●「自作関数」の構造の大雑把な説明とその重要性について

関数の構造

「関数」の名前。
カッコの前に書きます。
※「変数」のような役目も果たします。

「関数」に適用する「条件」。「引数（引数リスト）」と呼びます。
※このように「関数」には、必ず、
『カッコで囲まれた部分＝条件＝言わば 引数の集合体』があります。

```
Function　z　（ x As Long ,　y As Long ）　As Long

　　Let　z　= x + y

End　Function
```

このあいだに、処理したい内容を書きます。

『 " 関数 " の答え＝" 関数 " がユーザーに投げ返すデータ』、の、そのデータ型の設定。

第1引数。
および、その設定。
（＝「変数宣言」の一種）
この例では、ここに書かれた「x 」が第1引数、となります。

第2引数。
および、その設定。（＝これも「変数宣言」の一種）
この場合、カンマが1つだけなので、「第2引数までしかない・・・」、ということになります。
つまり、3引数は存在しません。
この例では、ここに書かれた「y 」が第2引数、となります。

「関数の名前」は、変数のような役目も兼務していて、ユーザーに投げ返すための「関数の答え」を設定するには、この例のように、
「関数の名前」に対し、投げ返したい値（答え）を「＝（イコール）」で代入します。
つまり、『関数の名前になんらかの値を 代入した』・・・、＝「ユーザーに答えを返した」、＝「ユーザーに答えを戻した」、・・・という意味になるわけです。
もう少し違う言い方で言うと、「返す」という「動き」を、この1文（1行）の命令で「実現した」ということになります。

現実には、多くの場合、ここには　こんなたったの1行ではなくて、複数行の命令文を書くことになるのですが、複数行の命令たちを書いたときは、その一番最後に、この形態の1文を書きます。（「代入ステートメント」と呼ばれる、「命令文の形態」のうちの1つ、です。）

※注意・・・・・・関数名への代入の際にはこの1文には、例えばデータを、「一般データとして値を返したい」という場合は、「Let」を書くか、もしくは それを省略してもOKですが、「オブジェクトとして返したい」という場合は、「Set」を必ず先頭に書きます。（「Set」は省略を許されていないので省略するとエラーになってしまうためです。
「Let」は、「省略を許されている」ので、エラーになりません。そして「ほとんどの場合」、省略されます（ただ、そもそも・それが間違いのもとの一つですが・・・。VBAは「省略」が多すぎるために「理解のし間違い」が多くなり、初心者や独学者は本当に苦労してしまいます）。
※なお、1行目のカッコの右側には、「返したい一般データやオブジェクト」の、各々に対応した変数設定も必要です。）

※ワークシート関数として使えるかどうかのテスト。Long を String に変えるとどういう現象が起こるか？ 数字、漢字、
※プログラミングとは、どんな言語でも、「変数」を「動かす＝操作する」ことで、「自動化を実現」する作業のこと。これが超基本。

自作関数の重要性

「自作関数」にこそが、脱初心者のための、最も重要な基本です。
理由は、「自作関数」が作れると、「部品化」（＝共用部品の作成）ができるので、効率化が進むからと、
「返す」「渡す」という仕組みをより理解でき、特に、同じように「返す」という仕組みを持つ、
「プロパティ、メソッド、関数、その他」といった命令たちへの理解も進むからです。
そのおかげで、ヘルプ、オブジェクトブラウザ、ウォッチウィンドウも読みやすくなります。

特に、「オブジェクトを返す色んなタイプの自作関数が思い通りに作れること」が重要で、
逆に、それが作れないと、100%、短期間のうちに挫折します。
（「同時に複数のファイルを扱う」とか、「同時に複数のシートを効率よく扱う」ということすらもできないため。）
理由は、「プロパティ」も「メソッド」も、「実は」、「オブジェクトを返す自作関数」とかなり似た仕組み・ルールで動いているからです。
それが理解できると、ヘルプとオブジェクトブラウザが更に読めるようになり、ある地点から、爆発的に上達します。

ひいては、「VBAの真の基礎」である、「クラスモジュール」への理解、にもつながります。
（結局のところ・・・、VBAは「クラスモジュール」というものが作れないと、本当のところは理解できません。
でも、クラスモジュールが分からなくても、「オブジェクトを返すタイプの自作関数」が思い通りに作れると、
「クラスモジュール」というものを理解する「助け」に、「すごく」なります。
結果、いつかはクラスモジュールの基礎が理解でき、そのおかげで、どこかの地点で、更に、飛躍的に上達できます。）

