# 基本命令

【準備事項】

* 「挿入」→「標準モジュール」で標準モジュールを作成し、プログラムを書き込める状態にして下さい。
* 1.4 で述べたように、モジュールの1行目にOption Explicitと書いて下さい。

## セルに何か入れる

　まずはセルの中に何か入れることから始めましょう。セルA1に数値12、セルB1に文字列 "abc" を入れるプログラムは以下のように書きます。= は右辺の値を左辺に代入する命令です。

Sub rei1()

 Range("A1").Value = 12

 Range("A2").Value = "abc"

End Sub

　プログラムは「Sub プログラム名()」という記述ではじめます。このように記述すると、自動的にEnd Subという文字列が入ります。SubからEnd Subまでが1つのプログラムです。rei1 というのはプログラムの名前です。プログラムの名前のつけ方は以下の通りです。

* 使える文字はA～Z, a～z, 0～9, \_ （日本語も使えるがトラブルの元なので推奨しない）
* 数字からはじめてはいけない
* 大文字小文字は区別されない

　1つのモジュールの中に複数のプログラムを書くことができます。ただし、同じ名前のプログラム名をつけることはできません。つけるとエラーが発生します。

プログラムの実行

　プログラムの実行は、VBA エディタ（Microsoft Visual Basic for Applicationsというウィンドウ名がついている）のアイコンをクリックする、あるいは「実行」→「Sub/ユーザーフォームの実行」と操作する、あるいはF5を押します。

　その時点でカーソルが置いてあるプログラムが実行されます。Sub ～ End Subまでの範囲内にカーソルがない場合は、実行するプログラムを選ぶウィンドウが開きます。

字下げ

　Rangeを含む2行は字下げしています。tabキーを押すと、1ランク字下げします。こうすることで、プログラムが見やすくなります。字下げはプログラムの動作に影響を及ぼしませんが、プログラムを見やすくするため、必ず字下げして下さい。

　このプログラムはセルA1に数値12を入れ、セルA2に文字列 "abc" を入れます。

　セルの中に入るものは「数値」か「文字列」です。文字列の場合は "abc" のように、ダブルクオーテーションで囲みます。

　「実行」→「Sub／ユーザーフォームの実行」として、プログラムを実行して下さい。セルA1とA2にそれぞれ数値12と文字列 "abc" が入ったことを確認してください。

数値と文字列の区別

　文字列は "abc" のようにダブルクオーテーションで囲む必要があります。

　次のようにプログラムを組むとどうなるでしょうか？

 Range("A2").Value = abc

　実行すると



図2.1　変数未定義のエラー

というエラーが発生してプログラムの実行を中断します。対処法は以下の通りです。

エラーへの対処

　プログラム実行時にエラーが発生し、図2.1のウィンドウが出現したときは、「OK」を押して下さい。

　黄色の矢印が示され、その場所でプログラムの実行が中断していることを示しています。「実行」→「リセット」で「プログラムを実行するモード」から抜けて、プログラムを編集するモードに戻って下さい。その後、エラーを修正してください。

　このエラーは「abcという名前の変数を使おうとしたが、宣言されていない」というエラーです。変数については、後ほど学習します。文字列は "abc" のように、ダブルクオーテーションで囲んで下さい。

（注意！）

　数値を "12" のようにダブルクオーテーションで囲むことは絶対にしてはいけません。数値の12と文字列の "12" は異なります。Excel VBAは文字列を数値に自動変換する機能を持っているので、問題が発生しない場合が多いですが、発見困難なバグ（プログラムの誤り）を引き起こす可能性があるので、読者は数値と文字列を厳格に区別する習慣をつけて下さい。

　エラーをもう少し体験してみましょう。次のようにプログラムを変更して下さい。

 Renge("A2").Value = "abc"

　Rangeと書くところ、Rengeとスペルミスをしています。



図2.2　SubまたはFunctionの未定義エラー

　実行すると、今度は図2.2のエラーが出て中断します。Rengeという名前の関数を知らないというエラーです。コンピューターは1文字でも間違いがあると、エラーを出して実行してくれません。

セルを空にする

　セルA1を空にするには以下のように書きます。"" は「何もない」ことを意味します。

 Range("A1").Value = ""

Valueプロパティの省略

　セルにものを入れるときに

Range("A1").Value = 10

のように書きました。.Value を省略して

Range("A1") = 10

と書くことができます。以後 .Valueは省略します。

セルを指定するもう一つの形式

　上記のプログラムではセルを指定するのにRange("A1")のような形式を使いました。もう一つの形式があります。

Sub rei2()

 Cells(4, 2) = 23

 Cells(4, 3) = "def"

End Sub

　Cells(4, 2)は4行2列目（B列）を表します。この形式は、4, 2といった数値の部分を変数で表すときに便利な形式です。

　列は次のように、数値ではなく文字を使うこともできます。ただし、列に英文字を使うときは、Range("C4") のように書いた方が分かりやすいので、この形式を利用する必要はないでしょう。

Sub rei3()

 Cells(4, "C") = 23

 Cells(4, "D") = "def"

End Sub

## セルの値を読み出す

　これ以降はプログラム先頭の「Sub プログラム名()」と末尾の「End Sub」は省略します。入力するときは「Sub プログラム名()」と「End Sub」を補って下さい。

　次のプログラムを実行して下さい。

 Range("B1") = Range("A1")

　= は右辺の値を左辺に代入する命令でした。セルA1の内容を読み出して、セルB1に代入します。

## 変数

　次のプログラムを見て下さい。

 Dim a

 a = Range("A1")

 Range("B1") = a

　ここで登場したaを変数といいます。図2.3に変数のイメージを示します。セルと同じように「数値」または「文字列」を入れることができます。「セル」と「変数」は似ています。主な違いは以下の通りです。

