LibreOffice　Excel_VBA→Libre_Basic マクロ移行マニュアル

※ 目次下線部をクリックすると該当ページへ移動します目次へ戻る場合は各ページのページ番号をクリックして下さい
※IPA フォントを利用しています、IPA フォント以外では表示が崩れる可能性があります。

～目次～

【実践編】 ページ

★ 本マニュアルは「マクロの記録」で作成した手続きレベルの Excel_VBA を手早く Libre_Basic で動作させることを目標にしています。（Function プロシージャなどについては解説しておりません）

～ マクロについて ～

【マクロとは？】
オフィスソフト（ワープロ・表計算）で特定の操作や手順をプログラムとして記述し、操作を自動化する機能のことです

【マクロ言語とは？】
自動化プログラムの記述に使う言語のことで、LibreOffice では Libre_Basic が MicrosoftOffice では VBA がそれぞれマクロ言語として採用されています。（LibreOffice では Javascript や Python でも記述可能です）

【マクロを使うことのメリットとは？】
- 「印刷」や「並び替え」等よく使う処理をマクロとして登録しておけば、Calc の操作に不慣れな人でも簡単に「印刷」や「並び替え」を実行できるようになります。
- 定型作業をマクロに任せることが可能となり、ミス低減と作業効率向上が期待できます。

【マクロを使う際に注意すべき点とは？】
- 仕様書等が正しく整備されていない場合、マクロ作成者しか処理内容が判らないブラックボックス化された状態に陥りやすい（エラーや不正値出力の原因となります）。
- 拡張性が低いマクロの場合、例外データ等に対応できず、かえって作業効率が低下します。

※マクロ機能はメリットと注意すべき点を考慮した上で便利に利用しましょう。

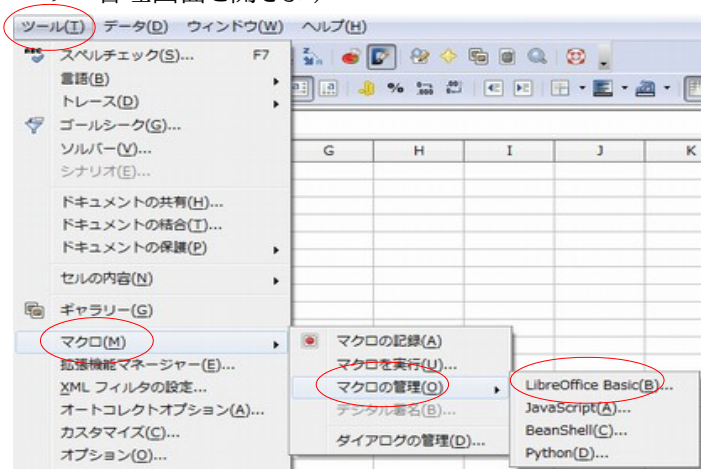
【ありがとうございます】
本マニュアルで利用している LibreBasic コードは、LibreBasic/OpenBasic 解説 Web サイトの掲載情報を参考に適宜修正・追加し作成しました。
Web ページ作成者の皆様にこの場を借りて御礼申し上げます。
参考 URL
http://hermione.s41.xrea.com/pukiwiki/pukiwiki.php?OOoBasic
http://openoffice3.web.fc2.com/OOoBasic_Calc.html
http://calibreblo.blogspot.jp/2011/04/blog-post_10.html

Libre_Basic

1. VBA から Libre_Basic への移行で事前に知っておくことは？

- Calc にはマイマクロとシートマクロの 2 種類のマクロがあります
  - マイマクロは PC に保存されるマクロで、自分の PC で使用する Ods ファイル全部で利用したいマクロを登録します。（自分だけが使うマクロを登録するイメージです）
  - シートマクロはシートに保存されるマクロで、その Ods ファイルを開く全ての PC で実行したいマクロを登録します。（Excel の VBA マクロはシートマクロを使って移行します）
    ※「Excel マクロを移行する場合はシートマクロを利用する」と覚えましょう。

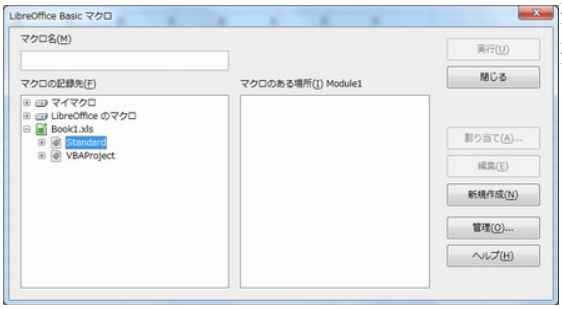
- マクロ管理画面を開きます



ツール → マクロ → マクロの管理 → Libreoffice Basic

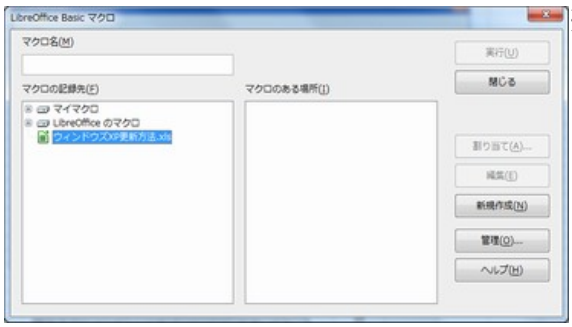
- マクロ画面の比較

VBA マクロ有り Excel シートを読み込んだ場合



※Standard と VBAProject が登録されています
※Calc 上で VBA マクロを操作･編集する場合は VBAProject 内の VBA モジュールを利用します

VBA マクロ無し Excel シートを読み込んだ場合

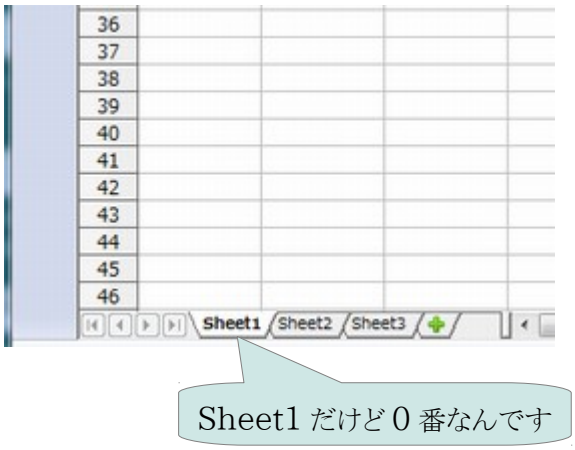


※Standard などモジュール登録箇所が一つもありません

（VBA を使わない「完全移行」を行う場合、この状態からスタートする必要があります）

- シート番号とセル番号の開始がゼロスタート

プログラム上のシート番号やセル座標がゼロから始まります



シート番号について
Sheet1 をプログラムで指定する際に利用する
シートNo.指定では 0 で指定します

※気づいた点※
一番左にあるシートをシート 0 と認識するようです、例えば Sheet2 を Sheet1 の左に移動させると、Sheet2 が 0 番シートになります。

※「いちばん左のシートがシート 0」って覚えましょう!

