目次

LISP のはなし

AUTOLISP とは 3
実行方法 3
.lsp ファイルの作り方 3
.lsp ファイルの編集 4
Emacs について(自身の愛用エディタがある場合はとばしてください) 4
インターフェース 5
ファイルを開く 6
括弧が光る 6
Tab キー操作 6
ショートカットキー 8
バックアップファイルについて 11
lsp ファイルの使い方 11
AutoLisp 基本関数編 11
defun 11
setq 12
quit 13
princ,alert 13
command 13
(entget(car(entsel))) 14
entmake 15
AutoLisp 例構文編 15
変数の名前について 15
コメントアウトについて 16
例 1：2 点間を直線でつないで両端に円を描く 17
例 2：オブジェクトの選択 18
例 3：2 点間を直線でつないでそれに直交する線を等間隔で作成する 20
例 4：テキストを複数選択しテキストの内容を左から順番に 1,2,3,4・・・とする 21

例 5：線分の長さによって色を分ける 24
例 6：サブルーチン 25
例 7：コマンドとオブジェクトスナップと一括 undo 26
例 8：拡張オブジェクトデータ 28
例 9：ポリラインの頂点に順番に番号を振る 30
例 10：外部ファイル書き出し、読み込み 30
例 11：エクセルからマルチテキストを作成してリスト化する 32
例 12：ブロックを分解してそれによって新たにできるブロックがなくなるまで分解する 35
例 13：ucs による影響 37
例 14：mapcar と lambda の組み合わせ 39
例 15：リアクター関数 39
例 16：エラー処理 42
＜おまけ編＞「entmake 生成.xlsx」について 43
＜ちょっと上級編＞基準座標系とポリラインの成り立ち 44
ポリラインの成り立ち 44
ポリラインの 3 次元配置計算その 1 46
ポリラインの 3 次元配置計算その 1.1 48
ポリラインの 3 次元配置計算その 1.1 裏 54
ポリラインの 3 次元配置計算その 2 55
＜とても上級編＞entmake でソリッドの作り方 61
AutoLisp 参考サイト 63

AUTOLISP とは

AutoCAD 専用のプログラミング言語で Autocad での作業を自動化するものである。

複数の操作（例えば線を引いてその先端にテキストを書くなど）からなる作業が繰り返しあるとき、ひとつのコマンドだけで実行できるようになる。

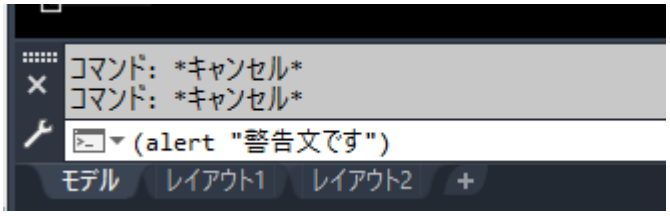
後から追記：この説明書および例文の変数名は「aa」などの意味のない文字列が使用されています。短いほうが書きやすいですが、**絶対に意味のある文字列を使用したほうが良いです。**たとえばフーチング底版幅であれば以前は「b」や「w」で書いていましたが、今は「real_width_footing」のように書いています。「line」や「dist」など既に定義されている関数やコマンドとかぶってしまうことが考えられますが、アンダーバーを使用しておけばほぼかぶることはあり得ません（「line_」、「dist_」などアンダーバーの後ろに書かなくてもよい。ちょっと見た目悪いけど）。今更すべてを直すのは困難であるため、**以降意味のない文字列の変数を見てもその書き方でいいのだと流されないでください。**

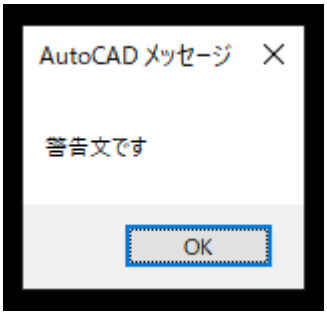
実行方法

短いものなら Autocad のコマンドラインに直接打ち込んで実行できる。

コマンドラインに『（alert "警告文です"）』と入力して Enter してみる。

⇒メッセージボックスが表示される。





ここで alert のことを関数と呼ぶ。

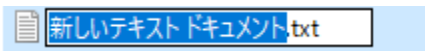
関数は基本的に

(関数　命令の内容)

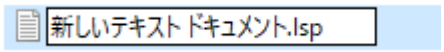
のように書かれカッコ「(」　で始まりカッコ「)」　で終わるのが基本である。

長くてもコマンドラインに打ちこんで実行できるが、エラーになったときどこがおかしいか調べるのが困難なので「～.lsp」ファイルを使用するべきである。

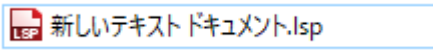
.lsp ファイルの作り方



テキストファイルを作成する

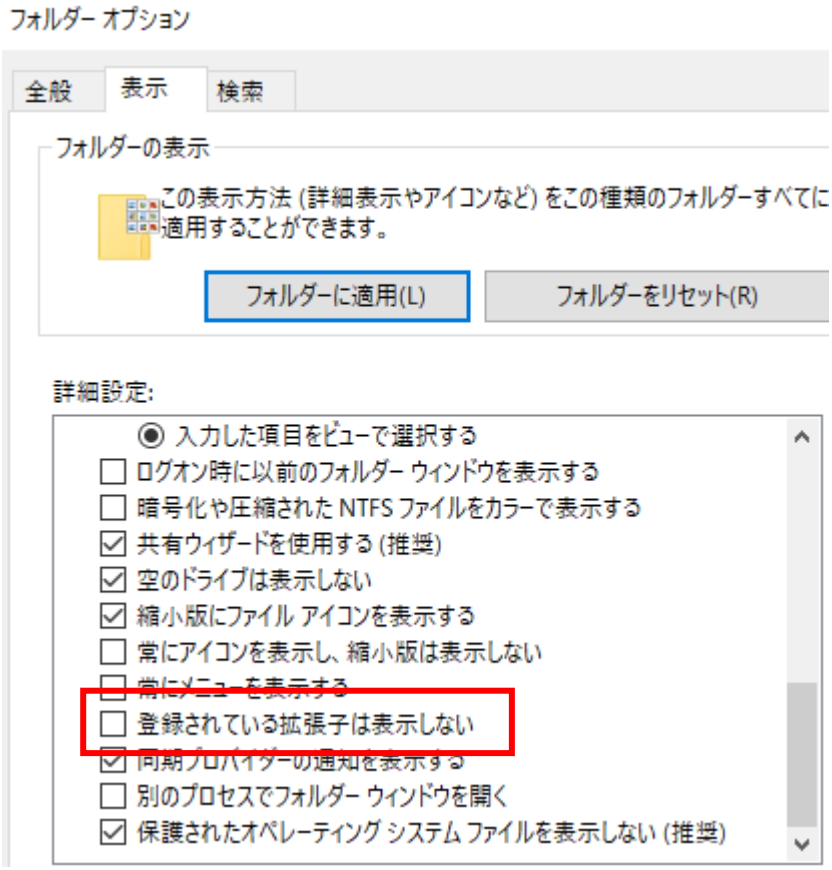


拡張子を lsp にする



赤色の LSP アイコンになる

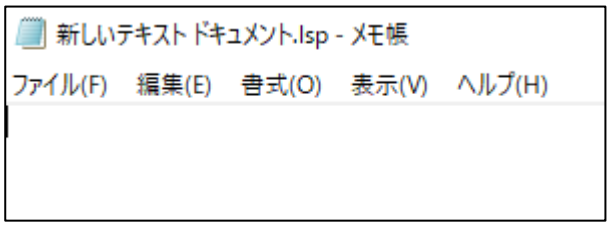
※拡張子が表示されない場合は



フォルダオプションの「登録されている拡張子は表示しない」のチェックをはずす

.lsp ファイルの編集

作成した lsp ファイルをダブルクリックするとメモ帳で開ける。



これでも編集可能であるが筆者は Emacs を推奨している(Emacs は LISP で書かれている)。
テキストが編集できるソフトならなんでもいいので強制するものではない。

Emacs について(自身の愛用エディタがある場合はとばしてください)

https://repo.jing.rocks/gnu/emacs/windows/



22~29 がありパソコンによっては動かないものもある

emacs-29.2.zip	18-Jan-2024 23:02	157706800
emacs-29.2.zip.sig	18-Jan-2024 23:02	438
emacs-29.2_1-installer.exe	01-Feb-2024 20:42	73457822
emacs-29.2_1-installer.exe.sig	01-Feb-2024 20:42	438
emacs-29.2_1-nodeps.zip	01-Feb-2024 20:41	142896852
emacs-29.2_1-nodeps.zip.sig	01-Feb-2024 20:41	438
emacs-29.2_1-sha256sums.txt	01-Feb-2024 20:42	266
emacs-29.2_1-sha256sums.txt.sig	01-Feb-2024 20:42	438
emacs-29.2_1.zip	01-Feb-2024 20:42	158662910
emacs-29.2_1.zip.sig	01-Feb-2024 20:41	438
emacs-29.3-installer.exe	27-Mar-2024 05:10	72966262
emacs-29.3-installer.exe.sig	27-Mar-2024 05:10	438
emacs-29.3-nodeps.zip	27-Mar-2024 05:09	142907953
emacs-29.3-nodeps.zip.sig	27-Mar-2024 05:09	438
emacs-29.3-sha256sums.txt	27-Mar-2024 05:10	260
emacs-29.3-sha256sums.txt.sig	27-Mar-2024 05:10	438
emacs-29.3.zip	27-Mar-2024 05:10	157762436
emacs-29.3.zip.sig	27-Mar-2024 05:09	438

Windows の場合は ZIP をどれかダウンロードして回答する。

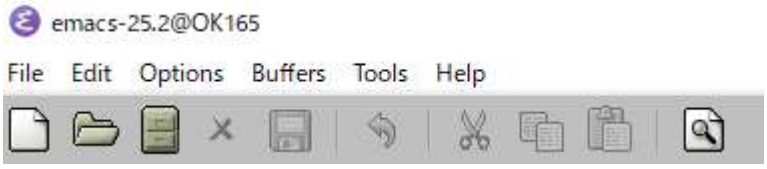
「bin」フォルダ内の exe を実行する



Emacs の特徴を簡潔に書く

インターフェース

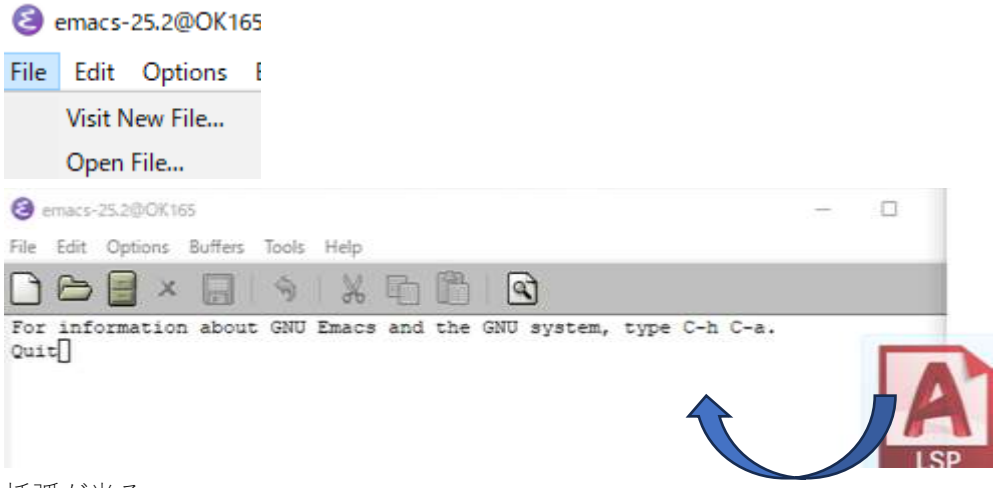
英語が基調。



日本語化パッチもあるらしいがやり方が難しい。

ファイルを開く

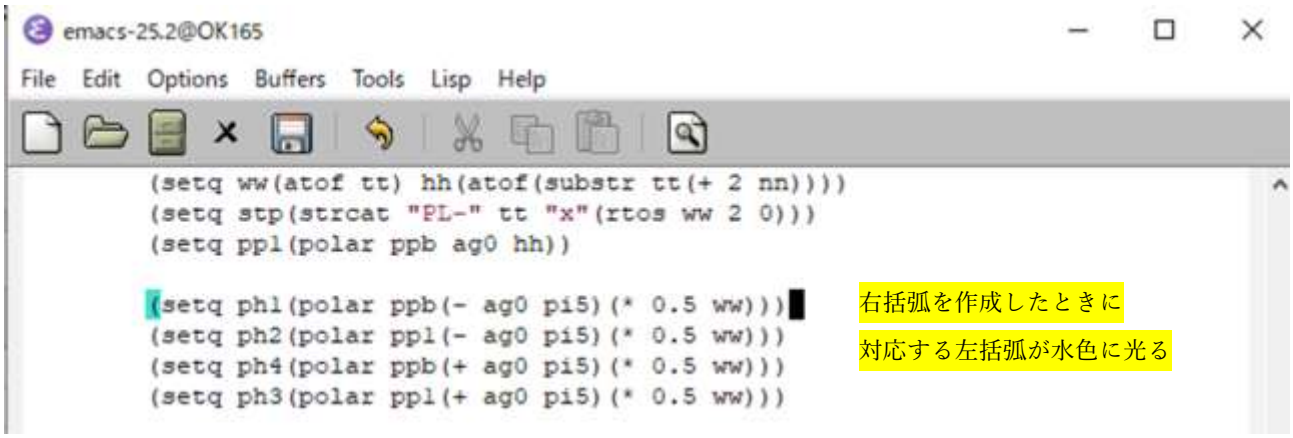
OpenFile はあるもののドラッグアンドドロップしたほうが早い。



括弧が光る

Emacs を使う最大のメリットのひとつ

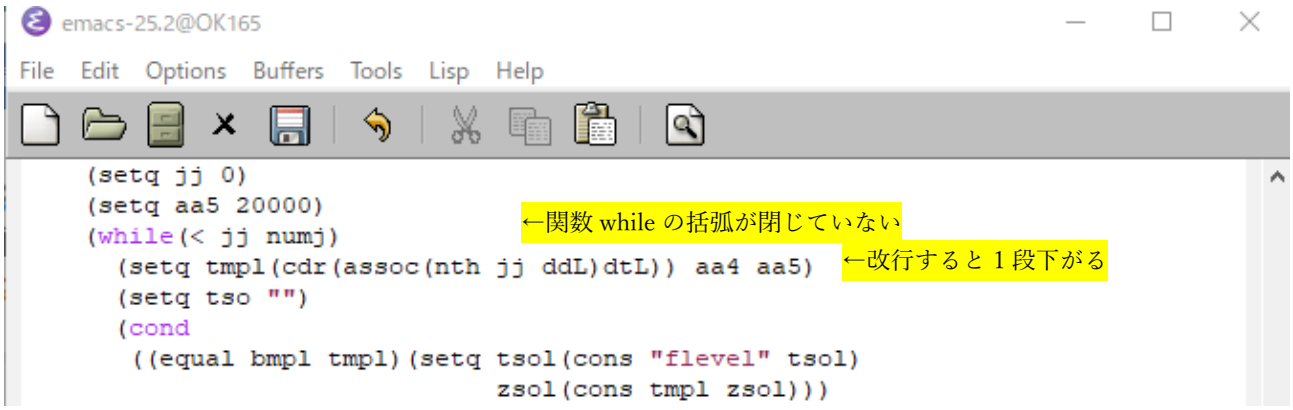
LISP は括弧に始まり括弧に終わるので括弧だらけになるためどの括弧が対になっているかわかりにくい状況が発生する。Emacs だと右括弧を書いて閉じるときに左括弧が光る。



