

マイコン部 2024

講習会資料

第2回

目次

目次

第 0 章 HSP の準備

- 0-0 はじめに
- 0-1 HSP とは
- 0-2 HSP の使い方
 - 0-2.1 スクリプトエディタ
 - 0-2.2 HSP アシスタント
 - 0-2.3 HSP ドキュメントライブラリ

第 1 章 基本の命令

- 1-0 HSP に命令を出す
- 1-1 screen 命令
- 1-2 メッセージ関連
 - 1-2.1 mes 命令
 - 1-2.2 font color
 - 1-2.3 pos
- 1-3 変数
 - 1-3.1 変数
 - 1-3.2 型変換
 - 1-3.3 配列
- 1-4 メディアの利用
 - 1-4.1 画像
 - 1-4.2 音楽
- 1-5 繰り返し
 - 1-5.1 repeat loop
 - 1-5.2 cnt

目次

1-5.3 wait await

1-5.4 break

第2章 基本の命令 ver2

2-1 条件分岐

2-1.1 if else

2-1.2 複数の条件分岐

2-1.3 end

2-2 数値計算

2-3 サブルーチン

2-3.1 ラベル

2-3.2 gosub return

2-3.3 goto stop

2-4 乱数

2-5 キー・マウスの読み取り

2-5.1 getkey

2-5.2 stick

2-6 オブジェクト

2-6.1 HSP のオブジェクト

2-6.2 オブジェクト管理命令

第3章 プログラム作成

3-1 画面の初期化

3-1.1 boxf

3-1.2 cls

3-1.3 redraw

3-2 時計の作成

目次

- 3-3 ボールを動かす
 - 3-3.1 ボールの移動
 - 3-3.2 ボールが跳ね返るようにする
 - 3-3.3 操作できるボールの追加
 - 3-3.4 ボールの数を増やす
- 3-4 じゃんけんゲーム
 - 3-4.1 素材の用意
 - 3-4.2 プログラムの作成
 - 3-4.2 勝敗の表示
- 3-5 プログラム紹介
 - 3-5.1 合成音声プログラム
 - 3-5.2 ブロック崩し
 - 3-5.3 タイピングゲーム
 - 3-5.4 RPG 風ゲーム

第4章 発展内容

- 4-1 オブジェクト・ウィンドウ ID
- 4-2 関数
- 4-3 ファイル操作
- 4-4 割り込みラベルジャンプ
 - 4-4.1 割り込み関数
 - 4-4.2 キー・マウスの読み取り発展版
- 4-5 プリプロセッサ
 - 4-5.1 プリプロセッサ命令
 - 4-5.2 HSP のプラグイン
 - 4-5.3 WindowsAPI

第 2 章 基本の命令 ver2

2-1 条件分岐

条件分岐はその名の通り、条件の違いによってプログラムの処理を分岐させる制御文です。HSP では基本的に **if 命令**を用いて条件分岐を行います。

2-1.1 if else

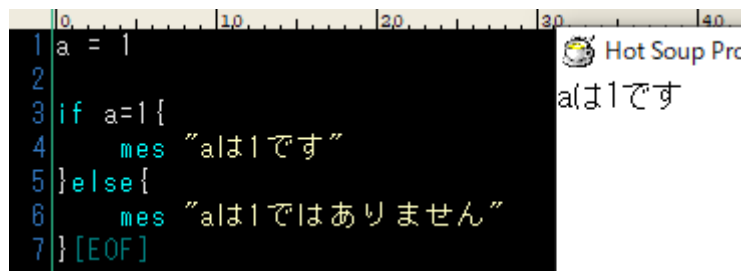
if-else 命令は HSP で条件分岐を行う上で最もポピュラーな命令です。使用する際は以下のように使用します。

```
if p1 //条件を満たしていればその行の命令を実行

p1: 条件式

else //条件を満たしていなければその行の命令を実行
```

if-else の文法はこの通りです。これから if-else 命令の使い方を詳しく見ていきましょう。まずは、以下の画像を見てみてください。



(図 2-1)基本的な if 文の使用法

上の画像を見てもわかる通り、基本的に if 文は条件に合ったときの処理の内容をカッコで囲んで表記します。なお、else は必ず書かなければいけないわけではなく、条件を満たしていなければ何もしない場合は省略することができます。（逆に if 命令を書かずに else 命令のみを書くことはできません）

if 条件式{

条件式が真のときに実行される命令

}else{

条件式が偽の時に実行される命令

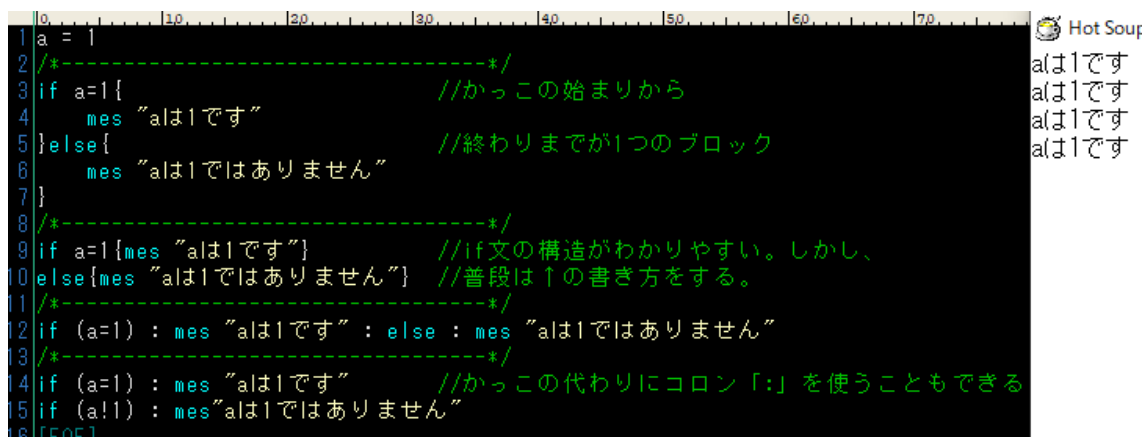
}

この表記方法では、かっこ{}の中身が一つのブロックとして扱われます。もしくは次のように書くこともできます。

if 条件式 : 条件式が真のときに実行する命令

if 条件式 : 真のとき実行する命令 : else : 偽の時実行する命令

これらはいずれも同じ意味の命令になります。



```
1 a = 1
2 /*-----*/
3 if a=1{                                     //かっこの始まりから
4     mes "aは1です"
5 }else{                                     //終わりまでが1つのブロック
6     mes "aは1ではありません"
7 }
8 /*-----*/
9 if a=1{mes "aは1です"}                     //if文の構造がわかりやすい。しかし、
10 else{mes "aは1ではありません"}           //普段は↑の書き方をする。
11 /*-----*/
12 if (a=1) : mes "aは1です" : else : mes "aは1ではありません"
13 /*-----*/
14 if (a=1) : mes "aは1です"                 //かっこの代わりにコロン「:」を使うこともできる
15 if (a!=1) : mes "aは1ではありません"
16 [END]
```

(図 2-2)様々な if 文の書き方

これで if 文の基本的な使い方は理解することができました。しかし、if 文には条件をつけるための条件式というものがようになります。そこで、ここからは**条件式**について解説していきます。

