

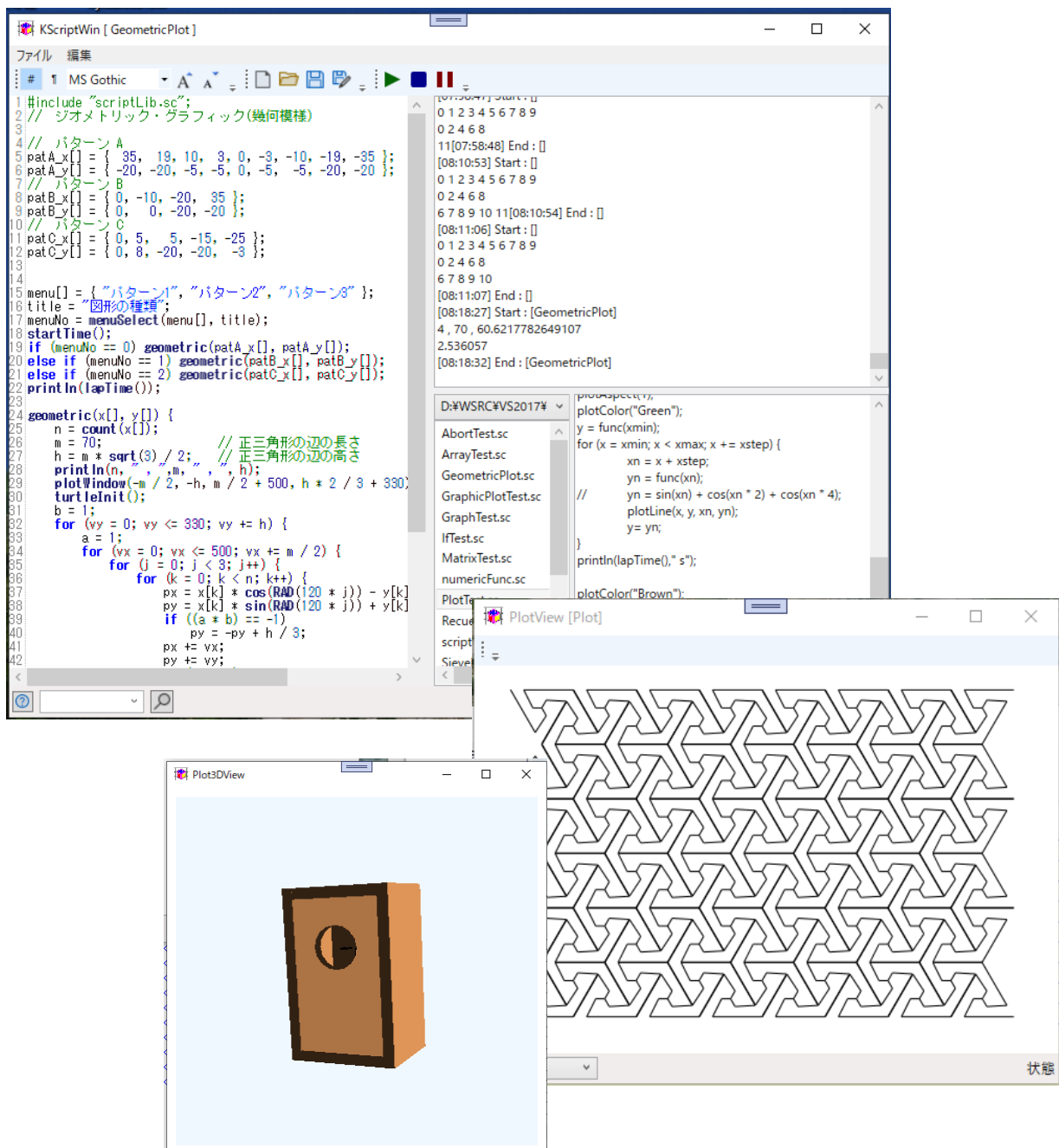
KScriptWin

C 言語風のスクリプト言語とその実行環境

KScript は C 言語に似せたスクリプト言語でインタプリタとして動作する。

自前の CAD ソフト (CadApp や Min3DCad) に組み込んで動作させることを目的に作成した。
基本的な制御構造と型指定のない変数で作られている。

KScriptWin は Windows 上で KScript のコード作成とそれを実行する環境である。

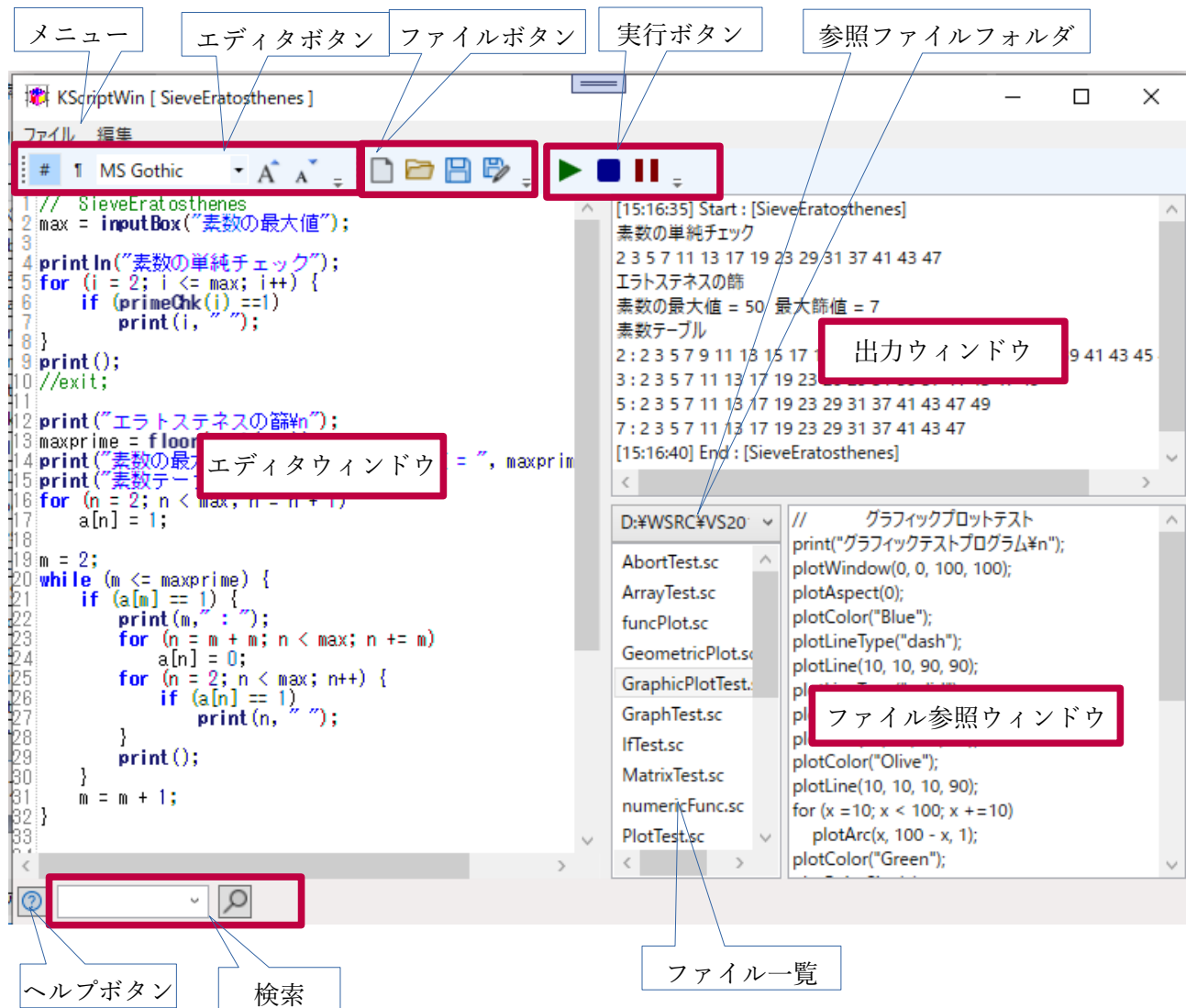


目次

1. 画面の説明.....	4
2. プログラムの入力と実行.....	7
プログラムの入力.....	7
プログラムの実行.....	7
プログラムの一時停止と中断.....	7
ファイル参照.....	7
入力候補.....	8
3. スクリプト文.....	9
4. スクリプト文の書き方.....	10
スクリプト文の書き方.....	10
コメント.....	11
ローカル変数.....	11
グローバル変数.....	11
配列.....	11
2次元配列.....	12
演算子.....	12
複合演算子.....	13
比較演算子.....	13
制御文.....	14
関数の定義.....	16
ディレクティブ.....	16
数式処理以外の関数.....	16
Win 版固有の関数.....	17
数式で使う関数.....	18
5. 配列関数.....	21
関数の種類.....	21
6. マトリックス関数.....	22
関数の種類.....	22
7. グラフィック機能.....	23
ウィンドウの設定.....	23
点の表示.....	24
線分の表示.....	24
円弧の表示.....	24

