# 少し細かい知識・テクニック

## 変数の型

　これまでに使用してきた変数は「数値」「文字列」のどちらも入れることができました。これをVariant型（バリアント型）変数と言います。

　それに対して「整数しか入らない変数」「実数（整数含む）しか入らない変数」「文字列しか入らない変数」を利用することができます。それぞれ次のように書きます。

 Dim i As Long ' Long は4バイト整数

 Dim a As Double ' Doubleは8バイト実数

 Dim s As String ' String は文字列型

 Dim v As Variant ' v はVariant型

 Dim v2 ' 型を省略するとVariant型なのでv2はVariant型

　iはLong型（長整数型）の変数です。4 byte使います。-2147483648～2147483647の範囲の整数を入れることができます。

　aはDouble型（倍精度浮動小数点型）の変数です。8バイト使います。15桁の精度で±1.7×10±308の範囲の数値を表すことができます。

　sはString型（文字列型）です。文字列を入れることができます。

　vはVariant型です。型を省略するとVariant（バリアント）型になります。v2はVariant型です。これまで使ってきた変数は、実はVariant型なのでした。

　ここでは4つの型を紹介しました。それ以外にも型はありますが、この4つが使えれば十分です。

 Dim i As Long

 i = "abc"

と書いて、整数型変数に文字列を入れようとすると、実行時に



図5.1　型が不一致のときのエラー

というエラーが出て、プログラムがストップします。

 i = "10"

と書くと、文字列 "10" が数値10に自動的に変換されてからiに入ります。

　整数型変数に実数を代入すると、何が起こるでしょうか？　調べてみましょう。

 Dim i As Long

 Dim j As Long

 Dim x As Double

 For i = 1 To 41

 x = (i - 1) / 10 ' x は 0 からはじまって 0.1 ずつ増加する[[1]](#footnote-1)

 j = x

 Range("a" & i) = x

 Range("b" & i) = j

 Range("c" & i) = WorksheetFunction.Round(x, 0)

 Next i

　C言語の場合、切り捨てが起こりますが、Excel VBAの場合、四捨五入に類似した丸めが起こります。C列はWorksheetFunction.Roundを使っているので、四捨五入です。問題のB列とC列を比べると、ほぼ同じですがi.5のときの処理が異なります。四捨五入ではi.5のときi+1になりますが、B列（整数型変数に実数を代入）の場合、偶数に丸められます。

　実数型変数に整数を入れるのは問題ありません。

　型が異なる複数の変数を1行で宣言する場合、以下のように書きます。

 Dim i As Long, a As Double, s As String

　Long型変数i, jを宣言するとき、次のように書きます。

 Dim i As Long, j As Long

　C言語に慣れている人は、次のように書いてしまうかも知れませんが、これは誤りです。

 Dim i, j As Long

　前節では型を指定せずに変数を宣言しました。その場合の変数はVariant型でした。従って、上記の宣言は

 Dim i As Variant, j As Long

と同等であり、iはLong型ではなくVariant型となります。

Variant型変数に関する注意

　次のプログラムを見て下さい。

 Dim a, b

 a = "10"

 b = 10

 If a = b Then

 Debug.Print "="

 Else

 Debug.Print "<>"

 End If

　a, bはVariant型変数なので、aは文字列 "10", bは数値10です。本来、文字列と数値を比較してはいけませんが、比較できてしまいます。If文でエラーは起こりません。そして、Ifの結果は "<>" となります。

 Dim a As Long, b As Long

と宣言しておけば、a = "10" のときに、自動的な型変換がおこり、aには数値10が入ります。Ifの結果は "=" となります。

　型宣言をする場合、Variant型は使わず、Long, Double, Stringのみを使うのが正統派のプログラミングだと思います。しかし、

 Dim a, b As Long

と書いてもエラーとはなりません。aの型を宣言するのを忘れていますが、エラーにならずa As Variantと解釈されます（型を省略した場合はVariant型になるというルールがあります）。使うのが望ましくないVariant型を使ってしまいます。筆者は