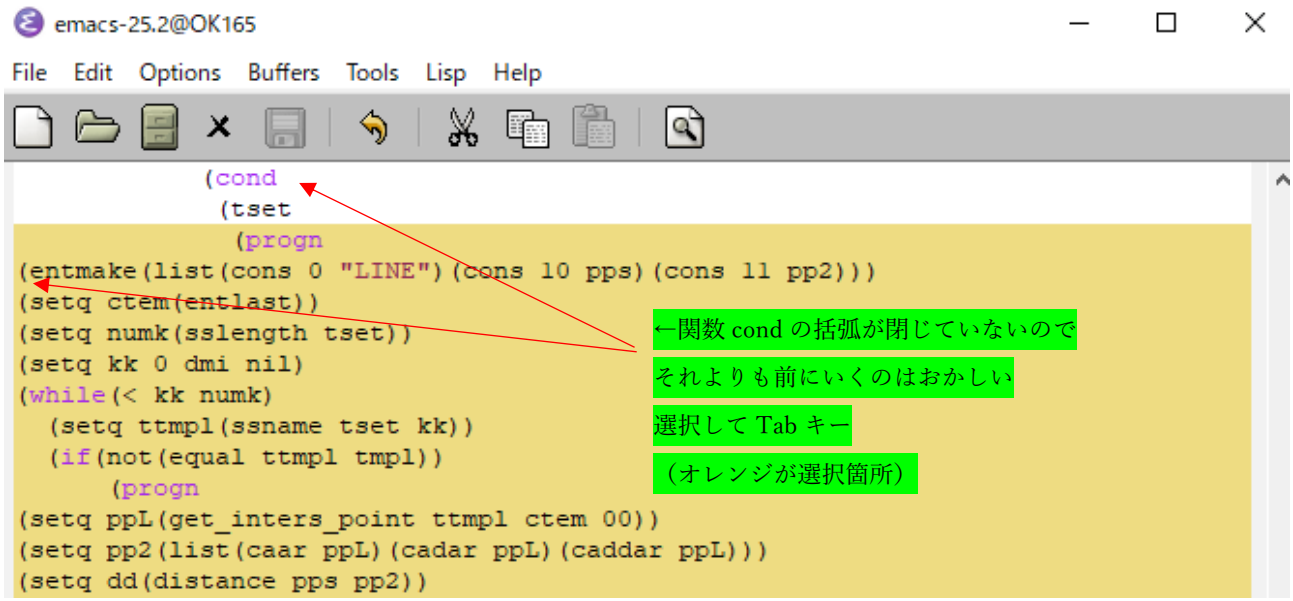
Tab キー操作

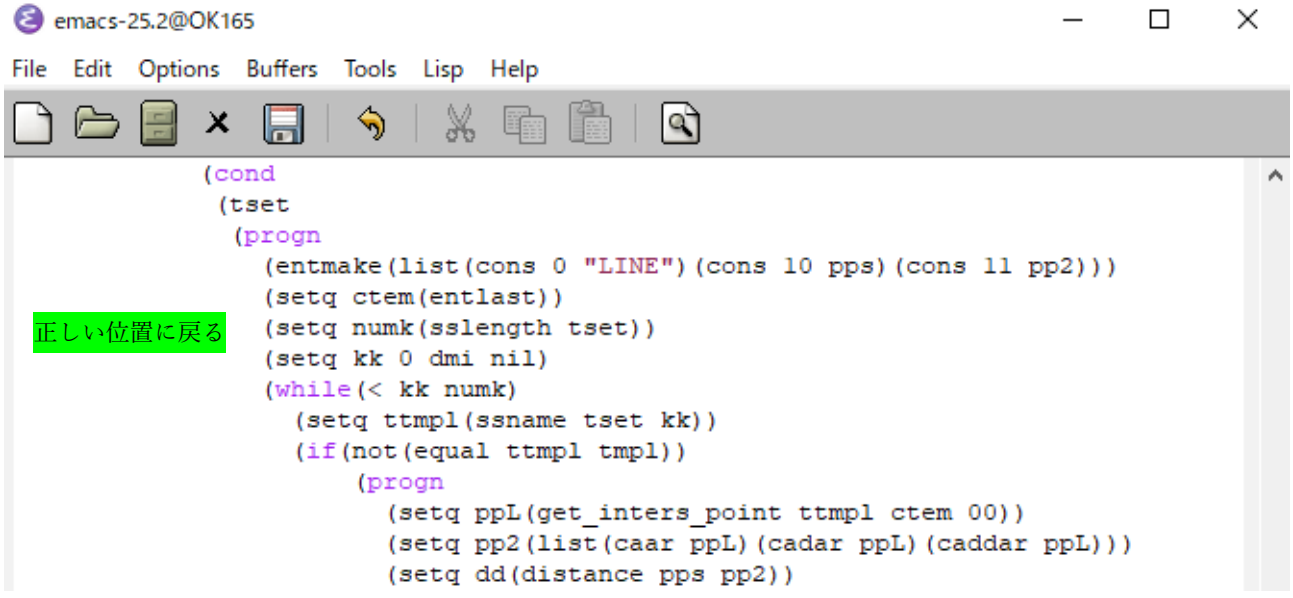
Emacs を使う最大のメリットのひとつ part2

括弧を閉じてない状態で Tab キーを押すと次の行から 1 段下がった配置になる。一部の関数はその中身が非常に長くなることもあるのでどこからどこまでがその関数の中なのかがわかりやすくなる。さらに何度も押しても必要以上に段が下がることはないのでとりあえず押しまくっておけばよい。



