2024.10.11

# その他の基本的な知識・テクニック

## 乱数

　サイコロを振ったとき、1～6の何が出るかは分かりません。このような数を乱数と言い、乱数を並べたものを乱数列と言います。サイコロを振って生成される乱数列は、1～6の数が不規則に並び、各数値の発生確率は1/6です。このような数を人間が人為的に作るのは困難です。規則性が生じてしまったり、確率が等しくなくなったりします。や*e*（自然対数の底）も乱数です。

　席替えや班分けなどのときに乱数が必要です。

　Excel VBAで乱数を使う方法以下の3通りがあります。

(1) ワークシート関数RandBetweenを使う

(2) ワークシート関数RANDを使う

(3) VBA関数Rnd()を使う

　ワークシート関数とはセルの中に「=SUM(A1:A3)」のように書き込める関数を指します。乱数系のワークシート関数は、どこかのセルの内容を更新するたびに、実行されます。

　乱数の質については和田維作氏による

<http://isaku-wada.my.coocan.jp/rand/rand.html>　良い乱数・悪い乱数

が非常に詳しくてわかりやすいです。このサイトによると、(1)(3) は質が良くないようですが、ここでは乱数の質はパスして次に進みます。

　3つの方法の中で一番わかりやすいのがRandBetween(a,b)を使う方法なので、RandBetweenを使う方法から話を始めます。

　シート「乱数1」を使います。

　以下のプログラムを実行すると、変数aに1～6の乱数が入ります。ワークシート関数をExcel VBAから利用するときは、WorksheetFunction.関数名　という形式で用います。ピリオドはオブジェクトの階層構造を表します。日本語の「の」に相当すると考えればよいです。この場合は「ワークシート関数」の「RandBetweenという名前の関数」と解釈してください。

a = WorksheetFunction.RandBetween(1, 6)

　セルA1～A20に1～6の乱数を入れるプログラムは以下のようになります。

For i = 1 To 20

Range("A" & i) = WorksheetFunction.RandBetween(1, 6)

Next i

　ExcelのWorksheetFunction.RandBetweenが作る乱数は良い乱数とは言えないように思われます。偏りがあります。

　C言語, pythonなど通常のプログラミング言語において、乱数を発生させる関数は、「0以上1未満」の値を返します。Excel VBAでは Rnd() が0以上1未満の乱数を返します。Rnd() を使って1～6までの乱数を得るには

Int(Rnd()\*6) + 1

という式を用います。理由は以下の表を見て下さい。

|  |  |
| --- | --- |
| 数式 | 取りうる範囲 |
| Rnd() | 0～0.9999…… |
| Rnd()\*6 | 0～5.9999…… |
| Int(Rnd()\*6) | 0, 1, 2, 3, 4, 5 |
| Int(Rnd()\*6)+1 | 1, 2, 3, 4, 5, 6 |

　このように、1～nの整数値をとる乱数を得るには、「0以上1未満」の乱数に対して、「nをかける → 整数部をとる → 1を足す」という操作をします。

For i = 1 To 20

Randomize

Range("A" & i) = Int(Rnd() \* 6) + 1

Next i

　Rnd() 関数はあらかじめ定められた乱数テーブルに従って乱数を返します。Randomize はシステムタイマーから取得した現在の時刻を元にして、使用する乱数テーブルを指定します。上記のRandmizeを省略してもプログラムとしては問題ないですが、毎回Randomizeすることで、乱数の質を良くすることができます。

【発展】

　和田氏のページによると、ワークシート関数のRAND() は良質な乱数を提供すると書いてあります。しかし、RAND()はWorkSheetFunction.RAND() と書いても使えません。以下のように書く必要があります。理由はよく分かりません。

a = Application.Evaluate("RAND()")

b = Int(a \* 6) + 1

　シート「乱数2」に切り替えて下さい。A列にA～Yまで25名の名前があります。もとの名簿を保存するため、いったんB列にコピーし、B列を操作することにします。掃除当番を決めたいと思います。名前をランダムに並べ直してから、1班3人としてE, F, G列に並べます。

方法１

1. B1～B25の25人から1人を選び、その人をB1に入れる。これでB1は確定。元々B1に入っていた名前は、選ばれた人の場所に入れる。
2. B2～B24の24人から1人を選び、その人をB2に入れる。これでB2は確定。元々B2に入っていた名前は、選ばれた人の場所に入れる。
3. 以下、この繰り返し。

　この考えをプログラムにすると、以下のようになります。

For i = 1 To 24

' i 行目を確定させる

a = WorksheetFunction.RandBetween(i, 25) ' aはi～25 の乱数

If a <> i Then

tmp = Range("B" & i) ' i行目とa行目を

Range("B" & i) = Range("B" & a) ' 入れ替える

Range("B" & a) = tmp

End If

Next i

　3.1節で学習したように、値を入れ替えるときは、一時的に値を保持する変数（ここではtmp）が必要です。

　上記のサンプルではa <> iのときに入れ替えていました。a = iの場合に、Ifの中を実行してもセルに変化はありません。従って、

For i = 1 To 24

a = WorksheetFunction.RandBetween(i, 25) ' i～25 の乱数

tmp = Range("B" & i) ' 入れ替え

Range("B" & i) = Range("B" & a)