文字列の表示.....	24
グラフィック関数.....	25
8. 3D グラフィック機能.....	26
3次元データ.....	26
3D グラフィック関数.....	27
3D 要素表示関数.....	27
3D 要素座標変換関数.....	30
3D 座標データの座標変換関数.....	31
3D 座標データ作成関数.....	31

1. 画面の説明



・メニュー

ファイル

新規

エディタウィンドウをクリアし新規スクリプトの作成

開く

ファイル一覧からスクリプトファイルを読み込む

保存

編集したスクリプトをファイルに保存

名前を付けて保存

編集したスクリプトをファイル名を指定してに保存

編集

切り取り

選択した文字列を切り取る

コピー

選択した文字列をクリップボードにコピーする

貼付け

クリップボードの文字列をカーソル位置に挿入

検索	下部検索ボックスの文字列を検索してカーソルを移動
置換え	未作成
コメント追加/解除	選択した行のコメント化またはコメント解除
タブ→スペース	選択行のタブコードをスペースに変換

・エディタボタン

行番号(#)	行番号の表示/非表示
行末記号(¶)	行末記号の表示/非表示
文字フォント	エディタの文字フォントの選択
拡大	エディタの文字の拡大
縮小	エディタの文字の縮小

・ファイルボタン

新規	エディタウィンドウをクリアし新規スクリプトの作成
開く	ファイル一覧からスクリプトファイルを読み込む
保存	編集したスクリプトをファイルに保存
名前を付けて保存	編集したスクリプトをファイル名を指定してに保存

・実行ボタン

実行	スクリプトの実行
中断	実行中のスクリプトの中断
一時停止	実行中のスクリプトの一時停止、再度押すと再開

・エディタウィンドウ

スクリプトを作成するウィンドウ
通常のエディタの機能以外に関数などの入力候補機能を追加

・出力ウィンドウ

print 文のメッセージを出力するウィンドウ

・参照ファイルフォルダ

参照したいファイルのフォルダの選択
マウスでダブルクリックするとフォルダの登録ができる

・参照ファイル一覧

指定されたフォルダのファイルの一覧を表示し選択したテキストファイルを参照ファイルウィンドウに表示する
マウスの右ボタンでコンテキストメニューを表示
コンテキストメニューではファイルのコピー、移動、削除、ファイル名の変更ができる

・参照ファイルウィンドウ

参照用のテキストファイルを表示するウィンドウ

・検索

入力した文字列の検索

・キー入力

F5	プログラム実行
----	---------

F8	入力候補
Apps(右 Win キー)	入力候補
Ctrl + C	コピー
Ctrl + D	一時停止
Ctrl + F	検索
Ctrl + N	新規
Ctrl + O	開く
Ctrl + S	上書き保存
Ctrl + V	貼付け
Ctrl + X	切取り
Ctrl + Y	リドゥ
Ctrl + Z	アンドゥ
Ctrl + ‘/’	コメント追加/解除

2. プログラムの入力と実行

プログラムの入力

プログラムの入力は左のエディタウィンドウから入力する。

エディタウィンドウの文字の種類や大きさは上ツールバーの左側のボタンで変更できる。

同じくツールバーの左側のボタンの中には行番号の有無、行末記号の有無の切替もできる。

入力する時に関数名などがよくわからない場合、関数名の一部の文字や関連する文字を入力して F8 キーまたは右 Win キー(Apps キー)を押すと関係する関数名などが表示される。

例えば平方根の `sqrt()` 関数を入力したい時は `sq` と入力して F8 キーまたは平方入力して F8 キーを押すといくつかの入力候補が表示されるのでその中から選択する。

その他にカット(Ctrl+X)、コピー(Ctrl+C) & ペースト(Ctrl+V) やアンドゥ(Ctrl+Z)、リドゥ(Ctrl+Y)が使用できる。

プログラムの実行

入力したプログラムの実行は F5 キーを押すかツールバーの右側の▶ボタンを押すと実行される。

<pre>a = 1; b = a + 3; println("a, b = ", a, ", ", b);</pre>		<pre>[09:06:28] Start : [test1] a, b = 1, 4 [09:06:28] End : [test1]</pre>
--	---	--

プログラムの実行結果は開始時間と終了時間を加えて右上の出力ウィンドウに表示される。

プログラムの一時停止と中断

プログラムを実行して無限ループなどに入っ場合など処理を中断させた場合には、上部ツールバーの右側の中断ボタンまたは一時停止ボタン(Ctrl+D)を押して中断させることができる。

一時停止ボタンの場合は再度ボタンを押すと処理が再開される。

ファイル参照

他のファイルを参照しながらプログラムを入力する場合には、右下のファイル参照ウィンドウにファイルを表示して参照することができる。

ファイルの選択はファイル参照ウィンドウの左上の参照フォルダのコンボボックスに登録してあるディレクトリを選択しその下のファイルリストでファイルを選択して表示する。

参照フォルダにディレクトリを追加する時は参照フォルダのコンボボックスをマウスでダブルクリックするとフォルダの選択ダイアログが表示されるのでそれでフォルダを選択して追加する。

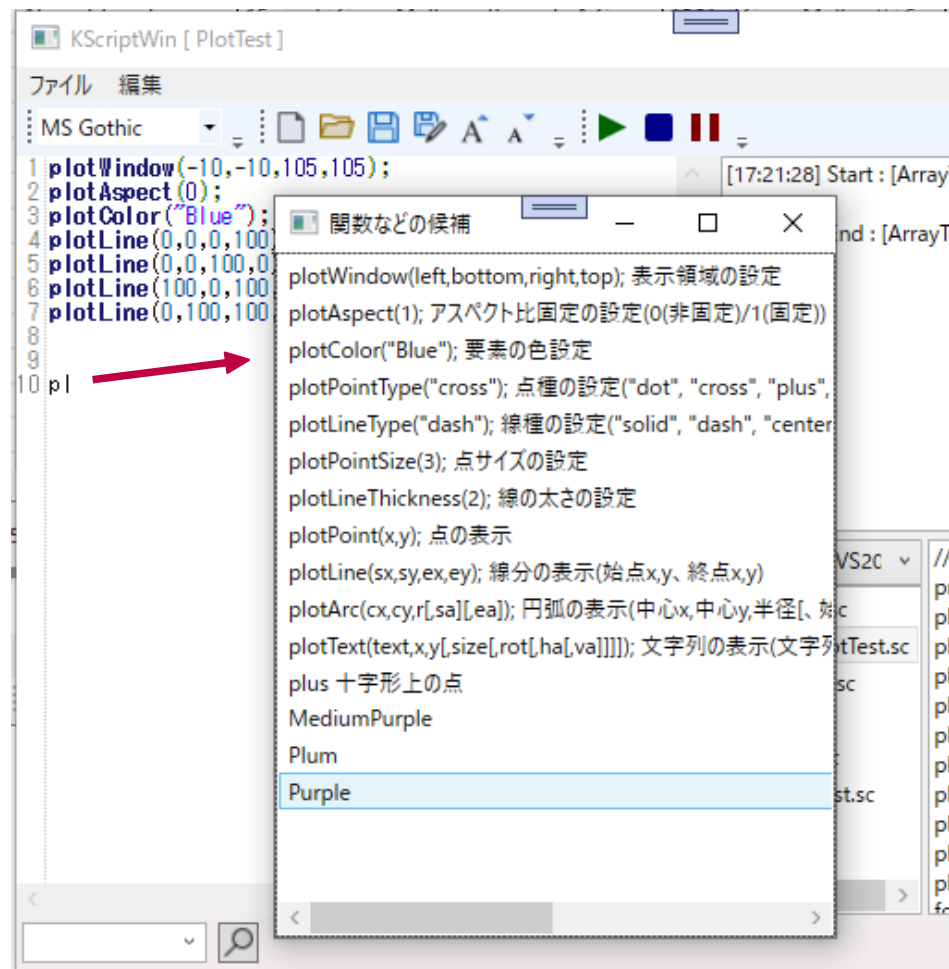
入力候補

エディタウィンドウで入力した文字から検索した関数名や定数などを説明文と合わせて表示し、該当の選択して入力する。

検索対象文字は関数名だけではなく、説明文に含まれる文字も指定できる。

使用方法是検索したい文字列の入力途中で Apps(右側 Win キー) または F8 キーを押すと該当する関数名などが表示される。

下図では pl と入力した後に F8 キーを押すとメニューダイアログが表示されこの中から選択する。



・ #include 文がある場合

インクルードするファイルから関数名とコメントを取り込んで入力候補に追加される。

下記の場合は 「roundTop(x) 最上位の桁で丸める」

新規に #include を追加した場合にはスクリプト文を再読み込みすると入力候補に出てくる。

// 最上位の桁で丸める

roundTop(x) {

 w = floor(log(x));

 return round(x, 10^w);

}

3. スクリプト文

C 言語風スクリプト言語

変数 : 型指定なし 実数または文字列(ダブルクォーテーション(")で囲む)

```
a = 12.5;  
a = "abcd";
```

配列 : 1 次元配列 a[2] = 25;
 a[m] = "abcd";
 fruit["apple"] = 50;

2 次元配列 a[1,2] = 30;
 a[m,n] = 10;

配列一括設定 a[] = { 1, 1, 5, 8 }
 b[,] = { { 1, 2, 3 }, { 2, 3, 4 } }

グローバル変数: 変数名の先頭が “g_” で始まるものはグローバル変数でそれ以外がローカル変数になる。

```
g_abcd = 2.5;
```

識別子: = (代入文) ; (文末)

演算子: * , - , * , / , %(剰余) , ^ (べき乗) , ++ , -- , += , -= , *= , /= , ^=

関係演算子: == , != , < , > , <= , >=

論理演算子: && , || , !