先ほども説明した通り、if 文は条件の内容によって、どの命令を実行するかを判断します。その条件を記述する式が条件式です。例えば、図 2-1 のプログラムでは if 命令の後ろに、条件式として「a=0」が記述されています。if 文の文法では、if の直後の条件式が真の場合の命令をその条件式の後ろに記述することになっているため、ここでは、a = 1 が「真」、つまり正しい場合、その直後にある「mes "a は 1 です"」の命令を実行することになります。

上記のプログラムでは条件式としてイコール(=)を用いていましたが、それ以外にも条件式は存在します。if 命令で使うことができる条件は以下の通りです。

(表 2-1)条件式の一覧

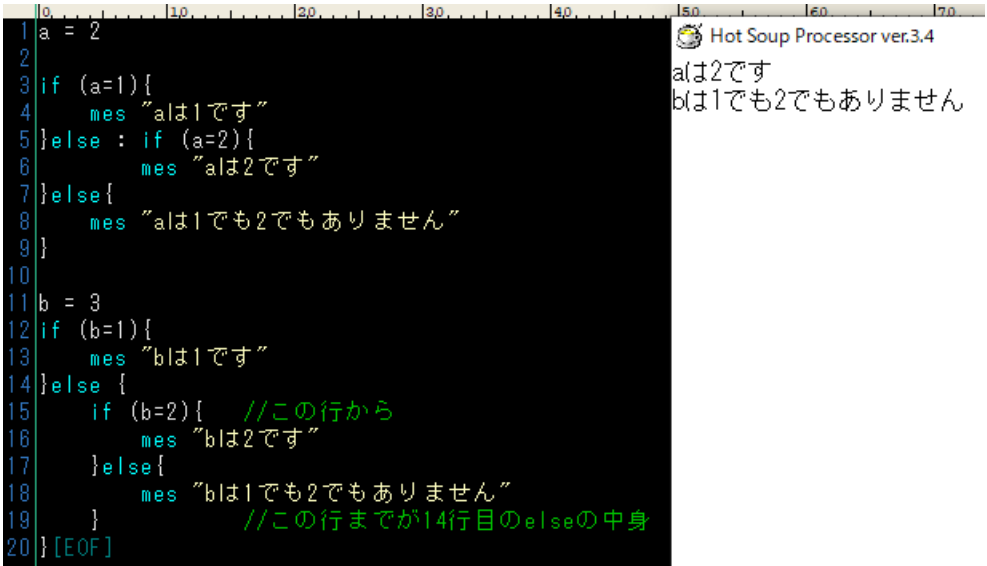
条件式	意味
a = b または a == b	a と b は等しい
a != b または a != b	a と b は等しくない
a < b	a は b よりも小さい (未満)
a > b	a は b よりも大きい
a <= b	a は b よりも小さいか等しい (以下)
a >= b	a は b よりも大きい等しい (以上)

2-1.2 複数の条件分岐

ここまでは2つの条件分岐について学びました。しかし実際にプログラムを書く際には複数の条件で分岐させたいといったこともあるかと思います。そこで、複数の条件を分岐させる方法について紹介します。

方法その1：else if 構文を使用する

1つ目の方法は、else if 構文を使用することです。else で実行される命令文の中に、さらにif文を入れる（入れ子構造）ことにより、複数の条件を分岐させることができます。



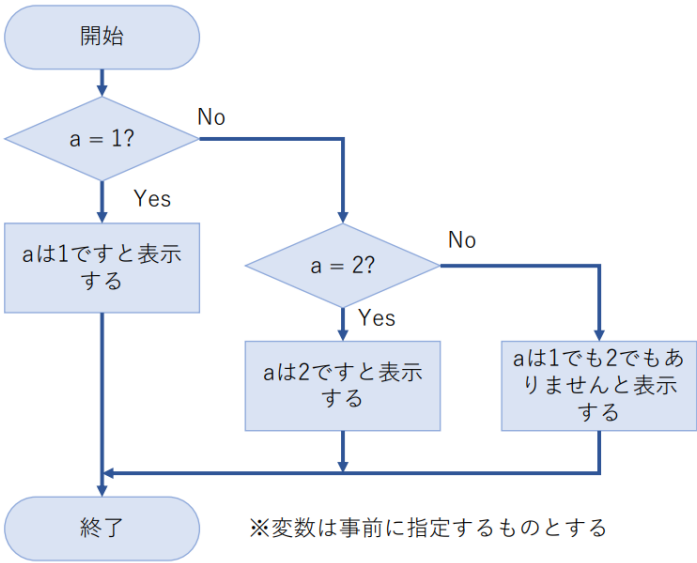
```
1 a = 2
2
3 if (a=1){
4     mes "aは1です"
5 }else : if (a=2){
6     mes "aは2です"
7 }else{
8     mes "aは1でも2でもありません"
9 }
10
11 b = 3
12 if (b=1){
13     mes "bは1です"
14 }else {
15     if (b=2){ //この行から
16         mes "bは2です"
17     }else{
18         mes "bは1でも2でもありません"
19     } //この行までが14行目のelseの中身
20 }[EOF]
```

(図 2-3) else if 構文を利用したプログラムの例

第2章 基本の命令 ver2

上のプログラムでは、5 行目の else の後に if 文が挿入されており、入れ子構造になっていることがわかります。また、下のプログラムでも、14 行目の if 文の 1 行下の 15 行目に if 文が挿入されています。

図 2-3 の場合、プログラムの流れを視覚的に表すと次のようになっています。このような図はフローチャートと呼ばれており、プログラムの動きを記述するためによく使われます。（詳細は別紙「フローチャート入門」を参照）



(図 2-4)プログラムの流れ

方法その 2：条件演算子を活用する。

2 つ目は条件演算子を利用する方法です。条件演算子とは、先ほど解説した、 $a=b$, $a!b$ などといったものを指します。条件演算子で複数の条件を付けることのできるものが AND（かつ）と OR（または）です。この 2 つは以下のように表わします。

(表 2-2)AND/OR の条件演算子

条件式	意味
条件式 A and 条件式 B または 条件式 A & 条件式 B	条件式 A が真かつ条件式 B が真のときに 真となる
条件式 A or 条件式 B または 条件式 A 条件式 B	条件式 A または条件式 B が真のときに真 となる

これらの条件演算子の使用例はこのようになります。


```

1 a = 2
2 b = 3
3
4 if (a = 2 or b = 0){
5     mes "a=2もしくはb=0もしくはa=2,b=0です"
6 }
7 /*-----*/
8 if a=2 & b=3{
9     mes "a=2でb=3です"
10 }else{
11     mes "a=2,b=3ではありません"
12 }[EOF]

```

Hot Soup Processor ver.3.4
a=2もしくはb=0もしくはa=2,b=0です
a=2でb=3です

(図 2-5) AND/OR の使用例

方法その3：switch 命令

HSP には、条件分岐のための命令として、if 命令のほかに switch 命令というものが用意されています。この命令では、if 命令のように2つに分岐させるのではなく、**1度にたくさんの分岐を作ることができます**。しかし、if 命令と比べて覚えることが多いため、最初のうちは if 命令で条件分岐になれることをお勧めします。switch 命令の使い方は以下の通りです。

switch p1 //比較ブロック開始

p1: 比較元

case p1 //比較値指定

p1: 比較値

swend //比較ブロック終了

default //デフォルト比較指定

swbreak //比較実行脱出指定

基本的な使い方は、switch で構文の開始、case でそれぞれの条件を指定して命令の実行を行い、すべての case を書き終わったら、swend で構文を終了させます。どのケース

