにかっこ内にカンマで区切って必要なだけの引数を記述する。ここで、変数名 x や y が引数を意味する。なお、引数が複数あるとき、その順番は重要な意味を持つので、使用に際しては注意しなければならない。

関数は、引数を基にして何らかの計算ないし処理を行うものであり、結果としての値を持つものであるので、計算ないし処理の行われた後では、その値によって関数の部分が置き直されるとみなされるものである。したがって、代入文の右辺等、データを記述できるところならばどこでも関数を記述することができる。例えば

```
a = Fgh(x)
b = 2 * Fgh(y) + x
If Fgh(x) > 0 Then ･･･
```

などが関数の使用例である。

（2）組み込み関数

組み込み関数は、一般的で汎用性のある関数を、計算公式を知ることなく利用できるようにしたものであり、VBA では 180 種類近くのものが用意されている。ここでは、代表的なもののみ説明する。

以下では、関数名の後ろに(･)を付けることで引数の個数を示す。(･)のついていない関数は引数なしを意味し、(･,･,･)とある場合は引数が 3 つ必要なことを意味する。引数は、データそのもの、変数名、算術式のいずれでもかまわない。

(a) 一般的な数値関数

Abs(･)	引数の絶対値を返す。

Sqr(･)	引数の平方根を返す。

Sgn(･)	引数の符号を整数型で返す。正のとき1、0のとき0、負のとき-1

Log(･)	引数の自然対数を返す。

対数には、10 を底にする常用対数と、ネピア数ｅ＝2.7182813･･･を底にする自然対数があるが、多くのコンピュータ言語においては、一般に自然対数が与えられるようになっている。引き数 x の常用対数が必要なときは、Log(x)/Log(10)とすればよい。

Exp(･)	ネピア数ｅに対する指数関数を返す。

ネピア数（自然対数の底）ｅ＝2.7182813･･･の引数乗を与える。したがって、Exp(1) はｅそのものとなる。

Rnd	0 から 1 の間の乱数を返す。

0 以上 1 未満の一様乱数を与える。1 から 10 の整数をランダムに発生させるためには

Int(10 * Rnd + 1)

とすればよい。

VBA が乱数を発生させるメカニズムは、乱数の「種」と呼ばれる数値を基に乱数を発生させ、次の乱数は前に発生させた乱数からある規則で作られる種を基に発生する。したがって、最初の乱数を発生させる「種」が同じであれば、常に同じ系列の乱数が発生されることになる。VBA を実行させるたびに同一の「種」に戻ることになり、以後同一の乱数系列が発生されることになる。異なる系列の乱数を発生させたいとき、すなわち、「種」の初期値を変更するためには、Randomize 文を用いなければならない。Randomize 文は、「種」の初期値を設定するためのものであり

Randomize

のように記述し、これだけで一つの命令となる。Randomize が実施されると、システムタイマーから取得した値が新しい「種」として使われる。

この関数の使用例は、次の通りである。

```
Randomize
k = 0
For i = 1 To 10000
    x = Rnd
    y = Rnd
    IF x * x + y * y <=1 Then k = k + 1
Next i
p = 4 * k / 10000
```

このプログラムは、円周率の近似値をシミュレーションで求めるためのものであり、その解釈は読者の練習問題とする。

(b) 三角関数

VBA では、三角関数として Sin、Cos、Tan、Atn の４種類が用意されている。これらの関数の引

数は一つの算術式であり、その算術式の計算結果に対する三角関数の値を与えるものである。ただし、引数としての角度は、ラジアン単位でなければならない。度数単位の角度に慣れている読者にとっては、ラジアン単位は不便なものかもしれないが、180 度がラジアン単位では円周率π＝3.1415926535897932 であり、90 度がπ/2、360 度が 2πとなる。なお、VBA の不便な点ではあるが、円周率の値が用意されていないので、円周率の値を定数としてプログラム内で与えなければならない。

Sin(･)	引数（ラジアン単位）の正弦値（サイン）を返す。
Cos(･)	引数（ラジアン単位）の余弦値（コサイン）を返す。
Tan(･)	引数（ラジアン単位）の正接値（タンジェント）を返す。
Atn(･)	引き数の逆正接値（アークタンジェント）を返す。

(c) 数値データの型に関する関数

数値データに三種類の型があることは先に述べた通りであるが、厳密な計算をするためには、型の変換が必要になることがある。ここで説明する関数は、引き数が一つの算術式になるものであり、その算術式の型を変換するためのものである。