制御文: 代入文: 変数/配列 = 変数|配列|値|文字列|関数|数式;

if 文: if (条件) { 実行文 } [else { 実行分 }]

while 文: while (条件) { 実行文 }

for 文: for (初期値; 条件; 更新処理) { 実行文 }

return 文: return 戻り値;

break 文: break;

continue 文: continue;

exit 文: exit;

関数定義: 関数名(引数 1, 引数 2...) { 実行分; ... return 戻り値; }

プログラムの構造

```
main() {                   main 関数(省略可)  
    a = 3;                 代入文  
    b = 2;  
    c = add(a, b);         関数呼び出し
```

```

        println(a + b, " ", c);
    }
    add(c, d) {                関数定義
        return c + d;
    }

```

4. スクリプト文の書き方

スクリプト文の書き方

スクリプト文(プログラム) は一つの main 関数と複数の sub 関数で構成される。

main 関数の main() { } は省略することもできる。

その中で使用する変数のうちローカル変数はその関数内ではしか使用できないがグローバル関数(g_で始まる変数)はすべての関数を通して使用できる。

```

// main 関数
main() {
    c = sub(a,b);
    println(c);
}

```

```

// main 関数
// main() { } は省略可能
c = sub(a,b);
println(c);

```

```

// 関数
sub(d, e) {
    return d - e;
}

```

:

```

// 関数
add(d, e) {
    return d + e;
}

```

関数内の文は基本的にはセミコロン(;)で終了する。

```
a = b + c;
```

{ } で囲まれた所には複数の文を書くことができる。

・ { } なしは 1 文のみ

```

if (a < b)
    println(a);
else
    println(b);

```

・ { } ありは複数文

```

if (a < b) {
    a = a + 2;
    println(a);
} else {
    b = b - 2;
    println(b);
}

```

コメント

// から行末までをコメントアウトする

/* から */ もコメントアウトする場合複数行になってもかまわない

1行のみコメント化

```

// コメント
a = a + 2;

```

以下はすべてコメント

```

/*
a = a + 2;
b = a * 4;
*/

```

ローカル変数

値を格納するもので特に型指定の必要はなく値としては実数と文字列が使える

数値と文字列を足すと文字列として結合する(日本語可))

```

a = 3;
b = "abcd";
c = a + b;
print(c);    → "3abcd"

```

```

住所 = "東京都";
println(住所);    → 東京都

```

グローバル変数

変数名の先頭が “g_” で始まるものはグローバル変数となる

```

g_val = 1;
val = 1;
inc();
println(g_val, ", ", val);    →      2, 1

```

```

inc() {
    g_val++;
    val++;
}

```

配列

複数のデータをまとめて扱うもので [] 中の数値を変えて値を格納する

配列のインデックスには文字列も使用できる。

配列の大きさは設定する必要はなく、配列にデータを入れることによって追加されていく。

```
a[0] = 10;
a[1] = 20;
a[2] = 30;
a["apple"] = 10;
```

値をまとめて設定するときは { } で括ってカンマ区切りで入力する

配列のインデックスには文字列も使用でき、値も文字列と数値の混在も可能

一括で値を設定する時は配列のインデックスは 0 から始まる

```
a[] = { 10, 20, 30 };
a[] = { "name", 18, "men"};
println(a[0], ",", a[1], ",", a[2]);    → name, 18, men
```

2次元配列

2次元配列に一括でデータを設定する場合には { } を2重に囲って行う。

```
a[,] = { { 1, 2, 3 }, {10, 20, 30} };
println(a[1,0], " ", a[1,1], " ", a[1,2]);    → 10, 20, 30
```

行ごとに設定する場合

```
a[2,] = { 100, 200, 300 };
for (i = 0; i < count(a[2,]); i++)
    print(a[2,i], " ");    → 100, 200, 300
```

配列のインデックスに文字列を使用

```
person["yamada",] = { "tarou", 68, "men" };
println(person["yamada",0], " ", person["yamada",1], " ", person["yamada",2,]);
    → tarou, 68, men
```

```
person["山田", "名"] = "太郎";
person["山田", "年齢"] = 68;
person["山田", "性別"] = "男";
println(person["山田", "名"]);    → 太郎
```

```
item[] = { "名", "年齢", "性別" };
for (i = 0; i < count(item[]); i++) {
    print(person["山田", item[i]], " ");
}    → 太郎 68 男
```

演算子

数式を処理するための演算子

```
+ : 加算    a = 2 + 3;
- : 減算    a = 3 - 2;
```

* : 乗算 `a = 2 * 3;`
/ : 除算 `a = 2 / 3;`
% : 剰余 `a = 2 % 3;` (2 / 3 の余り → 2)
^ : べき乗 `a = 2 ^ 3;` ($2^3 \rightarrow 8$)

複合演算子

2 つの演算子を組み合わせて演算をおこなう

`+=` : `a += 3;` (`a = a + 3` と同じ)
`-=` : `a -= 3;` (`a = a - 3` と同じ)
`*=` : `a *= 3;` (`a = a * 3` と同じ)
`/=` : `a /= 3;` (`a = a / 3` と同じ)
`%=` : `a %= 3;` (`a = a % 3` と同じ)
`^=` : `a ^= 3;` (`a = a ^ 3` と同じ)
`++` : `a++;` インクリメント (`a = a + 1` と同じ)
`--` : `a--;` デクリメント (`a = a - 1` と同じ)

比較演算子

条件文の中で 2 つの値を比較する演算子 (`if` 文, `while` 文, `for` 文の条件文で使用)

条件文の中に数値や文字が設定された場合は常に `true` として処理される。

`==` : 等しい
`!=` : 等しくない
`<` : 小さい
`>` : 大きい
`<=` : 小さいか等しい
`>=` : 大きいか等しい
`if (a == b) print(a);`

条件文の中に数値や文字が設定された場合は常に `true` として処理される。

`if (1) println("true"); else println("false");` → `true`

数値の前に `!` をつけると `false` となる

`if (!1) println("true"); else println("false");` → `false`

論理演算子

比較演算子の結果を演算する

`&&` : `a && b` a かつ b
`||` : `a || b` a または b
`!` : `!a` a ではない

※ 論理演算子と比較演算子が混在する場合、優先順位がつかないので括弧をつけて優先順位をつける

`a > 10 && a < 20` → `(a > 10) && (a < 20)`

`a = 5;`
`if ((4 < a) && (a < 8))`

```

    println("5 以上 7 以下");
else
    println("4 以下か 8 以上");

```

→ 5 以上 7 以下

制御文

if 文 : 条件によって分岐させる

実行分が複数の場合は { } で囲む必要があるが、単独の場合は省略も可能

if (条件) { 実行文; } [else { 実行文; }]

if (a > b) **println**(a); a より b が大きければ a の値を表示

if (a > b) **println**(a); a より b が大きければ a の値を表示
else println(b); そうでなければ b の値を表示

if (a > b) {
 println(a); a より b が大きければ a の値を表示
} **else if** (a == b) {
 println(a); a と b が同じときは a と b の値を表示
 println(b);
} **else** {
 println(b); そうでなければ b の値を表示
}

while 文 : 条件に入っている間繰り返す

while (条件) { 実行文; }

```

n = 0;
while (n < 10) {
    n = n + 1;
    print(n, " ");
}

```

→ 1 2 3 4 5 6 7 8 9 10

```

//      配列の中から s 以上の数を検索
a[] = { 6, 9, 12, 7, 2, 23, 10, 4 };
s = 15;
n = 0;
while ((n < count(a[])) && (a[n++] < s));
println(n, " ", a[n]);