* セルは中の値が常に見えますが、変数は見えない。
* 変数の方が値の代入や読み出しが高速
* 変数の方がタイプ量が少ない



図2.3　変数のイメージ

　aという名前の変数を使うには、あらかじめDim a と書く必要があります。これを変数の宣言と呼びます。変数の宣言を怠ると、エラーが発生します。Dim aの位置はaを使う場所よりも上であればどこでも良いですが、プログラムの先頭（Subの直後）に書くのが分かりやすいと思います。

　変数を2個以上使うときは、

 Dim a, b

のようにコンマで区切ります。

 Dim a

 Dim b

のように、2行に分けて書いてもよいです。

　変数名のつけ方はプログラム名と同じです[[1]](#footnote-1)。大文字と小文字は区別されません。ただし、変数名をDim Abcと宣言したなら、以後abcと打つと、VBAエディタがAbcと直してくれます。

　プログラム名と同じ変数名をつけてはいけません。実行時に図2.4のエラーが発生します。



図2.4　変数名とプログラム名が同一のときのエラー

　この例では1行目でセルA1の内容を変数aに代入し、2行目で変数aの内容をセルB1に代入します。結局、セルB1の内容をA1にコピーします。

　この場合は変数を使わず

 Range("B1") = Range("A1")

と書いた方が簡潔であり、変数を使う必要はありません。

　次節で学習する「ループを組む場合」などに変数が必要です。この例ではセルA1の内容を変数aに代入して、すぐにセルB1に代入していますが、aの内容に何らかの処理を加えたい場合などは、一旦変数にコピーしてから処理を記述する方がプログラムが書きやすく、見やすくなります。

　本書では以後、サンプルプログラムにおいて、変数の宣言は省略します。読者が例題を入力するときは、変数の宣言を補って下さい。

## イミディエイトウィンドウ

　VBAエディタのウィンドウの下方に「イミディエイト」という名前のウィンドウがあります。ない場合は「表示」→「イミディエイト」で表示させて下さい。この領域は2つの働きがあります。

* この中で単発の命令を実行することができる。
* この中に変数の内容などを表示することができる。

　イミディエイトウィンドウの中で

 Range("A1") = "nakyo"

と打ってEnterキーを押して下さい。セルA1に文字列 "nakyo" が入りました。このように、イミディエイトウィンドウの中で単発の命令を実行することができます。

　次のプログラムを実行しましょう。

 s = "abc"

 a = 10

 Debug.Print s

 Debug.Print a

　Debug.Printの後に書いた変数の内容がイミディエイトウィンドウに表示されます。

　2つ以上の変数の内容を表示したいときは以下のように書きます。

 s = "abc"

 a = 10

 Debug.Print s & " " & a

　& は文字列を接続します。この例ではsと " " は文字列、aは数値です。数値は文字列に変換して接続します。

　プログラム実行の途中で、変数の内容を確認したいときは、イミディエイトウィンドウに出力することで確認できます。

## プログラムの保存

　これまでに書いたプログラムを保存しましょう。新規作成したファイルを「ファイル」→「名前を付けて保存」とすると、図2.5のダイアログが表示されます。



図2.5　保存時の警告

　ここで「はい」は絶対に押してはいけません。プログラムは保存されず、Excelのシートのみが保存されるので、せっかく作ったプログラムが全て消えてしまいます。「名前を付けて保存」のダイアログの「ファイルの種類」で「Excelマクロ有効ブック \*.xlsm」を選びます。

　Visual Studio Basic for Applicationsの左上に「プロジェクト - VBAProjct」というタイトルのウィンドウがあります。ない場合は「表示」→「プロジェクトエクスプローラ」で表示して下さい。プロジェクトエクスプロ－ラには、現在開いているファイルのリストと各ファイルの構成要素が示されています。1つのExcelファイルはいくつかのシートと、ThisWorkbook（ファイルを開いた時に自動実行するプログラムを記述する場所）, 標準モジュール（存在しない場合もある）から構成されます。

## セルを変数で指定する

　セルC1～C5に数値12を入れるプログラムは以下のようになります。

 Range("C1") = 12

 Range("C2") = 12

 Range("C3") = 12

 Range("C4") = 12

 Range("C5") = 12

　C1～C5の数字の部分を変数で書くことができます。

 i = 1

 Range("C" & i) = 12

　このプログラムを実行するとC1に数値12を代入します。"C" & i の部分について説明します。iには1が入っています。& は文字列を接続する命令で、イミディエイトウィンドウのところで学習しました。数値の場合は文字列に変換してから接続します。ゆえに、"C" & i は "C" と "1" を接続することになり、"C1" となります。

　i = 2に書き換えるとC2に代入します。変数iの値を変えることで書き換えるセルを選択することができます。

### 【発展】

　ここでRange("C" & i) という形式を使いました。文字列を連結する基本的な方法は + で接続する方法です。以下のように書きます。

 Range("C" + CStr(i))

　+ と & は働きが少し異なります。+ は数値同士の場合は足し算をし、文字列同士の場合は接続します。文字列 "C" と数値 i を + で結ぶとエラーが起こります。CStr() は括弧内の数値を文字列に変換する命令です。文字列同士にすることで、エラーを起こさずに接続します。& は文字列を接続しますが、数値がある場合は文字列に自動的に変換してくれます。+ を使うより、& を使った方が簡潔に書けるので、本書では & を使います。

## For文

　次のプログラムを実行してみてください。

 For i = 1 To 5

 Debug.Print i

 Next i

　イミディエイトウィンドウに

1

2

3

4

5

と表示されたと思います。ForとNextは常にペアとなって用いられます。

　For ～ Nextはこの区間を繰り返し実行する命令です。i = 1 To 5と書くと、i = 1, 2, 3, 4, 5と1ずつ増やして5回繰り返します。1が最初の値、5が最後の値です。ループと呼びます。この例から分かるように、Forの後は字下げを行い、ループの範囲を明確にして下さい。