Range("B" & a) = tmp

Next i

と書いても動作は同じです。筆者はIfがあった方が分かりやすいと思うのですが、ここは好みの問題かも知れません。

　ランダムに名前を並べ替えるとき、多くの人が最初に思いつくのは以下の方法だと思われます。

方法2

2人を選び、その2人の場所を入れ替える

これを何回か繰り返す

　この考えをプログラムにすると以下のようになります。先述のように、Ifはなくてもかまいません。

For i = 1 To 30

a = WorksheetFunction.RandBetween(1, 25)

b = WorksheetFunction.RandBetween(1, 25)

If a <> b Then

tmp = Range("B" & a)

Range("B" & a) = Range("B" & b)

Range("B" & b) = tmp

End If

Next i

　この方法は良いとは言えません。ここでは30回繰り返していますが、何回繰り返したら十分ランダムになるかの基準が明確ではないからです。

　以上2つの方法を示しました。このようにしてB列にランダムに並べた名前を3人1班としてFGH列に並べるプログラムは既に学習しました。以下のようになります。

row = 1

column = 6

For i = 1 To 25

Cells(row, column) = Range("B" & i)

column = column + 1

If column = 9 Then

column = 6

row = row + 1

End If

Next i

　最後の人は、一人でやるのはかわいそうなので、手作業で8班に入れてあげましょう。

【例題】（100ます計算）

　100ます計算は陰山英男先生によって有名になった学習法です[[1]](#footnote-1)。このシートを作ってみましょう。まず、A2～A10に1～9の数字を入れ、B1～J1に1～9の数字を入れます。

For i = 1 To 9

Cells(1 + i, 1) = i ' A1～A9 に 1～9 を入れる

Cells(1, 1 + i) = i ' A2～I2 に 1～9 を入れる

Next i

　次にA2～A10, B1～J1のセルの数値をランダムに並べ替えます。

For i = 2 To 10

a = WorksheetFunction.RandBetween(i, 10) ' 1列目の

tmp = Cells(i, 1) ' a 行目と

Cells(i, 1) = Cells(a, 1) ' i 行目を入れ替える

Cells(a, 1) = tmp

Next i

For i = 2 To 10

a = WorksheetFunction.RandBetween(i, 10) ' 1行目の

tmp = Cells(1, i) ' a 列目と

Cells(1, i) = Cells(1, a) ' i 列目を入れ替える

Cells(1, a) = tmp

Next i

　これで100マス計算のシートができました。

　次に2重ループを利用して、答えを書き込んでみましょう。

For i = 2 To 10

For j = 2 To 10

Cells(i, j) = Cells(i, 1) + Cells(1, j)

Next j

Next i

【ビンゴゲーム】

　シート「ビンゴゲーム」にビンゴゲーム用のシートを作ります。1列目は1～15, 2列目は16～30, 3列目は31～45, 4列目は46～60, 5列目は61～75が入ります。ただし、真ん中のセルCells(3, 3) には "FREE" という文字を入れます。

　次のようなアルゴリズムで、シートを作ります。

1. column = 1 To 5 としてループを組む。

2. F列を作業領域として使う。F1～F15に列に対応する数字を入れる。

3. F列をランダムに入れ替える。

4. F列の1～5行目をcolumn列目の1～5行目にコピーする。

5 Cells(3, 3) に "FREE" と入れる。

a = 0

For column = 1 To 5

For i = 1 To 15

Range("F" & i) = a + i ' F列に数値を入れる

Next i ' a+1 ～ a+15

a = a + 15

For i = 1 To 15

r = WorksheetFunction.RandBetween(i, 15)

tmp = Range("F" & i) ' i 行と ｒ 行を

Range("F" & i) = Range("F" & r) ' 入れ替える

Range("F" & r) = tmp

Next i

For i = 1 To 5 ' F列目の 1～5行目を

Cells(i, column) = Range("F" & i) ' column列の 1～5行目に

Next i ' コピーする

Next column

Cells(3, 3) = "FREE" ' 中央の場所は FREE

## ボタン

　これまでの方法では、プログラムを実行するには「開発」タブを表示させてVisual Basic Editorを起動し、カーソルを実行したいプログラムの上に置き、「実行」→「マクロの実行」としていました。

　Excel VBAの使い方を知らない人にもプログラムを使ってもらえるようにしましょう。

　「開発」→「コントロール：挿入」→「フォームコントロール：ボタン」を選択すると、ボタンを作成するモードになります。ドラッグするとボタンを作成します。altを押しながらドラッグすると、ボタンの境界をセルの境目に合わせます。「マクロの登録」ウィンドウが開きます。「マクロ」とは「SubからEnd Subまでの1つのプログラム」を意味します。そのボタンに結びつけるプログラムを選びます。デフォルトでは「ボタン1\_Click」のような名前が表示されます。ここで「OK」を押してしまうとそのボタンが「ボタン1\_Click」という名前のマクロに結びつけられますが、そのようなマクロは存在しないので、ボタンを押すとエラーが発生します。「OK」を押してしまったときは、ボタンに結びつけるマクロを変更して下さい。