段落としがめちゃくちゃな場合でも全部選択して Tab を押すと正しい位置に戻る。



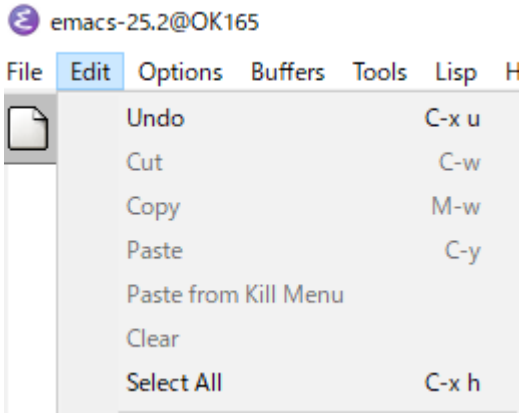


ショートカットキー

ショートカットキーが特殊であることが Emacs の唯一にして最大のデメリットである。

例えば一般的に保存として使われる「ctr+s」は検索である。

Ctr キーと Alt キーを多用する。



一部のショートカットキーはこのように右に書かれている

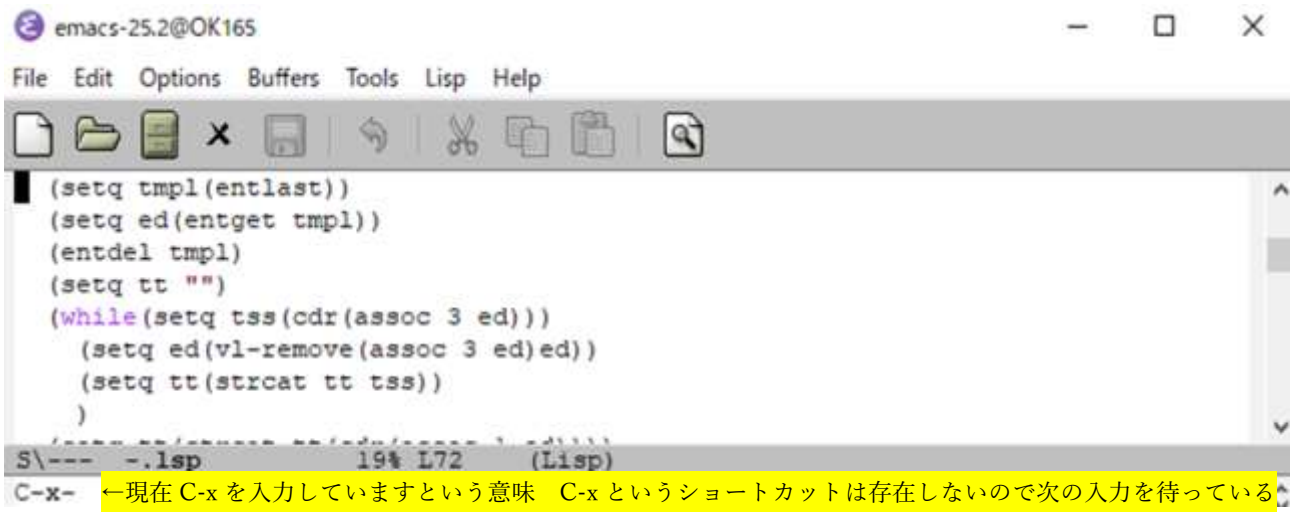
「C-●」：Ctr キーを押しながら●キーを押す

「M-●」：Alt キーを押しながら●キーを押す

「C-●　C-▲」：Ctr キーを押しながら●キーを押したあと Ctr キーを離さず▲キーを押す

「C-●　▲」：Ctr キーを押しながら●キーを押したあと Ctr キーを離して▲キーを押す

コマンド操作が 2 回以上必要なものもあるので Emacs の左下に現在押しているものが表示される



←現在 C-x を入力していますという意味　C-x というショートカットは存在しないので次の入力を待っている

よく使うものを以下で紹介する。

C-x C-s：ファイル上書き保存

C-g：キャンセル　2 回以上入力が必要なときに 1 回目を間違えてしまったときやり直すために押す。



左下に Quit と表示される。

動きがおかしくなったときにとりあえず C-g しておくと普通に戻ることがよくある。

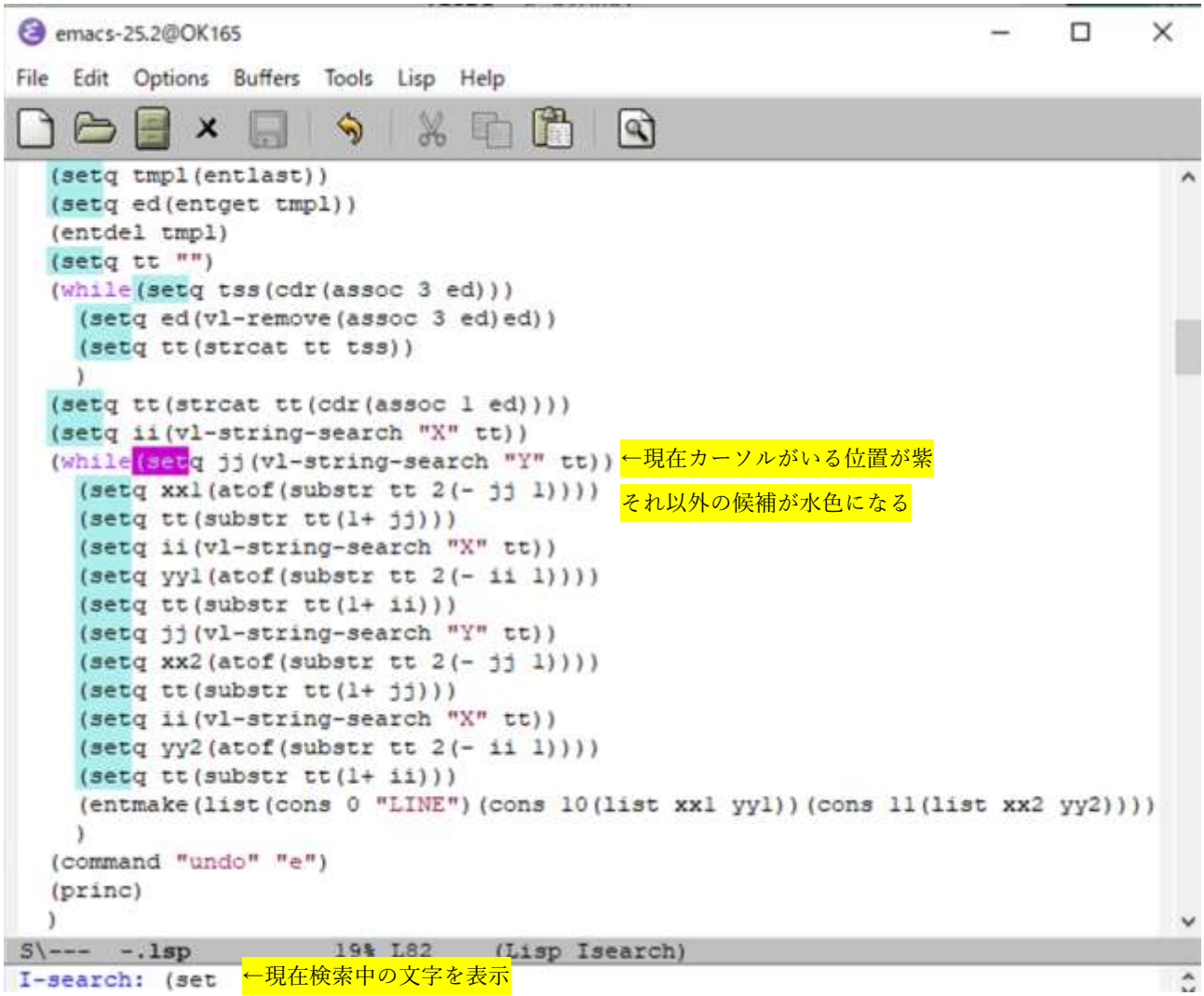
C-w：選択範囲を切り取り

M-w：選択範囲をコピー

C-y：上記を貼付け

C-s：検索（1 回押すたびに画面下向きに進む）

C-r：逆検索（1 回押すたびに画面上向きに進む）



C-a：行の先頭に移動

C-e：行の末尾に移動

Home キーと End キーを左手でも押せる程度の認識でよい

C-k：カーソルがある行のカーソルより後ろをすべて消す

C-/：アンドゥ

C-t：カーソル位置の文字とその一つ前の文字を入れ替える

（あまり使わないが C-y と間違えて押したとき注意）

カーソルキーショートカット（スマートなキーボードを使っている人は良く使うみたい）

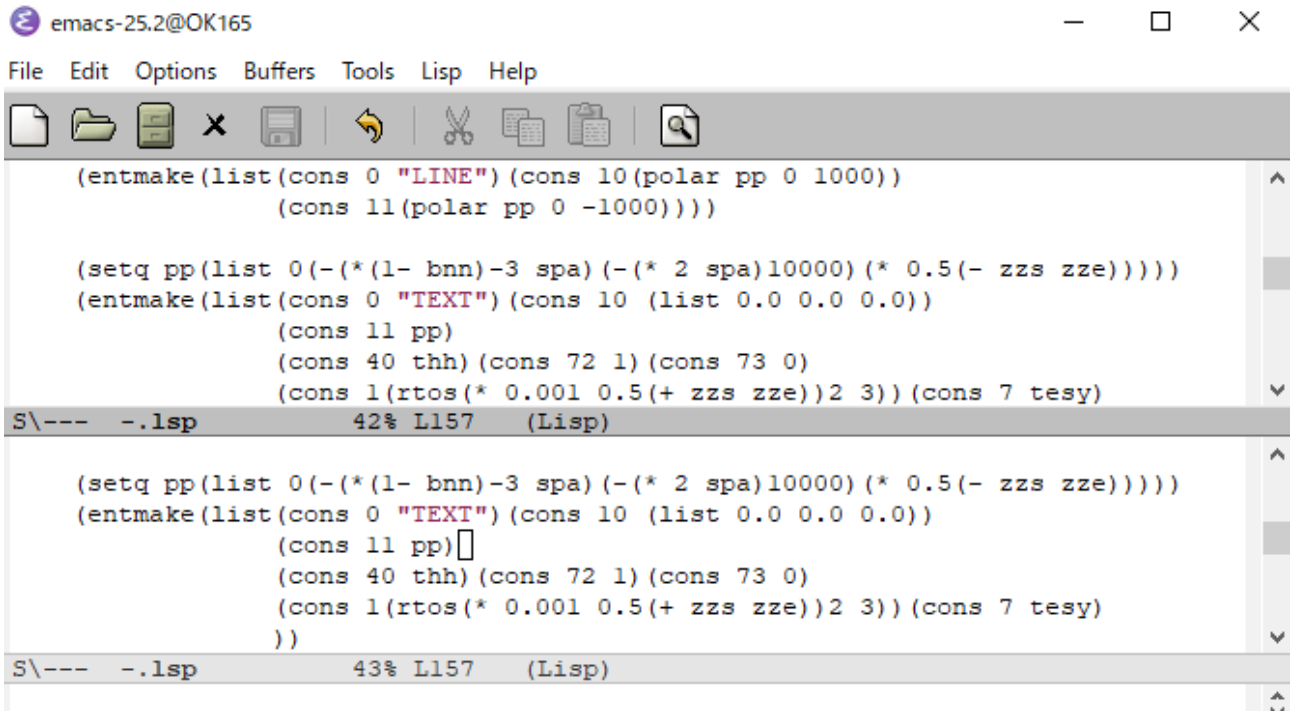
C-p：↑、C-n↓、C-b：←、C-f：→

C-d：Delete キー

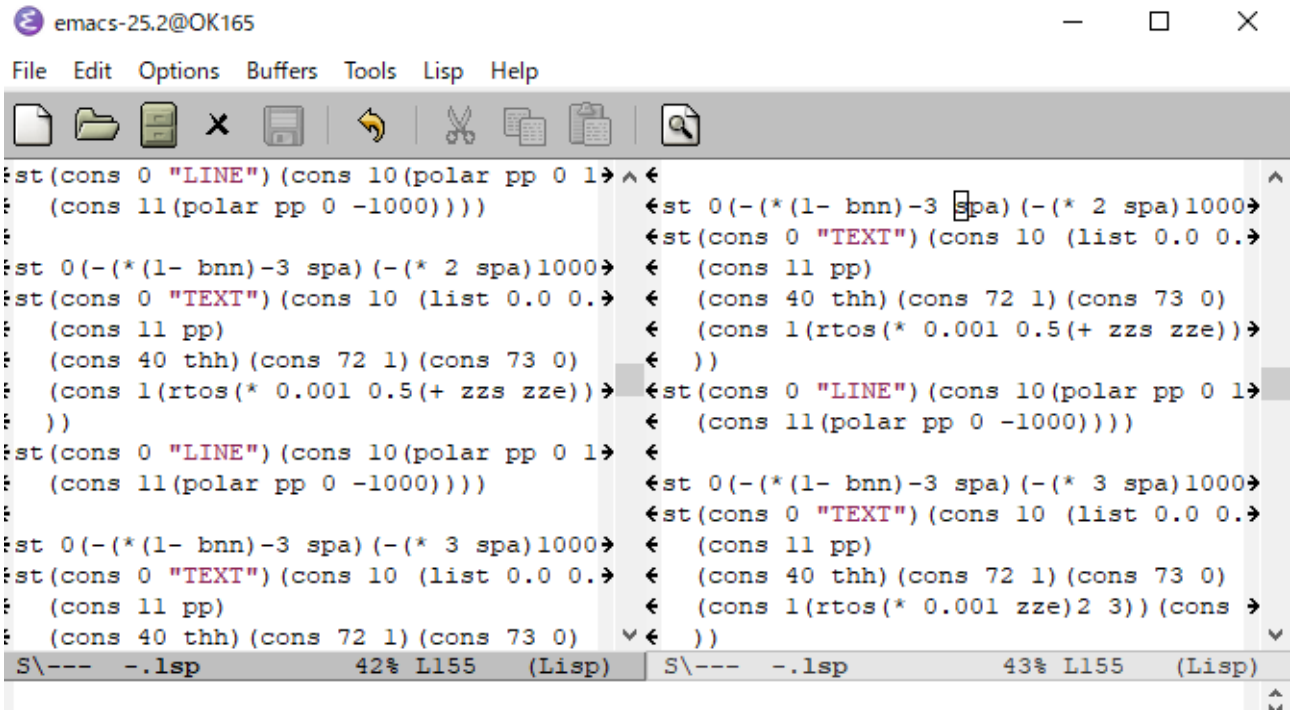
画面分割ショートカット

長いものを作るようになると画面を分けたくなることがある。

C-x 2：画面を上下に分ける



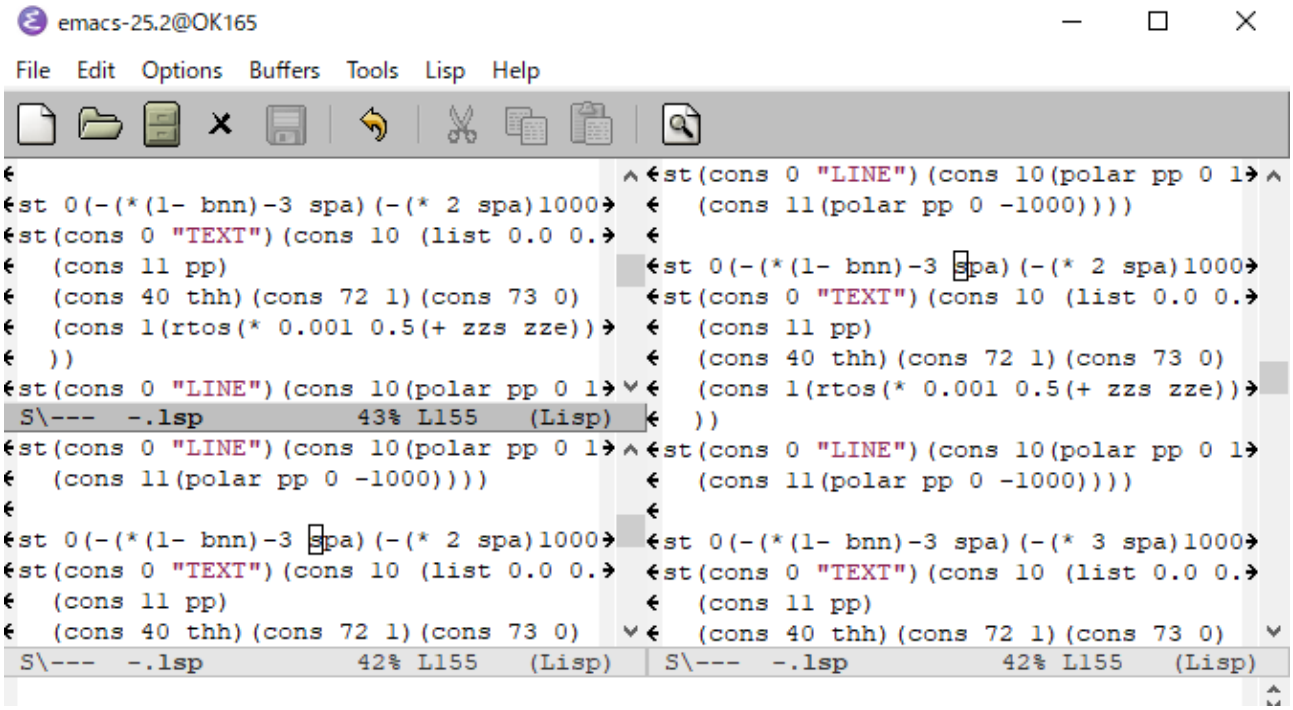
C-x 3：画面を左右に分ける



C-x o：分けた画面に移動する

C-x 1：分けた画面を 1 画面に戻す

横分割と縦分割は複合して何度でも可能



バックアップファイルについて

Emacs で編集して保存すると開く直前のデータが「~」付きでバックアップされる



これを外して拡張子を「lsp」にすれば元通りになる。

lsp ファイルの使い方

AutoLisp 基本関数編

基本関数編では関数を単体で紹介する。基本的にはウェブ検索すれば解説があるので最低限だけにする。
例構文編では構文としてまとまっているものに対して流れを解説する。

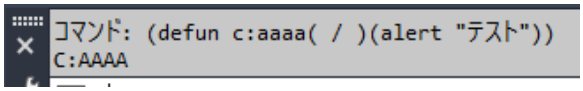
defun

コマンドを定義する関数。これがないとはじまらない。Define function の略らしい。

(defun c:○○ (/)
～いろいろな命令を書く～
)

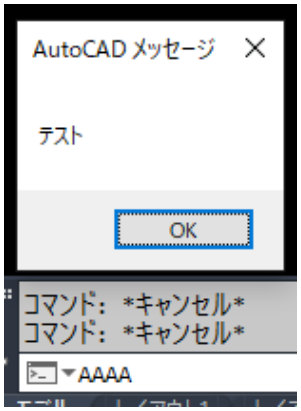
これで○○に入力した文字列をコマンドラインに打ちこめば実行できるようになる。
c: がない場合もある。出てきたときに解説
(/) は書いておく必要がある。意味は今説明するのは難しいのでしかるべきときに解説

コマンドラインに「(defun c:aaaa(/)(alert "テスト"))」と入力



aaaa というコマンドを作成した。

コマンドラインに aaaa と入力

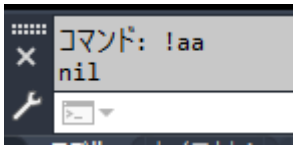


「aaaa」はメッセージボックスを出すコマンドになった。

Autocad を終了すれば defun はリセットされる。コマンド名に既存のコマンド「line など」を指定してしまった場合、もともとのコマンドは無効になってしまうが Autocad を終了すれば元に戻る。

setq

変数に値を設定する



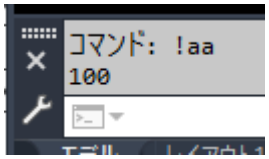
開いたばかりの図面でコマンドラインに「!aa」と入力して Enter すると「nil」とでる

nil は「なにも無い」という意味で「aa」という変数は何も値がないということがわかる。

コマンドラインに「(setq aa 100)」と入力する

aa は 100 であると定義する関数

もう一度「!aa」を入力すると「100」と表示される



aa = 100　となった

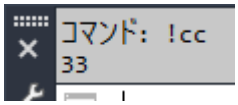
setq の後には「変数」、「値」、「変数」、「値」・・・と連続で変数に値を定義することができる。

前から順番に処理されていく。値に入れるのは数値だけではなく関数を含めた書き方もする。

コマンドラインに　「(setq aa 11 bb 22 cc(+ aa bb))」と入力する。

aa を 11 として定義⇒bb を 22 として定義⇒cc を aa+bb として定義　という流れで処理される。

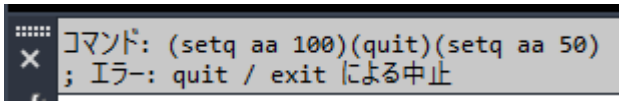
コマンドラインに「!cc」を入力して 33 になっていることを確認する。

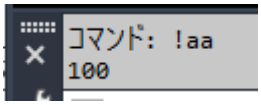


quit

強制終了する。入れてはいけない値を入力されたときに止めたり、エラーが出てしまうときにどこまで正常に進んでいるのかチェックするのに使える。

コマンドラインに「(setq aa 100)(quit)(setq aa 50)」と入力してから「!aa」で値を確認する。



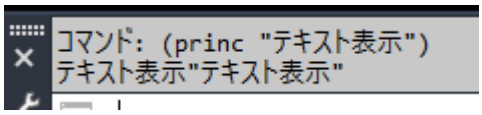


aa = 100 と定義した後、aa = 50 と定義する前に終了したので aa = 100 となる。

princ,alert

コマンドラインに文字を表示する princ とメッセージボックスを表示する alert

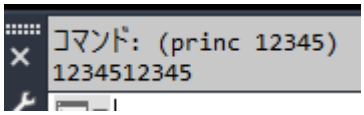
コマンドラインに「(princ "テキスト表示")」と入力



2 回連続で表示されるのは仕様

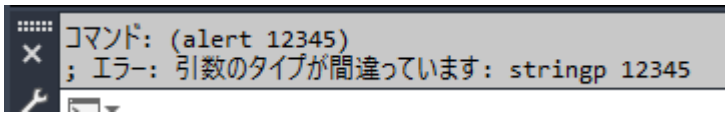
「""」(ダブルクォーテーション 2 つ) で囲まれると文字列となる。

コマンドラインに「(princ 12345)」と入力



数値でも表示される

コマンドラインに「(alert 12345)」と入力

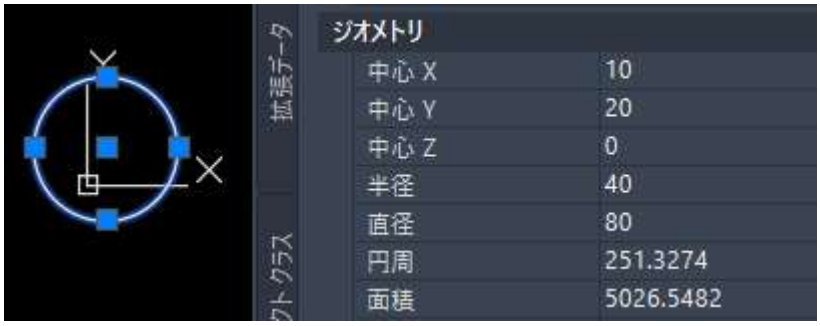


alert の場合はテキストしか受け付けない

command

Autocad にもともと備わっているコマンドを使うことができる。

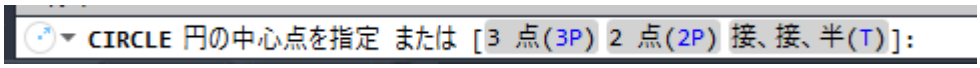
コマンドラインに「(command "circle" (list 10 20 0) 40)」と入力

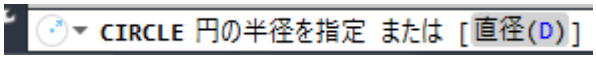


中心 X=10、中心 Y=20、半径=40 の円ができる。

(list x y z)は座標を表す

普通にコマンドラインに「circle」と入力したとき

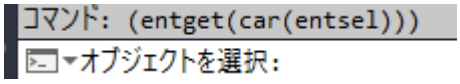




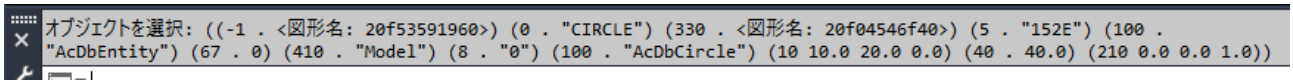
(command "circle" (list 10 20 0) 40)はこの順番で値を入力している。