 For i = 1 To 5 Step 2

と書くと、iの値を2刻みに増やします。すなわち、i = 1, 3, 5と3回ループを回ります。i = 1 To 6 Step 2と書いても同じ結果となります。

　ここでは1 To 6と書くとどうなるか書きましたが、説明が冗長になり、かえって理解を妨げているかも知れません。仕様や説明を見ても分からないときは、自分でサンプルプログラムを作って、実行して、確認してください。プログラムは「疑問を持ったら、動かして確認する」という試行錯誤が必要です。好奇心や疑問を常に持って、手を動かして（サンプルプログラムを動作させてみて）確認してください。

 For i = 5 To 1 Step -1

と書くと、iの値は5, 4, 3, 2, 1と変化します。Stepを省略したときはStep 1として実行します。

　先ほどRange("C" & i)という書き方を学習しました。以下のように書くと、セルC1 ～ C5に12を入れます。

 For i = 1 To 5

 Range("C" & i) = 12

 Next i

　iは1, 2, 3, 4, 5と変化してFor～Nextの間を5回繰り返すので、"C" & iの部分は "C1", "C2", "C3", "C4", "C5" となります。

　ここではForループ用の変数としてiを使いました。プログラミングの世界ではループ用の変数として、i, j, k, l, m, nを順番に使う[[2]](#footnote-2)のが慣習となっています。初期のプログラミング言語であるFortranではI, J, K, L, M, Nからはじまる変数は整数型変数、それ以外は実数型変数と決められており、ループ変数にはI～Nを使いました（初期のFortranは大文字しか使えませんでした）。その名残と思われます。それ以外の変数名を使っても、プログラムの動作に問題はありませんが、他の人にとって読みづらいプログラムとなってしまいます。皆さんも慣習に従って、i, j, k,... の順番で使うようにして下さい。

【例題】

　シート「For例題」を使います。セルA1～A10の内容をB1～B10にコピーするプログラムを作成しなさい。

【解答】

 For i = 1 To 10

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

 Next i

## プログラミングの神髄

　プログラミングが数学と大きく異なるのは変数を使う点です。変数は図2.3で見たように、「数値」あるいは「文字列」を入れる箱です。数学では のように に値を代入したとき、 の値を次々と書き換えることはしませんが、プログラミングでは変数の値を何回書き換えてもかまいません。本節では「変数」の値を更新しながら処理を進めるというプログラミング独特の発想を学びます。

【例題】

　1から10までの和を求めてイミディエイトウィンドウに表示するプログラムを作りなさい。

【解答までの過程】

　まず、愚直に計算することを考えます。

 sum = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10

 Debug.Print sum

　一気に計算せずに、以下のように書くこともできます。

 sum = 0

 sum = sum + 1

 sum = sum + 2

 sum = sum + 3

 (省略)

 sum = sum + 10

 Debug.Print sum

　プログラムにおいて = は左辺と右辺が等しいことを表すわけではなく、右辺の式を計算した結果を左辺の変数（またはセル）に代入することを意味します。

　1, 2, 3,...., 10 の部分をfor文で書きます。

 sum = 0

 For i = 1 To 10

 sum = sum + i

 Next i

 Debug.Print sum

　上記のsum = sum + iがプログラミングの神髄を表しています。sum という変数の値を更新しながら計算を進めています。プログラミング独特の式です。紙に書いたものを書き換えるのは困難ですが、変数の値は何回でも変更できるので、このような方法が成立します。「プログラミングができる」ことは「sum = sum + iという発想ができる」ことと私は考えます。

　1, 3, 5, 7, ...., 99のように、1から99までの奇数の和を求めるプログラムは以下のようになります。

 sum = 0

 For i = 1 To 99 Step 2

 sum = sum + i

 Next i

　変数sumの初期化について考察します。プログラムの最初のsum = 0を省略すると、どうなるでしょうか？　Excel VBAでは、Dim sumのように宣言した変数[[3]](#footnote-3)は初期値としてEmptyという値が入っています。数値として扱って演算を行うと0とみなされるので、sum = 0 を省略しても結果は同じです。しかし、デフォルト値を使うのは危険です。C言語の場合、変数の初期値はランダムな値が入っています。pythonで同様のプログラムを書くと、エラーとなります。変数を使うときは初期化する習慣を付けて下さい。

【例題】

　シート「For例題」を使います。セルA1～A10の内容をA1 → B1, A2 → B3, A3 → B5, A4 → B7のように、一つおきにコピーするプログラムを作成しなさい。

【解答1】

 For i = 1 To 10

 j = (i - 1) \* 2 + 1

 Range("B" & j) = Range("A" & i)

 Next i

　j = (i - 1) \* 2 + 1という式は、1→1, 2→3, 3→5の関係を表しています。この計算式を導出するのは少しややこしいですね。以下のように書くのがコンピューターらしい考え方です。

【解答2】

 j = 1

 For i = 1 To 10

 Range("B" & j) = Range("A" & i)

 j = j + 2

 Next i

　ここでもプログラミングの神髄と言えるj = j + 2という式を使っています。この式はjを2増やすことを意味しています。

　このように変数の値を更新しながら処理を進めるという概念を理解して下さい。

　B1, B4, B7,...のように2つおきにコピーする場合はj = j + 3と書けば良いです。B1, B2, B4, B7, B11...のように、間隔を1個ずつ増やしていく場合は、j = j + iです。

【例題】

　セルA1～A10の内容をC10～C1に逆順にコピーするプログラムを作成しなさい。すなわち、C1 ← A10, C2 ← A9, C3 ← A8 のように代入します。

【解答1】

 For i = 1 to 10

 j = 11 - i

 Range("C" & i) = Range("A" & j)