第2章 基本の命令 ver2

にも当てはまらない場合の条件は default に記述し、途中で構文を抜ける場合は swbreak 命令を使用します。詳しくはそれぞれで調べてみてください。

2-1.3 end

end 命令は、**プログラムを終了させるための命令**です。この命令はプログラム内のどこに記述してもよく、end 命令が実行されると、そのアプリケーションが終了します。実際、HSP では end 命令を記述しなくてもプログラムは実行できますが、end 命令を書かないまま条件分岐を行うと、本来は実行しないほしいプログラムまで実行してしまうことによってエラーが発生する可能性があります。したがって、自分でプログラムを作成する場合は、基本的にはプログラムの最後に end 命令を記述した方がよいでしょう。

```
end p1 //プログラム終了
```

p1(0): 終了コード

また、end 命令と似たような命令として stop 命令があります。両者の違いは、end 命令はアプリケーションを終了するのに対して、stop 命令は、後述するボタンなどのアクションが起こるまでプログラムを一時停止するということです。

```
stop //プログラム中断
```

2-2 数値計算

第1章の変数の節では変数同士の足し算を行いました。今回は、HSP での加減乗除算などについて、より詳しく学んでみましょう。

算術演算子

まずは、HSP で扱える**算術演算子**についてです。算術演算子とは、 $+$ 、 $-$ 、 \times 、 \div などの記号のことで、私たちが普段数学等で利用している演算子です。

(表 2-3)算術演算子の一覧

HSP での表記	普段の表記	意味
+	+	足し算、加算
-	-	引き算、減算
*	\times	掛け算、乗算

/	÷	割り算、除算
¥	… or あまり	割り算のあまり、剰余残

算術演算子の使用例は以下のようになります。

```
1 a = 13
2 b = 5
3
4 mes a+b 18
5 mes a-b 8
6 mes a*b 65
7 mes a/b 2
8 mes a#b [EOF] 3
```

7 行目の割り算に関しては a,b がともに int（整数）型のため、小数点以下は切り捨てられて答えは 2 となります。

また、8 行目の剰余残ではあまりのみが出力されます（この場合は 2 余り 3 なので 3 のみの出力）

(図 2－6)算術演算子を利用した計算の例

代入演算子

次は、**代入演算子**です。代入演算子とは、変数に対して何か値を代入するための演算子です。プログラミング言語では基本的に右辺の値を左辺に代入します。

(表 2－4)代入演算子の一覧

表記	説明
= or ==	右辺を左辺に代入
+=	右辺に左辺の内容を足して左辺に代入 a += b は a = a+b と同じ意味になる
-=	右辺から左辺の内容を引いて左辺に代入 a -= b は a = a-b と同じ意味になる
*=	右辺に左辺の内容をかけて左辺に代入 a *= b は a = a*b と同じ意味になる
/=	右辺に左辺の内容をわって左辺に代入 a /= b は a = a/b と同じ意味になる

```
1 a = 13
2 b = 5
3 mes a //元の数
4 a += b //a=a+b
5 mes a //+=された数
6 a -= b //a=a-b
7 mes a //さらに-=された数
8 a *= b //a=a*b
9 mes a //さらにさらに+=された数
```

Hot Soup P

13
18
13
65

(図 2-7)代入演算子の利用例

まずは基本的な代入演算子の説明を行いました。次は、少し特殊な代入演算子たちです。

(表 2-5)特殊な代入演算子

表記	名称	説明
++(+を 2 個)	インクリメント	1 の足し算 a++は、a=a+1 と同じ意味になる
--(－を 2 個)	デクリメント	1 の引き算 a--は、a=a-1 と同じ意味になる

2-3 サブルーチン

ここまでのプログラムでは、1 行目から順番にプログラムが実行されました。しかし、プログラムが大規模になるにつれて、1 つの大きな処理ではプログラムの変更が難しくなってきます。そこで、プログラムの中身をいくつかの小さな処理に分割することが必要となってきます。そこで使われるのがサブルーチンです。**サブルーチンとは、どこかから呼び出され、処理が終了したら元の位置に戻ってくる処理のこと**を言います。

2-3.1 ラベル

サブルーチンの処理を行う前に、まずはプログラムをいくつかの小さな処理に分割しなくてはなりません。その際使用するのが**ラベル**です。

プログラムをいくつかの処理に分割すると言いましたが、ラベルの機能のイメージは「**プログラムに旗を立てる**」です。このラベルを設定し、プログラムに印をつけることでその

第2章 基本の命令 vol2

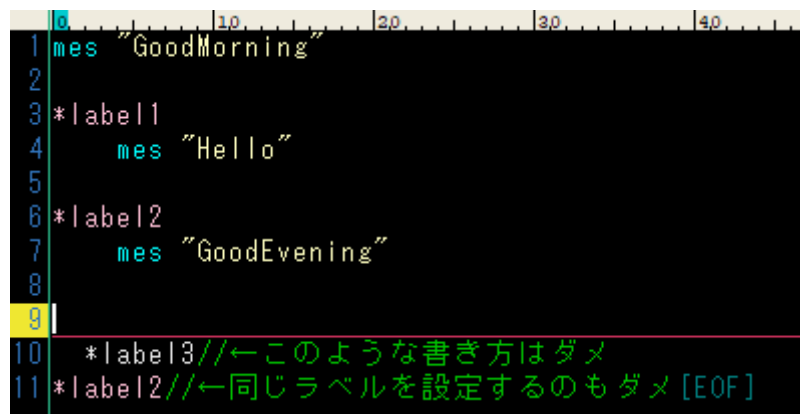
印へ飛んだり、戻ったりすることで、ひとかたまりのプログラムの処理を実行することができます。

HSP にはラベルを指定する際、いくつかの決まりごとがあります。

1. 命令と違って最初に TAB/Space を入れてはいけない
2. *の後に英文字（59 文字以内）で名前を付ける
3. 同じ名前が 2 つ以上あってはならない

実際には、1, 2 番の決まりを破っても HSP3.4 ではエラーは出ませんが、一応守っておいた方が良いでしょう。（3 番はエラーが出ます）

実際にラベルを設定してみるとこのようになります。



```
1 mes "GoodMorning"
2
3 *label1
4   mes "Hello"
5
6 *label2
7   mes "GoodEvening"
8
9
10 *label3//←このような書き方はダメ
11 *label2//←同じラベルを設定するのもダメ [EOF]
```

(図 2-8)ラベルの設定例 下部には推奨されない例も挙げた

このように、プログラムの機能ごとにラベルを設定し、後述する gosub 命令や goto 命令を利用してサブルーチンの機能を実現します。

2-3.2 gosub return

ラベルの設定が終わると、次はサブルーチンの設定を行います。サブルーチンを行うには、gosub-return 命令を使います。これらの命令は基本的には repeat-loop 命令のように 2 つで 1 セットとなっているので、両方とも忘れずに記述するようにしましょう。

```
gosub *label //指定ラベルにサブルーチンジャンプ
```