(entget(car(entsel)))

コマンドラインに「(entget(car(entsel)))」と入力する。



オブジェクトの選択を求められるので上記の円を選択する。



コマンドラインに長い文字列が表示される。

((-1 . <図形名: 20f53591960>) (0 . "CIRCLE") (330 . <図形名: 20f04546f40>) (5 . "152E") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle") (10 10.0 20.0 0.0) (40 . 40.0) (210 0.0 0.0 1.0))

図面オブジェクトはこの文字列によって作られている。この情報を作ることができればオブジェクトを作ることが可能。先頭についている数字が何の情報なのかを表している。これをグループコードという。覚えておくといいものを解説。

(-1 . <図形名: 20f53591960>)

図形オブジェクトの ID みたいなもの図面を開くたびに割り振られて変わるので特に覚える必要はない。

(0 . "CIRCLE")

オブジェクトのタイプ。ここでは CIRCLE。

(5 . "152E")

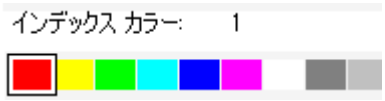
ハンドル ID と呼ばれるもの。-1 の図形名と違って変わることはないので利用することもある。

(8 . "0")

画層名。

(62. 256) ※上記では存在してない

色の情報でインデックスカラーの番号で表現される。ByLayer のときは 256 となり省略される。



RGB 値を調整するとインデックスカラーで表せない色を作ることができるが最も近いインデックスカラーの値が表示される。

(10 10.0 20.0 0.0)

座標値

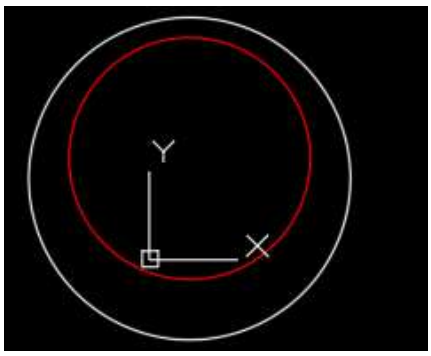
円は座標値が 1 つだがほかの図形のは座標値が 2 つ以上あることもある。11～が使われたりする。

(40 . 40.0)

円の半径
(210 0.0 0.0 1.0))
円の 3 次元的な向きを表す。難しいので今は深く説明しないが平面的なオブジェクトだと 0,0,1 になる。

entmake

上記で得られる情報を使ってオブジェクトを作成する。全ての情報が必要なわけではなく図形タイプによって必要な情報が違う。円の場合必要な情報は「0」と「10」と「40」だけ。
コマンドラインに「(entmake '((0 . "CIRCLE")(62 . 1)(10 10.0 25.0 0.0)(40 . 30.0)))」
「62」の色情報は見やすくするためにつけたので必須ではない。



中心 Y 座標が 5 上に移動、半径 30 の赤色の円ができる。

(command)関数は実際にそのコマンドを打ち込んでいるのと同じなので若干処理速度が落ちる。基本的に使わずに済ませられるのであれば使わないようにしたほうが良い。
たとえば mirror は計算次第で entmake で作成可能だが、その計算を書くのが難しすぎる場合は command を使ってよいと思う。

AutoLisp 例構文編

例構文編ではあまり長くなりすぎないように構文を作ってどのような流れになっているか紹介する
目安は Emacs で 1 画面分くらい。本解説書に構文のすべてを載せられないので、例ファイルを見ながら解説するものとする。ファイル名とコマンド名は「rei000」(000 は例番号の数字)とする。一度説明した関数は使い方が大きく異なる場合を除いて説明を省略する。

変数の名前について

変数は基本的に関数名やコマンドとかぶったりしないなら何を使ってもいいのでこれから紹介する例に倣う必要な全くない。
_（アンダーバー）を使うと関数やコマンドとかぶることはなくなるので各種サイトをみると使われていることが多いが、わざわざ Shift を押すのが面倒に感じるときもあるので絶対かぶらないとわかっている文字は付けなくてもいいと思う。
例：「aa01」、「aa02」、「line_A」、「line_B」など
Emacs を使っている場合は小文字の「 l 」（エル）と数字の「1」（いち）の見分けがつかないので「L」を使うのをおすすめする。変数に大文字小文字の区別はないのでたとえば「AA」も「aA」も同じ値として処理される。

コメントアウトについて

セミコロン「;」を付けるとそれ以降その行に書かれたものは無効となる。

Emacs の場合だと赤茶色になる。

何を書いてもプログラムとして扱われないので説明文を書くのに使える。

```
;各点のxyz座標を取り出す
 (setq xx1(nth 0 pp1) yy1(nth 1 pp1) zz1(nth 2 pp1)
       xx2(nth 0 pp2) yy2(nth 1 pp2) zz2(nth 2 pp2)
       xx3(nth 0 pp3) yy3(nth 1 pp3) zz3(nth 2 pp3))
```

行の途中に書いてもよい。

```
 (if(<(abs ag0)0.00001);;エラーによる終了
     (progn
       (alert"\n面の点が一直線上に並んでいます")
       (quit)
       ))
```

Emacs では範囲選択した後「M-;」を入力すると一括コメントアウトでき、コメントアウトされた範囲を選択してもう一度入力するとコメントアウトを一括解除できる。

・コメントアウトしたい範囲を選択

```
       ((= ii 1)(setq xx(nth 0 pp5) yy(nth 1 pp5)))
       )
      (setq agp(-(angle(list xx3 yy3)(list xx yy))ag1))
      (setq ds(distance(list xx3 yy3)(list xx yy)))
      (setq aa(/(-(* ds(cos agp))(/(* ds(sin agp)(cos ag0))(sin ag0)))ds1))
      (setq bb(/(/(* ds(sin agp))(sin ag0))ds2))
      (setq xx(+ xx3(* aa vx1)(* bb vx2))
```

M-;

```
       ((= ii 1)(setq xx(nth 0 pp5) yy(nth 1 pp5)))
       )
      ;; (setq agp(-(angle(list xx3 yy3)(list xx yy))ag1))
      ;; (setq ds(distance(list xx3 yy3)(list xx yy)))
      ;; (setq aa(/(-(* ds(cos agp))(/(* ds(sin agp)(cos ag0))(sin ag0)))ds1))
      ;; (setq bb(/(/(* ds(sin agp))(sin ag0))ds2))
      (setq xx(+ xx3(* aa vx1)(* bb vx2))
```

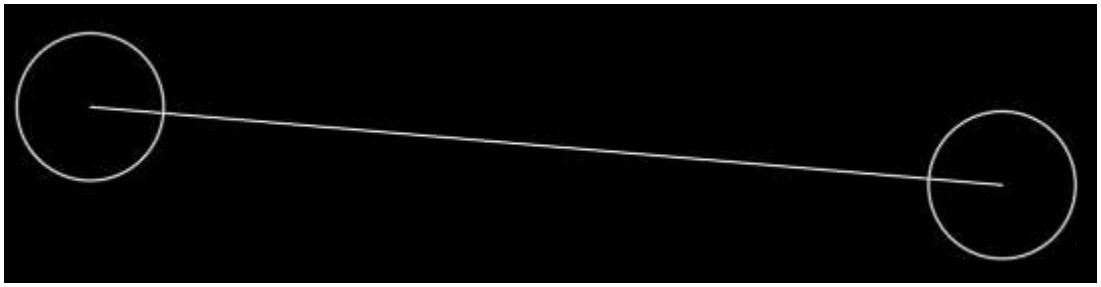
コメントアウトを解除したい範囲を選択

```
       ((= ii 1)(setq xx(nth 0 pp5) yy(nth 1 pp5)))
       )
      ;; (setq agp(-(angle(list xx3 yy3)(list xx yy))ag1))
      ;; (setq ds(distance(list xx3 yy3)(list xx yy)))
      ;; (setq aa(/(-(* ds(cos agp))(/(* ds(sin agp)(cos ag0))(sin ag0)))ds1))
      ;; (setq bb(/(/(* ds(sin agp))(sin ag0))ds2))
      (setq xx(+ xx3(* aa vx1)(* bb vx2))
```

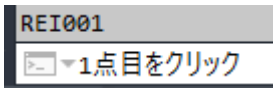
M-;

```
       ((= ii 1)(setq xx(nth 0 pp5) yy(nth 1 pp5)))
       )
      (setq agp(-(angle(list xx3 yy3)(list xx yy))ag1))
      (setq ds(distance(list xx3 yy3)(list xx yy)))
      (setq aa(/(-(* ds(cos agp))(/(* ds(sin agp)(cos ag0))(sin ag0)))ds1))
      (setq bb(/(/(* ds(sin agp))(sin ag0))ds2))
      (setq xx(+ xx3(* aa vx1)(* bb vx2))
```

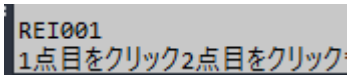
例１：２点間を直線でつないで両端に円を描く

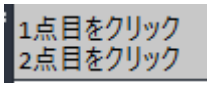


(getpoint)
図面内の座標をクリックして取得する。
(getpoint テキスト)　とすると座標取得時にコマンドラインにテキストが表示される。

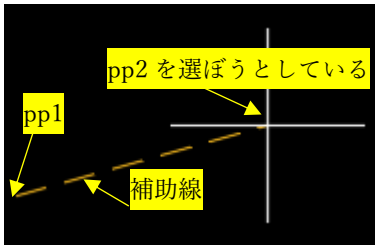


テキスト内の「¥n」は改行を表す。ありのときとなしのときは以下の通りになる。

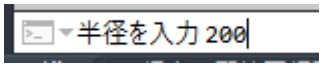




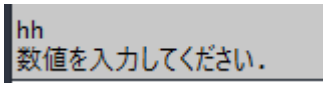
(getpoint 点の座標)とすると点を選ぶときに補助線が出る。



(getreal)
実数値の入力を要求する。
(getpoint)同様に後ろにテキストを書くことで要求時にコマンドラインにテキストを表示する。



実数値を入力する。



数値を入力しないと再入力を求められる。

(list 変数１ 変数２ 変数３ ・・・)
リストを作る。もともとバラバラだった複数の変数を箱に並べて入れて一塊にするようなイメージ
リストは新たに一つの変数という扱いになるのでリストの中にリストを入れることも可能。
(list X Y)や(list X Y Z)とすると座標を表すことができる。関数によっては要求される座標値が XY だけで Z があるとエラーになるものもあるので注意。いろいろな使い方があり、いたるところで出てくるので必要があれば都度解説する。

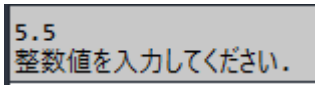
残りは entmake で線分と円を作成
線分は始点の座標と終点の座標が必要でグループコードは始点が 10、終点が 11。
円は中心の座標と半径が必要でグループコードは 10 と 40。

最後の(princ)について
コマンドを実行するとそのコマンド内で最後に実行したものがコマンドラインに表示される。
もし(princ)がなかったら最後の処理は entmake なのでコマンドラインに次の表示がされる。

```
((0 . "CIRCLE") (10 943803.0 58529.7 0.0) (40 . 80.0))
```

場合によってはこれがものすごく長くなり邪魔になることもある。
最後の処理を(princ)とすると最後の処理が見かけ上何もなくなる(何もない文字列を記入する処理)。
基本的には(princ)を最後につけると覚えておいてよい。
最後に(princ)をやってはいけないときがあるので別途紹介する予定。

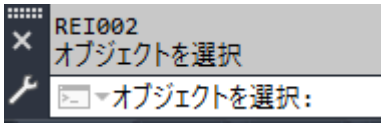
getreal と同種の関数
(getint)
整数値を入力する。

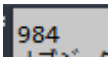


整数値を入力しないと再入力を求めれらる。

(getstring)
文字列を入力する。数字を入力しても文字列として扱われるのでそのまま計算には使えない。

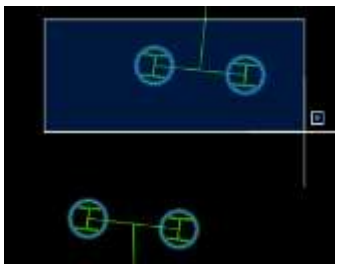
例 2：オブジェクトの選択
(ssget)
図面内のオブジェクトを選択して選択セットを作る getpoint、getreal などのように後ろにテキストを付けて要求時に表示することができないので ssget より前に princ でテキスト表示している。
(sslength 選択セット)
選択セットに含まれるオブジェクトの数を求める。



⇒ 

選択したオブジェクトの個数が表示される。

(ssget (list グループコード))
グループコードを限定して選択対象にする。例では(cons 0 “CIRCLE”)なので円だけが選択される。



円以外を囲んでも選択できない

(cons 変数 1 変数 2)
ドットペアリストというものを作る。変数 2 の前に変数 1 をくっつけてリストにする。グループコードはこれに該当する。(cons 0 “CIRCLE”)でグループコードの(0 . “CIRCLE”)ができる。

```
(setq eset(ssget(list(cons -4 "<OR")
                     (cons 0 "CIRCLE")(cons 0 "LINE")
                     (cons -4 "OR>"))))
```

(cons 0 "CIRCLE")と(cons 0 "LINE")が同時に成立する図形はありえないが
これらを(cons -4 "<OR")と(cons -4 "OR>")で挟むことによってどちらかが成立すればよいものとなる。
(setq eset(ssget(list(cons 0 "CIRCLE,LINE")))) でよいことが判明した

参考
https://help.autodesk.com/view/OARXMAC/2024/JPN/?guid=GUID-5CB54129-22A1-42B9-B97C-2D2F5597F90E

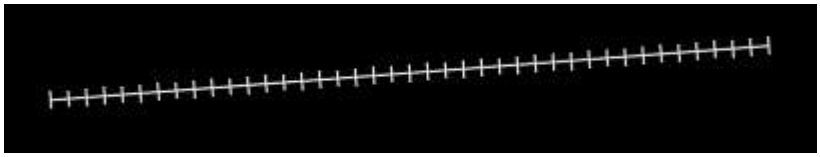
(alert(rtos num 2 0))
ここで num はテキストではなく数値である。alert は数値を扱うとエラーになるので rtos を使ってテキストにしている。
(rtos 数値 単位 桁数)
単位：次の中から選ぶ。2 しか使わないと思っていてもたぶん良い。
1 指数表記
2 十進表記
3 工業図面表記(フィートと小数インチ)
4 建築図面表記(フィートと分数インチ)
5 分数表記
桁数：小数点以下何桁まで表示するかを整数で書く。

ssget のオプション
種類が多いので割愛するが覚えておいたほうが便利なので参考 URL を紹介。

参考 1
http://docs.autodesk.com/ACD/2011/JPN/filesALR/WS1a9193826455f5ff1a32d8d10ebc6b7ccc-693e.htm

参考 2(例が書かれていてわかりやすい)
https://help.autodesk.com/view/OARXMAC/2024/JPN/?guid=GUID-7BE77062-C359-4D01-915B-69CF672C653B

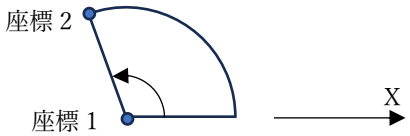
例 3：2 点間を直線でつないでそれに直交する線を等間隔で作成する



(setq hh(* 0.5 hh))
四則演算(+ , - , * , /)
数値の 1 番最初に書かれたものに対して続く数値を同じ演算子で計算する。
例では　hh = 0.5 * hh
※これは方程式ではないので式としておかしいということはない。プログラミングでは当たり前のこと。
hh は古い hh を使って計算された新しい hh となる。
(+ aa bb cc dd)
⇒　aa + bb + cc + dd
(- aa bb cc dd)
⇒　aa - bb - cc - dd