 Next i

　これでもよいですが、変数の値を更新して処理を進める次のプログラムの方が明快です。

【解答2】

 j = 10

 For i = 1 to 10

 Range("C" & i) = Range("A" & j)

 j = j - 1

 Next i

【例題】

　シート「数列」を使います。セルA1からA2, A3と下方へ向かって順番に1, 3, 5, 7....という数値をA10まで入れなさい。

【解答】

 num = 1

 For i = 1 To 10

 Range("A" & i) = num

 num = num + 2

 Next i

　セルの中に書き込む数値を保持する変数がnumです。2ずつ増やしています。

## デバッグ

　前節のセルを逆順にコピーするプログラムにおいて、For文を以下のように書き換えて下さい。

 For i = 1 To 11

　実行すると、図2.6のようなエラーが発生し、プログラムはストップします。



図2.6　エラー発生時の様子

　エラーの原因を追求しましょう。「デバッグ」をクリックすると、図2.7のようにエラーが発生した行の行頭に黄色矢印が表示されます（筆者のExcel VBAエディタは配色を変えています）。



図2.7　「デバッグ」を押したときの状況

　この行でエラーが発生しています。Rangeの上にカーソルを合わせると「'Range' メソッドは失敗しました」と表示され、jの上にカーソルを合わせると「j = 0」と表示されます。セルA0に1を入れようとしたが、A0という名前のセルはないので、エラーが発生しています。

　黄色で反転した行が表示されているとき、プログラムは中断した状態を継続しています。「実行」→「リセット」でプログラムを実行するモードから抜けてください。

　この場合は、カーソルを変数の上に合わせることで、エラーの原因が判明しました。イミディエイトウィンドウを使うと、さらに複雑な操作もできます。イミディエイトウィンドウで

Print j

あるいは

? j

と書いてEnterを押してください。? は Print の省略形です。j の値が0であることが表示されました。イミディエイトウィンドウで

j = 1

と打って下さい。jの値をその場で変更することができます。そして「実行」→「継続」と押すと、プログラムがストップした時点から、プログラムを再開することができます。

　プログラムがエラーでストップしたとき、イミディエイトウィンドウを使って、変数の値を調べたり、変数の値を変更したりすることができます。

## コメント文と継続行

　プログラムの中に注釈を書くことができます。以下のように、'（アポストロフィー）を書くと、その行の右側は何を書いても無視されます。プログラムの内容を分かりやすく解説しておくと、後日プログラムを見直したときや、他人がプログラムを見るときに役に立ちます。

'

' A1 から A10 まで 10 を入れるプログラム

'

 Dim i, num

 num = 10 ' 空行はどこに入れても良い。プログラムを

 ' 見やすくするため、適宜空行を入れよう

 For i = 1 To 10

 Range("A" & i) = num ' コメントは行の途中に書くこともできる

 Next i

　これまで学習したプログラムは1行が短いですが、今後は1行が長くなるかもしれません。以下のように、\_ (アンダースコア) を書くと、続きを次の行に書くことができます。

 Range("A" & i) = \_ ' アンダースコア \_ を入れると、続きを

 num ' 次の行に書くことができる。

## 四則演算の表記法

　以下のように書きます。

 a = b + c

 a = b - c

 a = b \* c

 a = b / c

　足し算、引き算は数学のときに使う記号と同じです。かけ算は "\*"（アスタリスク）、割り算は "/"（スラッシュ）を使います。数学では

のように "×" 記号を省略できますが、プログラムでは省略せずに

 x = 2 \* a

 y = b \* c

と書かねばなりません。

 x = 2a

 y = b c

と書くと構文エラーとなります。

　括弧は数学とは異なり、( ) のみを使います。たとえば、

は以下のように書きます。

 x = a \* ( b + c ) / (( d + e ) \* f )

　分母全体を括弧で囲むのを忘れて、

 x = a \* ( b + c ) / ( d + e ) \* f

と書いてしまうと、

という計算になってしまいます。分子に括弧をつけて

 x = ( a \* ( b + c ) ) / (( d + e ) \* f )

と書く必要はないですが、括弧があった方が分かりやすいです。

　累乗については、

を以下のように書きます。

 y = x ^ n

## If文

　条件が成立するか否かによって、実行する命令を切り替えるのがIf文です。シートを「If練習」に切り替えてください。A列に名前、B列に点数が入っています。C列に加点するか否かが入っています。1が入っている人は加点対象者です。

【例題】

　D列に点数を入れなさい。ただし、加点対象者は10点加えます。

【解答1】

　個々の行に対して図2.8に示す処理を施します。



図2.8　If文のフローチャート

　図2.8のように処理の流れを表した図をフローチャートと呼びます。フローチャートにおいて、「長方形」は「普通の処理」を表し、「菱形」は「If文（条件判断）」を表します。

　これをプログラムで表すと以下のようになります。

 Dim i

 For i = 2 To 11

 If Range("C" & i ) = 1 Then

 Range("D" & i) = Range("B" & i) + 10

 Else

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

 End If

 Next i

　データが2～11行目に入っているので、For文の範囲は2 To 11 です。条件によって処理を変えるにはIf文を使います。If文は以下の形式をとります。

If 条件式 Then

 Ifが成立したときの処理1 ' 処理の個数は何個でもよい

 Ifが成立したときの処理2 ' 極端な場合として 0 個でもよい

 .....

Else

 Ifが成立しないときの処理1

 Ifが成立しないときの処理2

 .....

End If

　条件が成立しないときの処理が不要な場合、Else以下を省略して以下のように書きます。

If 条件式 Then

 Ifが成立したときの処理1 ' 処理の個数は何個でもよい

 Ifが成立したときの処理2

 .....