```

→ 6 10 (検索された配列の次の値を表示)

```

n = -1;
while ((n < count(a[])) && (a[++n] < s));
println(n, " ", a[n]);

```

→ 5 23 (検索された値を表示)

for 文 : 初期値、条件、増分を指定して繰り返す

for (初期値; 条件; 増分) { 実行文; }

```
for (i = 0; i < 10; i++)  
    print(i, " ");
```

→ 0 1 2 3 4 5 6 7 8 9

```
for (i = 0; i < 10; i += 2)    増分を 2 とした場合  
    print(i, " ");
```

→ 0 2 4 6 8

return 文 : 処理を中断して呼出し元に戻る

return (戻り値);

```
add(a,b) {  
    return a+b;  
}
```

```
max(a,b) {  
    if (a>b)  
        return a;  
    else  
        return b;  
}
```

break 文 : ループの中断

```
n = 0;  
while (1) {                // 条件文に数値を入れると常に条件が成立  
    if(10 < n) break;  
    n++;  
}
```

continue 文 : ループの先頭に戻る

```
n = 0;  
while (n++ < 10) {  
    if(n < 6) continue;    6 未満の場合は表示しない  
    print(n, " ");  
}
```

→ 6 7 8 9 10

exit 文 : プログラムを終了させる

```
a= 1;  
b = a+2;  
exit;                ここでプログラムを打ち切る  
println(b);
```

関数の定義

複数の文をまとめて実行する仕組み(日本語可)

関数名(引数 1, ...) { 実行文; ... return 返値; }

```
add(a, b) {  
    c = a + b;  
    return c;  
}
```

呼出し側

```
s = 2;  
t = 3;  
sum = add(s, t);
```

引数に配列を使う場合

```
sum(a[]) {  
    size = arraySize(a[]);  
    sum = 0;  
    for (i = 0; i < size; i++) {  
        sum += a[i];  
    }  
    return sum;  
}
```

呼出し側

```
array[] = { 1, 2, 3, 4, 5 }  
total = sum(array[]);
```

ディレクティブ

#include 指定したスクリプトファイルを追加読み込みをおこなう。

作成した汎用の関数を使用するときに使う。

#include “スクリプトファイル名”;

#include “scriptLib.sc”;

フルパスを指定しない場合には実行するスクリプトファイルと同じディレクトリが指定される。

数式処理以外の関数

print(a, b, "\n");	: コンソールに文字出力(Win 版では出力ウィンドウに出力)
println(a, b);	: print と同じで末尾に改行コードがつく
a = input();	: コンソールからの文字列入力(Win では使用不可、代わりに inputBox() を使用)
a = inKey();	: キー入力(1 文字)
sleep(n);	: 指定時間停止(約 n m 秒)
cmd(command);	: Windows コマンドの実行 ex. cmd(“dir”);, cmd("KScriptWinManual.pdf");

配列関数(array.xxxx())

c = array.contain(a[2]);	: 配列の有無(0:なし 1:あり)
size = array.count(a[]);	: 配列のサイズ
size = array.count(a[,]);	: 2次元配列の全体の数
size = array.count(a[1,]);	: 2次元配列(2行目)の列数
max = array.max(a[]);	: 配列の最大値
min = array.min(a[,]);	: 配列の最小値
sum = array.sum(a[]);	: 配列の合計
ave = array.average(a[,]);	: 配列の平均
vari = array.variance(a[]);	: 分散
std = array.stdDeviation(a[]);	: 標準偏差
cov = array.covariance(a[, b[]];	: 共分散
corr = array.corrCoeff(x[,y[]];	: 配列の相関係数
array.clear(a[]);	: 配列をクリア
array.remove(a[,st[,ed]);	: 配列の要素削除(1次元のみ)
array.sort(a[]);	: 配列のソート(1次元のみ)
array.reverse(a[]);	: 配列の逆順化(1次元のみ)
c[] = array.concat(a[,b[]];	: 配列の結合

マトリックス関数(matrix.xxxx())

a[,] = matrix.unit(size);	: 単位行列
d[,] = matrix.transpose(a[,]);	: 転置行列
d[,] = matrix.multi(a[,], b[,]);	: 行列の積
d[,] = matrix.add(a[,], b[,]);	: 行列の和
d[,] = matrix.inverse(a[,]);	: 逆行列 a^{-1}
d[,] = matrix.copy(a[,]);	: 行列のコピー

Win 版固有の関数

a = inputBox();	: 文字入力ダイアログ
messageBox(outString[, title]);	: 文字列のダイアログ表示
n = menuSelect(menu[,title];	: 選択メニューを表示して選択項目 No を返す

グラフィック関数(plot.xxxx(), graph.xxxx())

plot.Window(left,bottom,right,top);	: 表示領域の設定",
plot.Aspect(1);	: アスペクト比固定の設定(0(非固定)/1(固定))
plot.Color("Blue");	: 要素の色設定
plot.PointType("cross");	: 点種の設定("dot", "cross", "plus", "box", "circle", "triangle")
plot.LineType("dash");	: 線種の設定("solid", "dash", "center", "phantom")
plot.PointSize(3);	: 点サイズの設定
plot.LineThickness(2);	: 線の太さの設定
plot.Point(x,y);	: 点の登録
plot.Line(sx,sy,ex,ey);	: 線分の登録(始点 x,y、終点 x,y)

<code>plot.Arc(cx,cy,r[,sa][,ea]);</code>	: 円弧の登録(中心 x,中心 y,半径[, 始角][、終角])
<code>plot.Text(text,x,y[,size[,rot[,ha[,va]]]]);</code>	: 文字列の登録(文字列,X 座標,Y 座標,サイズ,回転角,水平 アライメント,垂直アライメント)
<code>plot.Disp();</code>	: グラフィックデータを表示
<code>graph.Set(x[,y][[,Title]);</code>	: グラフデータの設定(X[,Y][[,Title])
<code>graph.FontSize(5);</code>	: グラフのフォントサイズの設定"

3D グラフィック関数(plot3D.xxxx())

<code>plot3D.setArea(min[,max]);</code>	3DView の初期化と領域設定
<code>plot3D.setAxisFrame(axis,frame);</code>	軸とフレームの表示(0:非表示/1:表示)
<code>plot3D.disp();</code>	要素の表示
<code>plot3D.setColor("\Blue");</code>	色の設定
<code>plot3D.plotTranslate(vec[]);</code>	移動(移動量 vec[x/y/z])
<code>plot3D.plotRotate(angle,"\X");</code>	回転(回転角度,回転軸 X/Y/Z 軸)
<code>plot3D.plotScale(scale[]);</code>	縮小拡大 (拡大率 scale[x/y/z])
<code>plot3D.plotReset();</code>	座標変換マトリックスをクリア
<code>plot3D.plotPush();</code>	座標変換マトリックスをスタックに保存
<code>plot3D.plotPop();</code>	座標変換マトリックスをスタックから戻す
<code>plot3D.plotPeekMulti();</code>	座標変換マトリックスにスタックの値をかける
<code>plot3D.plotLine(sp[x/y/z],ep[x/y/z]);</code>	線分の設定
<code>plot3D.plotLines(plist[n,x/y/z]);</code>	複数線分
<code>plot3D.plotPolyline(plist[n,x/y/z]);</code>	ポリライン
<code>plot3D.plotPolyloop(plist[n,x/y/z]);</code>	ポリラインループ
<code>plot3D.plotPolygon(plist[n,x/y/z]);</code>	ポリゴン(塗潰し)
<code>plot3D.plotTriangles(plist[n,x/y/z]);</code>	複数三角形(塗潰し)
<code>plot3D.plotTriangleStrip(plist[n,x/y/z]);</code>	連続三角形
<code>plot3D.plotQuads(plist[n,x/y/z]);</code>	複数四角形
<code>plot3D.plotQuadeStrip(plist[n,x/y/z]);</code>	連続四角形
<code>plot3D.plotTriangleFan(plist[n,x/y/z]);</code>	扇形の連続三角形
<code>plot3D.translate(pos[,vec[]);</code>	3D 座標の移動(3D 座標,移動量)
<code>plot3D.rotate(pos[,cp[,angle,face];</code>	3D 座標の回転(3D 座標,回転中心,回転角,回転軸)
<code>plot3D.scale(pos[,cp[,scale];</code>	3D 座標の拡大縮小(3D 座標,拡大中心,拡大率)