Int(･)	引数の値を越えない最大の整数値を返す。

Int(5) → 5　　Int(-3) → -3　　Int(3.7) → 3

Int(-0.1)→ -1　　Int(-2.1) → -3　　Int(-4.8) → -5

Fix(･)	引数の小数点以下を機械的に取り去った整数値を返す。

Fix(5) → 5　　Fix(-3) → -3　　Fix(3.7) → 3

Fix(-0.1)→ 0　　Fix(-2.1) → -2　　Fix(-4.8) → -4

CInt(･)	引数の小数点以下を四捨五入した整数値を返す。

CInt(5) → 5　　CInt(-3) → -3　　CInt(3.7) → 4

CInt(-0.1)→ 0　　CInt(-2.1) → -2　　CInt(-4.8) → -5

(d) 文字列に対する関数

文字列の処理に関しては、これまで詳しく説明してこなかった。しかし、VBA においては、文字列の演算も可能であり、このことに関してここで説明しておく。文字列の演算としては、文字列の連結と文字列の比較があり、多少程度の高いプログラムを作成する場合に、大いに役立つものである。

文字列の連結とは、文字列データ（その前後を引用符 ” で囲んだ文字列）ないし文字列変数名を&記号でつないだものであり、演算結果は、&記号の前後の文字列を連結した文字列となる。例は次のようなものである。A$ = "名古屋"、B$ = "市立"、C$ = ”大学” であるとき、A$ & B$ & C$ の演算結果は、”名古屋市立大学” となる

文字列同士の大小比較を行うこともできるが、かなり高級なプログラムを除いて、文字列が等しい

かどうかの比較以外は行わないので、ここでも等号による比較についてのみ説明する。VBA で比較を利用するのは、If 文ないし While～Wend 文における条件としてであり、そのときの使用例は次の通りである。

```
If a$ = "Y" Then ・・・
If z$ <> "N" Then ・・・
If p$ = "" Then ・・・
```

最後の例において、引用符が二つ続いたものを使用しているが、これは「ヌルストリング」と呼ばれるものであり、文字数が 0 である文字列すなわち文字が何も無い文字列を意味している。

Len(・)	文字列型引数の文字数を返す。

LenB(・)	文字列型引数のバイト数を返す。

Str(・)	数値型引数の表す文字列を返す。

引き数は１つの算術式であり、その値（数値データ）を文字列に変換する関数であり、関数の値は文字列となる。例は

```
a$ = Str(123)  →  a$ = "123"
```

Val(・)	文字列型引数を数値に変換して返す。

Str 関数の逆であり、引数は数値を表す文字列でなければならず、関数の値は数値型となる。引数が数値を表すものでないときには、関数の値は 0 となる。例は

```
x = Val("-12") + 3  →  x = -12 + 3
If Val(D$) > 0 Then ・・・
```

Right(・,・)	文字列型引数の右側から指定の文字数分の文字列を返す。

引数は２つであり、１番目の引数は文字列（文字列データないし文字列型変数名）で、２番目の引数は算術式でなければならない。この関数の値は、１番目の引数が示す文字列の右側から、２番目の引き数が示す個数分だけ取り出した文字列となる。例は

```
x$ = Right("ABCDEFG", 5)  →  x$="CDEFG"
a$ = RIGHT$(m$, Len(m$) - 2)  →  a$は m$から先頭の 2 文字を除いた文字列
```

２番目の引き数の値が 0 のときは、ヌルストリング（長さ 0 の文字列（""））を返すことになり、２番目の引き数の値が１番目の引き数の文字数よりも大きいときは、1 番目の引き数が示す文字列そのものが関数の値となる。

Left(・,・)	文字列型引数の左側から指定の文字数分の文字列を返す。

右と左の違いを別にすれば、Right 関数と全く同じであるので説明を省略する。

Mid(・,・,・)	文字列型引数の指定された位置から指定された文字数分の文字列を取り出し返す。

引き数は３つであり、１番目の引き数は文字列（文字データないし文字型変数名）で、２番目、３番目の引き数は算術式でなければならない。この関数の値は、１番目の引数の文字列の、２番目の引

数が示す値番目の文字から、３番目の引数が示す個数分の文字列となる。例は

x$ = Mid("ABCDEFG", 3, 4)　→　x$ = "CDEF"

１番目の引数の文字列の文字数よりも、２番目の引き数が示す値が大きいときは、関数の値としてヌルストリングを返す。

String(・,・)	指定した文字を指定した数だけ並べた文字列を返す。

この関数の引数は２つであり、１番目が算術式、２番目が文字列を与えるものでなければならない。ただし、２番目の引数が示す文字列の文字数が複数であるときは、最初の文字だけがとられる。

この関数の値は、２番目の引数が示す文字を、１番目の引数が示す値の個数だけ続けたものである。例は

a$ = String(10, "*")　→　a$ = "**********"

x$ = String(5, "ABC")　→　x$="AAAAA"

Space(・)	引数で指定された数のスペースからなる空白文字列を返す。

引き数は一つの算術式であり、その値だけの空白を続けた文字列を与える関数である。したがって、String 関数で２番目の引数を " " としたときと同じになる。例は

p$ = "AB" + Space(5) + "XY"　→　p$ = "AB　　XY"