End If

【解答2】

　図2.9のような考え方もできます。



図2.9　もう一つの考え方

　暫定値を入れておき、加点者は値を更新するという考え方です。これをプログラムにすると、以下のようになります。

 Dim i

 For i = 2 To 11

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

 If Range("C" & i) = 1 then

 Range("D" & i) = Range("D" & i) + 10

 End If

 Next i

　条件が成立しないときは、何もしないので、Elseはありません。

　解答1と解答2のどちらが分かりやすいか？　と問われたなら、多くの人は「解答1の方が分かりやすい」と感じると思います。人間の思考過程をそのままプログラムとして表現しているからでしょう。解答2は「いかにもコンピューターらしい」という感じがします。「最初に値を入れておいて、あとから書き換える」というのは、紙ではやりにくいですが、コンピューターでは容易です。加点要素が複数ある場合は、解答2の方式でないと対応できません。

　話をIf文の文法の話に戻します。条件が成立しないときのみ処理を行いたいときは、以下のように書きます。

If 条件式 Then

Else

 Ifが成立しないときの処理1

 Ifが成立しないときの処理2

 .....

End If

　条件式は以下のように書きます。

　文字列を比較する場合は「等しい」か「等しくない」のどちらかです。

 a = b　　　　aとbは等しい

 a <> b　　　 aとbは等しくない

　空白も一つの文字なので、"ab" と "ab "（abの後に半角の空白が一つ）は「等しくない」と判定されます。

　数値の場合、以下の6パターンがあります。

 a = b　　　　　等しい

 a <> b　　　　 等しくない

 a < b　　　　　＜

 a <= b　　　　 ≦

 a > b　　　　　＞

 a >= b　　　　 ≧

　>= のように、不等号とイコールを書く場合、不等号を先に書きます。=> と書くと、Visual Basic for Applicationsのエディタが自動的に >= に直してくれます。

　「等しくないときに処理を実行する」という処理をしたい場合、以下の2つの書き方が考えられます。

 If a = b Then ' a = b のときは何もせず、そうでないなら c = 10

 Else

 c = 10

 End If

 If a <> b Then ' a = b でないとき c = 10

 c = 10

 End If

　どちらが分かりやすいかは、場合によると思います。

2個以上の条件を使う場合

　「a > b かつ c > d」のように、2つの条件が両方とも成立するときのみ、何らかの処理をするケースを考えます。素直に書くと以下のようになります。このようにIfの中にIfを書くことができます。入れ子構造（ネストする）と呼びます。

 If a > b Then

 If c > d Then

 ここに実行したい処理を書く

 End If

 End If

　If a > b と If c > d の順番を逆にしても、処理結果は同じです。これを一つにまとめて、以下のように書くことができます。

 IF a > b And c > d Then

　「5 < a < 10」という条件の場合、そのまま書くとエラーになります[[4]](#footnote-4)。以下のように2つに分解して下さい。

 If 5 < a And a < 10 Then

　「a > b または a < c」のように2つの条件のどちらかが成立するとき（もちろん両方成立するときも含まれます）にThen以下の命令を実行したいときは、以下のように書きます。

 IF a > b Or c > d Then

　ここで間違えやすいパターンが一つあります。「a = 1またはa = 2」という条件を以下のように書いてはいけません。

 If a = 1 Or 2 Then

　これは「a = 1」という条件と「2」という条件のいずれかが成立したときにThen以下を実行することを意味しており、「2」という条件は常に成立する[[5]](#footnote-5)ので、このIf文は常に成立します。以下のように書いてください。

 If a = 1 Or a = 2 Then

【例題】

　E列に最終得点を記入しなさい。原則としてD列の得点を転記しますが、100点を超えた場合は、100点にします。

【解答1】

 Dim i

 For i = 2 To 11

 If Range("D" & i) > 100 Then ' >= 100 でもよい

 Range("E" & i) = 100

 Else

 Range("E" & i) = Range("D" & i)

 End If

 Next i

【解答2】

 For i = 2 To 11

 Range("E" & i) = Range("D" & i)

 If Range("E" & i) > 100 Then ' >= 100 でもよい

 Range("E" & i) = 100

 End If

 Next i

　前の例題と同様に、解答2はあとから値を書き換えるというコンピューターらしい方法です。これまでの例題の解答は変数を使わずRange( ) を使っていました。解答2を変数を使って書き直すと、以下のようになります。

【解答3】

 Dim i, score

 For i = 2 To 11

 score = Range("D" & i)

 if score > 100 Then

 score = 100

 End If

 Range("E" & i) = score

 Next i

　解答2と解答3を比べると、どちらがよいプログラムでしょうか？　解答3は冗長な感じがしますが、以下の長所があります。

* セルの内容を変数に入れる → 処理をする → 変数の内容をセルに戻す　という流れが分かりやすい。
* 変数名から処理内容が類推できる。
* Rangeより変数名の方がタイプしやすく、見やすい。

　この例は単純な場合であり、大きな優劣はなさそうに見えます。好みの範囲かもしれません。

【変数名のつけ方について】

　上の例では得点を表す変数名としてscoreを用いました。日本語のローマ字綴りのtokutenを使う方法もあります。日本語のローマ字綴りの変数名を使うことに対しては、以下のように賛否両論があります。

長所

* 英語表現が分からないときに調べる必要がない。
* 英語では表現しづらい場合がある。

短所

* ローマ字綴りは読みづらい。
* ローマ字の綴り方には訓令式とヘボン式があり、一意でない場合がある。
* 日本語を知らない人には読めないプログラムになる。

　私が個人で使うプログラムには、日本語のローマ字綴りの変数名と英語の変数名が混在しています。

　「変数名に全角文字を使って良いか否か」についても賛否両論があります。使うとプログラムは見やすくなりますが、「変数名をタイプするときに日本語変換が必要となるので、タイピング速度が落ちる」「全角文字を使えない処理系があるので移植性が落ちるん」などのデメリットがあります。何となく違和感があるので、私は使っていません。

　2000年頃は日本語のファイル名を付けることにも違和感がありましたが、今は違和感なく日本語のファイル名を使っているので、変数名についても今後は使うことに違和感がなくなるかも知れません。

【例題】

　F列に合否を記入しなさい。60点以上のとき合格とします。合格のとき "○", 不合格のとき "×” を記入します。同時に、不合格のセルはRGB(255,200,255)の色で塗りつぶし、合格のセルは「塗りつぶしなし」に設定しなさい。