(distance 座標 座標)
2 点間の距離を求める。2 次元座標でも 3 次元座標でもよい。2 次元だと Z=0。
結果は必ず正の数。
座標リストの中身がおかしいとこのエラーが出ることがある。

(angle 座標 1 座標 2)
2 点間の角度を求める。3 次元座標で入力しても結果は Z=0 の平面角度となる。
結果は弧度法(rad)。
座標 1 から X 方向を 0 として反時計回りが正方向となる。



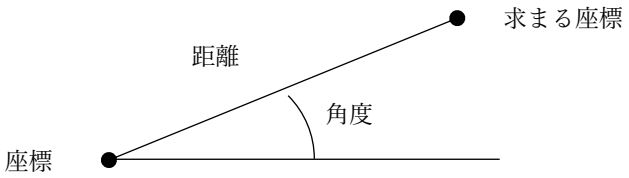
(atan Y 値 X 値)
X 値と Y 値からなるアークタンジェントを求める。
(atan Y 値/X 値) でも同じものが求められる(変数が 1 つになると関数の処理が変わる)。
ただし、90° のときは X=0 となり割り算ができないので前者を使うのが望ましい。直角はいろいろ使えるので(atan 1 0)はよく使う。(* 2(atan 1))でもいいけど。

(while 条件式 いろいろな処理)
条件式が nil(偽)にならない限り処理を何度でも繰り返し行う。

(<= 数値 1 数値 2)
数値 1 が数値 2 以下であれば T(真)、そうでなければ nil(偽)となる。
数値は複数設定可能で最初の数値が後に続く数値すべてに対して T のときのみ T となる。
(<= 1 2 3 4 5 6 7 8 9)
⇒T
(<= 5 6 7 8 9 1)
⇒nil

その他の演算子参考
https://qiita.com/tydesign/items/1f97fa316d25140ac6b6

(polar 座標 角度 距離)
座標から角度の方向に距離進んだ座標を求める。



(1+ 数値)
数値を 1 増やす。「1+」が関数であるため間は空けてはいけない。
同種の関数で（1- 数値）がありこれは数値を 1 減らす。見た目から 1 から数値を引くと勘違いしないように。「2+」、「3+」という関数は存在せず、1 だけ。

while のループのたびに ii が 1 ずつ増えるので ii が nn より大きくなるとループが終了する。

例 4：テキストを複数選択しテキストの内容を左から順番に 1,2,3,4・・・とする
CAD 図面上にテキストを用意する。内容は何でもよいが 1 文字のほうが見やすい。





(setq xxL(list)
ttL nil)
この 2 つはほぼ同じ意味。(list)で空のリストができる。nil は何もないという意味なのでどちらも空っぽの箱を作るイメージ。この後の while ループの中でここに値を入れていく。

(ssname 選択セット 整数値)
選択セットに含まれる図形を番号順に取り出し、その図形名を取得する。
ここで整数値の最初の値は 0 から始まるルールになっている。
選択セットが 10 個ならば 0〜9 の値を入れることになる。この範囲外の値を入れるとエラーになる。
なので「(while(< ii num)」では ii が 0 から num-1 までになるように「<=」ではなく「<」を使っている。

選択セットの 0 番目とはどのオブジェクトのことなのか。
選択セットの順番は「選択された順」＞「作成された順」になる。
ひとつずつオブジェクトをクリックして選択するとその順番で若い番号を割り当てることになり、囲み選択などで一括選択するとその中で新しく作られたものから若い番号が割り当てられることになる。

(entget 図形名)
図形名の図形のグループコードのリストを取得する。

(assoc 値 リスト)
リストからドットペアリストの先頭の値が値と一致するものを取り出す。
リストはドットペアリストの集合リストである必要がある。上記で取得した図形のグループコードのリストはそれに該当する。
テキストオブジェクトのグループリストは以下のようになっている。値が 10 のとき
((-1 . <図形名: 17753a258c0>) (0 . "TEXT") (330 . <図形名: 176f8da89f0>) (5 . "EA613C") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "VIEW") (100 . "AcDbText") (10 -1.58378e+07 -2.74605e+07 0.0) (40 . 6000.0) (1 . "a") (50 . 0.0) (41 . 1.0) (51 . 0.0) (7 . "MS ゴシック") (71 . 0) (72 . 1) (11 -1.58358e+07 -2.74616e+07 0.0) (210 0.0 0.0 1.0) (100 . "AcDbText") (73 . 1))
この中でドットペアリストの先頭が 10 のものを取り出す。

もし、同じ値があるときはリストの前のものが取得される。
（ポリラインでは座標値の先頭はすべて 10 なので出てきたときに解説）

(cdr リスト)
(car リスト)
cdr はリストの先頭の要素を消す。car はリストの先頭の要素だけを取り出す。

(setq aa(list 1 2 3 4 5 6 7 8 9 0))

(setq bb(car aa))

⇒bb は aa の先頭である 1 になる

(setq bb(cdr aa))
⇒bb は aa から先頭を消した(2 3 4 5 6 7 8 9 0)になる

この 2 つは a と d の組み合わせで使うことができる(6 個まで)。
(setq bb(cadddr aa))
c と r の間は a d d d　となっており、処理の順番は後ろからなので
「先頭を消す(1 回目)」⇒「先頭を消す(2 回目)」⇒「先頭を消す(3 回目)」⇒「先頭を取る」となる。
先頭を消す(1 回目)　⇒　(2 3 4 5 6 7 8 9 0)
先頭を消す(2 回目)　⇒　(3 4 5 6 7 8 9 0)
先頭を消す(3 回目)　⇒　(4 5 6 7 8 9 0)
先頭を取る　⇒　4
なので bb は 4 になる。

ファイル内では
(setq pp(cdr(assoc 10 ed)))
(setq xx(car pp))
としているが、ここで pp を定義してもその後使ってないので定義する必要性は薄い。
短いほうが読みやすいという利点はある。
(setq xx(cadr(assoc 10 ed)))
でもよい。

(setq xxL(cons xx xxL))
取得したテキストの X 座標を xxL というリストの先頭にくっつける。これをテキストの数だけ繰り返すので X 座標だけのリストが完成する。

(setq ttL(cons(cons xx ii)ttL))
(cons xx ii)を ttL というリストに収める処理をテキストの数だけ繰り返す。

(vl-sort リスト '演算子)
リストを演算子で決められた順に並べなおす。演算子の前には'(アポストロフィー)が必要。
'<と'>しか使ったことがないので他の演算子については解説できない。
値を小さい順にならべるか大きい順に並べるか。

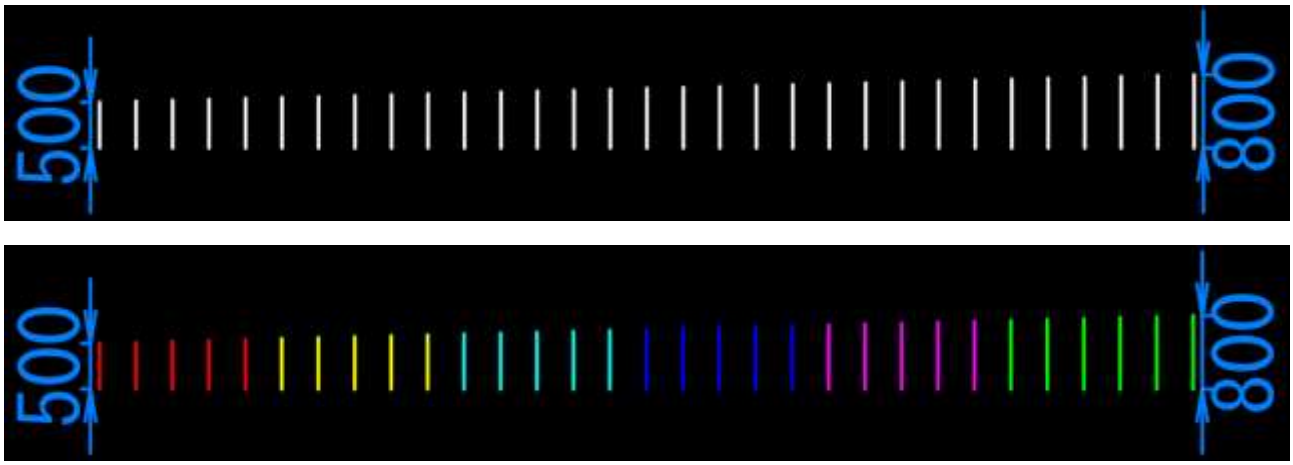
(nth 整数値 リスト)
リストの中の整数値番目の値を取り出す。始まりは 0 から数える。
(setq xx(nth ii xxL))で ii が増えるにしたがって X 座標が小さいものから順に取り出している。

(setq jj(cdr(assoc xx ttL)))
ttL は X 座標が小さい順に整列されていないが、先に取り出した X で assoc を使うことによって小さい順に図形の番号を知ることができ、それを jj とする。

(subst 値 1 値 2 リスト)
リストの中から値 2 を見つけ出し、見つかった場合値 1 に置換する。
(setq ed(subst(cons 1(rtos ii 2 0))(assoc 1 ed)ed))
これによりグループコードのリストからテキストの内容を表す 1 の中身を入れ替えている。

(entmod グループコードのリスト)
グループコードのリストの内容で図形を更新する。
ここでようやく図形オブジェクトが変化する。

例 5：線分の長さによって色を分ける
長さが 10 単位で 500 から 800 まで変化する線分を用意する(コマンド「rei005_p」)。
線分の色は「ByLayer」以外にしておくこと。



(cond (条件 1 処理 1) (条件 2 処理 2) (条件 3 処理 3) (条件 4 処理 4))
条件分岐により処理を変える。
条件 1 に該当するものは条件 2 以降の判定は行われない。
dd は線分の長さなので、
条件 1：線分の長さが 550 より小さいもの
条件 2：条件 1 に該当しないもののうち線分の長さが 600 より小さいもの
（以下同）となる。
条件が T は真の意味なので分岐は発生せず、その時点で残っているものすべてが対象となる。

(progn 処理 処理 ・・・)

複数の処理を 1 つの処理として扱う。
cond で」分岐後の処理は複数の式を認識できる。if のように progn で 1 つの式にまとめる必要はない。(progn)の内側は 1 つの処理となるので 2 つ以上の処理に対応できる。例の場合では処理が 1 つしかないためやる必要がないが説明のためにしている。

オブジェクトの色が「ByLayer」以外でなければならない理由
オブジェクトの色を表すグループコード 62 は色が「ByLayer」のとき省略されてしまうので、(assoc 62 ed)の結果が nil になってしまい、(subst)で何も置換することができない。
Bylayer でもいい方法がコマンド「rei005_b」として下のほうに書いてある。

VLA オブジェクトを使う方法について
コマンド「rei005_v」でも同じ処理ができる。
VLA 関数は総じてとても長い(ハイフンで単語をつないでいる)のでなかなか覚えられないが、覚えるとコードがすっきりするようになって便利。

(vlax-ename->vla-object 図形名)
図形名のオブジェクトの VLA オブジェクトとしての名前を取得する。

(vla-get-length VLA オブジェクト)
VLA オブジェクトの長さを取得する。線分、ポリラインに適用可能。
円弧に対して使うとエラーになるが円弧は
(vla-get-arclength VLA オブジェクト)
で対応できる。
グループコードを使う方法だと 2 点の座標を取り出して距離を計算していたのに対し 1 関数ですむ。

(vla-put-color VLA オブジェクト インデックスカラー番号)
VLA オブジェクトの色をインデックスカラー番号が表す色にする。
こちらを使う場合はオブジェクトのもともとの色が「ByLayer」であっても問題なく処理できる。

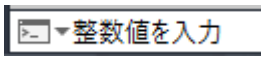
例 6：サブルーチン
よく使う処理をサブルーチンとして作成し、使いまわすたびに同じ式を書かなくていいようにできる。一見複雑なのでとりあえず実行してみるとよい。

(defun rei006_sub(aa bb cc / ii)
これまで defun の後ろは「c:コマンド名」であったがここでは「c:」がない。この場合、ここに記載の文字は関数として登録される。
これまで「(/)」しか書いていなかったところに文字が書かれている。
/(スラッシュ)より前の文字列

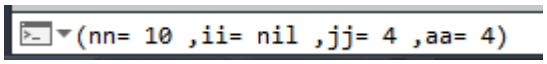
関数として使用するときの変数を意味する。
/(スラッシュ)より後の文字列
ここに書かれた文字は処理時に値が与えられても処理終了時には強制的に nil にされる。
これまでと違い最後の行が「(princ)」ではなく、「(progn aa)」となっている。最後の行に書かれたものがこの関数で得られる結果となる。progn は別になくてもよい

c:rei006 における処理
(setq nn(rei006_sub nn bb aa))
関数「rei006_sub」に nn、bb、aa を与えたときの計算結果を nn とするという意味。
サブルーチン内に取り込むことができるのはここに書かれた文字。
サブルーチンに入るとここでの
nn ⇒ aa
bb ⇒ bb
aa ⇒ cc
となり、サブルーチン前後で文字自体に関連性はなくなる。

(princ(list "nn=" nn ",ii=" ii ",jj=" jj ",aa=" aa))



ここで 2 を入力したとすると



これが表示される。

サブルーチン内において ii と jj は同じ計算をしているが ii はサブルーチン終了後に nil となるようにされているためサブルーチンの外では値がない。また、サブルーチン内では aa は 10 となっているが、サブルーチン外の aa には影響を与えない。

(getint)
数値入力を要求するだけでその数値を何に使うわけでもない。一時停止ができるのでこのような使い方をしている。Enter もしくは何か数値を入力することで次に進むことができる。

(strcat 文字列 1 文字列 2 文字列 3 ・・・)
文字列をつなぎ合わせて一つの文字列にする。
1 回目で使わなかったのは「(rtos nil 2 0)」だとエラーになってしまうため。
紹介しておきたかったので別の書き方をした。

例 7：コマンドとオブジェクトスナップと一括 undo
オブジェクトスナップ「端点」が ON になっている状態で座標(0,0)付近で「rei007」を実行する。

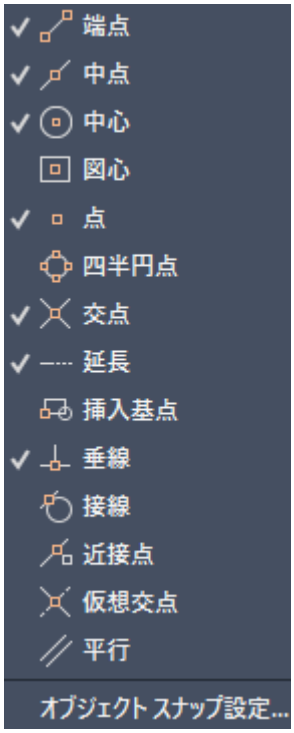


同じく「rei007_s」を実行する

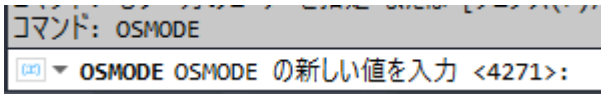


この 2 つのコマンドの(while)ループの中身は全く一緒なのにできるものが違う。
線分を連続して作成していく際に一つ前に作られた線分にスナップしてしまい、正しい位置に作成することができなくなっている。「rei007_s」では線分作成前にオブジェクトスナップを切り作成後に元に戻すという処理を行っている。