	A	B	C
1	A1	B1	C1
2	A2	B2	C2
3	A3	B3	C3
4	A4	B4	C4
5	A5	B5	C5
6	A6	B6	C6
7	A7	B7	C7

A1 セルのセル座標は(0,0)
B1 セルのセル座標は(1,0)
D6 セルのセル座標は(3,5)
ゼロから開始される点が Excel と違います、また VBA では(行,列)で指定しましたが、CalcBasic では(列,行)で指定するようになっている点に注意が必要です。

- このマニュアルではシート指定をシート番号で行っています、したがってシート位置の変更に伴いシート番号が変更され、処理エラーとなる可能性があります、シート名による指定に変更したい場合は適宜、モジュール本体の書き変えとパラメーターの型を変更して下さい。

【現行】
シート番号で指定している例(SheetNO は数値型の変数です)
例 1:ThisComponent.Sheets(SheetNO)

例 2:ThisComponent.Sheets.getByIndex(SheetNO)

【変更例】
シート名で指定した例(SheetMei は文字列型の変数です)
ThisComponent.Sheets.getByName(SheetMei)

2. VBA マクロを Libre_Basic マクロへ移行するための事前準備

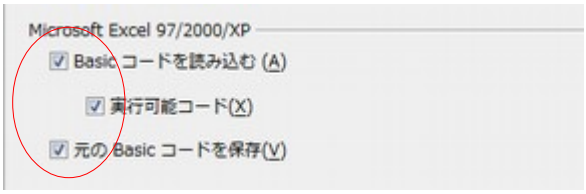
VBA マクロのコードを保持したままでは Libre_Basic で記述したマクロの一部でエラーが発生します。
ここでは、VBA マクロが有る Excel オブジェクトの中から Sheet オブジェクトだけを Calc に移し、マクロは Libre_Basic を使う手順をご案内します。

【効率よく作業するために】
マクロ移行作業は VBA コードを参照したり、コピー貼り付けする作業を多用しますので、Excel と Calc の両方がインストールされた PC で作業する方が効率的です。

【作業】

- Calc の標準設定では自動的に VBA コードも一緒に読み込みます、Excel シートを読み込む前に Calc の読み込み設定を変更し VBA マクロを読み込まない設定に変更します。

① 手順(この設定変更は VBA マクロ付き該当ファイルを読み込む**前**に実施します)
新規ファイルを開きます → メニュー → ツール → オプション → 読み込みと保存 → VBA 属性

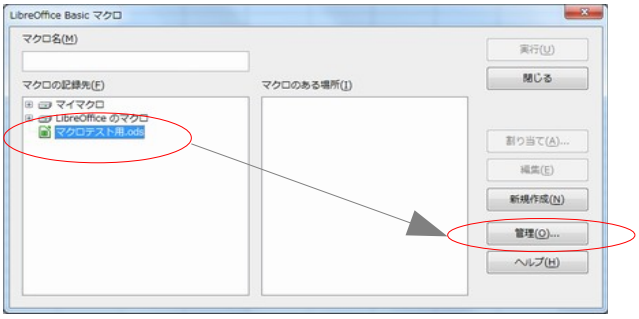


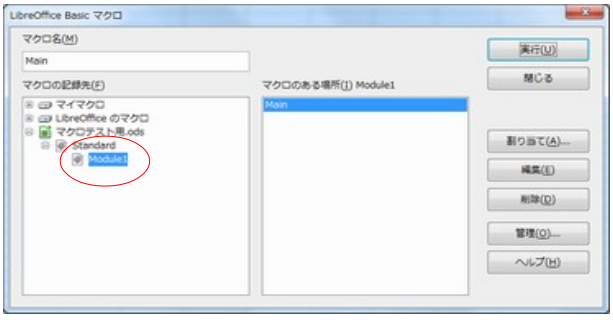
○ 枠内のチェックを外します

② OK をクリックします(これで VBA マクロを除く Sheet だけを取り込むことが可能となりました)
③ 該当のマクロ付き Excel シートを Calc で読み込みます。
④ 必ず ODF 形式(拡張子 ODS)で保存します。
※マクロ管理画面を開き、Standard や VBAProject が登録されていないことを確認します

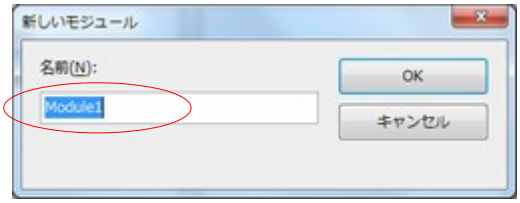
- Libre_Basic マクロの保存先を作成します
※新規に ods ファイルを作成した場合は自動的に Standard が作成されます、Standard を選択し、その配下に新規作成しますので注意してください。

① ツール → マクロ → マクロの管理 → LibreOffice Basic で管理ダイアログが表示されます
② マクロの記録先(F)でマクロを保存する ods ファイルを選択した後、新規作成(N)をクリックします



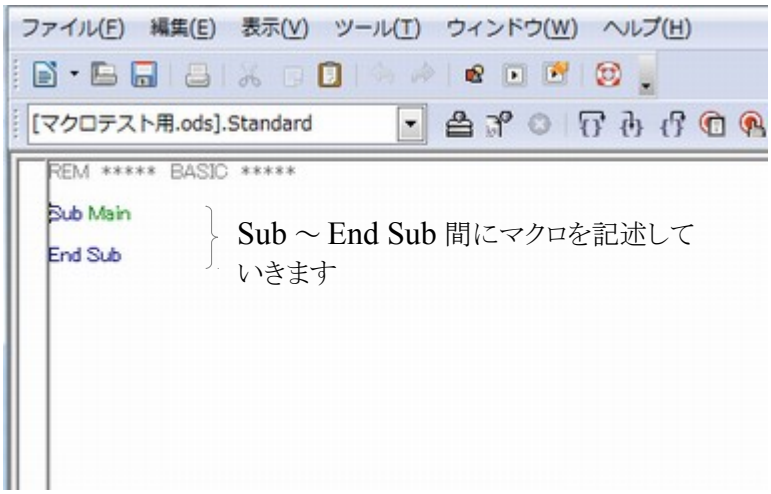


③ 新しいモジュール画面になります
名前は「Module1」のままでも変更してもかまいません。
OK をクリックします



④ Standard の下に Module1 が作成されました
マクロの記述は Standard 以下にある Module1 の中に記述していきます。

⑤ モジュールを操作する画面が起動しますので、マクロを記述していきます



あらかじめ以下の例が記述されています

```
Sub Main

End Sub
```

この例をそのまま利用しても、消去して新しく命令を記述してもかまいません。

プロシージャは VBA のように

```
Private Sub test1()
End Sub
```

と記述しても大丈夫です。

LibreBasic には Private や Public 等の区分が存在しない為、Private や Public などを記述していても無視され、全て Public として動作するようです。(Libre3.4.4 時点)

※Private や Public 区分を登録しているとコード判読時にわかりやすいです。

⑥ コード記述時に注意すること(VBA Editor と比較してみます):2012/04 時点

a) End Sub・End Function を自分で入力する必要があります
(VBA の場合は自動挿入されます)

b) 全角スペースがそのまま入り実行時エラーになりますので、意識して半角スペースを入れる必要があります(VBA の場合は半角スペースに自動変換されます)、If 文などのインデント下げにはスペースを使わず TAB キーを使って調整する方法をお勧めします。

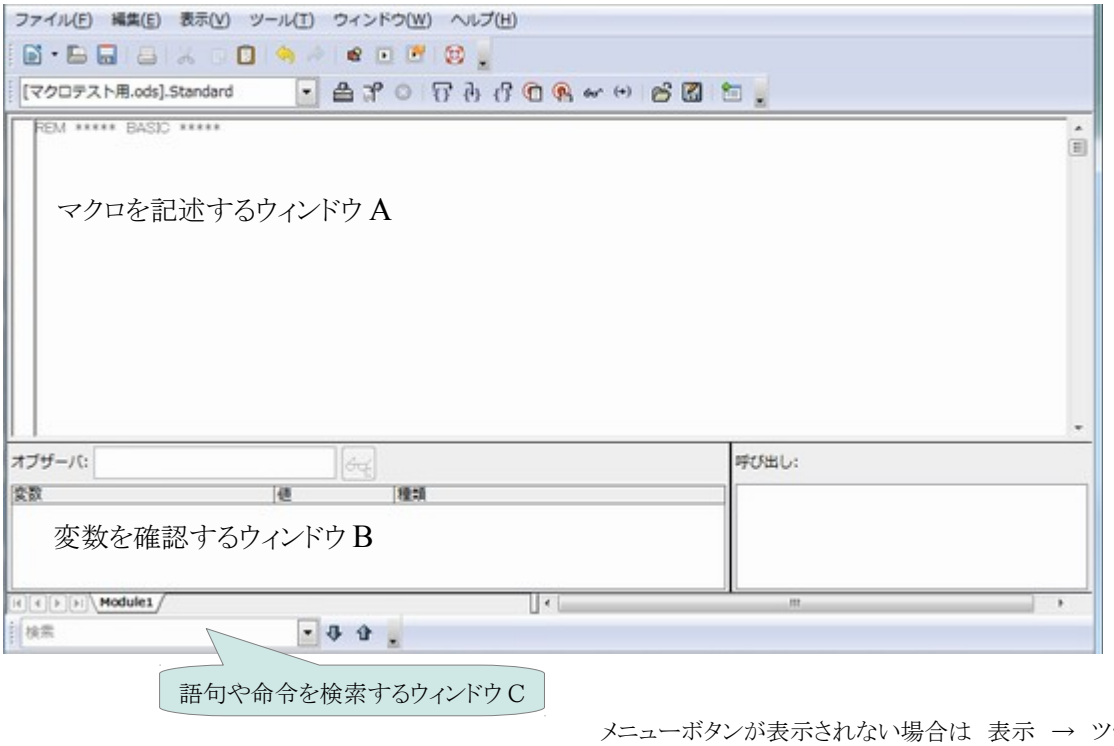
c) 構文ミスがあっても、そのまま記述できてしまいますので、コード実行時まで構文エラーが判りません。
(VBA の場合、行単位でコンパイルエラーメッセージが表示されます)