【解答】

 Dim i

 For i = 2 To 11

 If Range("E" & i) >= 60 Then

 Range("F" & i) = "○"

 Range("F" & i).Interior.Pattern = xlNone

 Else

 Range("F" & i) = "×"

 Range("F" & i).Interior.Color = RGB(255, 200, 255)

 End If

 Next i

　セルの塗りつぶしは以下のように指定します。

 Range("A1").Interior.Color = 色の情報

　色の情報はRGB(r, g, b) のように指定します。r, g, bはそれぞれ0～255の範囲で指定します[[6]](#footnote-6)。

　「塗りつぶしなし」のときは

 Range("A1").Interior.Pattern = xlNone

と指定します[[7]](#footnote-7)。

【例題】

　E列が80点以上を成績優秀者とする。成績優秀者のリストを作ることにした。成績優秀者の名前をH2, H3, H4,.... のセルに並べなさい。

【解答】

 Dim i, j

 j = 2

 For i = 2 To 11

 If Range("E" & i) >= 80 Then ' 80点以上の人は H 列に記入

 Range("H" & j) = Range("A" & i) ' j は行番号

 j = j + 1 ' 記入したら次の行に進める

 End If

 Next i

　j は「成績優秀者を記入するときの行番号」を保持する変数です。初期値2からスタートし、1人記入するたびに1増やします。ここでの重要事項は以下の通りです。

* 初期値を設定するj = 2はForの手前に書く（ループの外に書く）
* j = j + 1はIfの内側に書く

## ElseIf

　シート「ElseIf練習」を見て下さい。名前と得点が記入されています。プログラムが正しく動くことを確認するため、名前や点数はプログラムの動作確認用の値が入っています。

【例題】

　B列の値に従ってC列に評価を書き込みます。90以上は "A", 80以上90未満は "B", 70以上80未満は "C", 60以上70未満は "D", それ以外は "F" です。IFを使います。点数が低い方から考えてもいいですが、ここでは高い方から考えることにします。

　フローチャートを書くと図2.10のようになります。Rangeを何回もタイプするのが苦痛なので、変数scoreを導入しています。



図2.10　ElseIfのフローチャート

　プログラムに書き下すと以下のようになります。

 Dim i, score, grade

 For i = 2 To 13

 score = Range("B" & i)

 If score >= 90 Then

 grade = "A"

 Else

 If score >= 80 Then

 grade = "B"

 Else

 If score >= 70 Then

 grade = "C"

 Else

 If score >= 60 Then

 grade = "D"

 Else

 grade = "F"

 End If

 End If

 End If

 End If

 Range("C" & i) = grade

 Next i

　Ifの中にIfがあり、入れ子構造（ネスト）が深くなっています。入れ子構造が深くなるとプログラムが見づらいです。ElseIfという命令を使って以下のように書くことができます。

【解答】

 Dim i, score, grade

 For i = 2 To 13

 score = Range("B" & i)

 If score >= 90 Then

 grade = "A"

 ElseIf score >= 80 Then

 grade = "B"

 ElseIf score >= 70 Then

 grade = "C"

 ElseIf score >= 60 Then

 grade = "D"

 Else

 grade = "F"

 End If

 Range("C" & i) = grade

 Next i

　慣れるまではこの形式に違和感があるかも知れませんが、定番表現なので、処理が複数に分かれるIfの使い方に慣れて下さい。ElseIfを使うことで「End Ifが多数連続して字下げが深くなって見づらい」という状況を回避することができます。

【注意】

　入力していると、以下のようにElseの後にコロン「:」が発生することがあります。

 Else: grade = "F"

　Excel VBAでは以下のように、コロンで区切ることにより、1行に複数の命令を書くことができます。

 a = 1: b = c

　コロンを使って1行に複数の命令を書く形式は、初期のBasic[[8]](#footnote-8)との互換性を保つための仕様と思われます。筆者はお勧めしません。Elseと同じ行にgrade = "F" と書くと、Excel VBAエディタが自動的にコロンを挿入します。

≪危険なプログラム1≫

　次のようなプログラムを組む人がいるかも知れません。

 Dim i, score, grade

 For i = 2 To 13

 score = Range("B" & i)

 If score >= 90 Then

 grade = "A"

 ElseIf score < 90 And score >= 80 Then

 grade = "B"

 ElseIf score < 80 And score >= 70 Then

 grade = "C"

 ElseIf score < 70 And score >= 60 Then

 grade = "D"

 Else

 grade = "F"

 End If

 Next i

　このプログラムは一見、分かりやすいように見えますが、冗長かつ危険なプログラムです。

ElseIf score < 90 And score >= 80 Then

の部分（5行目）について考えます。この部分に到達するのは、一つ手前の条件である「score >= 90」が成立しない場合なので、「score < 90」は必ず成立します。ゆえに、Andの左側の条件「score < 90」は必ず成立するので不要です。

　『score < 90 And score >= 80 と書く方が「90点未満」かつ「80点以上」がより明瞭なので、分かりやすい』と考える人がいるかも知れません。しかし「score < 90」は冗長であるばかりでなく、落とし穴を掘るような危険性を秘めています。90点という境界を92点に変更した場合を考えます。【解答】として示したプログラムでは90 → 92と1箇所変更すれば良いですが、このプログラムの場合、90が2箇所あるので、両方とも変更する必要があります。もし、最初の「If score >= 90 Then」の90は変更し、次の「ElseIf score < 90 And score >= 80 Then」の90を変更し忘れたとすると、91点と90点の人がどこにも引っかからなくなり、最後のElseの部分に引っかかり、成績 "F" がついてしまいます。

≪危険なプログラム2≫

　次のプログラムも危険なプログラムです。Elseがありません。

 Dim i, score, grade

 For i = 2 To 13

 score = Range("B" & i)

 If score >= 90 Then

 grade = "A"