*label: ラベル名

```
return p1 //サブルーチンから復帰
```

p1: システム変数代入値

gosub 命令を実行することで、指定したラベルまでジャンプして、そのラベルの下
からプログラムを実行します。その後、return 命令があると、ジャンプ前の gosub 命
令の次の行に戻ってきて、プログラムの実行を続けます。また、gosub-return 命令はネ
スティング（入れ子構造）にすることが可能です。return 命令にはシステム変数の代入を
行うことができます。サブルーチンを関数のように扱う場合は利用することもありま
すが、基本的にはパラメータは何も設定しなくて大丈夫です。

```

1 mes "GoodMorning"
2 gosub *label2
3 mes "こんばんは"
4 gosub *label1
5 mes "こんにちは"
6 stop
7
8 *label1
9     mes "Hello"
10    gosub *label2
11 return
12
13 *label2
14     mes "GoodEvening"
15 return[EOF]
    
```

Hot Soup Processor ver.3.4

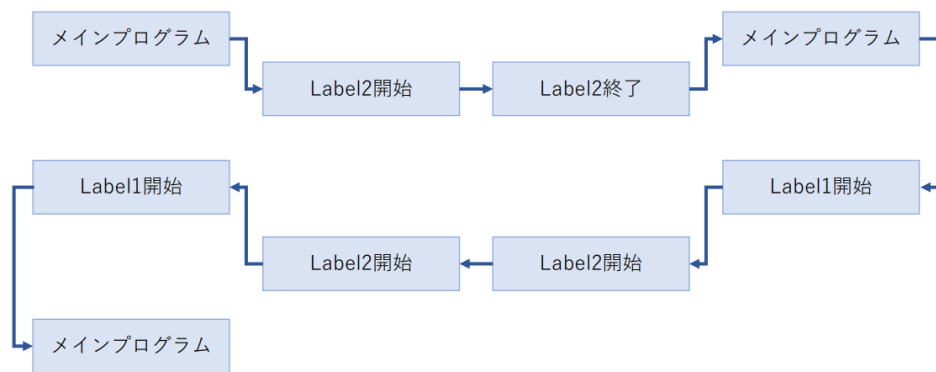
```

GoodMorning
GoodEvening
こんばんは
Hello
GoodEvening
こんにちは
    
```

(図 2-9)サブルーチンの使用例

図 2-9 のプログラムでは、1 行目が実行された後、gosub 命令で 12 行目にジャンプ、
14 行目の return で 3 行目に戻り、4 行目の gosub 命令で今度は 8 行目にジャンプしま
す。すると、label1 内で入れ子構造になっている gosub 命令により、再び label2 にジャ
ンプします。label2 が終了し、11 行目の return 命令に戻ると、そのまま label1 も終了
して元のプログラムに復帰し、5 行目から実行されます。

第2章 基本の命令 vol2



(図 2-10)図 2-9 のプログラムの流れ

※**gosub-return** 命令を使用する際、メインのプログラム（図 2-9 だと 1～6 行目）の最後には必ず **stop** 命令を入れてください。（6 行目）HSP は上から順番に命令を最後まで実行します。したがってメインのプログラムが最後まで命令が実行されると、プログラムはそこで停止せずにその下に記述されたラベルの中身まで実行します。その後、**return** 命令の箇所でエラーが発生してしまいます。

2-3.3 goto stop

gosub-return 命令と似たような命令として、**goto** 命令があります。

```
goto *label //指定ラベルにジャンプ
```

```
*label: ラベル名
```

goto 命令も **gosub** 命令と同様に、指定したラベルへジャンプしてくれます。**goto** 命令と **gosub** 命令の違いは、**gosub** 命令は **return** 命令が実行されると元の場所に復帰して実行を続けるのに対し、**goto** 命令はジャンプしたきりでそのまま実行を続けます。

- **gosub** : ラベルにジャンプして **return** があれば帰ってくる
- **goto** : 一度ジャンプすると行ったきり

2-4 乱数

ここまでは、プログラム自体の動きを制御する制御文を学んできました。ここからは、制御文からは少し外れて HSP で扱われる乱数について解説します。

第2章 基本の命令 vol2

乱数とは、簡単に説明すると「**ランダムな数字**」のことです。何かランダムに物事を決めたい場合、現実ではサイコロを振ったりくじを引いたりすることができます。しかし、コンピュータではそのようなことを行えません。したがってその代わりに乱数の命令を利用してランダムな物事を決定します。乱数の使い方は以下の通りです。

```
val = rnd(p1) //乱数の発生
```

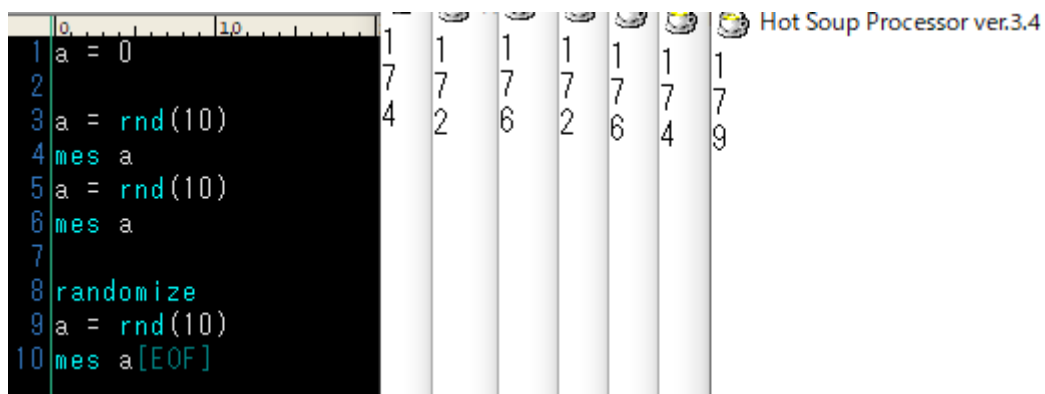
p1=1~32768: 乱数の範囲 ※val は変数のこと

```
randomize p1 //乱数発生の初期化
```

p1=0~(不定): 乱数の初期化パラメータ

このように、rnd の後のパラメータに値を入力することで、変数に 0~その値-1 までの乱数を代入することができます。例えば、「a = rnd(10)」とすると、変数 a には 0~9 までの 10 パターンのうち、いずれかの整数が代入されます。

しかし、実は乱数のパターンはパソコンが起動したときに一度決まっており、パソコンを再起動するまでその値が残ります。したがって、同じプログラムの中で複数回乱数を発生させたい場合は、乱数の初期化を行わなくてはなりません。乱数の初期化の際、パラメータを設定しなければ、ランダムな変数に初期化されます。



(図 2-11) 複数回プログラムを実行した場合

randomize を行った 3 つ目のみ、毎回値が変化していることがわかる

2-5 キー・マウスの読み取り

ゲームを作成する際、キーボードやマウスの入力を読み取る必要があります。ここでは、簡単なキーボード入力とマウス入力の検出方法を紹介します。

2-5.1 getkey

キーボード、マウス入力を読み取るために、最もよく使われる方法が `getkey` 命令を使用することです。この命令は、キーボードの入力、マウスのクリックを取得することができます。

```
getkey p1,p2 //キー入力チェック
```

`p1=変数` : 読み込むための変数

`p2=1～(1)` : キーコード

設定したキーコードのキーボードのキーが押されているまたはマウスがクリックされている場合は、変数に 1、されていない場合は変数に 0 が入ります。設定できるキーコードは以下の通りです。