　ボタンに結びつけられたマクロの変更は「ボタンの上で右クリック」→「マクロの登録」でそのボタンに結びつけるマクロ（プログラム）を変更します。

　ボタン上に表示するテキストの設定は「右クリック」→「テキストの編集」です。ESCで編集モードから抜けます。

　ボタンの消去は「右クリック」→「切り取り」です。

【例題】

　カウンタを作ります。A1にはあらかじめ数値が入っていることを仮定します。

ボタンは2つあります。ボタンAを押すと、1増加するようにしなさい。ボタンBを押すと、1減少するようにしなさい。ただし0以下にはならないようにしてください。ボタンCを押すとクリア（0にする）しなさい。

　ボタンA, B, Cの表面にはそれぞれ「カウントアップ」「カウントダウン」「クリア」と表示しなさい。

【解答】

Sub count\_up()

Range("A1") = Range("A1") + 1

End Sub

Sub count\_down()

Range("A1") = Range("A1") - 1

If Range("A1") < 0 Then

Range("A1") = 0

End If

End Sub

Sub clear()

Range("A1") = 0

End Sub

## マクロの記録

　例えば、「行高を指定したい。どのような命令を書けばよいか？」を知りたいとき、

以下の方法があります。たいていは1. 2. でこと足りると思いますが、ネットが使えない場合は3. 4. が必要です。

1. ChatGPTに聞く
2. Webで調べる。「Excel VBA　セル　行高」としてgoogleで検索すると該当ページが見つかるでしょう。
3. 書籍で調べる。「できる大事典　Excel VBA」のような辞書的な本を用いて「セルの高さ」の項を見ると書いてあります。
4. 自分で調べる。今から説明します。

マクロの記録

　以下のように操作して下さい。

1. 「開発」→「コード：マクロの記録」をクリックします。その場所のアイコンが「記録終了」に変わります。
2. どこかの行を選択し、行高を変更します。
3. 「記録終了」を押します。一連の作業が1つのプログラムとして記録されます。

　VBAエディタのオブジェクトエクスプローラの欄（左上の「プロジェクト - VBAProjectと書いてあるウィンドウ）を見て下さい。プロジェクトエクスプローラのウィンドウが見当たらない場合は「表示」→「プロジェクトエクスプローラ」で開いて下さい。新しいモジュールが一つ追加されています。ダブルクリックして中を見て下さい。6行目のセル高さを50.25にしたのなら、以下のようになっていると思います。

Sub Macro1()

'

' Macro1 Macro

'

'

Rows("6:6").RowHeight = 50.25

End Sub

　Macro1()を実行すると、6行目のセル高さを50.25にします。50.25を40に書き換えて実行したり、"6:6" の部分を "2:4" に変更して実行したりみて、動作を確認して下さい。

　このことから、n行目～m行目の高さをxにするには、以下のように書けば良いことが分かります。

Rows(n & ":" & m).RowHeight = x

（注意）

　Rows(n の直後に半角空白が必要。これを怠るとエラーになる。

　文字列を連結するときに & を使うと、数値は自動的に文字列に変換されます。連結に + を使うときは、以下のようにCStrを使って数値を文字列に明示的に変換する必要があります。

Rows(CStr(n) + ":" + CStr(m)).RowHeight = x

【例題】

　シート「名前記入」は名前記入用の用紙の作成途上です。図4.1のような体裁にしなさい。ふりがなの行の高さは20、名前の行は40です。



図4.1　名前書き込み用シート（上から2段を表示）

【解答1】

　「奇数行と偶数行で高さを変える」という考えをプログラムにすると、以下のようになります。i Mod 2はiを2で割ったときの余りを返します。余りが1のときは奇数行、0のときは偶数行です。

For i = 1 To 16

If i Mod 2 = 1 Then

Rows(i & ":" & i).RowHeight = 20

Else

Rows(i & ":" & i).RowHeight = 40

End If

Next i

【解答2】

　行高を変数hに入れます。「行高20の次は行高40、その次は行高20・・・」という考えをプログラムで表すと、以下のようになります。

h = 20

For i = 1 To 16

Rows(i & ":" & i).RowHeight = h

If h = 20

h = 40

Else

h = 20

End If

Next i

　h = 40とh = 20の部分は、それぞれh = h + 20, h = h - 20としても良いです。

　セルに罫線を引く方法も調べてみます。「セルB6の周囲に罫線を引く」という動作を「マクロの記録」で記録すると、以下のようになります。

Sub Macro2()

'

' Macro2 Macro

'

'

Range("B6").Select

Selection.Borders(xlDiagonalDown).LineStyle = xlNone

Selection.Borders(xlDiagonalUp).LineStyle = xlNone

With Selection.Borders(xlEdgeLeft)

.LineStyle = xlContinuous

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThin

End With

With Selection.Borders(xlEdgeTop)

.LineStyle = xlContinuous

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThin

End With

With Selection.Borders(xlEdgeBottom)

.LineStyle = xlContinuous

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThin

End With

With Selection.Borders(xlEdgeRight)