* 型宣言を忘れてVariant型を使ってしまう。
* 全ての変数の型宣言を行うのは煩雑である。

という理由により、Option Explicitは宣言しますが、変数の型は宣言せず、デフォルトのVariantを使っています。すなわち、タイプミス防止のためのみに、変数の宣言を行っています。

セルの書式設定による自動的な型変換

　セルに代入するとき、セルの書式設定によって、自動的な型変換が起こる場合があります。

 Dim a, b

 Range("A1") = 10

 Range("B1") = "10"

 a = Range("A1")

 b = Range("B1")

を実行すると、a, bには文字列 "10" か数値 10 のどちらが入るでしょうか？

　A1, B1の「セルの書式設定」が「標準」あるいは「数値」の場合は以下のようになります。A1には数値10が入るのでaは数値10です。B1に代入時、自動的な型変換が行われ、B1には文字列 "10" ではなく数値10が入ります。ゆえにa, bともに数値です。

　A1, B1の「セルの書式設定」が「文字列」の場合、自動的な型変換は起こりません。A1には数値10が入り、B1には文字列 "10" が入ります。ゆえにaは数値でbは文字列です。

型変換の関数

　「セルへの代入」や「型が決まっている変数への代入」のときに起こる「自動的な型変換」を使うと、プログラムが分かりにくくなり、バグの温床となります。型を変換する場合、型変換の関数を使い、型変換することを明示しましょう。

　Excel VBAには型変換のための関数があります。以下のように使います

 i = CLng(x) ' x をLong型に変換

 a = CDbl(y) ' y をDouble型に変換

 s = CStr(a) ' a をString型に変換

　CStrのよく出てくる使用例は、Range命令でセルを指定するときに以下のように指定する方法です。

 Range("A" + CStr(i))

　+ は文字列を連結します。CStrを怠ると、文字列と数値を接続できないのでエラーが発生します。CStrによって数値を文字列に変換します。

　この場合はRange("A" & i) という別のシンプルな書き方があります。どちらを使うべきかは議論の余地がありますが、私はタイプ量の少ない Range("A" & i) を使っています。

変数の初期値

　これまではVariant型変数のみを使ってきました。Variant型変数の初期値は空[[2]](#footnote-2)でした。本節で学習した数値のみを入れることができる変数（Long, Double）の初期値は0です。Stringは "" です。

通常は使わない型について

　Integer型（短整数型：2 byte使用）を使うことは滅多にありませんが、整数定数の奇妙な振る舞いを理解するのに必要なので、説明します。

 Dim i As Integer

と宣言すると、iはInteger型です。2 byte使います。2 byte = 16 bitなので表せる数の種類は216です。-32768～+32767という非常に狭い範囲の整数しか入れることができません。すぐにオーバーフローするので、通常は使いません。範囲外の数値を代入しようとすると、オーバーフローエラーが発生します。たとえば、

 i = 40000

あるいは

 i = 32767

 i = i + 1

を実行すると、図5.2に示すオーバーフローエラーが発生します。



図5.2　オーバーフローエラー

　Excel VBAが開発されたのは1994年であり、その頃のパソコンは16 bitのOSであるMS-DOSで動作していました。当時のパソコンのC言語で

 int a;

と変数宣言すると、2 byte使用する短整数型になりました。2 byte整数は16 bit CPUが使われていた時代の遺物です。

定数

 a = 256 \* 256

というプログラムを実行すると、図5.2と同一のエラーが発生します。これは256がInteger型の定数であり、Integer型×Integer型の結果はInteger型になるのですが、それが-32768～+32767の範囲を超えるのでエラーが発生しています。これを回避するには、以下のように書きます。

 a = 256& \* 256

　256& は整数256をLong型として扱うことを意味しています。演算をするとき、その結果は精度が高い方に合わせられるので、256 \* 256 のうち、少なくともどちらか片方に & を付けておくと、オーバーフローエラーは起こりません。

【余談の知識】変数を宣言せずに使用する

　本テキストではモジュールの先頭にOption Explicitを入れて変数宣言を強制する方法を採用しています。Option Explicitを省略すると、宣言せずに変数を使用することができます。一見、便利なようですが、落とし穴があります。以下のプログラムを見て下さい。

 tensuu = Range("A" & i)