d) Basic コードを保存するシートオブジェクトがありません、作成した Module に保存するのみです。
(VBA の場合、シートオブジェクトと標準モジュールにマクロを分散して保存できました)

以上で Libre_Basic マクロに移行する準備が整いました。

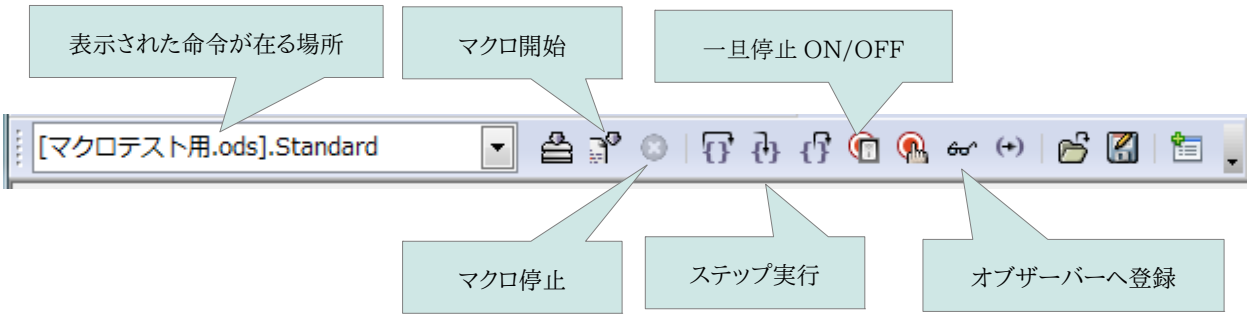
3. Libre_Basic 操作画面で知っておいて欲しいこと

① マクロを記述する画面です
ツール → マクロ → マクロの管理 → Libreoffice Basic → ファイル名 → Standard → 該当モジュール名(Module1 等)



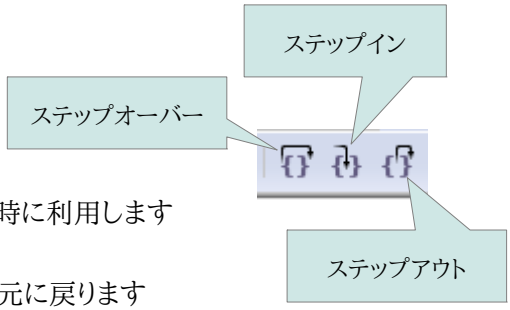
メニューボタンが表示されない場合は 表示 → ツールバー(T)
→マクロ(D)にチェックを付けてください

② 通常利用するコマンドメニューボタン



通常使うボタンとして上記の内容を把握しておきましょう。
マウスで変数をドラッグして選択し、メガネアイコンの「オブザーバー登録」ボタンをクリックすると、変数確認用のウィンドウ B に登録され、変数に代入される値の確認が容易になります。

③ ステップ実行ボタン

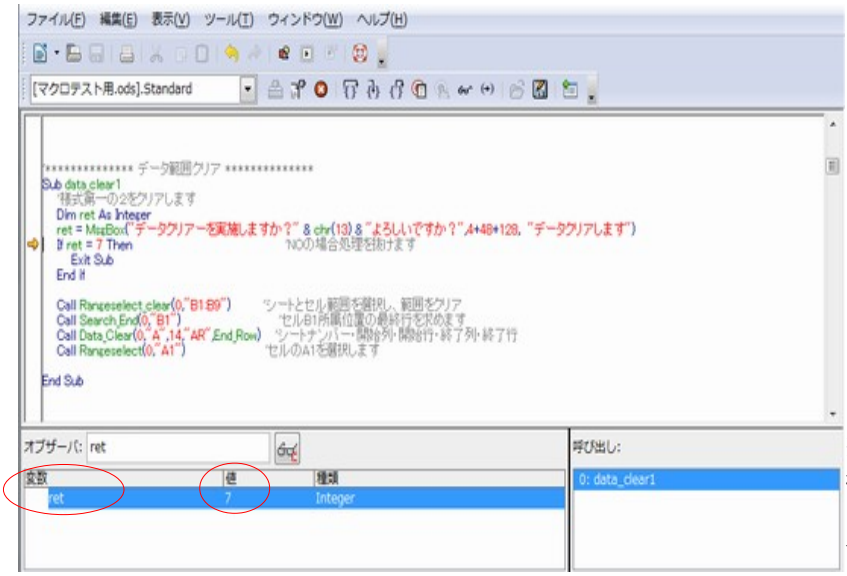


ステップ実行ボタンも非常によく使いますので、基本的な動作を把握しておきましょう。
ステップオーバー:呼び出し命令をスキップ(跳ばす)時に利用します
ステップイン:クリック毎に命令を 1 行ずつ実行します
ステップアウト:呼び出された命令を抜けて、呼び出し元に戻ります
※簡単なマクロで利用するのは通常ステップインだけです

④ マクロ記述後に行う動作確認作業の流れ

a) マクロを記述します
b) ステップインを使って１行毎に動作をチェックします
c) チェック終了後、マクロ開始ボタンで実行してみます
d) エラーが出たらエラー箇所を修正し、再度 a～c を実行します

⑤ 実際の画面での動作を見てみます

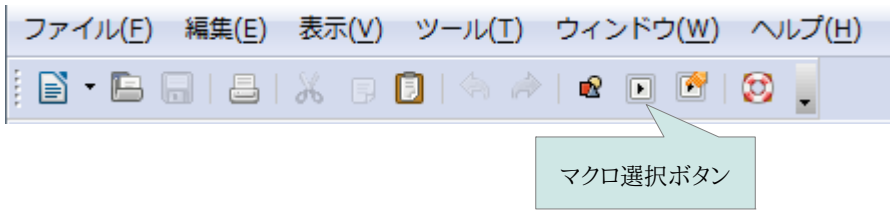


a) まず実行したい命令（モジュール）にカーソルを移動させます
b) ステップインをクリックして処理を開始します
黄色の→が実行行に表示されます
c) ステップインをクリックする毎に処理が進んでいきます

※左の例ではオブザーバーに変数 ret を登録しています（赤枠）、MsgBox で「いいえ」を選択した為、ret に 7 が表示されています

※マクロ実行中は「マクロ停止ボタン」 が有効になります、途中で停止したいときはクリックしてください。

【ワンポイント!】



登録したマクロが多くなってくるとマクロを探すのに一苦労します、「マクロ選択ボタン」を使うとマクロの一覧が表示されますので一発で該当マクロを見つけることができて便利です。