.LineStyle = xlContinuous

.ColorIndex = xlAutomatic

.TintAndShade = 0

.Weight = xlThin

End With

Selection.Borders(xlInsideVertical).LineStyle = xlNone

Selection.Borders(xlInsideHorizontal).LineStyle = xlNone

End Sub

　今度はかなり複雑です。想像力を働かせて解読しましょう。

Range("B6").Select

Selection.Borders(xlDiagonalDown).LineStyle = xlNone

の部分はB6を選択しなくてよいなら、以下のように書き換え可能です。この方が分かりやすいでしょう。

Range("B6").Borders(xlDiagonalDown).LineStyle = xlNone

　次に以下の部分を解読しましょう。

Range("B6").Select

（略）

With Selection.Borders(xlEdgeLeft)

.LineStyle = xlContinuous

（略）

End With

　先ほどと同様にB6を選択しなくてよいなら以下のように書き換え可能です。

Range("B6").Borders(xlEdgeLeft).LineStyle = xlContinuous

　上記の命令で、セルの左端の罫線を実線に設定することが推測されます。セルをB6からB8に変えてみてください。セルB8の左側に罫線が引かれたと思います。

　このように「マクロの記録」で得たプログラムのうち、必要な部分を抽出して使います。

　これ以外の部分について説明します。以下は罫線の色を指定する部分です。

Range("B6").Borders(xlEdgeLeft).ColorIndex = xlAutomatic

　色番号（整数）を指定しています。何番がどういう色なのかを調べるのは面倒なので、色の指定は以下の方法がよいでしょう。

Range("B6").Borders(xlEdgeLeft).Color = RGB(0, 0, 0)

　LineStyleについてはxlContinuous(実線) 以外に xlDot(点線), xlDashDot(一点鎖線), xlDouble(二重線)などが指定可能です。

　Weightは罫線の太さを指定する項目でxlThin, xlMedium, xlThickなどが指定可能です。

　それ以外の部分は滅多に使わないと思いますが、何を指定する項目なのかを説明しておきます。

* Borders(xlDiagonalDown).LineStyle …… 左上から右下へ引く罫線の属性
* Borders(xlDiagonalUp).LineStyle …… 右上から左下へ引く罫線の属性
* Borders(xlInsideVertical).LineStyle …… 複数をセルを選択したとき、内部に引く縦罫線の属性
* Borders(xlInsideHorizontal).LineStyle …… 複数のセルを選択したとき、内部に引く横罫線の属性
* Borders(xlDegeLeft).TintAndShade …… -1～+1の範囲で明度を指定。-1のとき黒, +1のとき白, 0のとき指定した色です。

## シートの切り替え・異なるシートのデータの操作

　本節で学習するプログラムは「標準モジュール」の中に書いて下さい。シートの中やThisWorkbookの中に書くと、正しく動作しません。

　これまでは操作したいシートを選んだ状態で、プログラムを実行しました。現在選択しているシートを「アクティブなシート」と呼びます。例えばRange("A1") = "abc"とすると、アクティブなシートのセルA1に文字列 "abc" が入ります。それ以外のシートのセルを操作するには以下の方法があります。

1. Sheets("シート名").Range("A1") のようにシート名も書く
2. シートを切り替える

　1.の方法について説明します。以下のようにシートを指定することで、異なるシートのセルを操作することができます。

　読み出し例：

a = Sheets("シート名").Range("A1")

書き込み例：

Sheets("シート名").Range("A1") = 10

　シートを指定する方法として、名前を指定するのではなく、通し番号を指定する方法があります。シートには通し番号が1から順番に付いています。一番左端のシートが1番です。通し番号3のシートのセルA1に10を入れるには以下のように書きます。

Sheets(3).Range("A1") = 10

　この方法を使うと、変数を使ってシートを指定することができます。シートの個数を変数n\_sheetに入れるには以下のように書きます。

n\_sheet = Sheets.Count

　次に2. の方法について説明します。アクティブなシートを切り替えるには以下のように書きます。

Sheets("シート名").Select

あるいは

Sheets(3).Select ' 3 はシートの通し番号

　アクティブなシートの名前を変数sheet\_nameに入れるには以下のように書きます。

sheet\_name = ActiveSheet.name

【例題】

　シート「Sheet1」のA1～A8をシート「Sheet2」のA1～A8にコピーするプログラムを作成しなさい。

【解答】

For i = 1 to 8

Sheets("Sheet2").Range("A" & i) = \_

Sheets("Sheet1").Range("A" & i)

Next i

　シート「Sheet1」がアクティブなときは、Sheets("Sheet1").Range("A" & i)と書かずに単にRange("A" & i) でも構いません。シート「Sheet2」がアクティブなときはSheets("Sheet2") は不要です。

## Forにおける注意

　以下のようなことをしてはいけません。

For i = 1 To 10

i = 3

Next i

　For文として使うループ変数に値を代入しています。このFor 文においてiの値は1→2→3→4→……→8→9→10となり、10回目のNext iを実行してループを抜けた後のiの値は11になります。上のように、ループの中でi = 3として、ループ変数iの値を強制的に変更してしまうと、永遠に終了条件を満たさなくなってしまい、無限ループになってしまいます。場合によっては、Excelがハングアップします。

　原則として、Forループの中でループ変数を変更してはいけません。

　Forが終わった後、変数iの値がどうなっているかを確認しましょう。次のプログラムを実行して下さい。

For i = 1 To 10

Debug.Print i

Next i

Debug.Print i

　最後のDebug.Printの実行でiの値は11になっています。この例のように、ループ変数iが1ずつ増えていく場合、Forループ終了時点でのiの値は「終了値 + 1」です。ループ終了後、ループ変数の値を使うことはないと思いますが、知っておいて損はないと思います。