LCase(・)	文字列型引数の大文字をすべて小文字にして返す。

UCase(・)	文字列型引数の小文字をすべて大文字にして返す。

(e) 日付・時刻に関する関数

Year(・)	日付型データから年を表す値を抜き出し整数型で返す。

Month(・)	日付型データから月を表す値（1～12）を抜き出し整数型で返す。

Day(・)	日付型データから日（1～31）を表す値を抜き出し整数型で返す。

Weekday(・)	日付型データの曜日を整数型で返す。1：日曜、2：月曜、･･･、7：土曜

Hour(・)	日付型データから時刻（0～23）を表す値を抜き出し整数型で返す。

Minute(・)	日付型データから分（0～59）を表す値を抜き出し整数型で返す。

Second(・)	日付型データから秒（0～59）を表す値を抜き出し整数型で返す。

Now	パソコンの現在の日付と時刻を取得する。

Date	パソコンの現在の日付を取得する。

Time	パソコンの現在の時刻を取得する。

Timer	午前 0 時からの経過時間を単精度浮動小数点数型で返す。

DateSerial(･,･,･)	３つの引数で年、月、日を整数型で指定したとき、指定された日付を日付型で返す。

TimerSerial(･,･,･)	３つの引数で時、分、秒を整数型で指定したとき、指定された時刻を日付型で返す。

DateValue(･)	引数を 2009:2:4 のように日付を表す文字列としたとき、指定された時刻を日付型で返す。

TimerValue(･)	引数を 11:22:33 のように時刻を表す文字列としたとき、指定された時刻を日付型で返す。

DateAdd(･,･,･)	指定した日付を指定した間隔だけ進ませた日付を文字列型で返す。

引数は、①間隔の単位を表す文字列、②間隔の個数（負の場合は遅らせる）、③基準となる日付の順である。間隔単位は次の文字列で指定する。

yyyy：年、　q：四半期、　m：月、　ww：週、　d：日、　h：時、　n：分、　s：秒

DateDiff(･,･,･)	２つの日付を指定しその差を指定した間隔単位で計った値で返す。

引数は、①間隔の単位を表す文字列、②第１の日付、③第２の日付の順である。間隔単位は Date Add 関数の場合と同じ。

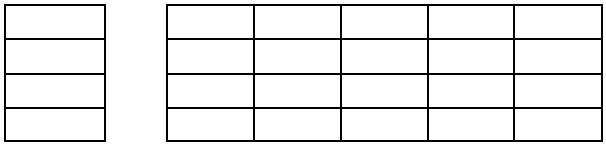
DatePart(･,･)	指定した日付の指定した時間間隔部分を整数型で返す。

引数は、①間隔の単位を表す文字列、②日付の順である。間隔単位は DateAdd 関数の場合と同じ。

６．配列

VBA では、データを保存する場所として変数を用いることを先に説明したが、“配列”と呼ばれるデータ記憶場所もよく使われる。配列とは、指定した大きさの表形式の保存場所である。表形式という意味では Excel のワークシートと同じであるが、Excel ワークシートの大きさは縦 65,536、横 256 に定まっているが、VBA の配列は自分で大きさを指定するものである。また、Excel ワークシートの名前は Sheet1 や Sheet2 のように最初から付いているが、VBA の配列の名前（配列名という）は、自分で決めなければならない。配列名のルールは変数名のルールと同じである。注意すべきことは、配列名と変数名で同一名を重複して使用してはならないということである。

配列として一つにまとめられた複数の記憶場所は、すべて同じ型の記憶場所になる。したがって、型の異なる記憶場所を一つの配列にまとめることはできない。また、配列へのまとめ方として、１次元配列と２次元配列の二種類のものがあり、それぞれは右図のようなものである。各枠が一つずつの記憶場所に相当する。したがって、１次元配列とは、いくつかの記憶場所



１次元配列　　２次元配列

を１列にまとめたものであり、２次元配列とは表のような形にまとめたものである。

配列を使用するときは、１次元配列か２次元配列かを指定し、配列の大きさ、すなわち、何個の記憶場所をまとめるかを指示しなければならない。このための命令は、「配列宣言文」と呼ばれ、配列を使用する前に実行されなければならない命令である。配列宣言文の例は、次のようなものである。

ReDim　A$(10), B(5,10)

である。この命令は、配列名が A$と B の２つの配列を用意せよというものであり、A$と B という名前を配列名として使用することを意味している。また、A$は文字列型の配列であり、文字列データを記憶するためのものであること、B はバリアント型の配列であり、バリアント型データを記憶するためのものであることを示している。

さらに、配列名に付けられたかっこ内の数値は、配列の次元数と大きさを示しており、数値が一つであれば１次元配列、二つであれば２次元配列を意味する。また、それぞれの数値の値は、配列の大きさを示す。A$は大きさ 10 の１次元配列（数学でのベクトルに相当する。）、B は縦 5 横 10 の２次元配列（数学での行列に相当する。）となる。