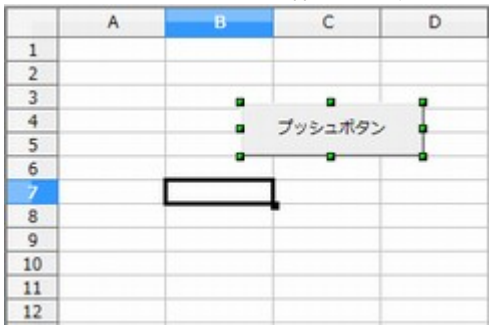
「VBA のようにプロシージャを一発選択できるウィンドウが欲しいところです」

4. ボタンクリックでマクロが動くようにするには？

シート上に配置したボタンクリックでボタンに登録されているマクロが動作するようにするには、シート上にボタンを配置し、配置したボタンに実行したいマクロを割り当てる必要があります。

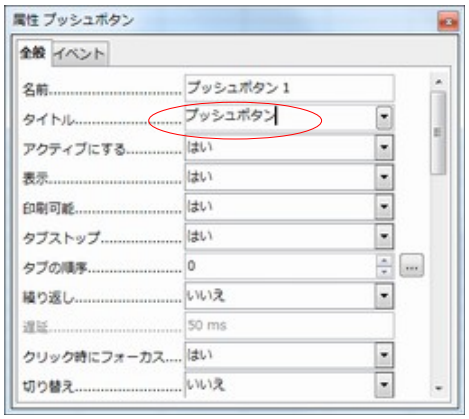
① マクロボタンを作成してみましょう
表示 → ツールバー → フォームコントロール　にチェックが付いていない場合はチェックを付けます
フォームコントロールバーが表示されますので、OK マークのボタンを選択します。

② ワークシート上にボタンを作成します



ワークシート上にカーソルを移動し、ドラッグしたままマウスを動かしてボタンを描画します。

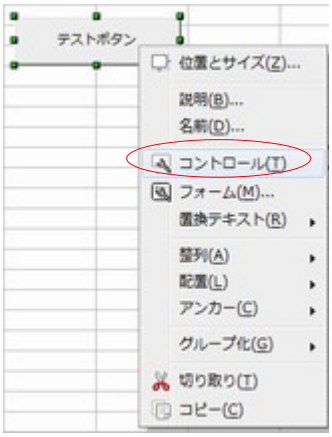
③ ボタンの表示名を変更します（表示名を変える場合は名前ではなくタイトルを変更します）

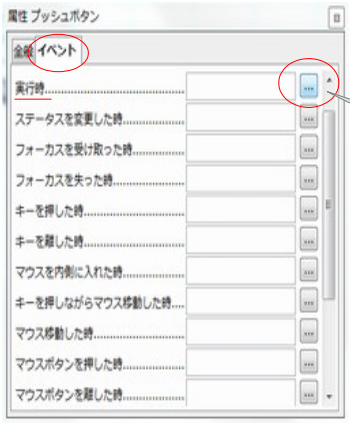


ボタン上で右クリック　→　コントロール(T)
全般タグのタイトルを「テストボタン」に修正します

※
「名前」はプログラム上での呼び名と考えてください、左の例ではプッシュボタン1が名前になります。

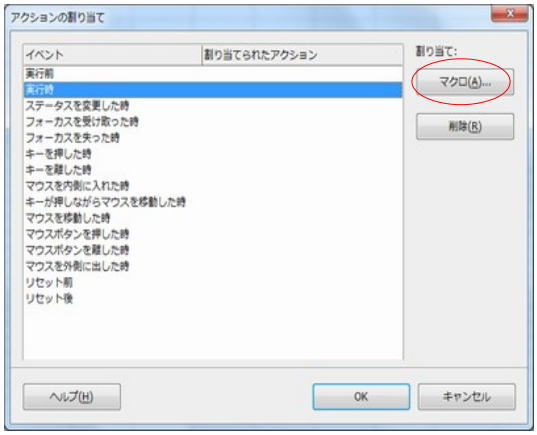
④ 作成したボタンにマクロを割り当て関連付けします
ボタン上で右クリック　→　コントロール(T)　→　イベントタブを選択　→　実行時の横にある四角ボタンをクリックします



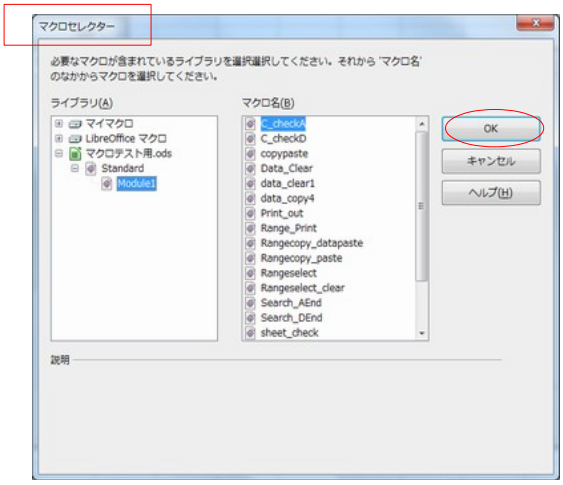


ココをクリックします

マクロボタンを右クリックできない場合は次ページ注 1 を参照してください

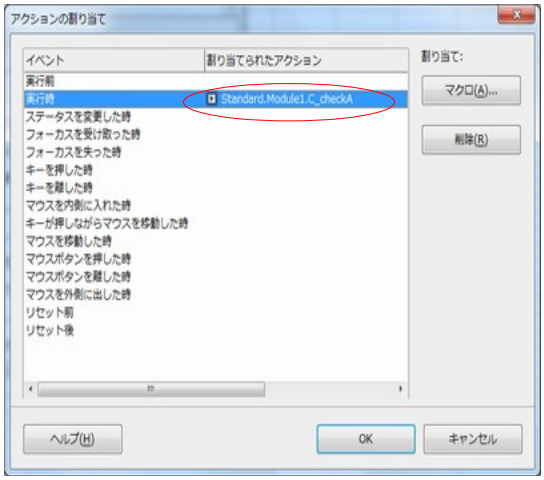


⑤ 実行時が選択されている状態のまま、マクロ(A)ボタンをクリックすると、マクロセレクターが表示されますので、「実行時」アクションに割り当てするマクロを選択します。



⑥ 例では、Module1 内にある C_checkA マクロを選択しています

選択後「OK」をクリックします



⑦ アクションが割り当てられました

以上でマクロの割り当ては終了です

注 1)マクロボタンを右クリックしても何も表示されない場合

デザインモードが OFF になっています、ボタンの設定変更等を行う場合はデザインモードを ON にするようにしましょう。　　↓の画像は OFF の状態です

ココをクリックしてオン・オフを切り替えます

マクロを実行する際はデザインモードを OFF にしてください。

5. メッセージボックスの利用と設定(改行・戻り値など)

マクロではメッセージを表示する、メッセージボックスを良く利用します、ここではメッセージボックスの設定について説明します。

- メッセージボックスで利用する各種パラメータ

◎表示するボタンを指定

0	- OK ボタンのみ表示。
1	- OK およびキャンセルの各ボタンを表示。
2	- 中止、やり直し、および取消しボタンを表示。
3	- はい、いいえ、キャンセルの各ボタンを表示。
4	- はい、いいえの各ボタンを表示。
5	- やり直しおよびキャンセルの各ボタンを表示。

◎標準(あらかじめ選択状態で表示する)ボタン設定

0	- 1 番目のボタンを標準ボタンに設定。
256	- 2 番目のボタンを標準ボタンに設定。
512	- 3 番目のボタンを標準ボタンに設定。

◎表示アイコン設定

16	- ストップ(進入禁止?)記号のアイコンを表示。
32	- 疑問符アイコンを表示。
48	- 感嘆符アイコンを表示。
64	- ヒント記号(電球?)のアイコンを表示。
	(64 を指定すると何故か全て 0 の「OK ボタンのみ表示」になってしまうようです)

パラメーターの設定方法は以下の設定例を参考にしてください
以下の命令を実行します

```
MsgBox "処理を続けますか?", 4+0+48 ,"確認しています"
```