　次の例を見て下さい。iの初期値が1、終了値が0です。これを実行するとどうなるでしょうか？

Debug.Print "begin"

For i = 1 To 0

Debug.Print i

Next i

Debug.Print "end"

　この例のように、forの初期値＞終了値のときは、For ～ Nextの間は1回も実行されません。

## データが入っている最後の行を取り出す

　これまでのプログラムではFor i = 2 To 11 のように書きました。11はデータの個数によって変わります。データの個数が変わると、11の部分を書き換える必要があります。プログラム中に11と書いてある部分が何か所もある場合、書き換え忘れミスが起こるかもしれません。この11のような数をマジックナンバーと言い、プログラムの中に書いてはいけないとされています。

　これを回避するには、変数を1つ導入します。例えば、プログラムの最初で

last\_row = 11

とし、プログラム中で11という数が必要なときはlast\_rowを使います。こうすれば、11という数値を変更する場合、書き換える箇所は1か所で済み、誤りが混入しにくくなります。

　さらにlast\_rowを自動的に取得する方法があります。A列でデータが入っている最後の行を取り出すには、以下のように書きます。

last\_row = Range("A" & Rows.Count).End(xlUp).Row

　Rows.Countはシートの行数を表します。ゆえにRange("A" & Rows.Count) はRange("A1048576")と同等です。B列でデータが入っている最終行を取り出すには、"A" の部分を "B" に置き換えて下さい。

　A列で最初に現れる空セルの1つ前の行を取得するには次のように書きます。

last\_row = Range("A1").End(xlDown).Row

　ただしこの方法はA1のみにデータ入っている場合には使えません。last\_row = 1048576となります。Range( ) で指定するセルとその下のセルの最低2個のセルにデータが入っている必要があります。

　本節で学習した

Range("A" & Rows.Count).End(xlUp).Row

Range("A1").End(xlDown).Row

は頻繁に使う命令ですが、筆者は覚えていません。ChatGPTに聞いて教えてもらいます。

## 条件を含むDo Whileループ

　3章でDo Whileループを学習しましたが、無限ループとして使う例を扱いました。Do While ループの一般形は以下の形をとります。

Do While 条件式

条件が成立したときの処理

条件が成立したときの処理

......

Loop

　Do While ～ Loopまでがループであり、先頭に条件式を置きます。条件式が成立する場合は、Loopまでの処理を行い、成立しない場合はLoopの次の行から実行します。

　例を示します。

i = 1

Do While i <= 10

Range("A" & i) = i

i = i + 1

Loop

　ループ先頭に i <= 10 という条件式があります。成立しているなら、Loopまでの区間を実行します。ループを1回実行するたびにiの値は1増えますから、A1～A10のセルに1～10の数値を書き込みます。

※　本書では無限ループを最初に学習する理由

　筆者は1979年頃にBasicからプログラミングに入門しました。当時のBasicにはループとしてFor文しかありませんでした。Do WhileループはIfとGotoで実現していました[[2]](#footnote-2)。

　Do Whileループは「条件が成立している間はループを実行」です。「無限ループとIf, Exit Doの組み合わせ」は「条件が成立したらループから脱出」です。条件の与え方が逆です。筆者はDo Whileループがない時代にプログラミングに入門したため、未だにDo Whileループに違和感があり、「ループが継続する条件を書く」のが苦痛です。「無限ループとIf, Exit Doによる脱出」の方が分かりやすいと判断し、本書では3章でそのパターンを最初に提示しました。これは、一般のプログラミングの教科書とは順序が逆です[[3]](#footnote-3)。

　Do Whileの直後に条件を与えるループの場合、条件を与える場所はループの先頭1箇所です。それに対して筆者が推奨する「無限ループ＋Exit Do」は、(1) 脱出条件を書く場所がループ中のどこにあるのか分からない　(2) ループから脱出する箇所を複数設定できてしまう、という観点からプログラムが分かりにくくなるかもしれません。

　プログラムを分かりやすくするため、関数の出口 (C言語の場合はreturn) は1箇所にするべきという考え方があります。ループからの脱出箇所も1箇所だけにすべきかも知れませんが、Exit Doを使った方法は、脱出箇所を複数設定できてしまいます。

　私の知人のプログラマーは「『条件が成立するまでループ実行』は分かりにくくない。日常生活でもこういうケースはある。たとえば、風呂に水を入れるとき、満杯になるまで水を入れ続ける。これは『満杯になる』という条件が成立するまで水を入れ続けることである。」と言っていました。私の感覚を読者に押しつけてしまったかもしれません。

## 配列

　配列変数というタイプの変数があります。以下のように使います。

Dim a(3)

a(0) = 0

a(1) = 10

a(2) = 20

a(3) = 30

Debug.Print a(0)

Debug.Print a(1)

Debug.Print a(2)

Debug.Print a(3)

　Dim a(3) と宣言すると、a(0), a(1), a(2), a(3) の4個の配列変数が使用可能になります。配列変数はこのようにカッコ付きの変数で、カッコの中に整数を書きます。