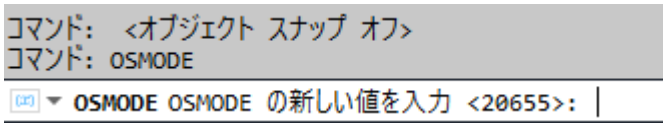
(getvar システム変数名)
システム変数の値を取得する。システム変数が何かわからない場合は調べること。



オブジェクトスナップのシステム変数「OSMODE」はチェックの入っているものの組み合わせで決まる。コマンドラインに「OSMODE」と入力すると現在の値がわかる。



オブジェクトスナップをオフにすると 16384 が足された値になる。
（全部チェックを外す方法でオフにすると 0 になる。）



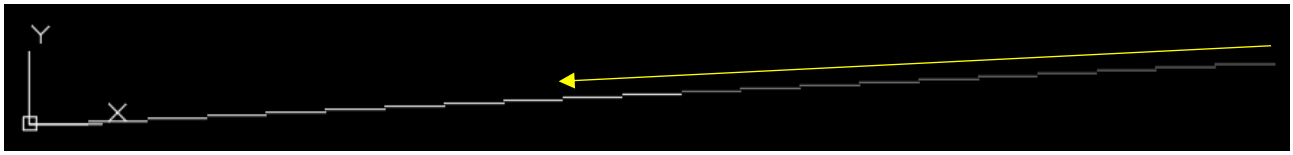
つまり 16384 より大きいときはオブジェクトスナップがオフになっている。

(if(< oss 16384)(setvar "OSMODE"(+ oss 16384)))
OSMODE の値が 16384 より大きいのにさらに 16384 足してはいけないので 16384 より小さいときだけ 16384 を足すようにしている。

(setvar システム変数名 システム変数の値)
システム変数名の値をシステム変数の値にする。

OSMODE を 0 にしないのはもしエラーで止まったときもともとのスナップの組み合わせがわからなくなるから。

線分が作成されている状態で undo すると一つずつ消えていく。



「rei007_u」のようにすると一回の undo で作成したものすべてが消えるようになる。

(command “undo” “be”)
undo の開始位置を設定
(command “undo” “e”)
undo の終了位置を設定

1 回の undo で“be”の位置まで戻れるようになる。

コマンド「rei007_e」は entmake を使う場合。entmake だとスナップを切る必要はないし一括 undo もできる。entmake を使えるときは極力使って command を使わざるを得ないときはスナップや undo に注意する。

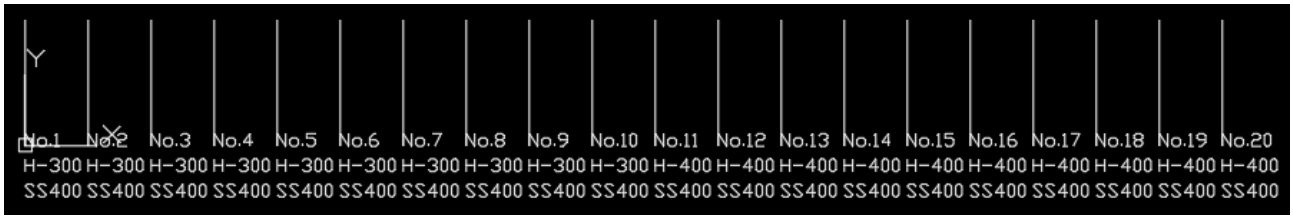
例 8：拡張オブジェクトデータ
オブジェクトのテキストデータには基本的にオブジェクトを構成するのに必要な内容しか入れることができない。拡張オブジェクトデータを使えば例えば線分に材質のデータを与えることができる。
いつでも使うものではないが知っておけば何かに使えるだろうと思われる。

コマンド「rei008」実行

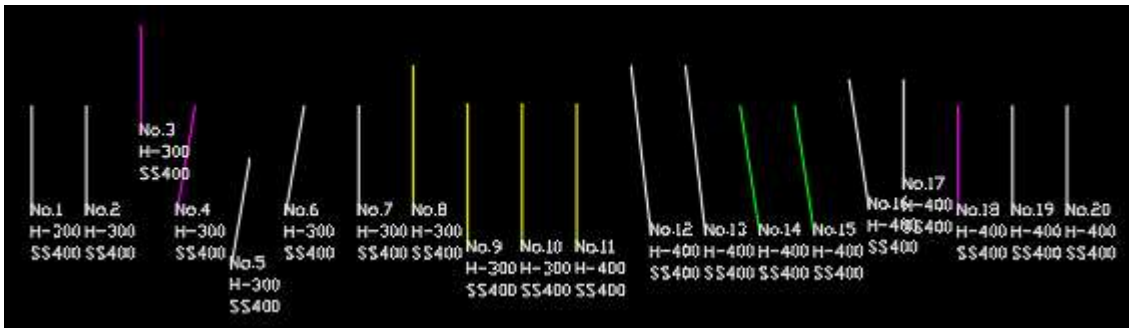


20 本の線分が作成される。プロパティを見ても特に違いの分からない線分だが、固有パラメータが設定されている。

コマンド「rei008_r」実行



それぞれの持つパラメータが表示される。



変形したり色を変えてもパラメータは変わらない。

(regapp 拡張データ名)

拡張データとして拡張データ名を使えるように定義する。

```
(entmake(list(cons 0 "line")
              (cons 10 pp1)(cons 11 pp2)
              (cons -3(list(list "hkou"
                                 (cons 1000(strcat "No."(rtos(1+ ii)2 0)))
                                 (cons 1000 "H-300")
                                 (cons 1000 "SS400")
                                 )))
```

□entmake で線分を作るのに必要な項目

□拡張データ

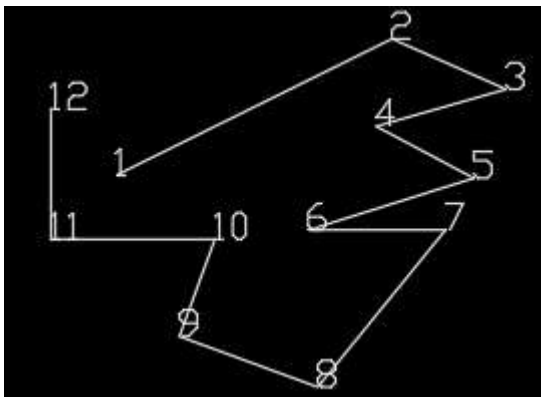
拡張データとして「hkou」を使えるようにしたのでグループコード-3 の中に記載する。

拡張データのグループコードの一覧
https://wiki.gz-labs.net/index.php/%E6%8B%A1%E5%BC%B5%E3%83%87%E3%83%BC%E3%82%BF

(mapcar '関数 リスト 1 リスト 2 ・・・・)
リストの構成要素に同じ関数を適用する。

(setq pp1(mapcar '+ pp1(list 50 0 0)))
ここで関数は「+」、リスト 1 とリスト 2 は 3 次元座標となっているので
もともとの pp1 を(x1,y1,z1)とすると
pp1 は (x1+50,y1+0,z1+0)という意味になる。
複数の変数に適用できる関数の場合はリストを複数書くことができる。

例 9：ポリラインの頂点に順番に番号を振る



ポリラインの頂点座標のグループコードはすべて 10 になっているので順番に選べるようにしている。
(assoc 10 ed)としたとき、10 が複数あると一番最初のものだけが選択される。

(ssget ":s")
選択するオブジェクトを 1 つに限定する。クリックした瞬間次に進むのであまり好きではない。

(vl-remove 変数 リスト)
リストの中に変数が存在している場合それを除去する。
このときリストが(a a a b c d)となっており変数が a のとき最初の a だけでなくすべての a が除去され、(b c d)となるので注意。

例 10：外部ファイル書き出し、読み込み
(open ファイルパス 読み込みオプション)
ファイルパスを開いてファイルディスクを作成する。

ここで、ファイルパスは"C:¥¥temp¥¥sampl.txt"となっている。「¥」は制御文字なので「¥¥」とする。
制御文字
http://docs.autodesk.com/CIV3D/2014/JPN/index.html?url=filesACD/GUID-C8E55E82-0104-4305-95E5-1671628A2FF3.htm,topicNumber=ACDd30e566326

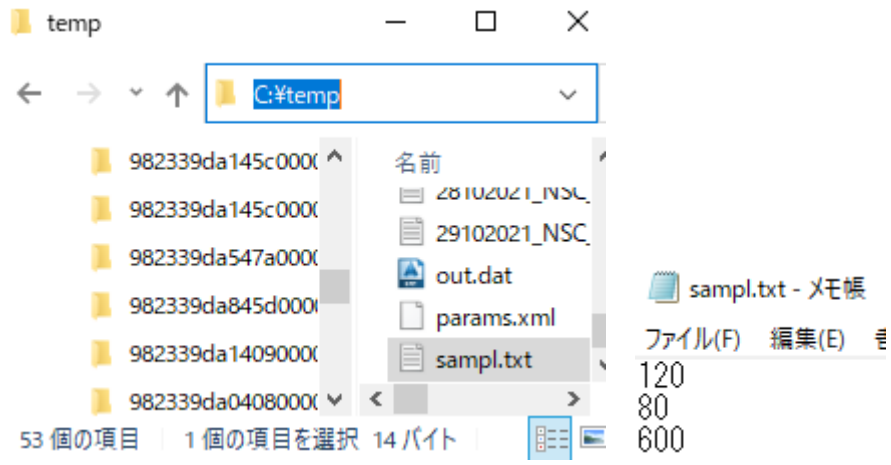
読み込みオプション
W：write　開いたファイルに上書き書き込みする。
A：add　開いたファイルに追記書き込みする。
R：read　開いたファイルに記載されたデータを読み込む。

(write-line テキスト ファイルディスク)
ファイルディスクによって指定されたファイルにテキストを書き込む。このとき書き込まれたテキストの後ろは改行となる。

(read-line ファイルディスク)
ファイルディスクによって指定されたファイルからテキストを 1 行読み込む。読み込んだ後、改行処理がなされ次に読み込むときは続きから読み込む。

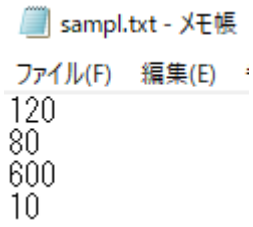
(close ファイルディスク)
ファイルディスクのファイルを閉じる。それまでに実施された改行は初期化される。

コマンド「rei010_w」を実行。



C:¥temp に「sample.txt」ができる。

コマンド「rei010_a」を実行。



テキストファイルの内容が追加される。

コマンド「rei010_r」を実行。

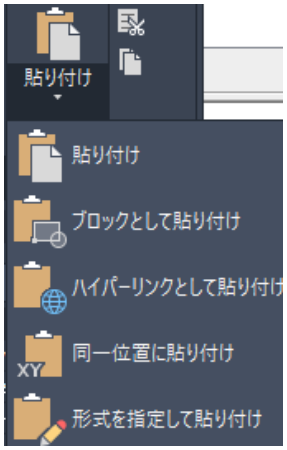


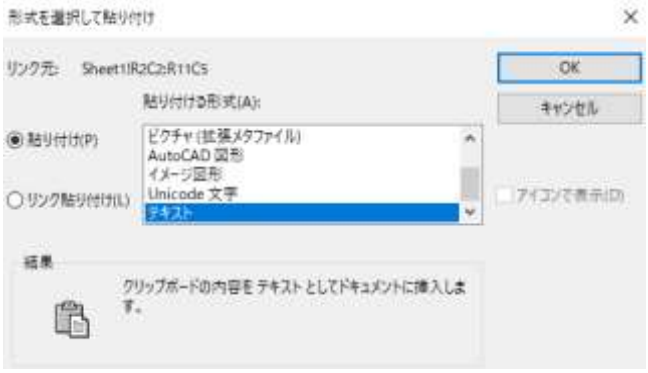
テキストファイルに書かれた数値をすべて合計した値が表示される。

例 11：エクセルからマルチテキストを作成してリスト化する

先にエクセルデータとマルチテキストの関係から解説する。

エクセルでデータ管理して DWG 上で使いたい場合は





「形式を指定して貼り付け」＞「テキスト」とするとマルチテキストで貼り付けられる。

マルチテキストの文字内容が格納されているグループコードは基本的には「1」だが、長すぎると特殊なことになる。

0611	仮設構造(架台,支保工他)	設計条件整理	設計条件整理	設計条件整理－設計条件整理
0621	仮設構造(架台,支保工他)	設計計画	設計計画	設計計画－設計計画
0631	仮設構造(架台,支保工他)	設計計算	設計計算	設計計算－設計計算
0632	仮設構造(架台,支保工他)	設計計算	動的解析,FEM解析	設計計算－動的解析,FEM解析
0634	仮設構造(架台,支保工他)	設計計算	計算断面図作成	設計計算－計算断面図作成
0641	仮設構造(架台,支保工他)	図面作成	一般図	図面作成－一般図
0643	仮設構造(架台,支保工他)	図面作成	詳細図	図面作成－詳細図
0644	仮設構造(架台,支保工他)	図面作成	配筋図	図面作成－配筋図
0645	仮設構造(架台,支保工他)	図面作成	施工要領図	図面作成－施工要領図
0646	仮設構造(架台,支保工他)	図面作成	手順図	図面作成－手順図
0647	仮設構造(架台,支保工他)	図面作成	その他図	図面作成－その他図
0651	仮設構造(架台,支保工他)	数量計算	数量計算	数量計算－数量計算
0661	仮設構造(架台,支保工他)	照査	設計計算照査	照査－設計計算照査
0662	仮設構造(架台,支保工他)	照査	図面照査	照査－図面照査
0663	仮設構造(架台,支保工他)	照査	数量計算照査	照査－数量計算照査

このデータを DWG に貼り付けてグループコードを取得する。

((-1 . <図形名: 27a249d9af0>) (0 . "MTEXT") (330 . <図形名: 282029a59f0>) (5 . "F6D6F") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "Defpoints") (100 . "AcDbMText") (10 200492.0 -189786.0 0.0) (40 . 140.0) (41 . 10046.3) (46 . 0.0) (71 . 1) (72 . 1) (3 . "0611¥t 仮設構造(架台,支保工他)¥t 設計条件整理¥t 設計条件整理¥t 設計条件整理 - 設計条件整理¥¥P0621¥t 仮設構造(架台,支保工他)¥t 設計計画¥t 設計計画¥t 設計計画 - 設計計画¥¥P0631¥t 仮設構造(架台,支保工他)¥t 設計計算¥t 設計計算¥t 設計計算 - 設計計算¥¥P0632¥t 仮設構造(架台,支保工他)¥t 設計計算¥t 動的解析,FEM 解析¥t 設計計算 - 動的解析,FEM 解析¥¥P0634¥t 仮設構造(架台,支保工他)¥t 設計計算¥t 計算断面図作成¥t 設計計算 - 計算断面図作成¥¥P0641¥t 仮設構造(架台,支") (3 . "保工他)¥t 図面作成¥t 一般図¥t 図面作成 - 一般図¥¥P0643¥t 仮設構造(架台,支保工他)¥t 図面作成¥t 詳細図¥t 図面作成 - 詳細図¥¥P0644¥t 仮設構造(架台,支保工他)¥t 図面作成¥t 配筋図¥t 図面作成 - 配筋図¥¥P0645¥t 仮設構造(架台,支保工他)¥t 図面作成¥t 施工要領図¥t 図面作成 - 施工要領図¥¥P0646¥t 仮設構造(架台,支保工他)¥t 図面作成¥t 手順図¥t 図面作成 - 手順図¥¥P0647¥t 仮設構造(架台,支保工他)¥t 図面作成¥t その他図¥t 図面作成 - その他図¥¥P0651¥t 仮設構造(架台,支保工他)") (1 . "¥t 数量計算¥t 数量計算¥t 数量計算 - 数量計算¥¥P0661¥t 仮設構造(架台,支保工他)¥t 照査¥t 設計計算照査¥t 照査 - 設計計算照査¥¥P0662¥t 仮設構造(架台,支保工他)¥t 照査¥t 図面照査¥t 照査 - 図面照査¥¥P") (7 . "Standard") (210 0.0 0.0 1.0) (11 1.0 0.0 0.0) (42 . 9567.88) (43 . 3595.98) (50 . 0.0) (73 . 1) (44 . 1.0))