 If tensu >= 80 Then

 hyouka = "A"

 （以下略）

　2行目はタイプミスしてtensuuとすべきところtensuとしています。tensuは初期値である "" が入っているので、Ifのどの条件にも一致しません。変数の宣言を省略することは、このように、変数名のタイプミスが深刻なバグを引き起こします。

　Option Explicitは通常は省略してはいけませんが、数行の使い捨てプログラムを作るときなどは、Option Explicitなしでプログラムを組んでしまうかも知れません。このときに知っておくべきことを以下に書きます。

　変数名としてsumと入力すると、Sが大文字に自動的に変換されてしまいます。Excelにおいて既にSumという変数が使われているとき、先頭が大文字になります。Add, Averate, Count, Max, Min, Name, Pi, Rowなどの変数名も先頭が大文字になるので、Excelが既に使っているようです。

　Colorという変数は定数として宣言されているようです。Colorに値を代入しようとすると、エラーが発生します。

　Dim Abcと宣言した後、abc = 10と入力すると、変数名が自動的にAbcに直されます。Excelは変数名として大文字と小文字は区別しません。しかし、宣言時と同一の大文字小文字になるようにExcel VBAエディタが自動的に変換します。ゆえに、変数名の先頭が大文字になるとき、その変数はExcelが既に使っていると思われます。

　Excelが既に使っている変数を我々が使っても通常は大丈夫ですが、悪影響が出るケースがあります。プログラムは標準モジュールだけでなく、Sheet1(シート名) にも書くことができます。Sheet1(シート名) の中でName = "a" とすると、シートの名前が "a" に変わります。

　なお、標準モジュールに書いたプログラムからシートの名前を操作するときは、以下のように3通りの書き方があります。

 ActiveSheet.Name = "a"

 Sheets(1).Name = "a" ' 1 はシートの番号

 Sheets("シート名").Name = "a"

　標準モジュールに書いたコードは全てのシートを操作できるのに対して、Sheet1(シート名)に書いたコードは、そのシート内だけしか操作することができません。

　Excelではややこしいことが起こります。

 Dim name

と書いてEnterを押すと、Excel の内部変数の名前が Name → name と書き換わるようです。Dim name の定義文を削除しても、以降そのファイルにおいて name という変数名を使うと、全て小文字のnameにExcel VBAエディタが自動変換します。そして、Sheet1(シート名) の中で

 name = "b"

を実行すると、シート名が "b" に変わります。

【余談の知識】色々なプログラミング言語における変数宣言

　プログラミング言語には変数宣言を強制するものと、そうでないものがあります。C, C++, Javaは変数宣言を強制します。Pythonは宣言不要です。ただしPythonでは変数aに値を代入していない状態で

 b = a

と書くとエラーが発生します。

　Excel VBA, Fortranはデフォルトで宣言不要です。これらの設計が古い言語は、代入する前に、変数の値を引用した場合、デフォルトの値が使われます。変数名のタイプミスが深刻なバグの原因となるので、Excelの場合はOption Explicitと書くことで変数宣言を強要します。Fortranの場合も同様にファイル先頭にimplicit noneと書くことで変数宣言を強要できます。

## セル、変数の中のデータの型

　セル、変数の中に入っているデータが「数値」なのか「文字列」なのかを調べる方法があります。以下のように書きます。

 Dim a, b, c, d

 a = 10

 b = 100000

 c = 10.5

 d = "10"

 Debug.Print TypeName(a) & " " & TypeName(b) & " " & TypeName(c)

 Debug.Print TypeName(d)

 Range("A1") = 10

 Range("A2") = "10"

 Debug.Print TypeName(Range("A1").Value) & " " & \_

 TypeName(Range("A2").Value)

　TypeNameを使うと変数に入っているデータの型を調べることができます。a はInteger（2バイト整数）, bはLong（4バイト整数）, c はDouble（倍精度実数）, dはString（文字列）となります。Variant型変数の中に入っているデータは、数値が入っている場合、その値を表すために最低限必要なバイト数を持つ型を返すようです。