逆に、自作関数の仕組みが理解できないと、「返す」の意味すら良く分からず、ヘルプも何も読めないし、無駄なコードばかり書く羽目になってしまいます。
また、自作関数の仕組みが理解できないと、初心者本を終えた後、「直後」に、「大抵」挫折します。
理由は・・・、繰り返しになりますが、「返す」という仕組みが理解できないために、「渡す」も理解しづらく（特にオブジェクトの場合の「返す・渡す」）、
「型のリレー」が「正しくできずに」エラーを多発させますし、「部品化」も結局できないからです。
そして、「部品化」ができないと、プログラムを・・・、例えば、「おもちゃのブロックを組み替えるような」イメージで、
「共用部品を組み合わせたり・差し込んだり・差し替えるだけで目的の半分以上をラクに達成する・・・・」、という経験がまるでできません。
その体験ができないと、ものごとを「部品単位」「ブロック単位」で「少し大き目なくくりで」、「全体的な目で見る・考える」、ということができないので、
結局、**『「自分のしたい事」をするためには、「自分が何をどう作ったらいいのか？どう部品を作ったらいいか？」』など**がまるでイメージできません。
つまり、「自力での設計」がまったくできないままになってしまうのです。なので、「結局何もできないまま・・・」ということになってしまいます。
だからほとんどの人は、初心者本を終えたあと、「ランクが ほんの少し上がっただけの本」を読んだだけでも、
「自作関数が作れないがために」、「急に難しく感じて」、挫折してしまいます。

同じようなことですが、独学者や初心者が100%挫折する理由のひとつは・・・
→ Amazonでもその超重要な、ExcelVBAの「ちゃんとした自作関数（＝ユーザー定義関数）」の本が1冊もヒットしないため。です。
中でも重要な、「オブジェクトを返す自作関数」の本は、一冊も売られていません。Amazonですら・・・。
挫折するのも「当然」といえます。

自作関数の各部の意味

※自作関数＝正式には「ユーザー定義関数」と呼ぶようです。

```
Function　　z　（ x As Long ,　y As Long ）　As Long

　　Let　　z　　= x + y

End　Function
```

「自作関数」では、基本、「 Function 」の行 ～「 End Function 」の行の間に、実行したい処理内容を書きます。
（「 Function 」～「 End Function 」の部分が、「 Sub 」～「 End Sub 」の場合もあります。一般的にはSubで始まるものは 「関数ではない」
という考え方ですが、まだ初心者のうちはあえて「自作関数」に含めて考えてみます。より詳しく理解できるようになったとき、使い分けます。
（https://pcacademy.jp/vba_04_2019　を参照）

で、ここでは「Let z = x + y」の1行が主な命令文、ということになります。（通常は複数行の命令文の集まりになります。）
「 z 」の部分（つまり、カッコの直前）は、自作関数の「名前」です。この関数は、「 z という名前の自作関数（Functionプロシージャ）である。」といえます。
なので、『関数「z」は、内部で「Let z = x + y」という計算をやっている・・・』といえます。
あるいは、逆から見ると
『「Let z = x + y」という面倒な計算を、「z」と 1文字書くだけで代用・呼出し・実行・答えをもらうことができてしまう！便利！』、ともいえます。

また、「自作関数」というものには、必ずカッコで囲まれた部分があります。カッコのない関数は存在しません。
この例だと1行目の「Function　z ・・・」のあとのカッコです。
（実は、自作関数だけでなく、すべてのプログラム＝プロシージャにはカッコが存在します。）

カッコの中、つまり、「（x As Long, y As Long）　」の部分は、自作関数「 z 」に適用する、ある意味「条件」です。
そしてその条件のことを「引数リスト」と呼びます。「引数」は、「ひきすう」と読みます。
「引数」は、カッコの中においては、1個だけのこともあれば、複数個・存在する場合もあります。
そして、その「条件（引数リスト）」の中においては、カンマで区切られるごとに、「第1引数」、「第2引数」、「第3引数」、・・・・となっていきます。

で、たとえば、「第1引数」と呼ぶ場合は、「ひとつめの条件値」という感じの意味です。
「第2引数」は、「2つめの条件値」という感じの意味です。
「第3引数」は、「3つめの条件値」という感じの意味です。
上記の例では、カンマが1つだけなので、存在するのは 第2引数までで、つまり、「条件値は2つ」という意味になります。第3引数はありません。
でも、もしカンマが2つあれば、第3引数までが存在する（＝条件値は3つ）・・・という意味になります。
逆に言うと、『 作りたい引数（条件値）の数だけ、カンマで区切ればいい・・・ 』ということでもあります。