　配列変数が便利なところは、カッコの中（添字と呼びます）を変数で表せることです。上記のプログラムを変数を使って書き直すと次のようになります。

Dim a(3)

For i = 0 To 3

a(i) = i \* 10

Next i

For i = 0 To 3

Debug.Print a(i)

Next i

　Excel VBAで a(10) と宣言すると、a(0)～a(10) まで11個の変数が使えますが、これは他の言語とは異なります。C言語では a[10] と宣言すると、a[0]～a[9] まで10個の変数が使えます。Fortranではa(10)と宣言すると a(1)～a(10) まで10個の変数が使えます。10と宣言して11個使えるというExcel VBAは少し風変わりな処理系です。

　a(i) のように添字が1個の配列を1次元配列と呼びます。

Dim a(10, 20)

と宣言すると、添字が2個ある配列変数を扱うことができます。2次元配列と呼びます。使うことはあまりないと思いますが、3次元や4次元の配列も使用可能です。

　C, Javaなどの言語と異なり、Excelではセルという2次元配列が最初から用意されています。従って、配列変数を使う機会は他の言語よりは少ないかもしれません。

　「セル」と「2次元配列」を比べると、「2次元配列」の方がアクセス速度が速い（セルに値を代入すると画面表示が変更されるので画面処理の時間が必要ですが、2次元配列に値を代入するのは瞬時です）という長所があります。プログラムを高速化するため、セルの値を一旦2次元配列にコピーし、その2次元配列に対して処理を行った後、セルに戻すという技法があります。

　後の節で文字列を分解する関数を学習しますが、分解結果は1次元配列で受け取ります。

## プロシージャ

　1つのプログラムはSub program\_name() の行で始まり、End Subの行で終わります。プログラムから別のプログラムを呼ぶことができます。下の例を見て下さい。

Sub sub1()

Debug.Print "1"

Call sub2

Debug.Print "3"

End Sub

Sub sub2()

Debug.Print "2"

End Sub

　sub1を実行します。途中にCall sub2という文があります。Call sub2を実行すると、Sub sub2() の先頭から実行を始めます。そして、sub2の実行を終了したら（End Subまで実行したら）、sub1のCall sub2の次の行に実行場所を戻します。従って、sub1を実行すると、イミディエイトウィンドウは以下のようになります。

1

2

3

　これまで、Sub ではじまり、End Subまでをプログラムあるいはマクロと呼んできました。これ以降は、正式の呼び方であるプロシージャ (procedure) と呼ぶことにします。

　プログラムにおいて同じ処理を何回も重複して書く必要があるとき、重複する部分を別のプロシージャとして独立させることにより、以下のメリットがあります。

(1) プログラムが見やすくなる。

(2) 変更が必要となったとき1箇所を変更するだけで済む。

　Sub1とSub2の間で情報を渡すことができます。まずは以下のプログラムを見て下さい。

Sub sub1()

Dim a

a = 10

Call sub2(a)

Debug.Print "2: a = " & a

End Sub

Sub sub2(a)

Debug.Print "1: a = " & a

End Sub

　sub1からsub2を呼ぶときにsub2(a) と書くことで、sub1の変数aをsub2に引き渡すことができます。sub2では変数aを受け取っています。sub2においてプロシージャ名の後に (a) のようにカッコで囲んで書く変数のことを引数（ひきすう）と言います。

　実行すると、イミディエイトウィンドウには

1: a = 10

2: a = 10

と表示されます。

　sub1はExcel VBAのエディタから「実行」→「Sub/ユーザーフォームの実行」で実行できますが、sub2は実行できません。引数を持つプロシージャは、他のプロシージャからcallされて実行される必要があります。

　sub2を以下のように変更してみましょう。

Sub sub2(b)

Debug.Print "1: b = " & b

End Sub

　イミディエイトウィンドウは b = 20 と表示されます。sub2で宣言する引数の名前は、call sub2(a) のときに使う変数名と一致させなくても構いません。

　sub2を次のように変えるとどうなるでしょうか。

Sub sub2(b)

Debug.Print "1: b = " & b

b = 20

End Sub

　sub2で変数bの値を更新すると、sub1における変数aの値も更新されています。このことはsub1のaとsub2のbは同一であることを意味しています。

　2つ以上の変数を渡すこともできます。

Sub sub1()

Dim a, b

a = 10

b = 20

Call sub2(a, b)

Debug.Print "2: a = " & a & " b = " & b

End Sub

Sub sub2(x, y)

Debug.Print "1: x = " & x & " y = " & y

x = 100

y = 200

End Sub

　上記のようにsub1においてCall sub2(a, b)と書き、Sub sub2(x, y)として受け取ると、sub2のxはsub1のaと同一であり、sub2のyはsub1のbと同一です。順番に結合されます。

　引数を使って値のやりとりをすることによって以下のようなことができます。

Sub sub1()

Dim a, b

a = 10

Call sub2(a, b)

Debug.Print b

End Sub

Sub sub2(c, d)

d = c \* 2

End Sub

　sub2において、cは入力用引数で、dは出力用引数です。sub2はcを受け取って2倍してdに代入します。すなわち、sub2は入力引数を2倍して返すプロシージャです。このように何らかの機能を一つのSubの中に実装し、それをCallして使うことにより、プログラムが見やすく、作りやすくなります。