　セルの中はどちらもDoubleとなっています。Range("A2") = "10" は実行時に自動的な型変換が行われ、文字列 "10" 数値10に変換されてからセルに代入されたようです。ただし、セルの表示形式が「G/標準」のときは自動的な型変換が行われますが、「文字列」に設定されているときは、String型になります。このあたりのExcel VBAのふるまいは大変複雑であり、予期しないバグを引き起こす可能性があります。Excel VBAの設計が古いので、やむをえないことでしょう。セルA1は表示形式が「G/標準」「文字列」のどちらの場合でも、Doubleになります。

　TypeNameの中はRange("a1").Valueと書いています。.Valueを省略するとデータの型はRangeとなってしまいます。本書ではセルに代入するとき、Range("a1") = 10のように書いてきましたが、これは省略形です。省略せずに書くと

 Range("a1").Value = 10

です。Range("a1")はオブジェクトであり、

 Range("a1").Interior.Color = RGB(255, 204, 255)

のように、Rangeの後はピリオドの後、プロパティを書きます。プロパティを省略すると、「.Value」がデフォルト値として用いられます。

　VarTypeという関数もあります。VarType(a) とすると、変数aの型を調べることが出来ます。返り値は数値で返ってきます。

|  |  |
| --- | --- |
| 返り値 | 型 |
| 0 | Variant |
| 2 | Integer |
| 3 | Long |
| 4 | Single |
| 5 | Double |
| 8 | String |
| 9 | Object |

## 小数の扱い

　コンピューターは2進数で数値を保持しています。2進数の小数について考察します。

|  |  |
| --- | --- |
| 2進数 | 対応する10進数 |
| 100 10 1 0.1 0.01 0.001 |  4 2 1 0.5 0.25 0.125 |

　表のように、2進数の小数点以下の位は、10進換算すると1/2の位、1/4の位、1/8の位‥‥‥というようになっています。小数点第n位の重みは2-nです。10進数の0.1は、2進数では0.000110011001100....という0011を繰り返す循環小数です。

　1/3を10進数で表すと0.333333...です。3回足し合わせると0.999999...となり、1にはなりません（iPhoneの電卓では1÷3×1は1になります。どうやっているのでしょうか‥‥‥）。一方で、3進数なら1/3は0.1です。3進数の計算では0.1 + 0.1 + 0.1 = 1となり、3回足し合わせると1になります。

　これと同じことが起こるので、Variant型変数やDouble型変数に0.1を代入すると、内部表現は2進数なので、循環小数となり、丸め（切り捨てor切り上げ）が起こります。ゆえに、0.1を10回足しても1にはなりません。Excel 2016の場合、セルに関しても同様です（昔のバージョンのExcelはセルの中の小数は10進数で保持していたように思うのですが‥‥‥）。

　しかし、セル関数のIFとExcel VBAのIfは動作が異なります。例えば、A1～A10に0.1を入れて、A11を =SUM(A1:A10) とします。セル関数

 =IF(A11=1,"○","×")

は "○" を返します。一方、Excel VBAで

 If Range("a11") = 1 Then

 Debug.Print "○"

 Else

 Debug.Print "×"

 End If

を実行すると "×" が表示されます。セル関数のIFは「非常に近い数値のとき、= が成立するとみなす」という判断をするようです。

　Excelの小数の扱いは難解です。「実数を扱うときは要注意。10進数とは異なる」と覚えて下さい。

## プログラムからプログラムを呼ぶ

　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

　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。sub2を以下のように変更してみましょう。

Sub sub2(b)

 Debug.Print "1: a = " & b

End Sub

　イミディエイトウィンドウの表示は変わりません。sub2で宣言する引数の名前は、sub1における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。

　ややこしい例を一つあげます。

Sub sub1()

 Dim a, b

 a = 10

 b = 20

 Call sub2(a, b)

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

End Sub

Sub sub2(b, a)