配列を使用するときには、必ず配列宣言文で定義しておいてから使用しなければならない。配列宣言文は、いくつあってもかまわないし、プログラム中のどこにあってもかまわないが、実際に配列を使用する前に実行し、指示された配列用の記憶場所を VBA に用意させなければならない。

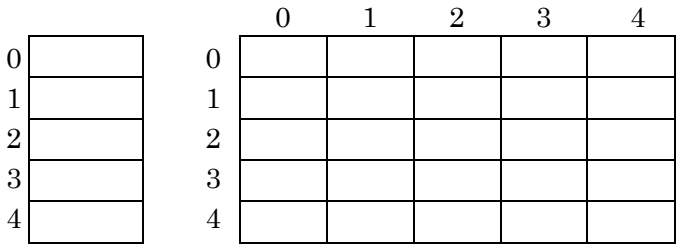
配列の各要素にデータを代入する方法は、変数名の場合と全く同じであるが、配列の多数ある要素中どの要素に代入すべきかを指示しなければならない。この指示のし方は、次のようなものである。

A$(5) = "XYZ"

B(3, 8) = 5.68

すなわち、配列名に続いてかっこを付け、かっこ内に配列中の位置を記入することで、一つの要素を示すことになる。上の例では、配列 A$の 5 番目の要素に XYZ という文字データを代入させること、配列 B の上から 3 番目左から 8 番目の要素に 5.68 という数値を代入させることを意味している。このように、配列中の位置を示すためのかっこで囲まれた数値は「添え字」と呼ばれ、配列を使用するときには必ず付けなければならないものである。

配列中での位置は、右図のように縦横共に 0 番目から始まるようになっている。したがって、先の配列宣言文の例の場合は、配列 A$は 0 番目から 10 番目までの要素が用意されることになり、配列 B は縦方向に 0 番目から 5 番目まで横方向に 0 番目から 10 番目までの要素が用意されることになる。このことから、配列宣言文で使用する配列の次元と大きさを指定するとき、添え字が最大となる要素の名前を記入すればよいことに気付くであろう。



１次元配列　　２次元配列

配列の各要素にデータを代入する方法は、変数名の場合と全く同じと説明したが、代入されているデータを算術式等で使用する方法も変数名の場合と同じであり

```
x = F(6)
P(3) = Q(6, 2) * R(5) + 1
```

などのようになる。このように、添え字付きの配列名は、変数名と全く同様に扱うことができ、変数名の使用可能なところには、添え字付きの配列名も使用可能なことを意味する。

配列を使用することの利点は、配列の要素（位置）を指し示す「添え字」として、算術式が使用可能なことである。ただし、添え字は順番を表すものであり、整数でなければならないが、実数型の変数名を添え字として使用することも可能である。このとき、実数型の添え字は、整数に四捨五入される。ただし、添え字の値が、用意された配列の添え字の範囲を越えないように注意しなければならない。もしも指示された添え字の値が範囲を越えると、VBA はそのことをメッセージで表示し、処理を中断する。添え字として算術式を使用した例は

```
Redim A(n)
A(k) = 1.2
B(n + 1) = C(n) + D(n)
E(i, j) = F(i, j) + G(i, j)
```

などであり、それぞれの命令の意味は明らかであろう。なお、上記例の最初の命令は「大きさ n の１次元配列 A を用意せよ。」というものである。n は変数名であり、この命令の前に n にデータが保存されていなければならない。

配列を使用するときには、データが多数ある場合が一般的であり、また、配列を用いての処理は、同型の命令になることが多い。したがって、配列を用いての処理は、For～Next 文によって記述可能となることが多い。

７．繰り返し命令１（For～Next 文）

繰り返し命令は非常に重要なものであり、繰り返し命令の存在しないプログラムは手作業で行った方が簡単であると言っても過言ではない。繰り返し作業があるからこそ、プログラムを作成し、使用すると言えるであろう。いくつかの繰り返し命令が用意されているが、ここでは最も代表的な For～Next 命令について説明する。For～Next 命令の基本形は

```
For 変数名 ＝ 初期値 To 終端値 Step 増分
        :
Next 変数名
```

なるものである。その意味は、まず、指示された「変数名」に「初期値」を代入し、For 文以下 Next 文までの間の命令を実行する。Next 文に突き当たったら、For 文のところに戻り、「変数名」の値を「増分」だけ増やし、For 文と Next 文間の命令を再び実行し、Next 文に達したら For 文に戻る。以下、「変数名」の値が「終端値」を越えない限り同様な処理を繰り返す。For 文に戻り、「変数名」の値に「増分」を加えたものが「終端値」を越えたときは、この For～Next 文が終了したこ

とになり、Next 文の次の命令に進む。したがって、For～Next 文は、For 文と Next 文の間の命令を何回か繰り返させる命令であり、何回繰り返すかは初期値、終端値、増分によって決まる。