合計値の 52 とだけ記述しても OK ですが 3 つに分散して記述した方が判りやすいです

パラメータ　4+0+48　の意味

はい・いいえ & 1番目ボタンを標準選択 & 感嘆符アイコン

メッセージボックスに複数のボタンを表示した場合、通常は、どのボタンが選択されたのかを確認する処理が必要となります。
このような処理には、次の戻り値を利用します。

◎戻り値の種別（どのボタンが押されたのかを判別する戻り値）

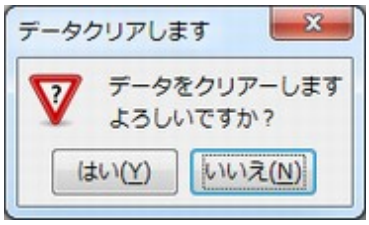
1　- OK
2　- キャンセル
3　- 終了
4　- やり直し
5　- 無視する記号・文字
6　- はい
7　- いいえ

◎戻り値を使った Msgbox 利用例

```
Dim modorichi As Integer
modorichi = MsgBox("データをクリアーします" & chr(13) & "よろしいですか?",4+256+32,  "データクリアします")
If modorichi = 7 Then        '「いいえ」が押された場合処理を抜けます
        Exit Sub
End if
```

改行文字

コメント文※1



Msg ボックス内で改行する場合は　chr(13)　を使います
※ExcelVBA では vbCrLf を使っていました

※1【コメント文について】
半角文字のシングルコーテーション'を付けて記述した部分はコメント文と認識され、プログラム実行時には無視されます。
処理の説明や注意点などをコメントとして記録しておくと後でプログラム内容が判りやすくなります。

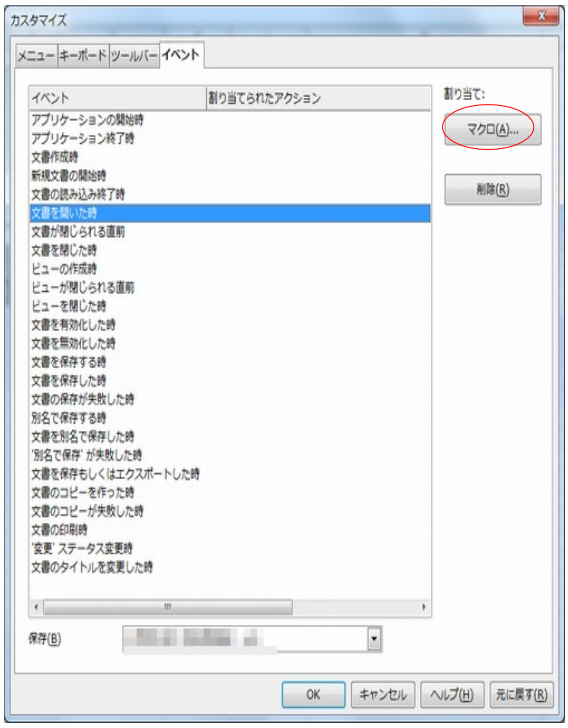
※全角スペースに注意!!
記述したマクロの構文中に全角スペースが入っているとエラーになります
構文に問題が無いのにエラーになる場合は該当行に全角スペースが入っていないか?を一度チェックしてみて下さい。
インデント（段ずらし）にはスペースで調整せず TAB キーを使うようにしましょう

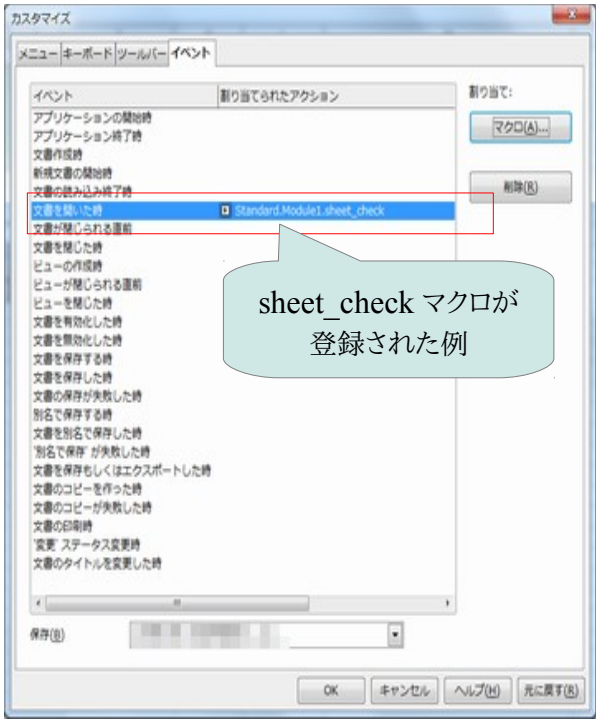
6. シートを開くと同時にマクロを動作させるには？

Excel_VBA では Auto_Open と言うマクロ名で記述していた処理です。
Calc では、処理させたいマクロをツール → カスタマイズ → イベントタブにある、「文書を開いた時」イベントに登録することでシートを開くと同時に実行させることができます。

【手順】

① 処理したいマクロを作成する（プロシージャ名は何でも OK）
② ツール → カスタマイズ → イベントタブ内にある「文書を開いた時」イベントにマクロを登録します





※「文書を開いた時」の他にも多数のイベントが登録されていますので必要に応じて使い分け可能です。

ワンポイント!

処理したい命令（プログラム）が、きっかけとなるイベントの発生によって実行されることをイベントドリブンと言い、イベントで処理が開始されるプログラムをイベントドリブン型プログラムと言います。
イベントドリブン型プログラム作成では、自分が実行したい命令を、どのイベントで実行させるのが一番良いのか?を正しく把握しておくことが重要です。

7. メインルーチンとサブルーチンとは？　パラメーターって？

【解説】

メインルーチンやサブルーチン等ありますが、**そもそもルーチンとは何でしょう?**

ルーチンとは、決まりきった手続きや手順を意味しています。

普段の生活でも日常の決まり仕事や日課などを「ルーチンワーク」などと呼んでいますよね?

プログラムでは、プログラム中のひとまとまりの機能をもつ命令群のことをルーチンと呼んでいます。

主に最初に実行されプログラム全体の進行を管理するルーチンを「メインルーチン」、プログラムの実行中に他のルーチンから呼び出されて動作するルーチンを「サブルーチン」と呼んでいます。

なぜルーチンを切り分ける必要があるのでしょう?

汎用性の高いコードをサブルーチンとして切り出すことにより、同じコードを何度も記述することが不要となり、コード量の削減や開発効率の向上、不具合発生率の低減などが見込めるからなのです。

もう一つ、サブルーチン処理に必要な事柄としてパラメーターがあります、**パラメーターとは**サブルーチンを呼び出す際に**「動作を指定する為に設定する値の事」**で、この値を変化させることで処理の対象やデータ範囲などを変化させることが可能となります。

☆ 本マニュアルでは、マクロで処理したい作業をメインルーチンとして記述し、プログラムをサブルーチンとして呼び出すことで作業が完了するようにしています。

※マクロ記述例

下記はメインルーチン sheet_check から、サブルーチン Rangeselect をパラメーター（シートNo.とデータ範囲）指定して呼び出しています

```
Sub sheet_check()
'シート番号とセル位置を指定し実行すると、該当シートに飛びます
        Call  Rangeselect(0,"A1")        'Sheet1 の A1 を選択する処理
        Call  Rangeselect(1,"A2")        'Sheet2 の A2 を選択する処理
        Call  Rangeselect(2,"A3")        'Sheet3 の A3 を選択する処理
End Sub

Sub Rangeselect(SheetNo as integer,DataArea as string)
'*************************************************************
'* 機能      : データセルを選択します
'* パラメーター  :   シートNo. , データ範囲
'*************************************************************
        Dim LController As Object
        Dim LSheet As Object
                LController = ThisComponent.getCurrentController()
                LSheet = ThisComponent.Sheets(SheetNo)

                LController.select (LSheet.getCellRangeByName(DataArea))   'アクティブなセルを選択
End Sub
```

メインルーチン

0,”A1” 等がパラメーターです

Call で呼び出す

サブルーチン

上記の例では、パラメーターとしてシートNo.と選択セル座標を指定するだけでセルを選択する処理が完了します。
パラメーター指定しサブルーチンを呼び出すことで、処理記述が少なくて済み、処理内容も把握しやすくなっています

8. 変数と定数について

マクロを記述する際に登場するものに変数と定数があります。
本マニュアルでは変数を使った構文等の作成を行いませんので、詳しい説明は割愛いたしますが、マクロを理解する上で最低限必要と思われる知識をお知らせします。

変数とは?