 Debug.Print "1: b = " & b & " a = " & a

 b = 100

 a = 200

End Sub

　名前とは関係なく、順番に結合されるので、sub2のbとsub1のaは同一であり、sub2のaはsub1のbと同一です。

　引数を使って値のやりとりをする場合、以下のように使います。

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。ここではsub2はcを受け取って2倍してdとして返しています。すなわち、sub2は入力引数を2倍する関数です。

（重要！）

　ここまでの例では、Sub2では引数の型宣言を省略しています。型宣言を省略した場合、呼び出し元の型が適用されます。上の例ではcは呼び出し元ではa, dは呼び出し元ではbです。aとbはどちらもVariant型ですから、cとdはVariant型です。

　引数c, dの型を宣言するときは

sub2(c As Long, d As Double)

のように宣言します。型宣言をした場合、呼び出し元の変数の型と一致させる必要があります。呼び出し元と呼び出し先の変数の型が一致しないときは、図5.3のエラーが発生します。



図5.3　引数の型が一致しない場合のエラー

　以下のように書くと、図5.4のエラーが発生します。

Sub sub2(c, d)

 Dim c, d



図5.4　変数を重複して宣言した場合のエラー

　c, dは引数として扱う変数であるにもかかわらず、sub2の中で新規に使う変数としても宣言しているため、バッティングしています。

　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。

　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」で実行できましたが、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。

【課題1】

　シート「プロシージャ課題1」を使います。A1～A5に数値が入っています。1からその数値までの和を求めてB1～B5に書き込むプログラムを作ります。数値nを与えたとき、1～nまでの和を返す部分はプロシージャwaとして独立させることにしました。それ以外の部分を以下のように作りました。

Sub procedure\_kadai1()

 Dim i, a, b

 For i = 1 To 5

 a = Range("A" & i)

 Call wa(a, b)

 Range("B" & i) = b

 Next i

End Sub

　Sub wa(n, sum) を完成させなさい。ただし、和を求めるときは公式n \* (n + 1) / 2を使わずに、Forループを使いなさい。

【課題2】

　組み合わせの個数を表す$\_{m}C\_{n}$は以下の公式で表されます。

$$\_{m}C\_{n}=\frac{m!}{n!\left(m-n\right)!}$$

　mとnが与えられたとき、組み合わせ$\_{m}C\_{n}$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) としています。Sub 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

## 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, a

 a = 1

 For i = 1 To n

 a = a \* i

 Next i

 myfact = a

End Function

　以下のように利用します。

Sub func\_check()

 a = 5

 b = myfact(a)

 Debug.Print b

End Sub

　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。

　関数の返り値の型を宣言する場合、

Function myfact(n) As Long

のように宣言します。プロシージャと同様に、関数内で引数nの型は呼び出し元の変数の型になります。引数nの型を

 Function myfact(n as Long) As Long

のように宣言すると、呼び出し元のaはVariant型、呼び出し先のnはLong型なので「ByRef　引数の型が一致しません」というエラーになります。呼び出し元で

 Dim a As Long

と宣言して、呼び出し元と呼び出し先（関数）で引数の型を一致させる必要があります。

≪注意≫

　C言語のように、関数の呼び出し元において、

 Dim myfact As Long

のように、関数の型を宣言すると「配列がありません」というエラーになります。呼び出し元における宣言はしてはいけません。

≪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

でもいけますが、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")

【課題3】

　入力は1～5を仮定する。1→2, 2→3, 3→4, 4→1, 5→5という変換を行い、その結果を返す関数を作りなさい。1～5以外の数値が入ったときは-1を返します。

1. 2進数で0.1は循環小数です。ゆえに、0.1を10回足すと、1にはならず、1からわずかに外れた値となります。0.1ずつ変化する数値を得たいときは、この例のようにカウントは整数で行い、丸め誤差が蓄積されないようにします。 [↑](#footnote-ref-1)
2. Variant型の初期値は厳密にはEmptyです。しかし、If文でEmptyと "" を比較すると「等しい」と判定されるので、Variant型の初期値を "" とみなしても差し支えないと思います。 [↑](#footnote-ref-2)