また、同時に、「 x 」と「 y 」は、「変数」と呼ばれるモノでもあります。中学の数学で習う「変数」と同じです。（小学校の算数では「未知数」と習ったかも？）
ただ、中学の数学で習う「変数」は、数字しか代入できませんが、プログラミングの場合の変数は、
数値、文字列、日付データ、True/False(Yes/Noのような意味)なども代入できます。
前述の「引数」の設定で、データの型を宣言することで、それが可能となります。

一般的に、「変数の宣言（＝変数の設定）」は、「 Function 」の行 ～「 End Funcion 」の行の間で行うことが多いのですが、
この例のように、条件設定（引数リストにおける引数の設定）の部分でも行えます。また、そうすることも結構多いです。
それぞれの「やり方」の「意味」は異なるのですが、いずれにしても、両方ともの やり方を、確実に覚えないと、
100%、初心者本を実践し終えた時点ですぐに、「挫折」します。
自分のしたいことが、「どうやったら実現できるか」や「効率化」ができなくて。
なお、上記の例の場合、
「 x As Long 」は、1つ目の引数の設定であり、と同時に、「 x 」という変数の変数宣言（＝データ型の指定）をもしています。
「 y As Long 」は、2つ目の引数の設定であり、と同時に、「 y 」という変数の変数宣言（＝同上）をもしています。
「 x As Long 」は、「1つ目の引数を、" x "という名前のLong型の変数のかたちで用意しとくね！」という意味になります。
「 y As Long 」は、「2つ目の引数を、" y "という名前のLong型の変数のかたちで用意しとくね！」という意味になります。
（「Long型」とは、−2,147,483,648 ～ 2,147,483,647 の整数 を扱いたいときに使います。「長整数型」ともいいます。
その範囲をはみ出た整数、つまり、−2,147,483,649以下や、＋の 2,147,483,648 以上の数を扱おうとすると、エラーになります。
カンタンに言うと、「 プラスの21億強～マイナスの21億の整数を扱うだけなら Long型の変数をつかえばOK！ 」ということになります。）

そして、もし、「（x As Long, y As Long）　」の部分が無かったとしたら、「 z As Long 」となりますが、
これは、『 関数 z のデータは、Long型のデータとしてユーザーに答えを返しますよ～！ 』という意味になります。
（※ Long型 ＝　つまり、「 −2,147,483,648 から 2,147,483,647 までの整数、として返しますよ～ 」、という意味になります。）

「 z = 」の「 = 」は、「等しい」という意味ではなくて「代入しろ」という「命令」です。
「代入」とは、小中学校で習った、「 x に 1 を代入する 」、とか、「 y に 2 を代入する 」、とかの「代入」、と同じ意味の「代入」です。

もちろん中学校の数学では、「＝」は「等しい」という意味だったですが・・・。
一応、VBAでも「 ＝ 」は、「比較の意味でのイコール」というケースのときもありますが、「代入する」という「命令」である場合も結構多いです。
なお、「 ＝ 」のような「記号めいた命令」のことを「演算子」と呼びます。「+、−、*(×)、/(÷)」も演算子です。
「演算子」は、その右や左に「引数めいたモノ」をもち、それらの値を材料に、計算や比較などをして、ユーザーに答えを返してくれる「命令の一種」です。

ちなみに「 ＝ 」が、「比較の意味でのイコール」となるのは、「 ＝ 」の存在する行の先頭に、「If」「Select」「For」「Do」などのような、
「ステートメント名（＝特にフロー制御系のステートメント用のキーワード）」と呼ばれる、命令語句の一種があったときです。
命令文の先頭に「Set」が書かれている場合と、「Let」が書かれている場合は、「代入しろ」という命令の意味になります。
ただ、「Set」は省略されません（できません）が、「Let」は基本、省略されるので、最初のうちはわかりづらいかもしれません。
いきなり「変数名」から始まって、その次がすぐに「＝」だったら、確実に「Let」が省略されています。結果、「＝」は「代入しろ」という命令です。
「 ＝ 」の存在する行の先頭に、「If」「Select」「For」「Do」などのような命令たちが「無い」場合も同様です。

● VBAにおける「データ＝ " 値 " 」と、「一般データ」と、「オブジェクト」について

Excelに限らず、VBAの場合、「データ＝ " 値 " 」には、大別すると、「一般データ」と「オブジェクト」の「2つ しか」あります。
「関数」がユーザーに対して「返す」「値」についても同様で、大別すると、その「2つしか」ありません。

そして、「一般データ」とは、「文字列、数値、日付、TrueやFalseなどの論理値（＝ブール値）」などの「文字系のデータ」のことをいいます。

「オブジェクト」とは、「装置」、「ユニット」、「マシン」、「ロボット」、的なイメージのモノのことをいいます。その内部には、「プロパティ、メソッド、イベント」と呼ばれる命令群（＝機能）を「メンバー（一員）」として含んでいます。そんなイメージです。（＝内包・保持・所有、しているイメージです。）
プロパティやメソッドは、いわば、「特定のオブジェクトの上でしか動けない」、下記の(01)～(05)のような「自作関数と同じ動きをするもの」です。