計算結果や一時的に値などを保存するのに利用する入れ物をイメージしてください。
この入れ物は何を収納するのか?によって利用型を宣言して使います。
代表的な型として

整数型(Integer)　　-32768 ～ 32768
長整数型(Long)　　-2147483648 ～ 2147483647
単精度浮動小数型(Single)　3.402823 × 10E38 ～ 1.401298 × 10E-45
通貨型(Currency)　　通貨単位
文字列型(String)　　文字列
日付型(Date)　　日付
真偽型(Boolean)　　TRUE (-1) か FALSE (0)
オブジェクト型(Object)
変化型(Variant)

使い方
使う前に宣言して使う方法が一般的で
命令作成時にパラメーター格納用として利用したり

```
Sub Rangeselect(SheetNo as integer,DataArea as string)
```

SheetNO は整数　　DataArea は文字列　として宣言しています

処理の戻り値を入れる時に利用します

```
Dim modorichi As Integer　　'modorichiを整数型として宣言
modorichi = MsgBox("データをクリアーします?",4+256+32, "データクリア")
```

変数 modorichi に MsgBox でどのボタンが押されたか?の結果を格納しています

変数の前についている Dim って?

いまから変数を宣言します!という接頭語だと思って利用してください
「変数を宣言する前には Dim を付ける」と覚えましょう

定数とは?

定数とは、一度しか値を決められない定義名のようなものです
あらかじめ値が決定しているものについて定義します

CONST 定数名 = 値　という形で定義します
例えば円周率のように 3.14 で運用することが決定している場合

```
Const Pai = 3.14
```

このように宣言して利用します

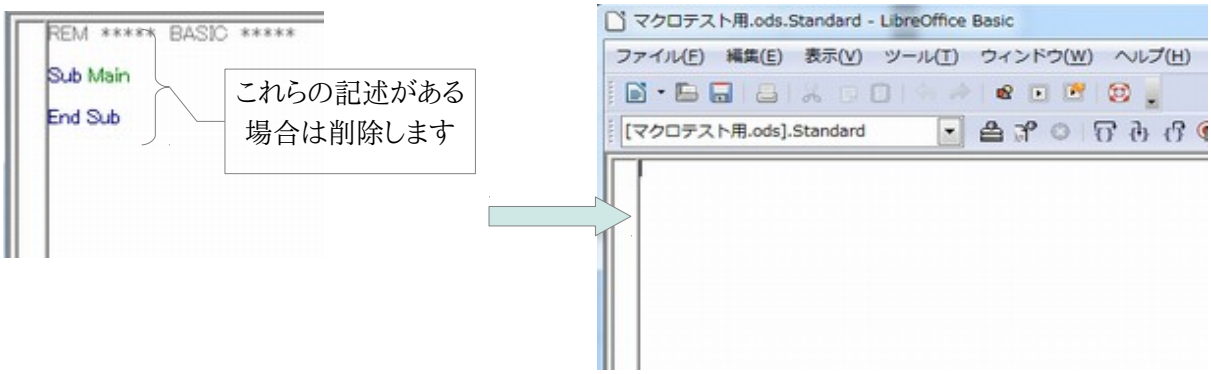
興味がある方はサブルーチンを見て、変数がどのように利用されているかを確認してください。

実践編

9. 事前処理：サブルーチンを使えるように登録します

マクロ処理をメインルーチンに登録する前に、あらかじめ作成しているサブルーチンリストをモジュール内にコピーして貼り付けします。
この作業を行うことで、サブルーチン呼び出しが可能となります。

① Calc マクロを登録する ods シートを開き、本マニュアルの「2. VBA マクロを Libre_Basic マクロへ移行するための事前準備」を実施後、Module を開いてマクロを記述する画面にします



② 別掲載しているファイル：LibreMacro.txt　をメモ帳で開き、内容を全てコピーします

メニュー　→　編集　→　すべて選択(A)
→　編集　→　コピー　します

③ マウスカーソルを Calc の Module 画面に移動させ貼り付けします

編集　→　貼り付け(P)

メインルーチン記述エリア

メインルーチン記述エリアに、処理マクロを記述していきます。

サブルーチンエリア内のモジュールは基本的に変更しませんが、必要な場合には修正可能です

10.メインルーチンマクロを作成してみましょう

メインルーチンマクロを作成してみます、このマニュアルで言うメインルーチンマクロとは「メッセージを表示する」「セルを選択する」「並べ替える」「印刷する」などのアクションの事です。

マクロ作成には「9. 事前処理：サブルーチンを使えるように登録します」の作業完了が必須条件です、作業未完了の場合は処理エラーになりますのでご注意ください。

例として以下の処理をマクロ化してみます

シート番号 0 のセル A1 を選択　→　A1：C19 をコピーして E1 に貼り付け　→　C 列を昇順で並び替え　→ A1：G19 を範囲指定して A4 横で印刷

上記例では、大きく分けて4つのアクションマクロを作成することになります。

```
'######　マクロ例　######
Sub macrotest()

        Call Rangeselect(0,"A1")                    'シート№0 のセル A1 を選択
        Call Rangecopy_paste(0,0,"A1:C19","E1")     'A1：C19 をコピーして E1 に貼り付け
        Call Sort_pro(0,2,"A1:G19",true,1)          'C 列を昇順で並び替え
        Call Range_Print(0,"A1:G19",2,1)            'A1：G19 を範囲指定して A4 横で印刷

End sub
```

パラメータ指定したサブルーチンを呼び出しています(Call 以下の部分）

通常であれば、途中に MsgBox で確認したり、Loop 処理によるデータ加工などを設定し、処理を進めていきます。

※　マクロ内容解説　※

```
Sub macrotest()
```

マクロ名は任意で OK

```
        Call Rangeselect(0,"A1")                    'シート№0 のセル A1 を選択
```

パラメータでシート 0 と A1 をセット

パラメータ：（選択シート番号，選択セル）

```
        Call Rangecopy_paste(0,0,"A1:C19","E1")     'A1：C19 をコピーして E1 に貼り付け
```

パラメータ：（コピー元シート番号，貼り付け先シート番号，コピー範囲，貼り付けセル）

```
        Call Sort_pro(0,2,"A1:G19",true,1)          'C 列を昇順で並び替え
```

パラメータ：（シート番号，並び替え列，ソート範囲，並び順，見出し行有無）

```
        Call Range_Print(0,"A1:G19",2,1)            'A1：G19 を範囲指定して A4 横で印刷
```

パラメータ：（シート番号, 印刷範囲，用紙方向，用紙サイズ）

```
End sub
```

各サブルーチンに関する詳細は次ページ以降に掲載していますのでご確認願います。

★マクロを記述するには?★

本 PDF マニュアルのサブルーチン呼び出し例をコピー貼り付けしパラメーターを適宜修正してください

11.指定範囲のデータをクリアー(消去)するには?(範囲可変設定可能)

- サブルーチン名
  Data_Clear(SheetNo as integer,Start_retsu as string,Start_gyou as long,End_retsu as string,End_gyou as long)
- 機能
  範囲指定してデータ消去(パラメータで指定してクリア可)
- 引数内容
  シート番号・開始列・開始行・終了列・終了行
- サブルーチン呼び出し例
  Call Data_Clear(0,"A",14,"AR",40)　'シート0のA14～AR40をクリアする例

12.範囲指定して印刷するには?(セル番地(A1～)で範囲指定)