テキストが格納されているグループコードは「3」が 2 つと「1」が 1 つある。
何文字かは知らないが文字数制限がありそれを超えるとグループコード「3」が使われるようになる。
並んでいる順につなげると元の文字列を復元できる。entmake の際は「3」ばかり使って「1」が一つもなくても自動で整理してくれる。極端な話、(3 . “a”)(3 . “b”)(3 . “c”)と一文字ずつ入力しても勝手につないで(1 . “abc”)にしてくれる。

制御文字について
¥t：タブをあらわす。エクセルで右のセルに移るとき使用される。
¥¥P：改行をあらわす。エクセルで下のセルに移るとき使用される。大文字。
¥¥n：改行をあらわす。普通は上記の「¥¥P」になるはずだが、一部環境でこちらが使われるのを確認している。原因は不明。

ここからコード解説

(while(setq tt3(cdr(assoc 3 ed)))	グループコード「3」が見つからなくなるまでて続ける。
(setq tt1(strcat tt1 tt3))	「3」の中身をつないだ文字列を作る。
(setq ed(vl-remove(cons 3 tt3) ed))	今回使った「3」を除去する。
)	ループ

(setq jj(vl-string-search "¥¥P" tt1))
文字列「tt1」から「¥¥P」を探して何番目の文字かを「jj」として返す。見つからないときは nil。

(substr テキスト 整数 1 整数 2)
テキストの整数 1 番目の文字から整数 2 個の文字列を作成する。
整数 2 は省略可。省略した場合残りすべての文字になる。テキストのもともとの文字数より大きい値を整数 2 に入れても問題なく残りすべての文字列が得られる。

(strlen テキスト)
テキストの文字数。

コマンド「rei011」でマルチテキストを選択することによりリスト「ctL」が得られる。
今回の例だと ctL の中身は以下の通り

(("13_3" . "図面照査") ("13_2" . "照査") ("13_1" . "仮設構造(架台,支保工他)") ("13_0" . "0662") ("12_3" . "設計計算照査") ("12_2" . "照査") ("12_1" . "仮設構造(架台,支保工他)") ("12_0" . "0661") ("11_3" . "数量計算") ("11_2" . "数量計算") ("11_1" . "仮設構造(架台,支保工他)") ("11_0" . "0651") ("10_3" . "その他図") ("10_2" . "図面作成") ("10_1" . "仮設構造(架台,支保工他)") ("10_0" . "0647") ("9_3" . "手順図") ("9_2" . "図面作成") ("9_1" . "仮設構造(架台,支保工他)") ("9_0" . "0646") ("8_3" . "施工要領図") ("8_2" . "図面作成") ("8_1" . "仮設構造(架台,支保工他)") ("8_0" . "0645") ("7_3" . "配筋図") ("7_2" . "図面作成") ("7_1" . "仮設構造(架台,支保工他)") ("7_0" . "0644") ("6_3" . "詳細図") ("6_2" . "図面作成") ("6_1" . "仮設構造(架台,支保工他)") ("6_0" . "0643") ("5_3" . "一般図") ("5_2" . "図面作成") ("5_1" . "仮設構造(架台,支保工他)") ("5_0" . "0641") ("4_3" . "計算断面図作成") ("4_2" . "設計計算") ("4_1" . "仮設構造(架台,支保工他)") ("4_0" . "0634") ("3_3" . "動的解析,FEM 解析") ("3_2" . "設計計算") ("3_1" . "仮設構造(架台,支保工他)") ("3_0" . "0632") ("2_3" . "設計計算") ("2_2" . "設計計算") ("2_1" . "仮設構造(架台,支保工他)") ("2_0" . "0631") ("1_3" . "設計計画") ("1_2" . "設計計画") ("1_1" . "仮設構造(架台,支保工他)") ("1_0" .

"0621") ("0_3" . "設計条件整理") ("0_2" . "設計条件整理") ("0_1" . "仮設構造(架台,支保工他)") ("0_0" . "0611"))

それぞれのデータの前に「行番号_列番号」というテキストがついているので assoc で行列を指定してデータを取り出すことができる。行列番号は 0 からスタートしている(その必要はなかったけど慣例で)。

↓例で使ったエクセルデータ

0611	仮設構造(架台,支保工他)	設計条件整理	設計条件整理	設計条件整理 - 設計条件整理
0621	仮設構造(架台,支保工他)	設計計画	設計計画	設計計画 - 設計計画
0631	仮設構造(架台,支保工他)	設計計算	設計計算	設計計算 - 設計計算
0632	仮設構造(架台,支保工他)	設計計算	動的解析,FEM 解析	設計計算 - 動的解析,FEM 解析
0634	仮設構造(架台,支保工他)	設計計算	計算断面図作成	設計計算 - 計算断面図作成
0641	仮設構造(架台,支保工他)	図面作成	一般図	図面作成 - 一般図
0643	仮設構造(架台,支保工他)	図面作成	詳細図	図面作成 - 詳細図
0644	仮設構造(架台,支保工他)	図面作成	配筋図	図面作成 - 配筋図
0645	仮設構造(架台,支保工他)	図面作成	施工要領図	図面作成 - 施工要領図
0646	仮設構造(架台,支保工他)	図面作成	手順図	図面作成 - 手順図
0647	仮設構造(架台,支保工他)	図面作成	その他図	図面作成 - その他図
0651	仮設構造(架台,支保工他)	数量計算	数量計算	数量計算 - 数量計算
0661	仮設構造(架台,支保工他)	照査	設計計算照査	照査 - 設計計算照査
0662	仮設構造(架台,支保工他)	照査	図面照査	照査 - 図面照査

例 12：ブロックを分解してそれによって新たにできるブロックがなくなるまで分解する
ブロックを分解すると新たなブロックが生まれるときそれを選択しなおさずに自動選択してまた分解するという流れ。
(vla-copy VLA オブジェクト)
VLA オブジェクトを同位置でコピーする

(entlast)
図面内で最後に作成されたオブジェクトの図形名を取得する。
ここでは直前でブロックをコピーしているのでコピーしてできたブロックが対象になる。
この図形名を取得するためだけにコピーした。コピーせずにもっといい方法があればそのほうがよい。

```
(while(< ii num)
  (setq tmpl(ssname eset ii))
  (setq nmL(cons tmpl nmL))
  (setq ii(1+ ii))
  )
```

選択したブロックの図形名で構成されたリストを作って nmL とする。

```
(while nmL                                          nmL がなくなるまで続ける。
  (setq tmpl(car nmL) nmL(cdr nmL))                 nmL の 1 個目を取得し、1 個目を削除する。
  (vla-explode(vlax-ename->vla-object tmpl))        上記で取得した 1 個目の図形を分解する。
  (setq eset(ssget "X"(list(cons 0 "INSERT"))))     図面内すべてを検索し、ブロックを取得する。
  (setq ii 0)
  (while(not(equal bmpl(ssname eset ii)))           コピーしたオブジェクトが見るかるまでループ。
    (setq nmL(cons(ssname eset ii)nmL))             分解してできたブロックがあれば nmL に加える。
    (setq ii(1+ ii))
    )
  )
```

(setq tmpl(car nmL) nmL(cdr nmL))
1 個目の図形は削除されているので 2 週目に 1 個目を選択すればそれはもともとの 2 個目である。

(ssget "X")
"X"は図面全体を選択するオプションなのですべてのブロックが eset になる。
(equal 図形名 1 図形名 2)
図形名 1 と図形名 2 が同じとき T、違うとき nil。本質的には(= a b)と同じだが、図形名に対して「=」を使うとどんな場合でもすべて nil になってしまうので「equal」を使わなくてはならない。
ここではさらに not がついているので図形名が異なるとき T となる。
ここで eset は図面内のすべてのブロックで(ssname eset ii)とすると、ii を大きくするにしたがって新しい順にブロックの図形名を取得することになる。

←旧　　新→
「対象となるブロック」 ⇒「その他の図形」 ⇒「コピーブロック」⇒「分解でできたブロック」
（分解されてもうない）

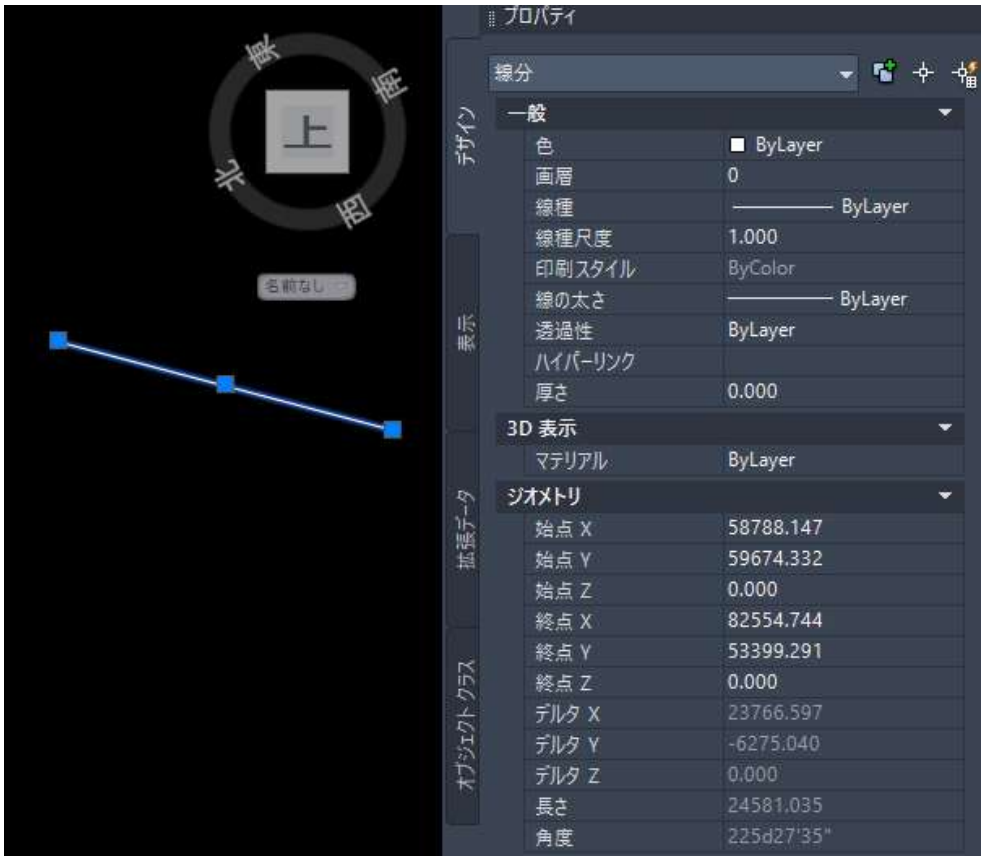
となっているので、新しい順にコピーブロックと図形名が同じかどうか調べて同じになるまで続ければ分解してできたブロックの図形名をすべて取得できたことになる。コピー以前に作られていたブロックは含まれない。
ここで、分解でブロックが新たに生成された場合 nmL が増えることになり、nmL が空になるまで処理を続ける。

(entdel 図形名)
図形名のオブジェクトを削除する。
コピーブロックは上記の新しいブロックを取得するために作ったのでもう用済み。残っては困る。

例 13：ucs による影響

グループコードの座標は図面の ucs に関わらずワールドの時のものになるが、ucs がワールドでないとき(getpoint)など図面内をクリックするものをつかうと、現在の ucs に合わせて座標を取得してしまうので思ったようにいかない場合がある。

適当な ucs がワールドでない図面を用意する。

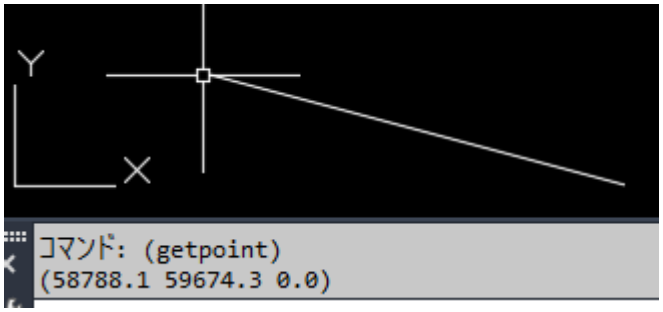


この線分のグループコードを取得すると

((-1 . <図形名: 28202b69f30>) (0 . "LINE") (330 . <図形名: 282027d49f0>) (5 . "9043") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbLine") (10 22637.5 -80651.2 0.0) (11 5396.14 -98171.5 0.0) (210 0.0 0.0 1.0))

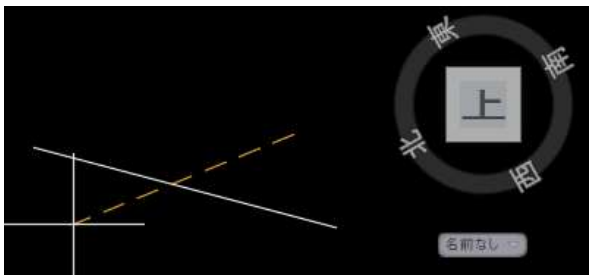
座標が全然違う。

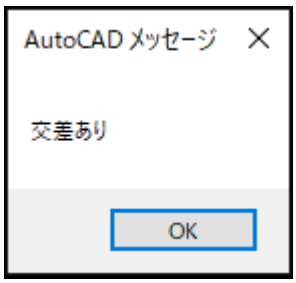
コマンドラインに(getpoint)と入力してこの線分の始点を取得する。



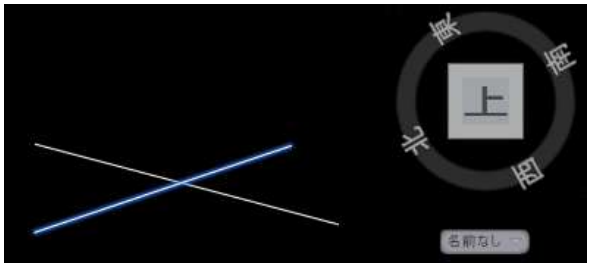
現在の ucs で座標が取得されてしまう。

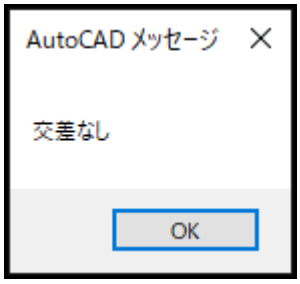
なにが困るか？
コマンド「rei013_1」を実行し線分と交差するように 2 点選択する。