【例題】

　組み合わせの個数を表すは以下の公式で表されます。

　mとnが与えられたとき、組み合わせを計算するプロシージャ kumiawase() を以下のように作成しました。階乗を計算する部分が3カ所あるので、その部分をプロシージャ Sub kaijyou(a, b) とすることにしました。aを入力するとa!を計算しbとして返すプロシージャをkaijyou(a, b) とし、以下のようにプログラムを組みます。kaijyou(a, b) を作成しなさい。

Sub kumiawase()

Dim m, n, i, bunsi, bunbo1, bunbo2, kekka

m = 10

n = 3

i = m - n

Call kaijyou(m, bunsi)

Call kaijyou(n, bunbo1)

Call kaijyou(i, bunbo2)

kekka = bunsi / (bunbo1 \* bunbo2)

Debug.Print kekka

End Sub

【解答】

　を計算するプロシージャは以下のようになりまｓ。

Sub kaijyou(a, b)

Dim i, n

n = 1

For i = 1 to a

n = n \* i

Next i

b = n

End Sub

　For i = 1 から始めていますが、For i = 2 として2から始めても結果は同じです。

　変数nを用いて階乗を計算し、最後にbに代入しています。nを使わずに

b = 1

(略)

b = b \* i

としても結果は同じです。使う変数の個数は1個減りますが、私の感覚としては、引数を何回も書き換えるのは違和感があります。引数bを変更するのは最後の1回だけというのがよいと思います。

## 関数

　sin関数を利用したいときはa = sin(x) のように書きます。sin, cos, expなどの基本的な数学関数はExcel VBAにあらかじめ備わっています。自作の関数を定義することができます。数値nを与えたとき、1～nまでの階乗を返す関数は以下のようになります (Excel VBAにはFactという関数があるので、myfactを作る価値はありませんが・・・)。

Function myfact(n)

Dim i, n

n = 1

For i = 1 To n

n = n \* i

Next i

myfact = n

End Function

　この例のように関数はFunction func\_name(n) で始まり、End Functionで終わります。関数の返り値は、End Functionの手前で「関数名 = 値」として代入します。この場合は関数名がmyfactなので、myfact = nという式で代入しています。ここで、スペルミスをしてmyfuct = nと書くとどうなるでしょうか？　モジュールの先頭でOption Explicitと書いておけば、未定義の変数を使ったというエラーが発生し、誤りを教えてくれます。Option Explicitを怠ると、返り値myfactがデフォルトの値（Empty）のまま関数からリターンすることになります。すなわち、ret = myfact(a) において、aにいかなる値を入れようと、retはEmptyになります。非常にたちの悪いバグを誘発することになるので、Option Explicitを怠ってはいけません。

　関数をプロシージャの中から呼び出す場合は、以下のように使います。

Sub func\_check()

a = 5

b = myfact(a)

Debug.Print b

End Sub

　セルの中に書く数式に自作関数を使うことができます。セルの中に

= myfact(A2)

と書くと、自作関数を使うことができます。

≪注意≫

　引数を持たない関数myfuncを利用する場合、

a = myfunc()

と書くことを推奨します。

a = myfunc

と書いてもエラーにはなりませんが、その手前で

Dim myfunc

と書いてしまうと、関数呼び出しではなく、変数の代入となってしまいます。() を付けておくと、誤ってDim myfuncと宣言してしまっても、「インデックスが有効範囲にありません」というエラーが発生し、誤りを発見することができます。

【例題】

　A列においてデータが入っている最後の行を取り出す方法は以下のように書くことを4.5節で学びました。

last\_row = Range("A" & Rows.Count).End(xlUp).Row

　上記のコードはプログラムの随所で書く必要があります。そして、覚えるのは困難です。関数にして簡単に扱えるようにしましょう。列の名前を引数として渡すと、その列にデータが入っている最後の行を取り出す関数は、以下のようになります。

Function get\_last\_row(column)

get\_last\_row = Range(column & Rows.Count).End(xlUp).Row

End Function

　C列のデータが入っている最後の行をlast\_rowに入れるとき、以下のように書きます。

last\_row = get\_last\_row("C")

## セルのクリア・色つけ

　Cellsは全セルを表します。イミディエイトウィンドウで

Cells.Select

と打ってみて下さい。シート上の全セルを選択します。

　セルのクリアは3通りの方法があります。

Cells.ClearContents セルの内容を消去しますが、高さや幅、罫線や塗りつぶし色はそのままです。

Cells.Clear セルの内容、罫線、塗りつぶしなどを削除しますが、列幅、行高はそのままです。

Cells.Delete 罫線、塗りつぶし、列幅、行高さなど全てをデフォルト値に戻します（シートを新規作成したのと同等）。

　セルA2の背景色をつける方法は、以下の通りです。

Range("A2").Interior.Color = RGB(255, 255, 200)

あるいは

Cells(2, 1).Interior.Color = RGB(255, 255, 200)

　RangeあるいはCellsのプロパティであるInterior.Colorに色を表す整数を代入します。RGB(r, g, b) は色を表す整数を返す関数です。r, g, bには0～255の整数を代入します。関数RGB(r, g, b) の返り値は以下の値です。

r + ( g \* 256 ) + ( b \* 256 \* 256 )