- サブルーチン名
  Range_Print(SheetNo as integer,P_Area as string,Houkou as integer,Psize as integer)
- 機能
  範囲指定して印刷します(セル番地(A1～)で範囲指定)
- 引数内容
  シート番号, 印刷範囲, 用紙方向(1:縦 2:横),用紙サイズ(A4=1,A3=2)
- サブルーチン呼び出し例
  Call Range_Print(0,"A1:AR40",1,2)　　'シート0の範囲A1～AR40を縦書きA3サイズで印刷する例

注意:印刷するシートがアクティブであることが必要です、非アクティブシートを印刷する場合は事前に印刷予定シートをアクティブにして下さい。

例)シート0に設定したマクロボタンで非アクティブなシート1を印刷する場合

```
Call ActiveSheetNo(1)　　'まずシート1をアクティブにします
Call Range_Print(1,"A1:AR40",1,2)　　'次にシート1の範囲A1～AR40を縦書きA3サイズで印刷
```

13.範囲指定して印刷するには?(数値座標で範囲指定)

- サブルーチン名
  Print_out(SheetNo as integer,s_retu as integer,s_gyou as long,e_retu as integer,e_gyou as long,houkou as integer,Psize as integer)
- 機能
  範囲指定して印刷します(数値座標で範囲指定します)
- 引数内容
  シート番号 , 開始列 ,開始行, 終了列 ,終了行 , 用紙方向(1:縦 2:横),用紙サイズ(A4=1,A3=2)
- サブルーチン呼び出し例
  Call Print_out(0,0,0,2,39,1,1)　　'シート0の範囲A1～C40を縦書きA4サイズで印刷する例
  ※設定するスタイルによって用紙方向指示が無効になる事例が出ています、1シート1様式で印刷する場合は、あらかじめスタイル設定で印刷用紙や方向を決めていた方が無難です

注意:印刷するシートがアクティブであることが必要です、非アクティブシートを印刷する場合は事前に印刷予定シートをアクティブにして下さい。

例)シート0に設定したマクロボタンで非アクティブなシート1を印刷する場合

```
Call ActiveSheetNo(1)　　'まずシート1をアクティブにします
Call Print_out(1,0,0,2,39,1,1)　　'次にシート1の範囲A1～C40を縦書きA4サイズで印刷
```

14.データセルを選択するには？

- サブルーチン名
  Rangeselect(SheetNo as integer,DataArea as string)
- 機能
  データセルを選択します
- 引数内容
  シート番号，選択範囲
- サブルーチン呼び出し例
  Call Rangeselect(0,"A1")　　'シート０　の　A1 を選択
  シート番号が判らない時に利用すると、簡単にシートを番号特定できて便利です
  ※注意：指定したシートが非表示設定になっていると、**隣のシート**が選択されてしまいます！

15.セル範囲を指定して内容を消去するには？

- サブルーチン名
  Rangeselect_clear(SheetNo as integer,DataArea as string)
- 機能
  セル範囲を選択した後、内容を消去します
- 引数内容
  シート番号，選択範囲
- サブルーチン呼び出し例
  Call Rangeselect_clear(0,"A1:B5")　　'シート０の A1:B5 を消去します

16.アクティブ範囲の最終行を求めるには？（表組の最終行など）

- サブルーチン名
  Search_AEnd(SheetNo as integer,RetsuCell as string,bangoukbn as integer)
- 機能
  アクティブ範囲の最終行を返す（表組の最終行など）
- 引数内容
  シート番号,列座標『（例)A1』,戻り値番号区分（行番号:1　セル位置:2）
- サブルーチン呼び出し例
  Call Search_AEnd(0,"A1",1)　　'シート０の A 列を基準にアクティブ範囲の最終行を返す
  この処理を実行すると共通変数：End_Row に求めた最終行の値が格納されます

17.指定列データの最終行を求めるには？

- サブルーチン名
  Search_DEnd(SheetNo as integer,Col as integer,Bangoukbn as integer)
- 機能
  指定列データの最終行番号を取得(セルの行番号とセル位置を選択可)
- 引数内容
  シート番号, 列番号『（例)A=0』, 戻り値番号区分（行番号:1　セル位置:2）
- サブルーチン呼び出し例
  Call Search_DEnd(0,0,1)　'シート０の A 列にあるデータの最終行を行番号で返す
  この処理を実行すると共通変数：End_Row に求めた最終行の値が格納されます

※行番号とセル位置とは?(戻り値番号区分によって求められる値が変化します)
例)10行目までデータが登録されている場合
行番号:1 の場合＝10　　通常の10行目という位置(A10 など位置指定で利用)
セル位置:2 の場合＝9　　getCellByPosition(列番号, セル位置行)で指定する場合等に利用

- End_Row を利用したサブルーチン呼び出し例(可変範囲での印刷指示)

```
Call Search_DEnd(0,2,2)              'シート 0 の C 列にあるデータの最終行をセル位置行で返す
                                     '上記処理で End_Row に値がセットされます
Call Print_out(0,0,0,2,End_Row,1,1)      'シート 0 の範囲 A1～C のデータの最終行までを A4 縦で印刷する例
```

18.指定範囲をコピー貼り付けするには?

- サブルーチン名
  Rangecopy_paste(CSheetNo as integer,PSheetNo as integer,CopyArea as string,Pastecell as string)
- 機能
  指定範囲をコピーした後、指定セルに貼り付けます
- 引数内容
  コピー元シート番号, 貼り付け先シート番号, コピー範囲, 貼り付けセル
- サブルーチン呼び出し例

```
Call Rangecopy_paste(0,1,"A1:B9","A20")      'シート 0 の A1:B9 を選択後、シート 1 の A20 に貼り付けます
```

19.指定範囲をコピー貼り付けするには?(値貼り付け)

- サブルーチン名
  Rangecopy_datapaste(CSheetNo as integer,PSheetNo as integer,CopyArea as string,PasteArea as string)
- 機能
  指定範囲をコピーした後、指定セル範囲に値貼り付けします
- 引数内容
  コピー元シート番号, 貼り付け先シート番号, コピー範囲, 貼り付けセル
- サブルーチン呼び出し例

```
Call  Rangecopy_datapaste(0,1,"A1:B5","A1:B5")    'シート 0 の A1:B5 をシート 1 の A1:B5 に値貼り付けします
```

20.シートを表示したり、非表示にするには?

- サブルーチン名

```
SheetShow(sSheet as String)    'シート表示
SheetHide(sSheet as String)    'シート非表示
```

- 機能
  シートを表示したり、非表示にします
- 引数内容 (シート名は”で囲みます)
  シート名
- サブルーチン呼び出し例

```
Call SheetShow("九州")    'シート名「九州」のシートを表示します
Call SheetHide("福岡")    'シート名「福岡」のシートを非表示にします
```

21.シートの保護・解除を設定するには?

- サブルーチン名
  SheetProtect(SheetNo as integer,Pas as String,Kaijyokbn as integer)
- 機能
  シートの保護・解除
- 引数内容
  シート番号, パスワード(設定しない場合は""), 解除区分(設定:1/解除:2)
- サブルーチン呼び出し例
  Call SheetProtect(0,"",1)　'シート0の保護開始(パスワード無し)

22.指定された項目で並び替え(ソート)するには?