キーコード : 内容		
1 : マウスの左ボタン	17 : [CTRL]	37 : カーソル[←]
2 : マウスの右ボタン	18 : [ALT]	38 : カーソル[↑]
2 : キャンセル	20 : [CAPSLOCK]	39 : カーソル[→]
4 : ホイールクリック	27 : [ESC]	40 : カーソル[↓]
8 : [BACKSPACE]	32 : スペースキー	48～57 : [0]～[9]
9 : [TAB]	33 : [PAGEUP]	65～90 : [A]～[Z]
13 : [ENTER]	34 : [PAGEDOWN]	96～105 : テンキー
16 : [SHIFT]	35 : [END]	112～121 : [F1]～
	36 : [HOME]	[F10]

2-5.2 stick

HSP には、キー/クリック検出のために、ほかの命令も用意されています。stick 命令は getkey 命令と同じく、キーやクリックの検出を行いますが、getkey と違うのは、一つの命令で同時検出ができる。（getkey 命令でも変数を複数用意することで同時検出は可能）getkey 命令は入力をしている間はずっと変数に数値が代入されるが、stick 命令はキーやクリックを押した瞬間のみ数値が代入される、検出できるキー/クリックの種類が getkey に比べて少ないなどの点です。

```
stick p1,p2,p3 //キー入力情報取得
```

p1=変数 : 読み込むための変数

p2=0~(0) : 非トリガータイプキー指定

p3=0~1(1) : ウィンドウアクティブチェック ON/OFF

stick 命令で検出出来るボタン情報は以下の通りです。

1 : カーソルキー左(←)

2 : カーソルキー上(↑)

4 : カーソルキー右(→)

8 : カーソルキー下(↓)

16 : スペースキー

32 : Enter キー

64 : Ctrl キー

128 : ESC キー

256 : マウスの左ボタン

512 : マウスの右ボタン

1024 : TAB キー

何もボタンが押されていない場合には 0 が代入されます。また、もし複数のボタンが同時に押されていた場合には、それらの数値がすべて加算されて変数に代入されます。

2-6 割り込みラベルジャンプ

HSP には、クリックが行われたりキー入力が行われたり、エラーが発生したり、windows のダイアログが出た瞬間に事前に指定したラベルへジャンプする（割り込み処理）命令が存在します。これらの命令は一度実行されると、その後 stop や wait、await で一時停止している間にずっとキーやマウスの入力を監視してくれます。これを応用することで、キー入力やクリックなどを検出することができます。

2-6.1 割り込み関数

割り込み処理の関数には以下のようなものがあります。

命令	内容
onclick	クリック時割り込み実行指定
oncmd	windows メッセージ割り込み実行指定
onerror	エラー発生時割り込み指定
onexit	終了後（ウィンドウ右上×ボタン）割り込み指定
onkey	キー割り込み実行指定

oncmd 命令以外の表記方法はほぼ同じ(oncmd の場合のみラベル名の後ろにメッセージ ID を入力できる)なので、代表として onclick を見てみましょう。

```
onclick goto/gosub *label //クリック割り込み実行指定
```

*label: ラベル名

onclick はマウスのボタンを押したときに自動でジャンプする場所を指定します。

同じように、onerror では HSP 上のエラーが出たときにジャンプする場所を指定し、onexit はウィンドウ右上にある赤い×印が押されたとき、onkey はキーボードのキーが押されたときにジャンプする場所を指定します。

注意点として onexit 命令は一度実行すると end 命令以外に終了する方法がなくなる（一応タスクキルで対処可能）ので必ず end 命令を記述してください。

また、割り込み関数はいずれも命令の直後に 0 を入力して実行すると一時的に割り込みを停止し、1 を入力すると割り込みを再開させることができます。

例：onclick の場合の割り込み処理の ONOFF

`onclick 0` //割り込み一時停止

`onclick 1` //停止した割り込みの再開

また、各命令を使用して割り込みジャンプを行うと、システム変数 `iparam`, `wparam`, `lparam` に値が代入されます。各命令によって代入される内容は以下の通りです。

割り込み要因	iparam	wparam	lparam
onclick	マウスボタン ID	window メッセージとして渡されたパラメータ	左に同じ
oncmd	メッセージ ID	window メッセージとして渡されたパラメータ	左に同じ
onerror	0(なし)	エラー番号	エラー発生行番号
onexit	終了要因 ※	終了の通知を受けたウィンドウ ID	0(なし)
onkey	文字コード	window メッセージとして渡されたパラメータ	左に同じ

※終了要因について

`onexit` でジャンプされた直後は、システム変数 `iparam` に終了要因が値として保存されています。`iparam` が 0 の場合は、ユーザの意思でプログラムを終了。`iparam` が 1 の場合は、Windows シャットダウン(再起動または電源 OFF)による終了です。

追加で、`oncmd` 命令は、割り込みが発生したウィンドウ ID を `ginfo` 関数で取得することができます。

2. - 6.2 キー・マウスの読み取り発展版

これらの割り込み処理を活用して、2-5 で行ったキーやマウスの読み取りをより高度に行うことができます。

2-7 オブジェクト

HSP では画面に文字や画像を表示させるほかに、実際に使用することのできるボタンやチェックボックスを扱うことができます。これらの要素のことを「オブジェクト」といい、標準では6つのオブジェクトを命令によって用意できます。

(表 2-6)オブジェクトの一覧

命令	オブジェクト名	説明
button	ボタン	指定ラベルのプログラムを実行する
input	入力ボックス	1行分の数値や文字列を入力
mesbox	メッセージボックス	複数行の文字列を入力
chkbox	チェックボックス	チェックマークの ONOFF を入力
combox	コンボボックス	複数要素の選択データを入力
listbox	リストボックス	複数要素の選択データを入力

ここからは、各オブジェクトの細かな解説です。

2-7.1 HSP のオブジェクト

HSP の標準命令で用いられる6つのオブジェクトの詳細は以下のようになります。

ボタン

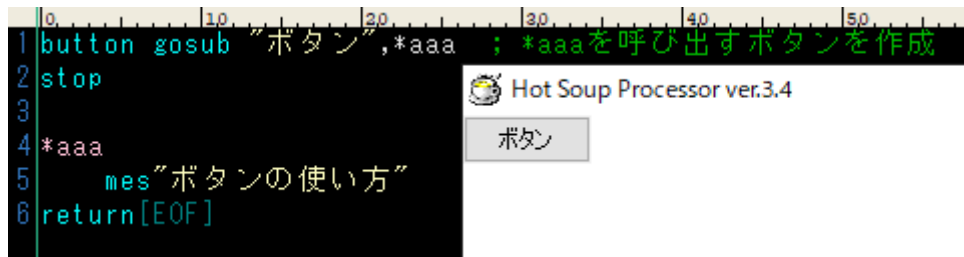
ボタンは HSP の実行画面上にボタンを表示します。

```
button goto/gosub "name",*label //ボタンの表示
```

"name": ボタンの名前

*label: 押した時にジャンプするラベル名

button の後に goto と記述した場合は goto 命令と同じ挙動を、gosub と記述した場合は gosub 命令を同じ挙動をします。また、“name”の部分を変更することで、ボタン内に表示される文字の内容を変更することができます。ボタンをさらに細かく設定したい場合は、objimage 命令（カスタムボタンの設定）を利用することによって実現できます。