注意事項は、For 文における「変数名」と Next 文における「変数名」が同一でなければならないこと、Step 以下が省略可能であるが、省略されたときは「増分」が 1 とみなされることである。また、「変数名」は、繰り返し回数を規定するものであるので、For 文と Next 文の間にある命令において、その値を変えるようなことがあってはならない。ただし、「変数名」の値を使用することは自由であり、一般的には、計算等に「変数名」の値を使用している。For～Next 文の繰り返しが終了する前に、GoTo 文等で、その範囲から飛び出すことは可能である。しかし、逆に、For～Next 文の範囲外から範囲内に飛び込むことは、「変数名」の値が設定されず、無意味な計算を行う可能性があるので、一般に禁止されている。

For 文における「変数名」としては、数値型変数名が使えるが、配列要素名およびブール型変数名は使用できない。「初期値」、「終端値」、「増分」のどれかが実数の場合は、「丸め誤差」の影響で想定した回数だけ繰り返しが行われないことがある。したがって、「初期値」、「終端値」、「増分」には、なるべく整数を用いるべきであろう。

簡単な例は、次のようなものである。

```
For  k = 1  To  10
   ◇◇◇◇◇
Next  k
```

この例では、Step 以下が省略されているので、増分は 1 であり「変数 k を 1 から 10 まで変化させ、Next 文までの間に記された命令を繰り返しなさい。」というものである。すなわち、最初に変数 k に 1 を格納し、Next 文までの命令を実施し、For 文に戻る。次に、k に 2 を格納し、Next 文までの命令を実施し、For 文に戻る。以下同様にして、最後に k に 10 を格納して Next 文までの命令を実施し、すべての繰り返しが終わったので、Next 文の次の命令に進む。この例の場合は For 文と Next 文の間に記された命令が 10 回繰り返されることになる。

また、次のような記述も許されている。

```
① For  k = 5  To  10
② For  k = 1  To  n
③ For  k = 0  To  10  Step  2
④ For  k = 1.5  To  2  Step  0.1
⑤ For  k = 5  To  1  Step  -1
```

①は 5 から 10 で 6 回繰り返すことになる。②は変数 n に格納されている値分だけ繰り返すことになる。③④⑤には Step が追加されているが、Step は増分を意味し、③は k を 0、2、4、6、8、10 と変えて繰り返すことを意味し、6 回繰り返すことになる。④は k を 1.5、1.6、1.7、1.8、1.9、2 と変えて 6 回繰り返す。⑤は k を 5、4、3、2、1 と変えて 5 回繰り返す。

For～Next 命令の例として、1 から 100 までの総和を求める問題を考えてみよう。総和を格納する変数名を s とし、s に 1 を足し込み、s に 2 を足し込み、･･･、s に 100 を足し込めばよい。それら

の命令は、

s = s + 1、s = s + 2、. . .、s = s + 100

であり、さらに、

k = 1、s = s + k、k = 2、s = s + k、･･･、k = 100、s = s + k

としても同じことであるため、For～Next 命令が使えることになる。プログラムは次のようになる。

```
s = 0
For   k = 1   To   100
   s = s + k
Next   k
```

先頭行（s = 0）は、変数 s に何かデータが格納されていた場合を避けるためのものである。しかし、変数 s が初めて使用されるときは、0 が格納されているので、なくてもかまわない。長いプログラムの場合は、すでに使用しており 0 以外のデータが格納されている場合もあるので、先頭行を付けることを習慣化しておくことが望ましい。この構文は応用の広いものであるので覚えてほしい。

For～Next 命令を二重三重に重ねて使用する場合もある。

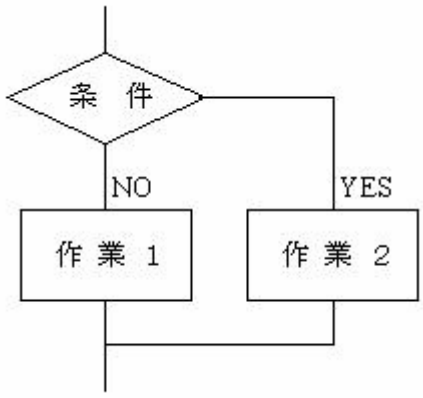
```
For   k1 = 1   To   n
   For   k2 = 1   To   m
      ◇◇◇◇◇
   Next   k2
Next   k1
```

破線で囲った部分を一つの命令とみなせばよいので、それほど難しくないと思われる。この場合、各 k1 に対して k2 が m 回繰り返されるので、全体で◇◇◇◇◇が n×m 回繰り返されることになる。

For～Next 命令で注意すべきことは、For 文に記述された変数（上の例では k1、k2、n、m）を、For 文と Next 文の間では代入文等で変更してはならないことである。

８．場合分け

なんらかの処理をさせるときに、「場合分け」をし、それぞれの場合に応じて別々の処理をさせることが必要なときがある。この「場合分け」なるものを「流れ図」で示すと、右図のようになる。