 ElseIf score >= 80 Then

 grade = "B"

 ElseIf score >= 70 Then

 grade = "C"

 ElseIf score >= 60 Then

 grade = "D"

 ElseIf score >= 0 Then

 grade = "F"

 End If

 Range("C" & i) = grade

 Next i

　最初の【解答】に示したIf ～ ElseIf ～ ElseIf ……～ Else ～ EndIfのプログラムは、必ずどこかの部分に引っかかります。上記のようにElseがないプログラムは、どの条件にも引っかからない場合が発生する可能性があります。例えば、得点の欄に「負の値」が入っている場合、i = 2の場合（1人目）はgradeはデフォルト値の "" となり、それ以外の場合（2人目以降）はgradeは一つ前の人の値になります。これは重大な事故を引き起こす可能性があります。

【発展】

　Excel VBAはセルに何も入っていないとき、Emptyという値が入っているとみなします。Emptyと数値を比較すると、Emptyは0とみなされて処理されます。従って以下のIf文は「真」となります。

 a = Empty

 If a = 0 Then

　本来ならEmptyと数値を比較するとエラーが発生するべきです。

　文字列（半角空白も含む）と数値を比較すると「文字列は如何なる数値よりも大きい」と判定されます。以下のIf文は「真」となります。

 a = " "

 If a >= 1000000 Then

　本来なら文字列を数値と比較した場合、エラーが発生するべきです。

　Excel VBAはこのようにエラーが発生するべきところ[[9]](#footnote-9)、処理を進めてしまいます。その結果、たちの悪いバグが発生する可能性があります。Excel VBAは危険な処理系です。そのような処理系は可能であれば避けたいですが、現時点ではExcelのセルやシートを操作するための実用的な処理系はExcel VBAだけなので、筆者は仕方ないと割り切って使っています。

　将来、ExcelのセルやシートをPythonで操作可能になるなら、pythonに乗り換えるべきでしょう。2024年4月の時点では、PythonでExcelを操作する2つの方法があります。

(1) openpyxlなどのライブラリを使用してpythonでExcelファイルを操作する

(2) Python in Excelを使う

　どちらもExcel VBAを代替可能とは言えない状況です。

文字列を扱うIf文

　セルに文字列 "A" が入っているか否かは次のように書きます。

 If Range("D" & i) = "A" Then

セルが空か否かを判断する

　セルが空か否かを判定するには、以下のように書きます[[10]](#footnote-10)。"" は長さ0の文字列を表します。

 If Range("D" & i) = "" Then

【例題】

　シート「セル操作」選択して下さい。A列に名前、B列に出欠の結果を入れています。空白のセルに "×" を入れるプログラムを作りなさい。プログラムがうまく動かなくてB列が破壊されたときは、C列からコピーしなさい。

【解答】

 Dim i

 For i = 2 To 10

 If Range("B" & i) = "" Then

 Range("B" & i) = "×"

 End If

 Next i

　セルが空白か否かは "" と比べればよい。

【例題】

　D列にレポートの評価値が入っています。E列にレポート点を記入しなさい。レポート点は評価がAのとき10, Bのとき8, Cのとき6, Dのとき4, Eのとき2, 未提出のとき0です。レポート未提出の場合D列のセルは空です。

 rep\_grade = Range("D" & i)

 If rep\_grade = "A" Then

 rep\_score = 10

 ElseIf rep\_grade = "B" Then

 rep\_score = 8

 ElseIf rep\_grade = "C" Then

 rep\_score = 6

 ElseIf rep\_grade = "D" Then

 rep\_score = 4

 ElseIf rep\_grade = "E" Then

 rep\_score = 2

 ElseIf rep\_grade = "" Then

 rep\_score = 0

 Else

 MsgBox("想定外の値が来ました")

 Exit Sub

 End If

　新しい技をここで使っています。MsgBoxは "" で囲まれた文字が書かれたダイアログボックスを表示します。Exit Subはプログラムを終了します（End Subまでジャンプすると考えればよい）。

　上記のプログラムはElseにおいて、"A", "B", "C", "D", "E", "" 以外の値が来たときの処理をしています。

 (略)

 ElseIf rep\_grade = "" Then

 rep\_score = 0

 End If

と書いてしまうと、想定外の値が来たときに、Ifのどの部分にも引っかからないので、rep\_scoreに値を入力せずに処理が進んでしまいます。前の人のrep\_scoreの値を使ってしまいます。

 (略)

 Else rep\_grade = "" Then

 rep\_score = 0

 End If

と書くと、必ずどこかに引っかかりますが、「未提出者の場合」と「想定外の値の場合」の両方のときに0点となってしまいます。

　皆さんはこれで、Excel VBAの基礎的な命令のほとんどを修得しました。

　残っている命令はDo While ～ Loop のループだけです。これをマスターしたなら、後は自分のアイデア次第で何でも組むことができるでしょう。

## よくあるエラーのパターン

　以下のプログラムは誤りが3箇所あります。どこか分かりますか？

 Dim i

 For i = 1 To 10

 If Range("A" & i) >= 10 Then And Range("B" & i) <= 30 Then

 ("A" & i) = 12

 End If

 Nexti

【解答】

1. If 文の中央付近の Then は不要
2. Range("A" & i) とするところを ("A" & i) としている
3. Nextとiの間にスペースがない。

　次のプログラムは「Nextに対応するForがありません。」というエラーが出ます。

 Dim i

 For i = 1 To 10

 If Range("A" & i) = "" Then

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

 Next i

　エラーの本当の原因はEnd Ifがないことです。Excelは「Nextを見つけた。このNextに対応するForがIfより下にあるはずだがない（Ifの中にForがある）」と判断するようです。ループ変数はiなので、「Next iに対応するForはIfの上にあるForである」と判断してほしいところですが……

　次のプログラムは何か錯覚をしています。どのような錯覚をしているか分かりますか？

 Dim i, a

 For i = 1 To 10

 a = Range("B" & i)