　「色なし」に設定するときは以下のように書きます（今後はRangeを使う方法を記します）。

Range("A2"). Interior.Pattern = xlNone

　xlNoneはExcel VBAがデフォルトで用意している定数で、Debug.Print xlNoneで値を見ると、-4142という数値が入っています。

　Interior.Patternではなく、以下のようにInterior.Colorを使っても「色なし」に設定できます。

Range("A2"). Interior.Color = xlNone

　Colorは通常はRGB値を表す整数を与えますが、試したところ負の値なら何を与えても「色なし」に設定されるようです。

【例題】

　シート「市松模様」のA1～G8の範囲において、セルの色を市松模様につけなさい。色はRGB(255, 255, 220)とRGB(220, 255, 255)を使い、左上はRGB(255, 255, 220)を使いなさい。

　愚直にプログラムを書くと以下のようになります。ここでcolorという変数名を使っています。本書ではモジュールの先頭にOption Explicitと書き、全ての変数宣言を強要することを推奨しています。本例のようにcolorを変数宣言する場合は問題は生じませんが、Option Explicitを用いず、変数宣言なしでcolorという変数に代入しようとすると、Excelがデフォルトで保持している定数colorを変更することになり、定数は変更できないのでエラーが発生します。

Dim i, j, color, left\_color

left\_color = 0 ' 左端の列の色の初期値

' left\_color は 0 → 1 → 0 → 1 と変化する

For i = 1 To 8

color = left\_color ' 今から描く行の色　1列目はleft\_color

For j = 1 To 7 ' 1 行描く

If color = 0 Then

Cells(i, j).Interior.color = RGB(255, 255, 220)

Else

Cells(i, j).Interior.color = RGB(220, 255, 255)

End If

color = color + 1 ' color の更新 0 → 1 → 0 → 1 ...

If color = 2 Then

color = 0

End If

Next j

left\_color = left\_color + 1 ' 次の行のためにleft\_color の更新

If left\_color = 2 Then

left\_color = 0

End If

Next i

　ここでは 010101... という数列を得る方法として「1を足して2になったら0に戻す」という操作をしましたが、スマートな方法として「2で割った余りをとる」という方法があります。Modを使うと以下のようにスマートになります。

For i = 1 To 8

left\_color = (i - 1) Mod 2

For j = 1 To 7

col = (left\_color + (j - 1)) Mod 2

If col = 0 Then

Cells(i, j).Interior.Color = RGB(255, 255, 220)

Else

Cells(i, j).Interior.Color = RGB(220, 255, 255)

End If

Next j

Next i

　さらにスマートに書くと以下のようになります。

For i = 1 To 8

for j = 1 to 7

If (i + j) Mod 2 = 0 Then

Cells(i, j).Interior.Color = RGB(255, 255, 220)

Else

Cells(i, j).Interior.Color = RGB(220, 255, 220)

End If

Next i

## メッセージボックス

　処理中に何か表示したいときは、Debug.Printでイミディエイトウィンドウに表示するか、どこかのセルに表示するという方法を今まで学びました。

　それ以外の方法として、メッセージボックスを表示する方法があります。例を示します。

MsgBox "これがメッセージです"

　これはOKだけが出るメッセージボックスを表示します。メッセージボックスにはYes/No, OK/Cancelなどいくつかのパターンがあります。

MsgBox "これがメッセージです", vbYesNo

とすると、Yes/Noの2つのボタンを持つメッセージボックスを表示します。押したボタンの値を取得するときは、次のように括弧付きで書きます。

a = MsgBox("これがメッセージです", vbYesNo)

　aにはYesのとき6, Noのとき7が入ります。この例のvbYesNoの部分にはvbOKonly, vbOKCancel, vbYesNoCancelなどが入ります。適切なものを選んで下さい。

　メッセージボックスに表示する文字列をその時々によって変えたいときは、以下のように書きます。

i = 2

moji = "i = " & i & " です。"

a = MsgBox(moji)

1. Wikipediaによると、昭和40年代に岸本裕史氏が担当するクラスの児童の発想により生まれた。 [↑](#footnote-ref-1)
2. 当時のBasicのプログラムは1行ごとに「文番号」と呼ばれるラベルが付いていました。Gotoは文番号を指定してその行へジャンプする命令です。Gotoはプログラムを分かりにくくする諸悪の根源なので、現在のプログラミングではGoto命令は使いません（Gotoがあちこちにあるため、処理の流れが把握しづらく分かりにくいプログラムをスパゲッティプログラムと呼びました）。Excel VBAにもGoto命令はありますが、本書では扱いません。

   　嫌われ者のGoto命令ですが、マシン語（CPUが理解できる命令）にはForやWhileという命令はありません。IfとGotoしかありません。Gotoは低レベル（CPUに近いという意味）の命令です。 [↑](#footnote-ref-2)
3. 筆者と同じ考えの人は他にもます。<https://www.wareko.jp/blog/best-loop-structure-for-programming>の執筆者も同じことを書いています (2021.10.8 アクセス)。「ワレコの講座 for/while/do-while/do-untilのどれが良い？ ループ処理」で検索すると見つかる。 [↑](#footnote-ref-3)