右図において、「条件」とは成り立つ（正しい）か成り立たない（誤り）かの結果が判定できるものであり、その条件が成り立つときには「作業２」が実施され、成り立たないときには「作業１」が実施されることを意味し、作業１ないし作業２が実施された後は、合流して以下の命令を行うことを示している。作業１と２は、それぞれ別々のものであり、一般に命令群となる。しかし、

場合によっては、いずれかの作業がただ一つの命令であったり、実際には存在しないこともある。

このように、条件に応じて別々のことをさせるための命令が、「IF 文」と呼ばれるものである。図のような作業の流れ方をさせるための IF 文は、次のようになる。

```
If  条件  Then
    作業２
Else
    作業１
EndIf
```

この命令の意味は「もしも条件が成り立つときは作業２を実施し、そうでないときは作業１を実施せよ。」というものであり、先の流れ図に対応したものである。

IF 文においては、次の例のように Else 部分が省略可能である。すなわち、作業１が存在しない場合には、Else 部分を記述しないことが許される。このとき、もしも条件が成り立たないならば、何もすることがなく、すぐに IF 文の次の命令に処理が移ることになる。ただし、Then 部分は省略できず、作業２が存在しないときには、条件を反転させ、作業１と２を入れ換えなければならない。

```
If  a = 0  Then
   ○○○○○
EndIf
```

この例の場合は、変数 a に格納されたデータが 0 のときのみ○○○○○を実施し、そうでないときは何もしないことになる。

条件文は、形式的には「二つの算術式を関係演算子でつないだもの」である。関係演算子は

=	<>（≠のことで、 >< でもよい。）
<	=<（≦のことで 、<= でもよい。）
>	>=（≧のことで 、=> でもよい。）

の６種類ある。x + 1 >= y*z のように両辺共に数式としてもよいし、移項して 1 >= y*z - x のようにしてもよい。すなわち、条件文として、数学で使う等式や不等式を使うことができる。しかし、あまり複雑な条件文は分かりにくくなるので避けた方がよいであろう。上の例でも、事前に r = y*z - x のようにして変数 r に格納しおけば、条件文は 1 >= r ないし r =< 1 でよいことになる。また、与えられた条件を反転させることも容易であり、条件「a > 0」、「x + 1 = y * y」、「p - q <> 5」を反転させると、「a <= 0」、「x + 1 <> y * y」、「p - q = 5」となることは明らかであろう。このように条件を反転させたとき、先の流れ図における作業１と２を入れ換えなければならないことも明らかであろう。

「２つの算術式を関係演算子でつないだもの」は「関係式」と呼ばれ、IF 文の条件の所に記述されるものである。そして、その IF 文が実行されるときにその条件（関係式）が評価され、評価の結果は「真（True）」ないし「偽（False）」のいずれかとなる。すなわち、その条件が成り立つときは「真」と評価され、成り立たないときは「偽」と評価される。

IF 文の条件には、関係式だけでなく、複数の関係式を「論理演算子」でつないだものを記述する

こともできる。論理演算子にはいくつかのものがあるが、ここでは「And」と「Or」についてだけ説明をする。「And」は「かつ」を意味し、その前後に記述された関係式が両方共に成り立つ（真である）ときにだけ、成り立つ（真）と評価されるものである。また、「Or」は「または」を意味し、前後の関係式の両方ないしいずれか一方が成り立つ（真である）ときに、成り立つ（真）と評価されるものである。このように関係式を論理演算子でつないだものは「論理式」と呼ばれ、IF 文の条件の所に記述されるものである。論理式の例として、次のものを考えてみよう。

（ａ）関係式１ And 関係式２ And 関係式３

（ｂ）関係式１ Or 関係式２ Or 関係式３

（ｃ）関係式１ And 関係式２ Or 関係式３

（ｄ）関係式１ Or 関係式２ And 関係式３

（ｅ）関係式１ And (関係式２ Or 関係式３)

（ｆ）関係式１ Or (関係式２ And 関係式３)

このように論理演算子を記述するときには、その前後に１つずつのスペースを記述しなければならない。また、ここでの例のように複数の論理演算子が用いられるときには、かっこがあればかっこ内の評価が優先され、同じかっこ内ないしかっこがないときには、「And」による評価が「Or」よりも優先される。例(ａ)は、３つの関係式がすべて「真」のときにのみ「真」と評価され、それ以外のときは「偽」と評価される。逆に、例(ｂ)は、３つの関係式のすべてが「偽」のときにのみ「偽」と評価され、それ以外のときには「真」と評価されることになる。例(ｃ)と(ｄ)の場合は、２種類の論理演算子が混在しているので、「And」による評価が先に行われ、その真偽が決定されてから「Or」による評価が行われる。すなわち、例(ｃ)の場合には、「関係式１ And 関係式２」が評価され、その結果と「関係式３」の評価結果との「Or」が取られることになる。例(ｄ)の場合も同様に、まず「関係式２ And 関係式３」が評価され、その結果と「関係式１」の評価結果との「Or」が取られる。例(ｅ)と(ｆ)の場合は、かっこが使用されているので、かっこ内の評価が優先される。したがって、例(ｆ)は例(ｄ)と全く同じものになる。