- サブルーチン名
  Sort_pro(SheetNo as integer,Sort_col as integer,Sort_area as string,Narabijun as string,Midashi as integer)
- 機能　(ソート項目が複数ある場合は26番:複数の項目で並び替えするには?を参考にしてください)
  指定された項目(1項目)で並び替えします
- 引数内容
  シート番号, 並び替え列(0～, ソート範囲(A1:B3形式), 並び順(昇順=True 降順=False), 見出し行有無(有り:1　無し:2)
- サブルーチン呼び出し例
  Call Sort_pro(0,0,"B1:C35",True,1)　'シート0のB1:C35の見出し有り 範囲をB列基準で昇順に並び替え

  ※並び替え行指定時の注意※(ソート範囲内の左端＝0)
  列番号　:ソート範囲中の列を左から順に列番号(0～)で設定します。
  例)ソート範囲(B3:E20)の場合、並び替え列 B=0,C=1,D=2,E=3
  　ソート範囲(C3:E20)の場合、並び替え列 C=0,D=1,E=2

23.行・列の挿入と削除

- サブルーチン名
  RC_sounyu(SheetNo as integer,RCNo as integer,Sounyusu as long,Gyouretukbn as integer)　　'行・列の挿入
  RC_sakujyo(SheetNo as integer,RCNo as integer,Sakujyosu as long,Gyouretukbn as integer)　　'行・列の削除
- 機能
  行・列の挿入と削除
- 引数内容
  挿入:シート番号,挿入開始行・列番号(1行目＝0、A列＝0),挿入する行数・列数,挿入する行列の区分(行＝1・列＝2)
  削除:シート番号,削除開始行.列番号(1行目＝0、A列＝0), 削除する行数・列数, 削除する行・列の区分(行＝1・列＝2)
- サブルーチン呼び出し例
  Call RC_sounyu(0,0,5,2)　　'シート0のA列から5列挿入します
  Call RC_sakujyo(0,0,8,1)　　'シート0の1行目から8行削除します

24.行・列の表示と非表示

- サブルーチン名
  RC_Visible(SheetNo as integer,RCNo as integer,Gyouretukbn as integer)　'行・列の表示
  RC_Hide(SheetNo as integer,RCNo as integer,Gyouretukbn as integer)　'行・列の非表示
- 機能
  行・列の表示と非表示
- 引数内容
  挿入:シート番号,表示行・列 No(1 行目＝0、A 列＝0), 行・列の区分(行＝1・列＝2)
  削除:シート番号,非表示行.列 No(1 行目＝0、A 列＝0),行・列の区分(行＝1・列＝2)
- サブルーチン呼び出し例
  Call RC_Visible(0,5,2)　'シート 0 の F 列目を表示します
  Call RC_Hide(0,8,1)　'シート 0 の 9 行目を非表示にします

25.オートフィルターを設定するには?

- サブルーチン名
  Auto_Filter()
- 機能
  指定範囲にオートフィルターを設定します(既にフィルター設定中の場合は設定解除になります)
- 引数内容
  なし
- サブルーチン呼び出し例
  Call Auto_Filter　'オートフィルターを設定します

  利用例)
  Call Rangeselect(0,"A2:D2")　'シート0の A2:D2 の範囲(データの項目欄を指定します)
  Call Auto_Filter　'上で指定した項目にオートフィルターを設定します

26.シートをアクティブにするには?(シート番号で指定)

- サブルーチン名
  ActiveSheetNo(sheetno)
- 機能
  指定番号のシートをアクティブにします
- 引数内容
  シート番号
- サブルーチン呼び出し例
  Call ActiveSheetNo(0)　'シート 0 をアクティブにします

'【シートをアクティブにするには?(シート名で指定)】
'VBA では　Worksheets("新規シート").Activate
'LibreBasic では?
ThisComponent.CurrentController.ActiveSheet = ThisComponent.Sheets.getByName("新規シート")

シート番号で指定する場合は「26. シートをアクティブにするには?」を参考にしてください

27.複数の項目で並び替えするには?(プロシージャ)

※以下のプロシージャの必要箇所を修正して実行してください

```
Sub Sample_Sort()

    Dim oObjSheet As Object
    Dim oObjRange As Object
    Dim aObjSortDesc(1) As New com.sun.star.beans.PropertyValue
    Dim aObjSortKeys(2) As New com.sun.star.util.SortField     'ソートフィールド用配列を3つ用意しています
    oObjSheet = ThisComponent.CurrentController.ActiveSheet
    oObjRange = oObjSheet.getCellRangeByName("A1:C14")

    '並べ替え時の第 1 優先キーは A 列。並び順は昇順
    aObjSortKeys(0).Field  = 0    'A 列
    aObjSortKeys(0).SortAscending  = True   '昇順     False 降順

    '並べ替え時の第 2 優先キーは B 列。並び順は昇順
    aObjSortKeys(1).Field  = 1   'B 列
    aObjSortKeys(1).SortAscending  = True   '昇順

    '並べ替え時の第 3 優先キーは C 列。並び順は昇順
    aObjSortKeys(2).Field  = 2   'C 列
    aObjSortKeys(2).SortAscending  = True   '昇順

    'ソートキー設定
    aObjSortDesc(0).Name = "SortFields"
    aObjSortDesc(0).Value = aObjSortKeys()

    'データに「行見出し」がない場合は  aObjSortDesc(1).Value = False
    aObjSortDesc(1).Name = "ContainsHeader"
    aObjSortDesc(1).Value = True        '行見出しがある場合は Ture

    '並べ替えの実行
    oObjRange.sort(aObjSortDesc())

End Sub
```

28.セルからの値取得や値代入など、よく使う命令(プロシージャ)

下記は記述方法の一例です。

※値取得する場合も代入する場合もデータに応じて string / value / Formula を使い分ける必要があります

文字列	string
数値	value
数式	Formula

'【セルから値取得】変数 hensu　に代入する例　hensu は宣言して利用しましょう

```
hensu = ThisComponent.Sheets.getByName("新規シート").getCellByPosition(7, 23).string
hensu = ThisComponent.Sheets.getByName("新規シート").getCellRangeByName( "B1" ).string
```

'【セルへの値代入】(プロシージャ)

'VBA では　Sheets("新規シート").Range("H24") = "H24 セル"
'LibreBasic では?セル番地で指定する時
ThisComponent.Sheets.getByName("新規シート").getCellRangeByName("H24").string = "H24 セル"
ThisComponent.Sheets.getByName("新規シート").getCellRangeByName("H25").value = 1234
ThisComponent.Sheets.getByName("新規シート").getCellRangeByName("H26").Formula = "=SUM(H24:H25)"

'Excel_VBA では　'Sheets("新規シート").Cells(24, 8) = "H24 セル"
'LibreBasic では? セル座標で指定する時 (Excel_VBA との相違点:行と列が逆で数が1つ少ない)
ThisComponent.Sheets.getByName("新規シート").getCellByPosition(7, 23).string = "H24 セル"
ThisComponent.Sheets.getByName("新規シート").getCellByPosition(7, 24).value = 1234
ThisComponent.Sheets.getByName("新規シート").getCellByPosition(7, 25).Formula = "=SUM(H24:H25)"

'【画面の更新を制御する】(ThisComponent なので今開いているファイルのみが対象です)**(プロシージャ)**

'画面更新を停止する
ThisComponent.LockControllers　　'VBA では Application.ScreenUpdating = False　'更新停止

'画面更新を再開する
ThisComponent.UnLockControllers　　'VBA では Application.ScreenUpdating = True　'更新再開

'【オートフィルターが設定されているか?をチェック】(プロシージャ)

'『データベース範囲の指定』で設定した範囲にオートフィルタが設定されているか?を判定
'ツール→「データ」→「範囲の指定」→「データベース範囲の指定」で範囲名を設定可能。(例では Range1)
'上記設定で、名前を付けたデータベース範囲でオートフィルタが有効にされているかどうか判定。

```
Sub Auto_Filter_check()
 Dim oDBArea as Object
        oDBArea = ThisComponent.DatabaseRanges.getByName("Range1")
        If oDBArea.AutoFilter Then
                MsgBox "オートフィルタ有効"
        Else
                 MsgBox "オートフィルタ無効"
        End If
End Sub
```

'【アクティブシートを指定部数印刷します】(用紙サイズ等を含め、現在の設定で印刷します)**(プロシージャ)**

'あらかじめページ設定で指定されている範囲を現在の印刷設定で指定部数印刷します
'モジュール

```
Sub ActiveSheet_Print(busuu as integer)
        Dim aPrinter(0) as object
        aPrinter(0) = new com.sun.star.beans.PropertyValue

                aPrinter(0).Name = "CopyCount"
                aPrinter(0).Value = busuu
                ThisComponent.print(aPrinter())
End Sub
```

'呼び出し例(現在のアクティブシートに設定されている印刷範囲を 2 部印刷します)

```
Call ActiveSheet_Print(2)
```