(図 2-12)ボタンの使用例

入力ボックス

入力ボックスは HSP の実行画面上に 1 行の入力ボックスを表示します。

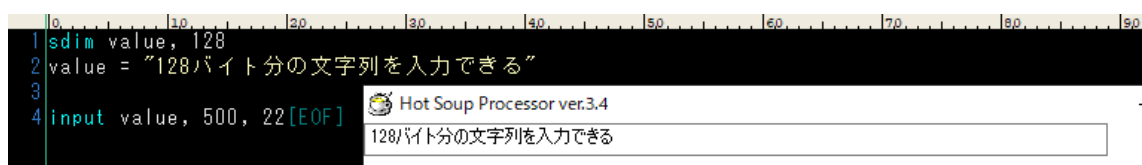
`input p1,p2,p3,p4` //入力ボックス表示

p1=変数 : 入力のための変数

p2,p3 : メッセージボックスのサイズ (ドット単位)

p4=0~ : 入力できる最大文字数 (0 だとほぼ無限になる)

入力ボックスはキーボードで入力することができる、小さなボックスです。入力した値は p1 の変数に代入されます。代入される要素の型は、変数の型に依存します。入力ボックスは初期状態では、p1 で指定した変数の内容がボックス内に表示されます。また、P4 では入力できる最大文字数を指定することができます。



(図 2-13)入力ボックスの使用例

メッセージボックス

メッセージボックスは HSP の実行画面上に複数行入力することができる入力ボックスを表示します。

`mesbox p1,p2,p3,p4,p5` //メッセージボックス表示

p1=変数 : 表示メッセージが代入された文字列型変数

p2,p3 : メッセージボックスのサイズ (ドット単位)

p4=0~(1) : メッセージボックスのスタイル

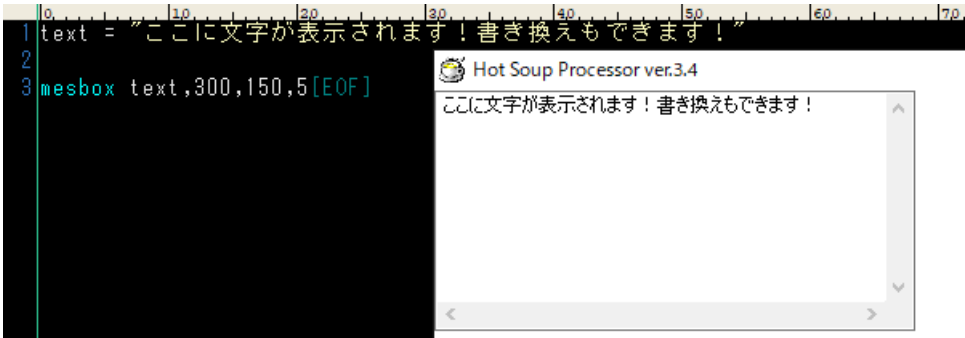
p5=0~(-1): 入力できる最大文字数

mesbox 命令には 2 つの使い道があります。一つ目は、スクロールできる msg 命令のように、メッセージボックスに長文を表示させること。もう一つはユーザが好きに文字を入力できる、簡易的なテキストボックスとして使用することです。パラメータには input 命令と同じように、p1 には変数が設定され、初期状態ではこの変数に代入された文字列が表示されています。また、P4 にはメッセージボックスのスタイルを設定することができます。スタイルの詳細は以下の通りです。

(表 2-7)mesbox のスタイル

p4 に入力する値	対応するキー
0	スクロール可能なエディットボックス (書き換え不可)
1	スクロール可能なエディットボックス (書き換え可能)
4	横スクロールバー付きエディットボックス (書き換え不可)
5	横スクロールバー付きエディットボックス (書き換え可能)

書き換え不可では上記の 1 の使い道、書き換え可能では上記の 2 の使い道に利用することができます。



(図 2-14)メッセージボックスの使用例

コラム：文字列の改行

HSP では、mes 命令や mesbox 命令に記述する文字列を開業することができます。改行するためには文章の中の改行したい場所に「¥n」を入力します。すると、¥n 以降の文

第2章 基本の命令 vol2

字列は次の行に記述されるようになります。また、文章中で¥マークが使いたい場合は、¥マークを2つ重ねる (¥¥) と、1つの¥マークとして認識してくれます。

チェックボックス

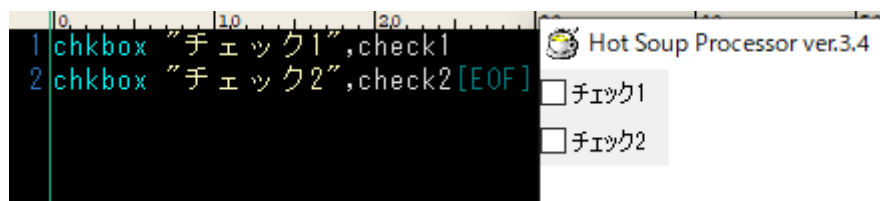
チェックボックスは HSP の実行画面上に項目のチェックボックスを表示します。

```
checkbox "strings",p1 //チェックボックス表示
```

"strings": チェックボックスの内容表示文字列

p1=変数 : チェックボックスの状態を保持する変数

チェックボックスは、strings で設定した文字列の左側に、カーソルで ONOFF を切り替えることのできるスイッチのついたオブジェクトです。p1 で設定した整数型変数の内容が0ならチェックは OFF、1なら ON になります。



(図 2-15)チェックボックスの使用例

コンボボックス

コンボボックスは HSP の実行画面上にドロップダウンリストを表示します。

```
combox p1,p2,p3 //コンボボックス表示
```

p1=変数 : コンボボックスの状態を保持する数値型変数

p2=0～(100) : 拡張 Y サイズ

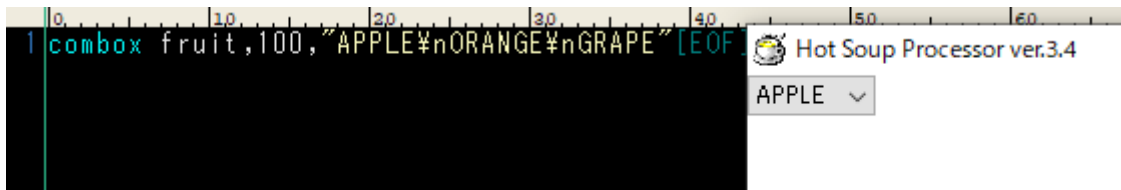
p3="strings": コンボボックスの内容を示す文字列

コンボボックスは複数の要素の中から1つを選択することができるオブジェクトです。p1 には後述するインデックス番号が代入されます。また、ドロップダウンしたときのリストの縦サイズを p2 で設定します。(100～150 推奨) さらに「¥n」で区切った文字列

第2章 基本の命令 vol2

を p3 で指定することで、選択する要素を設定することができます。（文章の改行と同じ要領）例えば、「APPLE¥nORANGE¥nGRAPE」という文字列を指定すると、

「APPLE」「ORANGE」「GRAPE」の中から1つを選択するコンボボックスになります。それぞれの要素には、0 から順番にインデックス番号がついています。前の例では、「APPLE」はインデックス 0、「ORANGE」はインデックス 1、「GRAPE」はインデックス 2 というふうに番号がついていきます。