数式で使う関数

<code>pi = PI;</code>	: 円周率(3.14..)
<code>e = E;</code>	: 自然対数の底(2.71..)
<code>rad = RAD(deg);</code>	: 度をラジアンに変換
<code>deg = DEG(rad);</code>	: ラジアンを度に変換
<code>hour = deg2hour(deg);</code>	: 度を時単位に変換(360° → 24 時間)
<code>deg = hour2deg(hour);</code>	: 時単位を度に変換(1 時間 → 15°)
<code>hour = rad2hour(rad);</code>	: ラジアンを時単位に変換(PI → 12 時間)

rad = hour2rad(hour);	: 時単位をラジアンに変換(12 時間 → PI)
a = mod(x,y);	: 剰余(x / y の余り)
a = pow(x,y);	: 累乗 (x^y)
a = max(x,y);	: x と y の大きい方
a = min(x,y);	: x と y の小さい方
a = combi(n,r);	: 組合せの数(nCr)
a = permu(n,r);	: 順列の数(nPr)
a = sin(x);	: 正弦
a = cos(x);	: 余弦
a = tan(x);	: 正接
a = asin(x);	: 逆正接
a = acos(x);	: 逆余弦
a = atan(x);	: 逆正接
a = atan2(x,y);	: 逆正接
a = sinh(x);	: 双曲線正弦
a = cosh(x);	: 双曲線余弦
a = tanh(x);	: 双曲線正接
a = asinh(x);	: 逆双曲線正弦
a = acosh(x);	: 逆双曲線余弦
a = atanh(x);	: 逆双曲線正接
a = exp(x);	: e の累乗
a = ln(x);	: e を底とする自然対数
a = log(x);	: 10 を底とする対数
a = log(x,y);	: x を底とする y の対数
a = sqrt(x);	: 平方根
a = abs(x);	: 絶対値
a = ceil(x);	: 切上げ(x 以上で最小の整数値)
a = floor(x);	: 切捨て(小数点以下の数の内最大の整数値)
a = round(x);	: 四捨五入(もっとも近い整数値)
a = trunc(x);	: 浮動小数点の整数部
a = sign(x);	: 符号示す値(1/0/-1)
a = round(x,y);	: x を y の倍数に丸める
a = equals(x,y);	: 等価判定 $x == y \Rightarrow 1, x != y \Rightarrow 0$
a = lt(x,y);	: 大小判定(less than) $x > y \Rightarrow 1$, 以外は 0
a = gt(x,y);	: 大小判定(greater than) $x < y \Rightarrow 1$, 以外は 0
a = compare(x,y);	: 大小判定 $x > y \Rightarrow 1, x == y \Rightarrow 0, x < y \Rightarrow -1$
a = deg2dms(x);	: 度(ddd.dddd) → 度分秒(ddd.mmss)
a = dms2dig(x);	: 度分秒(ddd.mmss) → 度(ddd.dddd)
a = hour2hms(x);	: 時(hh.hhhh) → 時分秒(hh.mmss)
a = hms2hour(x);	: 時分秒(hh.mmss) → 時(hh.hhhh)
a = fact(x);	: 階乗 ($a = x * (x-1) * \dots * 2 * 1$)
a = fib(x);	: フィボナッチ数列 ($F_n = F_{n-1} + F_{n-2}$, $F_2 = 1, F_1 = 0$)

$a = \text{gcd}(x,y);$: 最大公約数
 $a = \text{lcm}(x,y);$: 最小公倍数
 $a = \text{JD}(y,m,d);$: 西暦年月日からユリウス日を求める
 $a = \text{MJD}(y,m,d);$: 西暦年月日から準ユリウス日を求める
 $a = \text{JD2Date}(x);$: ユリウス日を年月日に変換して `yyyymmdd` の実数にする
 $a = \text{sum}(f([\text{@}]),n,k);$: 級数の和 n から k まで連続し値を計算式 $f([\text{@}])$ で演算した値の合計を求める
 $(\text{sum}([\text{@}]*[\text{@}],1,10) \rightarrow 1*1+2*2+\dots+10*10 \rightarrow 385)$
 $a = \text{sum}(f([\text{@}]),n1,n2\dots nm);$: 級数の和 $n1$ から nm まで値を計算式 $f([\text{@}])$ で演算した値の合計を求める
 $(\text{sum}([\text{@}]^{\textcolor{blue}{2}},\textcolor{blue}{1},\textcolor{blue}{2},\textcolor{blue}{3},\textcolor{blue}{4},\textcolor{blue}{5}) \rightarrow 1^2+2^2+3^2+4^2+5^2 \rightarrow 55)$
 $a = \text{product}(f([\text{@}]),n,k);$: 級数の積 n から k まで連続し値を計算式 $f([\text{@}])$ で演算した値の積を求める
 $(\text{product}([\text{@}],\textcolor{blue}{1},\textcolor{blue}{5}) \rightarrow 120)$
 $a = \text{product}(f([\text{@}]),,n1,n2\dots nm);$: 級数の積 $n1$ から nm まで値を計算式 $f([\text{@}])$ で演算した値の積を求める
 $(\text{product}([\text{@}]^{\textcolor{blue}{2}},\textcolor{blue}{1},\textcolor{blue}{2},\textcolor{blue}{3},\textcolor{blue}{4},\textcolor{blue}{5}) \rightarrow 14400)$
 $a = \text{repeat}(f([\text{@}],[\text{ \% }]),i,n,k);$: 計算式の $[\text{@}]$ に n から k まで入れて繰り返す, $[\text{ \% }]$ に計算結果が入る, i は $[\text{ \% }]$ の初期値
 $(\text{repeat}([\text{ \% }]*[\text{@}],\textcolor{blue}{10},\textcolor{blue}{2},\textcolor{blue}{6}) \rightarrow (((((10*2)*3)*4)*5)*6 \rightarrow 7200)$
 例 : 元本 1 万円を年利 2% で 5 年預けた場合の金額
 $\text{repeat}([\text{ \% }]*1.02,10000,1,5) \Rightarrow$
 $(((((10000*1.02)*1.02)*1.02)*1.02))*1.02 = 11040.808$

5. 配列関数

array. で始まる関数は配列に関する関数

使用できる配列の種類はインデックスが数値で 0 以上のものに限る。


関数の種類


<code>a = array.contains(c[2]);</code>	配列内で存在を調べる関数 (a = 0 : 存在しない a = 1 : 存在する)
<code>size = array.count(a[]);</code>	1 次元配列のサイズ
<code>size = array.count(a[,]);</code>	2 次元配列の全体サイズ
<code>size = array.count(a[m,]);</code>	2 次元配列 m 行目のサイズ(列数)
<code>size = array.count(a[,n]);</code>	2 次元配列 n 列目のサイズ(行数)
<code>c[,] = array.concat(a[,],b[,]);</code>	配列同士を結合して新しい配列にする
<code>array.clear(a[]);</code>	配列データをクリア
<code>array.remove(a[,start[,end]]);</code>	1 次元配列要素を範囲指定で削除
<code>array.squeeze(a[]);</code>	配列の未使用データを削除圧縮
<code>array.sort(a[]);</code>	1 次元配列のソート
<code>array.sort(a[,n]);</code>	2 次元配列を n 列でソート
<code>array.reverse(a[]);</code>	1 次元配列の順番を逆にする
<code>array.reverse(a[,]);</code>	2 次元配列の行の順番を逆にする
<code>array.reverse(a[,],1);</code>	2 次元配列の列の順番を逆にする(2 番目の引数が 1 の時)
<code>max = array.max(a[]);</code>	1/2 次元配列の最大値を求める
<code>max = array.max(a[m,]);</code>	2 次元配列の m 行目の最大値を求める
<code>min = array.min(a[]);</code>	1/2 次元配列の最小値を求める
<code>min = array.min(a[m,]);</code>	2 次元配列の m 行目の最小値を求める
<code>sum = array.sum(a[]);</code>	1 次元配列の合計
<code>ave = array.average(a[]);</code>	1 次元配列の平均
<code>ave = array.average(a[,]);</code>	2 次元配列の全体の平均
<code>ave = array.average(a[m,]);</code>	2 次元配列の m 行目の平均
<code>vari = array.variance(a[]);</code>	配列の分散
<code>std = array.stdDeviation(a[]);</code>	配列の標準偏差
<code>cov = array.covariance(a[,], b[,]);</code>	共分散
<code>coor = array.corrCoeff(x[,],y[,]);</code>	配列の相関係数