 If Range("A" & i) = "A" Then

 a = 10

 Else

 a = 5

 End If

 Next i

【解答】

　Forの次の行にa = Range("B" & i) とあります。これはa = Range("B" & i).Valueと解釈されて、変数aにセル（"B" & i）の内容が入ります。しかし、プログラマーはaに「Range("B" & i) というセル」を入れたと錯覚しています。

　a = 10とa = 5の部分は、それぞれRange("B" & i) = 10, Range("B" & i) = 5を意図しています。

【発展】

　変数aに代入するときに「a = 〇〇」と書くと、aには「数値」か「文字列」が入ります。セルも「数値」や「文字列」が入るので、変数とセルは似ていますが、少し違います。セルは「オブジェクト」と呼ばれます。実は変数は「数値」や「文字列」以外に「オブジェクト」を入れることができます。変数aにオブジェクトを代入するときは、以下のようにSetを使います。

 Dim i, a

 For i = 1 To 10

 Set a = Range("B" & i) ' オブジェクト変数への代入は Set 必要

 If Range("A" & i) = "A" Then

 a.Value = 10 ' Value を省略して a = 10 と書いても良い

 Else

 a.Value = 5

 End If

 Next i

　この場合、aはオブジェクト変数であり、Range("B1") などのセルを表します。セルの内容はa.Valueです（.Valueは省略可）。

　セルaの色を変更したいときは、

 a.Interior.Color = RGB(0, 255, 255)

のように書きます。また、型（後の章で学習します）を明示して宣言する場合は

 Dim a As Object

と宣言します。

## ちょっと便利な技

　イミディエイトウィンドウに出力するとき、似たような出力が連続すると、「今回のプログラム実行で出力しているのは、どこからだったっけ？　ということが起こります。Subの直後に以下のように書くと、分かりやすいです。

 Debug.Print

 Debug.Print Now()

　1行目は単に改行のみを行います。Now() は現在の日付と時刻を取得する関数であり、Now() の部分が現在の日付と時刻に置換されます。

1. 変数名として「漢字」「ひらがな」「カタカナ」を使うことができます。これらの文字を使うことの可否は賛否両論あります。プログラムは見やすくなりますが、日本語が分からない人には読めないプログラムとなります。また全角文字が使えない処理系もあるので、移植性が悪くなります。筆者は「使わない派」です。goukeiのように日本語をローマ字で書いた変数名を使うことに関しても賛否両論あります。対応する英語がすぐに分からないときや、難しい単語となるときは、筆者は使います。ただし、日本語のローマ字綴りは読みづらく、goukeiよりsumの方が分かりやすいです。また、gokeiかgoukeiのどちらを使うか、という問題もあります。この問題についてはIf文（2.12節）のところで再度取り上げます。 [↑](#footnote-ref-1)
2. 3章でForループの中にForループが入るという二重ループについて学習します。そのときにiの次の変数名としてjを使います。 [↑](#footnote-ref-2)
3. 後の章で、変数の型について学習します。Dim sum と宣言した変数はVariant型の変数であり、初期値としてEmptyという値が入ります。数値として扱うと0, 文字列として扱うと "" とみなされます。 [↑](#footnote-ref-3)
4. pythonではこの書き方が許されます。 [↑](#footnote-ref-4)
5. If文の中のa = bのような条件式は、条件が成立するならTrue, そうでないならFalseに置換されます。If True Thenと書くと、このIf文の条件は常に成立します。条件式の場所にIf 1 Then のように数値を書くと、0の場合はFalse, それ以外は Trueと判定されます。ゆえに、If a = 1 Or 2 は If a = 1 Or True と書くのと同じであり、このIf文は常に成立します。 [↑](#footnote-ref-5)
6. RGB(r, g, b) は整数値を返す関数です。イミディエイトウィンドウで print RGB(1, 0, 0) などと打つと返り値を確認できます。返り値はr + g \* 256 + b \* 256 \* 256 です。 [↑](#footnote-ref-6)
7. イミディエイトウィンドウで print xlNoneと打つと、-4142という数値が得られます。xlNoneは-4142という値を持つ定数のようです。

 Range("A1").Interior.Color = xlNone

でも塗りつぶしなしに設定されます。覚えやすさからは、(略).Color = xlNoneの方が覚えやすいですが、後の章で学習する「マクロの記録」を行って塗りつぶしなしに設定したとき、(略).Pattern = xlNoneが得られたので、本テキストでは (略).Patternを最初に紹介しました。 [↑](#footnote-ref-7)
8. 筆者がプログラミングを始めた1980年頃、スクリーンエディタ（文字を自由に挿入、削除できるアプリ：代表的なアプリはメモ帳）がなかったので、各行に文番号という番号を付け、特定の行を削除するには、番号を指定して削除命令をキーボードから入力する必要がありました。その時代はメインメモリが32 KB（今は32 GB, 100万倍!）という状況でした。文番号を減らして少しでもメモリを節約するため、1行にコロンで区切って複数の命令を書くことが行われました。マルチステートメントと呼びました。 [↑](#footnote-ref-8)
9. pythonでは文字列と数値を比較するとエラーが発生します。変数に値を代入せずに比較するとエラーが発生します。 [↑](#footnote-ref-9)
10. セルが空か否かの判定は奥が深い。この方法ではセルの中に数式「=""」を書いた場合もTrueとなってしまいます。セルが空か否かを判定するにはIsEmpty(セル)という関数を使います。

　一方で、変数aに何も代入していない状態で IsEmpty(a) はTrueを返しますが、a = "" のあとのIsEmpty(a) はfalseとなります。

　セルに数式「=""」が入っている場合も空であると判定してよいなら、「〇=""」はセル、変数のどちらにも使えます。本書では深入りせずに、空か否かを判断するには、「〇=""」を使うことを推奨します。 [↑](#footnote-ref-10)