(図 2-16)コンボボックスの使用例

リストボックス

リストボックスは HSP の実行画面上に選択できるリストを表示します。

`listbox p1,p2,p3` //リストボックス表示

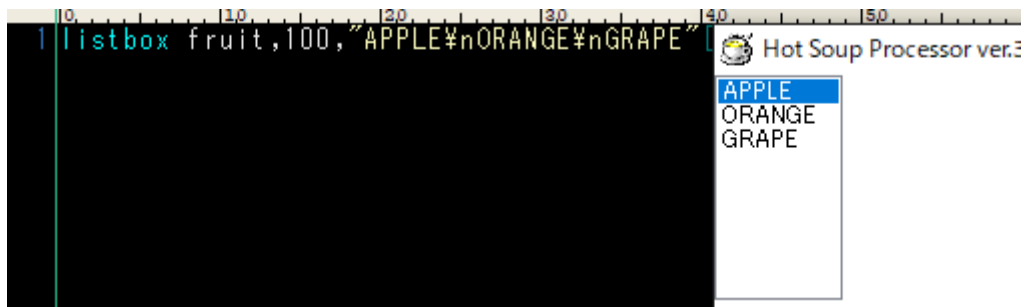
p1=変数 : リストボックスの状態を保持する数値型変数

p2=0～(100) : 拡張 Y サイズ

p3="strings": リストボックスの内容を示す文字列

リストボックスは複数の要素の中から1つを選択することができるオブジェクトです。

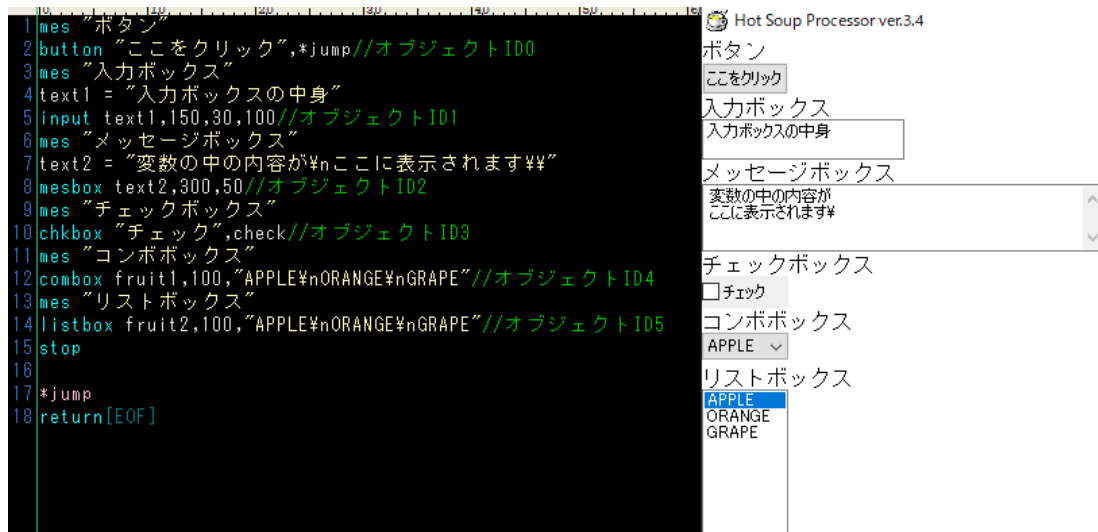
仕様はほぼコンボボックスと同じですが、リストボックスはボックスをクリックしたときにドロップダウンメニュー（下に垂れ下がるように表示されるメニュー）が出ます。



(図 2-17)リストボックスの使用例

コラム：オブジェクト ID

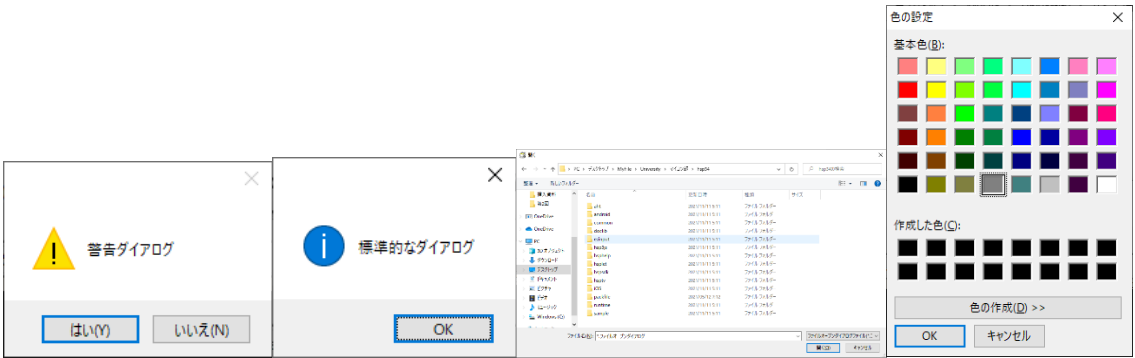
HSP には、オブジェクト ID という概念があります。オブジェクト ID とは、プログラム上でオブジェクトを生成するごとに、順番に付けられる通し番号のことです。オブジェクト ID は、後述するオブジェクト管理命令である `objprm` 命令、`clrobj` 命令、`objsel` 命令などに使用することになります。



(図 2-18)オブジェクト一覧とオブジェクト ID

コラム：[応用]ダイアログボックス

標準の 6 つの命令からは外れますが、HSP には少し特殊なオブジェクトである `dialog` 命令があります。これは、windows でよく見るポップアップしてくる各種ダイアログボックスを表示する命令で、HSP 内でも予期しない動作時のアラートの作成やファイル選択、はい/いいえの選択などに使うことができます。



(図 2-17~20)様々なダイアログ

```
dialog "message",p1,"option" //ダイアログを開く
```

p1=0~(0) : ダイアログのタイプ設定

ダイアログのタイプは、p1 に入力する値によって変わります。

(表 2-8)dialog のタイプ

タイプ	内容
0	標準メッセージボックス + [OK] ボタン
1	警告メッセージボックス + [OK] ボタン
2	標準メッセージボックス + [はい][いいえ] ボタン
3	警告メッセージボックス + [はい][いいえ] ボタン
16	ファイル OPEN(開く)ダイアログ
17	ファイル SAVE(保存)ダイアログ
32	カラー選択ダイアログ(固定色)
33	カラー選択ダイアログ(RGB を自由に選択)
64~	拡張ダイアログ

タイプ 64~の拡張ダイアログでは、拡張命令（ランタイムごとに用意される拡張ダイアログ）のために予約されています。標準命令では使用しません。

dialog 命令では、タイプの種類ごとに message と option の意味が変わってきます。

(表 2-9)各タイプの message、option、返り値の特徴

タイプ	message	option	返り値
0,1,2,3	表示文字列	タイトル名	システム変数 stat に押したボタンが 「OK」なら 1,「はい」なら 6,「いいえ」 なら 7 が代入される
16,17	選択拡張子	ファイルの種類	システム変数 stat には ファイルが選ばれなかったら 0, 選ばれた ら 1 が代入される システム変数 refstr には 選択されたファイルのパスが代入される
32,33	意味なし	意味なし	システム変数 stat には 色が選ばれなかったら 0, 選ばれたら 1 が 代入される システム変数 ginfo_r,ginfo_g,ginfo_b にそ れぞれ RGB 値が代入される

dialog のどのボタンを押したか/どのファイルを選んだか/どの色を選択したかはシステム変数に代入されることになっています。(stat,refstr,ginfo)