6. マトリックス関数


matrix. で始まる関数は行列の演算処理を行う関数


関数の種類


matrix.unit(size); 単位行列(2次元)の作成
a[,] = matrix.unit(3);  $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

matrix.transpose(a[,]); 転置行列(2次元行列 A の転置(A^T))
a[,] = {
 { 1, 2, 3 },
 { 4, 5, 6 },
 { 7, 8, 9 }
}
d[,] = matrix.transpose(a[,]);  $\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$

matrix.multi(a[,], b[,]); 行列の積 $A \times B$ ($c[,] = \dots$)
b[,] = {
 { 1, 2 },
 { 4, 5 },
 { 7, 8 }
}
d[,] = matrix.multi(a[,], b[,]);  $\begin{pmatrix} 30 & 36 \\ 66 & 81 \\ 102 & 126 \end{pmatrix}$

matrix.add(a[,], b[,]); 行列の和 $A+B$
e[,] = {
 { 10, 20, 30 },
 { 40, 50, 60 },
 { 70, 80, 90 }
}
d[,] = matrix.add(a[,], e[,]);  $\begin{pmatrix} 11 & 22 & 33 \\ 44 & 55 & 66 \\ 77 & 88 & 99 \end{pmatrix}$

matrix.inverse(a[,]); 逆行列 A^{-1}
a[,] = {{1,2},{3,4}};
b[,] = matrix.inverse(a[,]);  $\begin{pmatrix} -2 & 1 \\ 1.5 & -0.5 \end{pmatrix}$

matrix.copy(a[,]); 行列のコピー
d[,] = matrix.**copy**(a[,]);  $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

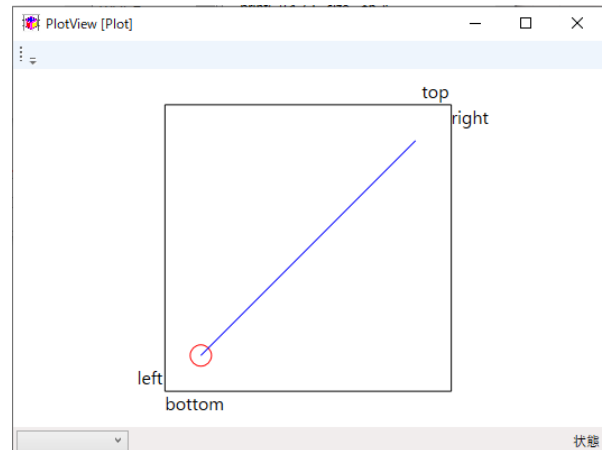
7. グラフィック機能

ウィンドウの設定

グラフィック関数を使うと別ウィンドウにグラフィック表示を行う。

グラフィック・ウィンドウは `plot.Window(left,bottom,right,top)` によってワールド座標で設定される。

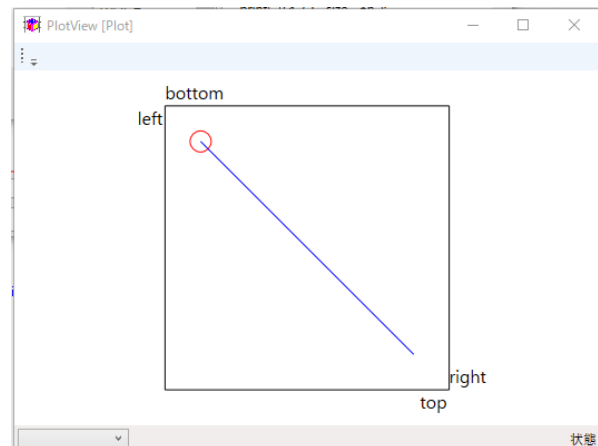
```
plot.Aspect(1);
plot.Window(0,0,100,100);
plot.Color("Red");
plot.PointType("circle");
plot.PointSize(5);
plot.Point(20,20);
plot.Color("Blue");
plot.Line(20,20,80,80);
plot.Color("Black");
plot.Line(10,10,10,90);
plot.Line(90,10,90,90);
plot.Line(10,10,90,10);
plot.Line(10,90,90,90);
fontSize = 5;
plot.Text("left",10,10,fontSize,0,2,2);
plot.Text("bottom",10,10,fontSize,0,0,0);
plot.Text("right",90,90,fontSize,0,0,0);
plot.Text("top",90,90,fontSize,0,2,2);
```



上図は `bottom < top` , `left < right` で設定した場合で `bottom > top` とすると上下が逆になる。

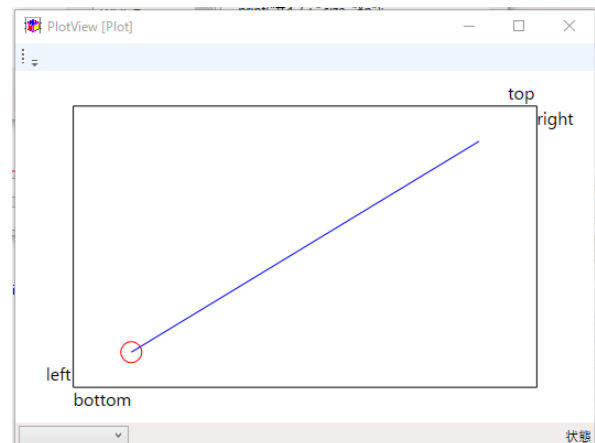
同様に `left > right` とすると左右が入れ替わる。

`plot.Window(0,100,100,0);`



また上図はアスペクト比固定の設定(縦横比が同じ、`plot.Aspect(1)`)で表示しているが非固定(`plot.Aspect(1)`)にするとウィンドウの形状に合わせた表示する。

なお `plot.Aspect()` は `plot.Window()` より前に設定しないと有効にならない。



点の表示

点を表示する場合の属性として 点種と点サイズの 2 通りがある。

- ・点種(PointType()) 点の形状 dot(点), cross(×), plus(+), box(□), circle(○), tryangle(△)
- ・点サイズ(PointSize()) 点のサイズはスクリーン座標のサイズで Window の大きさを変えても変わらない。

```
plot.PointType("circle");  
plot.PointSize(5);  
plot.Point(20,20);
```

線分の表示

線分を表示する場合、その属性として 色(plot.Color())、線種(plot.LineType())、太さ(plot.Thickness())が設定でき、次に設定されるまでその属性が有効になる。

ちなみに色はすべての要素に、線種と太さは円弧にも有効である。

次の場合は Blue、破線 で線分が表示される。

線分の色は「色」と入力した後 F8 キーまたは右 Win キー を押すと入力候補が表示され、線種も同様に入力候補を出すことができる。

ちなみに線種は solid (実線)、dash(破線)、center(一点鎖線)、phantom(2点鎖線)の4種類がある。

```
plot.Color("Blue");  
plot.LineType("dash");  
plot.Line(20,20,80,80);
```

円弧の表示

円弧を表示するときは下記の書式にしたがって行う。

開始角と終了角の単位はラジアンで省略可能である。

その場合、開始角 = 0、終了角 $= 2 \times \text{PI}$ となり円で表示される。

```
plot.Arc(x 座標, y 座標, 半径, 開始角, 終了角);
```

文字列の表示

文字列を表示する場合は次のような書式でおこなう。

- ・文字高さ ワールド座標で設定されるので Window の大きさを変えるとそれに合わせて変わる
- ・回転角 文字列の回転角を設定で 単位はラジアンで設定する。
- ・水平アライメント 文字列の水平方向の基準位置 0:左 1:中央 2:右
- ・垂直アライメント 文字列の垂直方向の基準位置 0:上 1:中央 2:下

※水平垂直のアライメントを省略すると基準位置は左上、回転角を省略すると回転角は 0° となる。

```
plot.Text("文字列", x 座標, y 座標, 文字高さ, 回転角, 水平アライメント, 垂直アライメント);
```



```
fontSize = 5;  
rotate = RAD(0);  
plot.Text("left",10,10,fontSize,rotate,2,2);
```

グラフィック関数

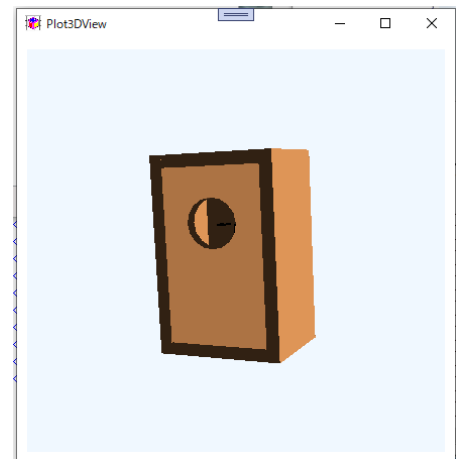
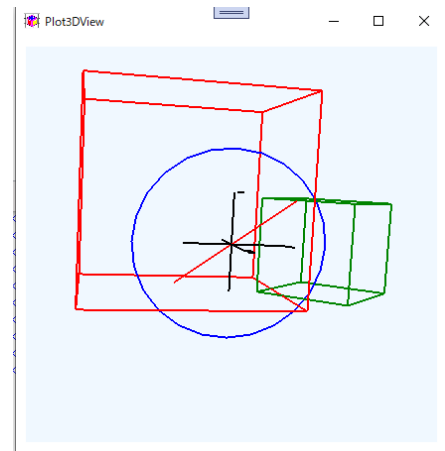
<code>plot.Window(left,bottom,right,top);</code>	: 表示領域の設定",
<code>plot.Aspect(1);</code>	: アスペクト比固定の設定(0(非固定)/1(固定))",
<code>plot.Color("Blue");</code>	: 要素の色設定",
<code>plot.PointType("cross");</code>	: 点種の設定("dot", "cross", "plus", "box", "circle", "triangle")
<code>plot.LineType("dash");</code>	: 線種の設定("solid", "dash", "center", "phantom")"
<code>plot.PointSize(3);</code>	: 点サイズの設定
<code>plot.LineThickness(2);</code>	: 線の太さの設定
<code>plot.Point(x,y);</code>	: 点の登録
<code>plot.Line(sx,sy,ex,ey);</code>	: 線分の登録(始点 x,y、終点 x,y)
<code>plot.Arc(cx,cy,r[sa][,ea]);</code>	: 円弧の登録(中心 x,中心 y,半径[、始角][、終角])
<code>plot.Text(text,x,y[,size[,rot[,ha[,va]]]]);</code>	: 文字列の登録(文字列,X 座標,Y 座標,サイズ,回転角,水平 アライメント,垂直アライメント)
<code>plot.Disp();</code>	: グラフィックデータを表示
<code>graph.Set(x[],y[][,Title]);</code>	: グラフデータの設定(X[],Y[][,Title])
<code>graph.FontSize(5);</code>	: グラフのフォントサイズの設定"