さらに、１つの「ブール型変数」を条件として使用することも可能である。当然のことであるが、その「ブール型変数」には、事前に論理データが格納されていなければならない。

９．繰り返し命令２（Do～Loop 文）

For～Next 文と IF 文を組み合わせたような繰り返し命令として Do～Loop 命令がある。Do～Loop 命令の基本形は次の４つとなる。

タイプ１

```
Do While 条件
   命令群
Loop
```

タイプ２

```
Do Until 条件
   命令群
Loop
```

タイプ３

```
Do
    命令群
Loop While 条件
```

タイプ４

```
Do
    命令群
Loop Until 条件
```

条件は、IF 文の条件と同じであり、「While」は条件が成り立つ限り Do と Loop 間の命令群を繰り返し実施せよというものであり、条件が成り立たなくなったとき終了することになる。タイプ１とタイプ３はよく似ているが、タイプ１の場合命令群が一度も実施されないこともあるが、タイプ３の場合最低１度は命令群が実施されることとなる。「Until」は条件が成り立たない限り Do と Loop 間の命令群を繰り返し実施せよというものであり、条件が成り立ったとき終了する。

命令群の中で IF 文により Do～Loop から強制的に抜け出させるための命令として

Exit Do

がある。この命令が実施されると、Loop 文の次の命令にジャンプすることになる。

Do～Loop 命令では、「While 条件」と「Until 条件」が省略可能である。これらを省略した場合は、命令群の中に「Exit Do」ないし次節で説明する「GoTo」等の強制抜け出し命令がなければならない。

１０．ジャンプ命令

VBA は、基本的に書かれている順番に上から下へ命令を処理するが、いくつかの命令を飛び越したり、戻ったりすることもでき、そのときに使う命令がジャンプ命令である。ジャンプ命令は

GoTo　ラベル

と記述するもので、その意味は「ラベルのところにジャンプせよ。」というものである。しかし、この命令を使うためには、ジャンプ先であるラベルがプログラム中になければならない。

ラベルには、文字（英字、漢字、ひらがな、カタカナ）で始まり、コロン（:）で終わる任意の文字の組み合わせを使うことができるが、同一プログラム内で重複するラベルを使うことはできない。大文字小文字は区別されない。また、ラベルは、行頭から記述しなければならない。

abc:

というラベルがあるとき、そこへジャンプする命令は次のようになる。

GoTo　abc

ジャンプ命令は、場合分け命令（IF 文）と組み合わせて使用されることが多い。

１１．各種サブプログラム

（１）GoSub～Return 文

GoSub～Return 文は、GoTo 文と対比されるものである。すなわち、GoTo 文が「鉄砲玉」のように行きっぱなしのジャンプ命令であるのに対し、GoSub～Return 文は、「ブーメラン」のように戻って来ることを前提にしたジャンプ命令である。

GoSub 文は、GoTo 文と全く同じ記述方法の取られる命令であり

GOSUB ラベル

である。このような命令が実行されると、GoTo 文の場合と同様に、指示されたラベルのところにジャンプし、そこの命令以下を実行することになる。しかし、GoSub 文が GoTo 文と異なる点は、GoSub 文の場合はジャンプするときの場所を記憶しており、Return 文に突き当たったときにジャンプ前の場所に戻るということである。

GoSub 文でジャンプした行より後に、戻るための Return 文がなければならないことは明らかであろう。

（2）Sub プロシージャと Call 文

Call 文は、GoSub 文と非常によく似た命令である。GoSub 文が同一プロシージャ内にあるラベルのところにジャンプさせる命令であるのに対し、Call 文は別の Sub プロシージャにジャンプさせる命令である。

同一プロシージャ内では１つの変数は１つの記憶場所を示すが、プロシージャが異なると、同一名称の変数名であっても別の記憶場所を示すことになる。したがって、プログラムが巨大化するような場合は、プログラムを多数の Sub プロシージャに分割することで、変数名の重複を避けることができる。

しかし、変数が共有されないことから、Call 文でジャンプするときおよび戻ってくるときに受け渡しが必要なデータを指示しなければならない。GoSub 文の場合は、このような配慮が不要となる。

Call 文の基本形は

Call Sub プロシージャ名()

であり、( )内に受け渡しに必要な変数をリストアップしなければならない。

```
Sub x2(a, b)
    b = a * a
Exit Sub
```

のような Sub プロシージャが作成されているとき

Call x2(x, y)

とすると、呼び出し側プロシージャの変数 x と y に格納されているデータを持って Sub プロシージャ x2 へジャンプする。Sub プロシージャ x2 側では、持ってきたデータを変数 a と b に格納する。このとき、データの順番が重要であり、双方での変数名は関係ない。