システム変数について

ここで、簡単にシステム変数について説明します。前章の繰り返し命令に出てきた、cnt や、今回の stat、refstr、ginfo などは事前にシステムによってそれぞれ決まったものを入れると決められている変数で、これをシステム変数といいます。例えば、cnt だとループの回数、stat だと命令の返り値（命令を実行した結果戻ってくる値（ステータス値））、refstr は命令を実行した結果の文字列など、それぞれのシステム変数ごとに役割があります。詳しくは、第4章の応用で解説します。

2-5.2 オブジェクト管理命令

オブジェクトは、そのままでは大きさや表示場所、内容を変更することができません。そこでオブジェクト管理命令を利用することによってそれらを変更して、より扱いやすいオブジェクトを作成することができます。

第2章 基本の命令 vol2

場所の変更

厳密にはオブジェクト管理命令ではありませんが、pos 命令によって、オブジェクトの表示位置を変更することができます。

それ以外の管理命令

ここでは、pos 以外の管理命令について紹介します。命令の細かい仕様については各自調べてみてください。

(表 2-10)オブジェクト管理命令の一覧

命令	命令名	内容
clrobj	クリアオブジェクト	オブジェクトをクリア
objmode	オブジェクトモード	オブジェクトモード設定
objprm	オブジェクトパラメータ	オブジェクトの内容を変更
objsel	オブジェクトセレクト	オブジェクトに入力フォーカスを設定
onjsize	オブジェクトサイズ	オブジェクトのサイズ設定
objenable	オブジェクトイネーブル	オブジェクトの有効・無効を設定

クリアオブジェクト

```
clrobj p1,p2 //オブジェクトをクリア
```

p1=0～(0) : 消去するオブジェクト ID(開始)

p2=0～(-1) : 消去するオブジェクト ID(終了)(-1 の場合は、最終の ID が指定されます)

実は、一度表示したオブジェクトは HSP 内では普通の方法で消去することができません。消去するにはこの clrobj 命令か、cls 命令が必要になります。

オブジェクトモード

```
objmode p1,p2 //オブジェクトモード設定
```

p1=0～(0) : オブジェクトフォント設定モード指定

p2=0～1 : フォーカス移動キー指定(0=OFF/1=ON)

第2章 基本の命令 vol2

button,input,mesbox などのオブジェクト命令で使用するスタイル等を設定するための命令です。

オブジェクトパラメータ

```
objprm p1,p2 //オブジェクトの内容を変更
```

p1=0～(0) : オブジェクト ID 指定

p2 : 変更するパラメータの内容

button 命令や input,mesbox 命令などで画面上に配置したオブジェクトの持つ内容やパラメータを変更します。

オブジェクトセレクト

```
objsel p1 //オブジェクトに入力フォーカスを設定
```

p1=0～(0) : オブジェクト ID 指定

p1 で指定したオブジェクト ID に入力フォーカスを合わせます。つまり、自動で入力ボックスにカーソルを移動させることができます。

入力フォーカスを合わせるにより、mesbox 命令や input 命令で配置した入力ボックスの中に入力カーソル(キャレット)を出すことができます。

オブジェクトサイズ

```
objsize p1,p2,p3 //オブジェクトサイズ設定
```

p1=0～(64) : オブジェクトの X 方向のサイズ (ドット単位)

p2=0～(24) : オブジェクトの Y 方向のサイズ (ドット単位)

p3=0～(0) : Y 方向の最低確保行サイズ (ドット単位)

ボタンや入力ボックスなどを配置する時のオブジェクトの大きさを設定します。

p3 でボタンやメッセージが置かれた後にカレントポジションが移動する最低量を指定することができます。

第2章 基本の命令 vol2

オブジェクトイネーブル

`objenable p1,p2` //オブジェクトの有効・無効を設定

p1=0～(0) : オブジェクト ID 指定

p2=0～(1) : 0 ならば無効、0 以外は有効

p1 で指定したオブジェクト ID の状態を変更します。

p2 で指定した値が 0 ならば、オブジェクトは無効化されます。

無効化されたオブジェクトは、画面上に存在しますが色に変更され、操作することはできなくなります。

第 2 章 基本の命令 vol2

第 2 章練習問題

練習問題 1

以下のプログラムを作成してください

- region 変数と timezone 変数を用意
- region には日本 or アメリカを、timezone 変数には時間帯（朝、昼、夕）を直接入力
- 地域、時間帯によって挨拶を変える(おはよう/こんにちは/こんばんは
/GoodMorning/Hello/GoodEvening)

練習問題 2

以下のプログラムを作成してください

1. プログラムを起動するたびに値が変更される変数を 5 個生成する
2. 5 個の変数の和を表示する
3. 5 個の変数の最大値を表示する
4. 5 この変数の最大値÷最小値の商とあまりを表示する
5. 2～5 を別のサブルーチンに分割し、gosub を用いて一連の操作を行う

練習問題 3

以下のプログラムを作成してください

- 練習問題 1 で作成したプログラムについて、region と timezone を、オブジェクトを利用してアプリケーション上で入力できるようにする


```
1 region = ""
2 timezone = 0
3
4 mes "地域"
5 input region,100,30,10
6 mes "時間帯"
7 listbox timezone,100,"朝\n昼\n夕"
8 button gosub "実行する!",*main
9 stop
10
11 *main
12 if region = "日本"{
13     if timezone = 0{
14         mes "おはよう"
15     }else : if timezone = 1{
16         mes "こんにちは"
17     }else : if timezone = 2{
18         mes "こんばんは"
19     }
20 }else : if region = "アメリカ"{
21     if timezone = 0{
22         mes "Goodmorning"
23     }else : if timezone = 1{
24         mes "Hello"
25     }else : if timezone = 2{
26         mes "GoodEvening"
27     }
28 }
29 [EOF]
```

(図 2-22) 練習問題 1 & 3 の回答例

第2章 基本の命令 vol2

```
1 randomize//変数の初期化
2
3 dim numbers,5//配列の宣言
4 sum = 0
5 max = 0
6 min = 0
7
8 mes "ランダムに生成された数"
9 repeat 5//ランダムな数字の生成
10     numbers(cnt) = rnd(10) + 1
11     mes "番号" + str(cnt+1) + ":" + str(numbers(cnt))
12 loop
13 gosub *plus
14 gosub *maxp
15 gosub *division
16 stop
17
18
19 *plus//変数の和を計算して表示するサブルーチン
20     repeat 5
21         sum += numbers(cnt)
22     loop
23     mes "5つの変数の合計値は" + str(sum) + "です"
24 return
25
26 *maxp//最大値を表示するサブルーチン
27     max = numbers(0)
28     repeat 4,1
29         if max <= numbers(cnt): max = numbers(cnt)
30     loop
31     mes "5つの変数の最大値は" + str(max) + "です"
32 return
33
34 *division//最大値÷最小値の商とあまりを表示するサブルーチン
35     min = numbers(0)
36     repeat 4,1//最小値を調べる
37         if min >= numbers(cnt): min = numbers(cnt)
38     loop
39     mes "最大値" + str(max) + "と、最小値" + str(min) + "の割り算は"
40     mes str(max) + "÷" + str(min) + "=" + str(max/min) + "あまり" + str(max % min) + "です"
41 return
42 [EOF]
```

(図 2-23)練習問題 2 の回答例