8. 3D グラフィック機能

KScriptWin では簡単な関数を使って3次元データの表示を行うことができる。

```
sp = -0.5;
ep = 0.5;
min[] = { sp, sp, sp };
max[] = { ep, ep, ep };
plot3D.setArea(min[],max[]);
plot3D.setAxisFrame(1,0);
plot3D.setColor("Red");

sp[] = { -1, -1, -1 };
ep[] = { 1, 1, 1 };
plot3D.plotLine(sp[],ep[]);
plot3D.setColor("Blue");
circle(2,24);
// 円 (半径, 角数)
circle(r, n) {
    step = 2 * PI / n;
    i = 0;
    for (ang = 0; ang < 2 * PI; ang += step) {
        plist[i,0] = r * cos(ang);
        plist[i,1] = r * sin(ang);
        plist[i,2] = 0;
        i++;
    }
    plist[i,0] = r * cos(0);
    plist[i,1] = r * sin(0);
    plist[i,2] = 0;
    plot3D.plotPolyline(plist[,]);
}
:
:
```



3次元データ

3次元の座標データは配列を使ってグラフィック関数に設定する。

3D のグラフィック関数にはこの配列データを使って行う

```
座標データ p[] = { x, y, z }
p[] = { 1.2, 2.0, 3.0 }
座標リスト plist[,] = { { x1, y1, z1 },
                        { x2, y2, z2 },
                        :
                        { xn, yn, zn } }
```

データの表示 plot3D.plotPolyline(plist);

3D グラフィック関数

`plot3D.setArea(min[],max[]);`

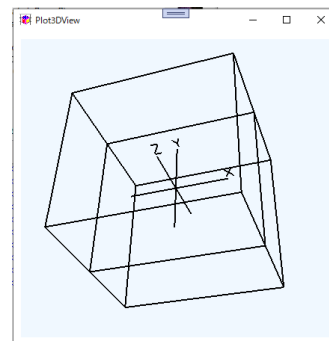
3DView の初期化と領域の設定

```
sp = -0.5; ep = 0.5;
min[] = { sp, sp, sp };
max[] = { ep, ep, ep };
plot3D.setArea(min[],max[]);
plot3D.setAxisFrame(1,0);
```

`plot3D.setAxisFrame(axis,frame);`

軸(axis)とフレーム(frame)の表示(0:非表示/1:表示)

`setArea()` で指定した領域の軸方向とフレームを表示する



`plot3D.disp();`

登録した要素の表示

`plot3D.plotLine()`等で登録したデータは通常はプログラム実行後に表示されるが実行途中の状態を表示したいときや再表示に使用する。

`plot3D.setColor("Blue");`

色の設定

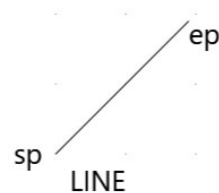
表示データの色設定を行う、「色」と入力してから F8 キーまたは右 Win キーを押すと使用できる色の一覧が表示される。

3D 要素表示関数

`plot3D.plotLine(sp[x/y/z],ep[x/y/z]);`

線分の設定

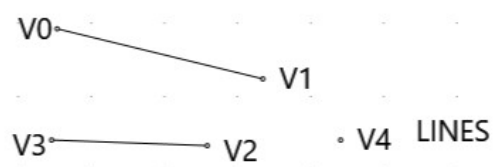
```
sp[] = { 1, 2, 2 };
ep[] = { 2, 3, 2 };
plot3D.plotLine(sp[],ep[]);
```



`plot3D.plotLines(plist[n,x/y/z]);`

複数線分(plist)

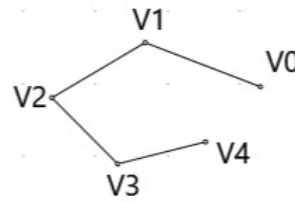
```
Plist[,] = {
    { 2, 5, 2 },
    { 6, 2, 2 },
    { 5, 1, 2 },
    { 2, 1, 2 },
}
plot3D.plotLines(plist);
```



plot3D.plotPolyline(plist[n,x/y/z]);

ポリライン

```
Plist[,] = {
    { 6, 5, 2 },
    { 4, 6, 2 },
    { 2, 4, 2 },
    { 3, 1, 2 },
    { 5, 2, 2 },
}
plot3D.plotPolyline(plist);
```

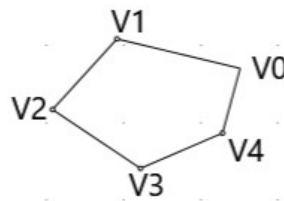


LINE_STRIP

plot3D.plotPolyloop(plist[n,x/y/z]);

ポリラインループ

```
Plist[,] = {
    { 6, 5, 2 },
    { 4, 6, 2 },
    { 2, 4, 2 },
    { 3, 1, 2 },
    { 5, 2, 2 },
}
plot3D.plotPolyloop(plist);
```

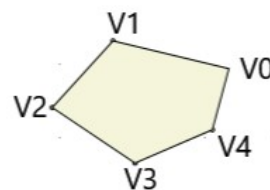


LINE_LOOP

plot3D.plotPolygon(plist[n,x/y/z]);

ポリゴン(塗潰し)

```
Plist[,] = {
    { 6, 5, 2 },
    { 4, 6, 2 },
    { 2, 4, 2 },
    { 3, 1, 2 },
    { 5, 2, 2 },
}
plot3D.plotPolygon(plist);
```

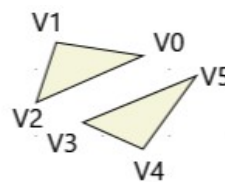


POLYGON

plot3D.plotTriangles(plist[n,x/y/z]);

複数三角形(塗潰し)

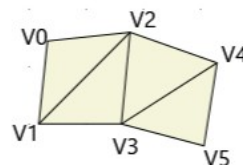
```
Plist[,] = {
    { 6, 5, 2 },
    { 4, 6, 2 },
    { 2, 4, 2 },
    { 3, 1, 2 },
    { 6, 0, 2 },
    { 7, 5, 2 },
}
plot3D.plotTriangles(plist);
```



TRIANGLES

plot3D.plotTriangleStrip(plist[n,x/y/z]); 連続三角形

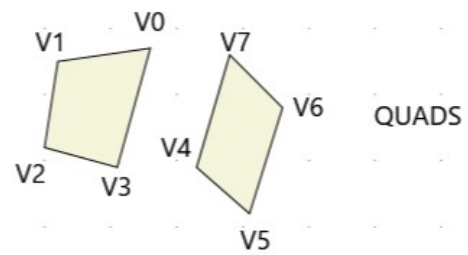
```
Plist[,] = {
    { 1, 5, 2 },
    { 1, 4, 2 },
    { 2, 5, 2 },
    { 2, 4, 2 },
    { 3, 4.5, 2 },
    { 3, 3.5, 2 },
}
plot3D.plotTriangleStrip(plist);
```



TRIANGLE_STRIP

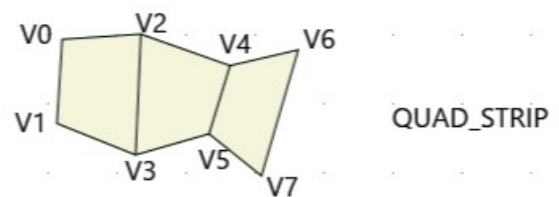
`plot3D.plotQuads(plist[n,x/y/z]);` 複数四角形

```
Plist[,] = {  
    { 5, 5, 2 },  
    { 3, 4, 2 },  
    { 1, 2, 2 },  
    { 4, 1, 2 },  
    { 6, 1, 2 },  
    { 7, -1, 2 },  
    { 8, 3, 2 },  
    { 7, 5, 2 },  
}  
plot3D.plotQuads(plist);
```