Sub プロシージャ x2 では、データを受け取った変数 a と b を用いて処理を実施し、終了後（Exit Sub まできたとき）変数 a と b に格納されているデータを持って呼び出された Call 文のところに戻る。ここで、変数 x と y に戻ってきたデータを格納し、Call 文の次の命令に進む。

ここで注意すべきことは、変数 x のデータが不変であり、最初に持って行った変数 y のデータが使用されないことである。これらのデータの受け渡しに使われる変数、すなわち( )に記述される変数リストは引数と呼ばれる。引数には配列を使用することもできるが、この場合は呼び出される Sub プロシージャ側でも配列で受け取らなければならない。すなわち、ReDim で配列宣言しなければ

ならない。配列の場合は大きさも同一でなければならないため、サイズを表すデータも引数に含めなければならない。このためには次のようにすればよい。

```
ReDim X(n, m)                    Sub abc(A, p, q)
    :                                ReDim A(p, q)
  Call abc(X, n, m)                    :
    :                              Exit Sub
```

呼び出し側　　呼び出される側

Excel マクロにおいて、新たな Sub プロシージャを作成するためには、既存 Sub プロシージャの Exit Sub 行の次の空白行で

```
Sub abc()
```

と入力し Enter キーを押せば新たな Sub プロシージャが作成される。その後、引数やプログラムを入力すればよい。

（３）Function プロシージャ

Sub プロシージャとよく似たものとして Function プロシージャがある。Function は利用者定義関数であり、組み込み関数と同様に使うことができる。

Function プロシージャは、Sub プロシージャと同様に Sub プロシージャの外側の空白行で

```
Function Xyz()
```

と入力し Enter キーを押せば新たな Function プロシージャが作成される。その後、引数やプログラムを入力すればよい。Function プロシージャの構成は次のようなものであり、Sub プロシージャと同列に扱われるが、Sub プロシージャとは異なる面を持っている。

```
Function Xyz(a, b)
    :
  Xyz = ・・・
Exit Function
```

Function プロシージャ内では、Function プロシージャ名（上の例では Xyz）が変数名として扱われ、戻るときには、この Function プロシージャ名に値を持たせて戻る。さらに、引数は参照するためのみに使用し、その値を変更しない。

この Function を使用するときは、x = Xyz(p, q) + w のように組み込み関数と同じ使い方をする。

（4）グローバル変数を定義する

基本的に各 Sub プロシージャで使用される変数はその Sub プロシージャ内だけで有効なもの（そのためローカル変数と呼ばれる）であり、Sub プロシージャ相互間でデータを受け渡すためには引数を使わなければならない。しかし、すべての Sub プロシージャで使用可能な変数（グローバル変数と呼ばれる）があれば便利な場合がある。VBA では、グローバル変数は Public 文で定義することになっている。グローバル変数を定義するための Public 文は、マクロのコードの最上部にどの Sub プロシージャにも含まれないように記述しなければならない。記述例は次のようになる。

Public x as Integer, c$ as String ← 複数の変数を定義することができる
Public A(5, 5) ← グローバル配列を定義することもできる
as 以下を省略するとバリアント型になる
複数行の Public 文を記述することもできる

１２．VBA と Excel のやりとり

VBA は、Excel シートからデータを読み込み、Excel シートに処理結果としてのデータを書き出すことを原則としている。このため、Excel シートとの間でのデータの入出力を理解する必要がある。

基本的に、VBA は Excel シートのアクティブセルからデータを読み込み、アクティブセルにデータを書き込むことになる。

① Excel のアクティブセルのデータを VBA の変数 n に読み込む VBA の命令は次式である。

```
n = ActiveCell
```

② VBA の変数 S に保存されているデータを Excel のアクティブセルに出力する VBA の命令は次式である。

```
ActiveCell = s
```

VBA が、Excel シートのアクティブセルを移動させるための命令が用意されている。

① 絶対参照によりアクティブセルを移動させる VBA の命令

この方法は、Excel シートのセル番号を直接使用する方法であり、アクティブセルを C2 に移動させる命令は次のようになる。

```
Range("C2").Select
```

さらに、シートを指定するための命令として次のものがある。

```
Sheets("Sheet2").Select
```

③ 相対参照で Excel のアクティブセルを移動させる VBA の命令

相対参照とは、現在アクティブであるセルを基準にして、そこから上下方向および左右方向にいくつ分離れた場所のセルに移動させるかを指示する命令であり、次のようになる。

```
ActiveCell.Offset(2, 1).Range("A1").Select
```

この命令は、「現在アクティブなセルから下に２つ右に１つ行ったところのセルをアクティブにせよ。」という意味である。

```
ActiveCell.Offset(-1, 0).Range("A1").Select
```

は、「現在アクティブなセルのすぐ上のセルをアクティブにせよ。」という意味である。

```
ActiveCell.Offset(-n, 1).Range("A1").Select
```

は、「現在アクティブなセルから n（変数であり、そこに保存されている値になる。）個上の右隣のセルをアクティブにせよ。」という意味である。