● 「超」重要なこと

あと、「超」重要なこととして、「自作関数」には、大別して、以下の5つのタイプがあります。
（「引数や既定値がある／なし」などを含めて。）

(01) 一般データを返すこと「だけ」、しかしないタイプの自作関数
(02) 何らかの動きをしつつ、同時に、一般データをも返すタイプの自作関数
(03) オブジェクトを返すこと「だけ」、しかしないタイプの自作関数
(04) 何らかの動きをしつつ、同時に、オブジェクトをも返すタイプの自作関数
(05) 何も返さずに、何らかの動きをする、だけのタイプの自作関数。

※ ここで言う、「何らかの動き」とは、
「一般データ操作の場合、主に、計算、抽出、切り出し、変換、比較、結合、分離、etc. ・・・ といった処理（＝動き）のこと」をさし、
「オブジェクト操作の場合、主に、開く／閉じる、追加、削除、編集、フィルタ、並べ替え、外部データ接続と吸込み、グラフ作成、メッセージ表示、etc. ・・・ といった処理（＝動き）のこと」 をさします。

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

そして、VBAの命令は、演算子、各種VBA関数、WorksheetFunctionオブジェクトのメソッド、プロパティ、メソッド、ステートメント名、API、など、色んなカテゴリに分かれていて、合計で（Excelの場合だけでも）、少なくとも1000個以上の命令単語がありますが、実は・・・、それらの1000個以上の命令たちは全部、上記の(01)～(05)の、5つの自作関数と同じ、あるいは似た動きをします。

「演算子、関数、プロパティ、メソッド、云々…」といったカテゴリ名、カテゴリ分けも、まあ、確かに大事といえば大事ですが、でも、「過度」にそのカテゴリ分けに「惑わされないで」ください。

その呼び名、カテゴリ分けの堀根・枠を飛び越えて、1000個以上の命令たちはすべて、この「5つのどれか」に当てはまります。

命令を使うシーン、命令の書き方のルール（命令の構造）、操作する対象物、などが異なるので呼び名やカテゴリが異なるだけで、「動きの本質はすべて同じ」です。
引数、あるいは引数めいたもの、等々が「有る／無い」はありますが、すべて、冒頭の(01)～(05)の動きの「どれか」に「あてはまります」。
「その動きの本質」は、1000個以上の命令たち、全部、「同じ」です。
（ちなみに「書き方と呼び出し方のルール」だけに「限って言えば、それが異なるのは「演算子とステートメント名だけ」です。あとは各種関数も、プロティも、WorksheetFunctionオブジェクトのメソッドも、通常のメソッドも、自作のそれらも、「全部」、「同じ」です。）

また、そう考えたほうが、VBA自体の構造やクラスモジュールの仕組みを理解しやすくなると思います。

（※ちなみに(02)は、「自作のワークシート関数」として、セルの数式の中でも使えるようになります。）

逆に言うと、「自作関数」のことが理解できて、
「オブジェクトを返すタイプの自作関数が思い通りに作れれば」、
「VBAの1000個以上の命令を全部理解できる」、
または、
「その素地ができる」、
ということになります。

だからこそ、「自作関数」が、「重要」なのです。

逆に、「自作関数」のことをやらなければ、1000個以上の命令の事どころか、「数個の命令の動き」すら、ちゃんと理解できません。（ついでに、「式」という重要な単位・考え方のこともわかりません。）

もちろん、そんなことでは、応用なんて利くわけがありません。

挫折して「当然」ということになります。

結果、「数個の命令の動きすら、ちゃんと理解ができなければ」、
初心者用の本を何冊マスターしたところで、
また、お金を払って習ったところで、
結局、
「即そのあとに挫折」
「自力で何の設計、組み立てもできない」
「自力でエラー解決なんてできるわけない」
「Q&Aサイトで質問しても回答者の回答の意味がわからない」
となってしまいます。
そしてその原因は、すべて、
この(01)～(05)の自作関数のことを、
「全くやらないから」、
です。

そして、そのことを、Excelが発売されてから 20年間以上、どのレジェンド講師や著者や出版社も、まともに、書籍では教えてきませんでした。
（隠してわざと教えずカモってきたのか？と勘繰りたくなるほどです。）

Amazonでも、VBAの自作関数の「ちゃんとした」本は、
「1冊も」、
「ありません」。

さらに、

「挫折して当然」といえます。