`plot3D.plotQuadeStrip(plist[n,x/y/z]);` 連続四角形

```
Plist[,] = {  
    { 0, 5, 2 },  
    { 0, 2, 2 },  
    { 2, 5, 2 },  
    { 2, 1, 2 },  
    { 5, 4, 2 },  
    { 4, 2, 2 },  
    { 7, 5, 2 },  
    { 6, 0, 2 },  
}  
plot3D.plotQuadeStrip(plist);
```



`plot3D.plotTriangeFan(plist[n,x/y/z]);` 扇形の連続三角形

```
Plist[,] = {  
    { 8, 5, 2 },  
    { 5, 6, 2 },  
    { 2, 4, 2 },  
    { 5, 1, 2 },  
    { 7, 2, 2 },  
}  
plot3D.plotTriangeFan(plist);
```



3D 要素座標変換関数

`plot3D.plotTranslate(vec[]);` 移動(移動量 `vec[x/y/z]`)

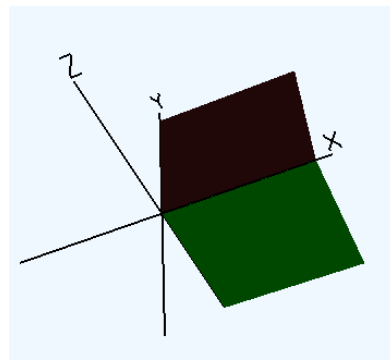
描画関数を実行する前に描画関数のデータを移動する(移動量を変換マトリックスに追加)

```
v[] = { 0, 0, d };  
plot3D.plotTranslate(v[]);  
plot3D.plotQuadeStrip(p[]);
```

`plot3D.plotRotate(angle,"X");` 回転(回転角度,回転軸["X"/"Y"/"Z"])

描画関数を実行する前に描画関数のデータを指定軸で回転する(回転データを変換マトリックスに追加)

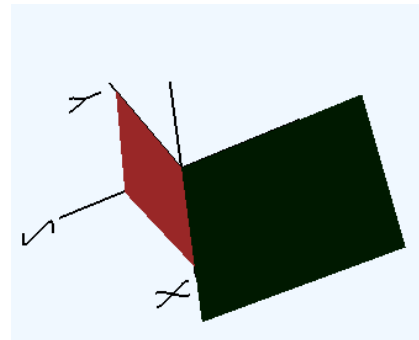
```
plot3D.setColor("Brown");  
p[] = plateData(1,1);  
plot3D.plotPolygon(p[]);  
plot3D.plotRotate(RAD(90), "X");  
plot3D.setColor("Green");  
plot3D.plotPolygon(p[]);  
  
plateData(w, h) {  
    p[0,] = { 0, h, 0 };  
    p[1,] = { w, h, 0 };  
    p[2,] = { w, 0, 0 };  
    p[3,] = { 0, 0, 0 };  
    return p[];  
}
```



`plot3D.plotScale(scale[]);` 縮小拡大 (拡大率 `scale[x/y/z]`)

表示要素を原点を中心に指定した倍率で拡大、縮小する。(拡大率を変換マトリックスに追加)

```
plot3D.setColor("Brown");  
p[] = plateData(1,1);  
plot3D.plotPolygon(p[]);  
  
sc[] = { 1.5, 1, 1.5 };  
plot3D.plotScale(sc[]);  
  
plot3D.plotRotate(RAD(90), "X");  
plot3D.setColor("Green");  
plot3D.plotPolygon(p[]);
```



`plot3D.plotReset();` 座標変換マトリックスをクリア

表示要素の移動や回転などを行う変換マトリックスの値をリセットする。

`plot3D.plotPush();` 座標変換マトリックスをスタックに保存

変換マトリックスを一時的に変更する場合などに使用する。一度スタックに保存した変換マトリックスは最後に保存した変換マトリックスから `plotPop()` で変換マトリックスが再設定される。

`plot3D.plotPop();` 座標変換マトリックスをスタックから戻す

`plotPush()` でスタックに保存された変換マトリックスを取り出し再設定する

`plot3D.plotPeekMulti();` 座標変換マトリックスにスタックの値をかける

3D 座標データの座標変換関数

`plot3D.translate(pos[,],vec[]);` 3D 座標の移動(3D 座標,移動量)

座標データの移動を行う関数で機能的には `plotTranslate()` と同じであるが `plotTranslate()` がグラフィック処理に登録したデータに対して行うのに対し、登録前の座標データに対して座標変換をおこなう。

```
cir[,] = circle(r,n);  
vec[] = { cx, cy, 0 };  
cir[,] = plot3D.translate(cir[,],vec[]);
```

`plot3D.rotate(pos[,],angle,axis);` 3D 座標の回転(3D 座標,回転角,回転軸["X"/"Y"/"Z"])

```
cir[,] = circle(r2,4);  
cir[,] = plot3D.rotate(cir[,],RAD(45),"Z");
```

`plot3D.rotateAxis(pos[,],angle,axis[]);` 3D 座標を指定軸で回転(3D 座標,回転角,回転軸ベクトル)

`plot3D.scale(pos[,],cp[,],scale);` 3D 座標の拡大縮小(3D 座標,拡大中心,拡大率)

```
cir[,] = circle(r,n);  
vec[] = { cx, cy, 0 };  
cir[,] = plot3D.translate(cir[,],vec[]);  
cir[,] = plot3D.scale(cir[,],vec[],0.5);
```

3D 座標データ作成関数

`plot3D.holePlateQuads(outline[,],innerLine[,],,);` 中抜き平面データの作成

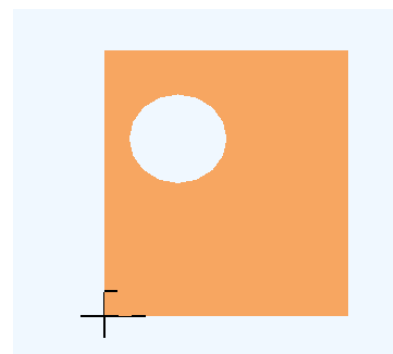
ポリゴンで囲まれた領域にポリゴンで穴が開けられた状態のデータを作成する。

外側のポリゴン領域(`outline[,]`)と内側のポリゴン領域(`innerLine[,]`)を指定する。

内側の領域は複数指定することができる。

表示する領域は複数の四角形(QUADS)に分割されており、`plot3D.plotQuads(p[,])`で表示する。

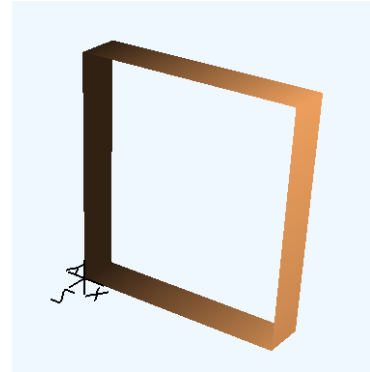
```
rect[,] = rectangle(w, h);  
cir[,] = circle(r,n);  
vec[] = { cx, cy, 0 };  
cir[,] = plot3D.translate(cir[,],vec[]);  
p[,] = plot3D.holePlateQuads(rect[,],cir[,]);  
plot3D.plotQuads(p[,]);
```



```
plot3D.polygonSideQuadStrip(polygon[,],thicknes);
```

ポリゴンの側面データを QuadStrip で作成

```
rect[,] = rectangle(w, h);  
rectSide[,] = plot3D.polygonSideQuadStrip(rect[,],-2);  
plot3D.plotQuadStrip(rectSide[,]);
```



```
plot3D.polygonSideQuads(polygon[,],thicknes);
```

ポリゴンの側面データを Quads で作成

```
plot3D.polylinRotateQuads(polyline[,],centerline[,],divAng,sa,ea);
```

ポリライNDERをセンタラインを中心に回転させたデータ

// 円データ (半径, 角数)

```
circle(r,n) {  
  if (n < 1) return pos[,];  
  step = 2 * PI / n;  
  i = 0;  
  z = 0;  
  for (ang = 0; ang < 2 * PI; ang += step) {  
    x = r * cos(ang);  
    y = r * sin(ang);  
    pos[i, 0] = x;  
    pos[i, 1] = y;  
    pos[i, 2] = z;  
    i++;  
  }  
  return pos[,];  
}
```

// 長方形データ (幅, 高さ) (x,y,0)

```
rectangle(w, h) {  
  w /= 2;  
  h /= 2;  
  p[0,] = { -w, -h, 0 };  
  p[1,] = { w, -h, 0 };  
  p[2,] = { w, h, 0 };  
  p[3,] = { -w, h, 0 };  
  return p[,];  
}
```