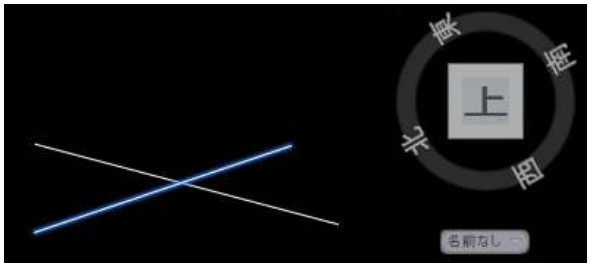


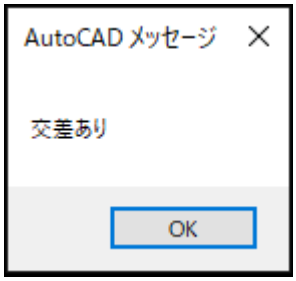
交差する線分を用意し、コマンド「rei013_2」を実行しこれを選択する。





コマンド「rei013_3」を実行し同様にする。





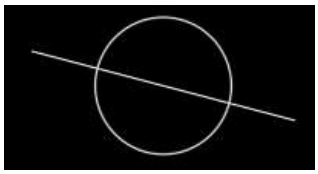
(ssget “F”(list 座標 座標 座標 ・・・))
フェンス選択でオブジェクトを取得する。
フェンス選択とはリスト内の座標でポリラインを構成し、そのポリラインと交差しているオブジェクトを選択するようなもの。
rei013_1 のとき、この座標は getpoint で取得しているため、固有 ucs のものとなり、フェンス選択できたが、rei013_2 のときオブジェクトから座標を取得しているためワールド座標となり、全然違う場所にポリラインが作成されることになったため交差なしとなった。
rei013_3 では rei013_2 と同じ方法で交差するようにしている。

(command "ucs" "w")
コマンドで ucs をワールドにしている。これによりワールド座標系でフェンス選択することになるので交差することになる。ここだけで問題は解決しているが、もともと固有 ucs だったものをワールドに変えてしまうので、元の ucs に戻しておきたい。前後の処理はそのために必要。

(command "-view" "D" "xixixixixiviewixixixixix")
(command "-view" "S" "xixixixixiviewixixixixix")
(command "ucs" "w")
(setq gset(ssget "F" (list pp1 pp2)))
(command "-view" "R" "xixixixixiviewixixixixix")
(command "-view" "D" "xixixixixiviewixixixixix")

"xixixixixiviewixixixixix"はビューの名前。既存のビューとして絶対に存在しなさそうなものにしたので、こう書かなければいけないということではない。これを登録し（S)、ucs を変更して処理が終わった後に呼び出し（R)、最後に削除する（D)。最初の D はもしエラーなどで途中で止まってビューが残ってしまった場合、削除しておかないと S のときに上書きするかどうか求められてコマンドが変わってしまうのであらかじめ消しておくために書いてある。

例 14：mapcar と lambda の組み合わせ
まずコマンド「rei014」を実行して 2 点選択する。



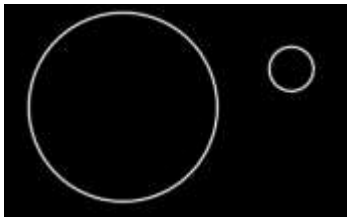
選択した 2 点をつなぐ直線とその半分の長さを直径とする円を中心に作る。
(lambda(変数)(数式))
関数を作成する関数
(lambda(x y)(* 0.5(+ x y)))
x と y を入力されたら 0.5*(x+y)を返す関数を一時的につくる。
(mapcar)と組み合わせることで 2 点の座標から中心点の座標を求めることができる。

直後にコメントアウトしてあるものはこれでも同じ計算ができるというもの。3 行使って長い。

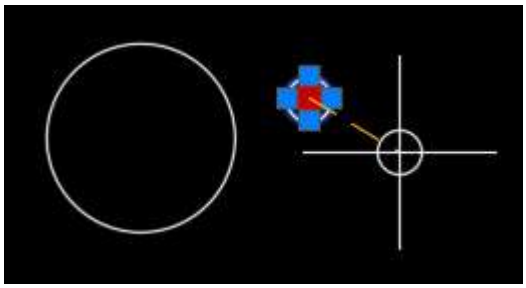
コマンド「rei014_1」ではちょっと長いけど cond を使うことでリスト内の値に対して条件付きで操作を選べる。
(rem 値 1 値 2)
値 1 を値 2 で除算したときのあまり
リスト(0 1 2 3 4 5 6 7 8 9)に対して奇数のときは負の値にし、偶数のときは 10 を足すという処理なので結果は(10 -1 12 -3 14 -5 16 -7 18 -9)
While を使っても似たようなことは書けると思うが、mapcar だと変数を一切使っていない。

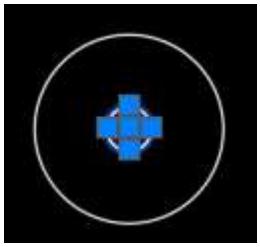
例 15：リアクター関数
まだ完全にマスターしていないので要研究。

円を二つ用意する（半径は違っていたほうが何が起きているかわかりやすい）



コマンド「rei015」で２つの円を選択。
コマンド終了後どちらか片方を動かすともう一方がついていくという処理が自動的に発生する。





この効果は図面を閉じるまで続く。図面を閉じると効果がなくなるので次回図面を開いたときに同じようにコマンドしなくてもいい方法をあとで説明する。

(cdr(assoc 5 ed0))
一つ目の円のグループコード(ed0)からハンドル(グループコード 5)を取り出す。
ハンドルとは各オブジェクトに固有で与えられている番号のこと。図形名も固有ではあるがこれは図面を開いて再作図するときに値が変わってしまうので記憶しておいても次回使えないがハンドルは変化しないので図面を開きなおしても使える。

(entmod(list(assoc -1 ed0)(list -3(list "rei015_xdata"(cons 1005 ha1)))))
一つ目の円のグループコード(ed0)から図形名を取り出し、拡張データ「rei015_xdata」にもう一つの円のハンドル(ha1)を追加する。これにより２つの円の関連性が拡張データを参照することによって調べることができるようになる。

(vlr-object-reactor (list va0) nil (list(cons :vlr-modified 'rei015_callback)))
オブジェクトに変化があったときに発動するリアクター関数を定義する。
(list va0)：リアクター関数を与える vla オブジェクト(va0)のリスト。
nil：リアクタ オブジェクトに関連付ける任意の AutoLISP データ。ここで使用している「vlr-modified」では参照するデータが必要ないため nil としている。
(list(cons :vlr-modified 'rei015_callback))：コールバック関数(‘rei015_callback)とコールバックが発生する条件(:vlr-modified)のリスト。条件(円の修正)が起きたときコールバック(中心を合わせる)を実行する。コールバック関数はここに直接書き込んでもよいが、基本的には長くなると思われるので別途定義しなおしている。

(defun rei015_callback (owner reactor params / serv ed0)
コールバック関数(rei015_callback)を定義する。ここで変数が３つ(owner,reactor,params)与えられているが後ろの２つはどうやって使うのかまだ知らない。owner は vlr-object-reactor で入力したリアクター関数を与える vla オブジェクトのリストの中身をあらわす。今回はリストの中身が１つだが、リストの中身が複数ならそのすべてに適用するはず。

(vlr-remove-all)
リアクターを除去する。とりあえずつけておけばいいのではないかと思う。今回はあってもなくてもなにが違うのかわからなかった。

(vlax-vla-object->ename VLA オブジェクト)
VLA オブジェクトの図形名を取得する。

(handent ハンドル)
ハンドルの図形名を取得する。

(if(not(equal(assoc 10 ed0)(assoc 10 serv))) (entmod(list(assoc -1 serv)(assoc 10 ed0))))
２つの円の中心座標を比較し、異なるときは円を移動させる。
ここで、if がなければ片方の円を動かすともう一方を動かし、その動きに連動して最初の円をまた動かすという処理が発生し無限ループとなってしまって Autocad がクラッシュするので注意。座標が全く動いていなくても entmod をすれば修正したという判定になるため if をつけて制限しなくてはならない。
ここで(equal(assoc 10 ed0)(assoc 10 serv))は座標リストを比較しており、「=」を使うとうまく判定できなかったので「equal」を使っている。

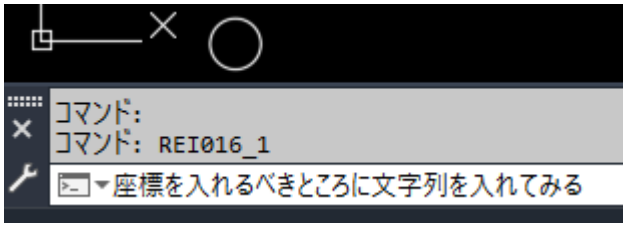
リアクターの自動設定について
コメントアウトでも書かれている通り、(defun)はコマンドを作成するためのものなのでこれで囲んでいないと lsp ファイルを読み込んだ瞬間に実行されることになる。

(if(setq s(ssget "_X"(list(cons 0 "CIRCLE")(list -3(list "rei015_xdata")))))
図面全体を選択して円であり、拡張データ rei015_xdata を持っているものを s として定義することができれば次に進む。該当するものがなければこの処理は発生しない。図面を開きなおしたときリアクターは無効になっているが、前回図面を閉じたときにリアクター関数を設定しているのならその図形は拡張データを持っているので対象にとることができる。

(repeat 数値 処理)
処理を数値回繰り返す。「while」でもほとんど同じことができるけど事前に i=0 としておかなくてよいので繰り返し回数がわかっているときはこっちのほうがよいかもしれない。条件に合致するオブジェクトすべてに対してリアクター関数を設定している。

例 16：エラー処理

「rei016_1」を実行



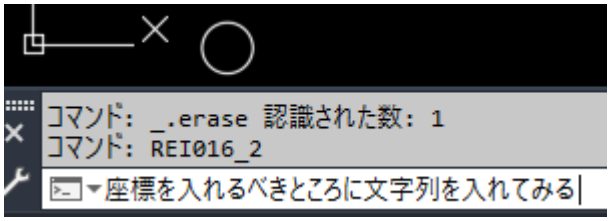
円を一つ作ってテキスト入力を要求してくる

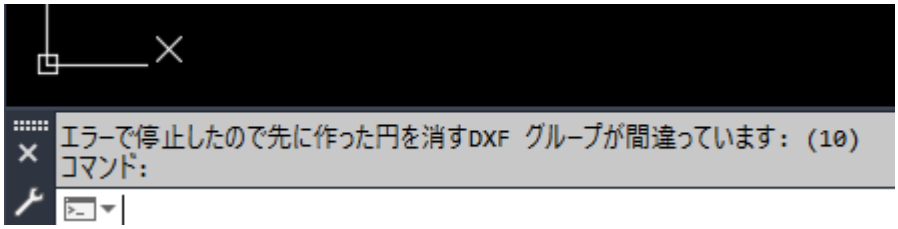
なんでもテキストを入力



エラーで停止する

「rei016_2」を実行





エラーに反応して円を消す処理が実行される

```
(defun *error*( msg )
  (entdel xx)
  (princ(strcat "¥n エラーで停止したので先に作った円を消す エラー内容:" msg))
  )
```

エラーが起きたときにこの中身を実行する。すべてのコマンドで同じ関数「*error*」を使う仕様になっているので最初に

(defun c:rei016_2(/ *error*)

としてこの関数固有のエラー処理であるとしておかなければほかの関数に共有されてしまうので注意。

変数「msg」にはエラーメッセージが自動的に入れられる。

ここでは msg は「DXF グループが間違っています: (10)」

(setq xx(entmakex ～))

作ったものの図形名を取得したいとき entmake ではなく entmakex を使うとよい。entlast でもいいと思うけど好みで。追記：別途定義しておいて呼び出したほうがいい。最後に普通に戻すのを忘れないこと。

＜おまけ編＞「entmake 生成.xlsx」について

いろいろな種類のオブジェクトを作るようになると entmake でオブジェクトを作る際、どのグループコードが必要なのかわからないこともあるので、必要なグループコードを探るために使用する。

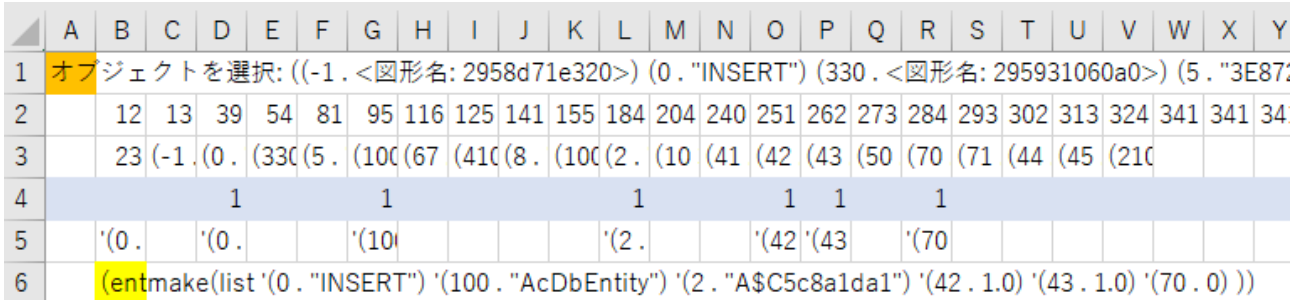
(entget(car(entsel)))でグループコードを取得



この部分をコピー（「オブジェクトを選択」はあってもなくてもいいのでわざわざ「((-1」の前にカーソル合わせず選択したほうが楽)。

F2 キーを押すとコマンドラインが別窓で開くのでやりやすくなる。

セル A1(オレンジのところ)に貼り付け。

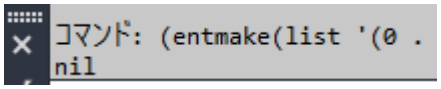


3 行目にグループコードの数字が見えるので、4 行目(青いところ)にそのグループコードが必要であれば「1」と記入する。

セル B6(黄色いところ)に必要なグループコードをまとめて entmake の型にしたものが表示される。

このセルをコピーしてコマンドラインに貼り付ける。

必要なグループコードが足りていないときは何も作成されず nil と表示される。

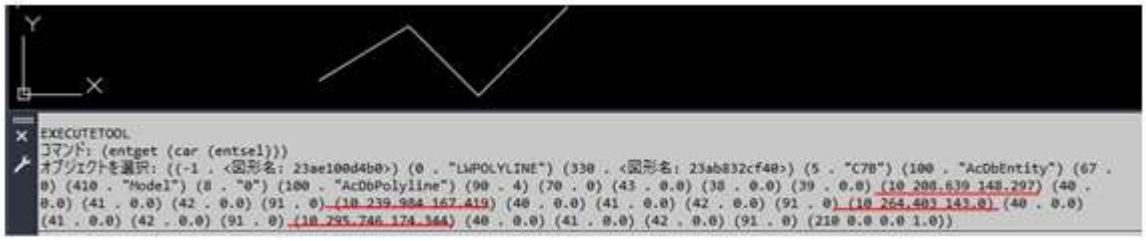


＜ちょっと上級編＞基準座標系とポリラインの成り立ち

グループコード 210 には法線ベクトルの情報が書かれている。ポリラインのグループコードがどうなっているかとともに解説する。

ポリラインの成り立ち

ポリラインのグループコードの座標には X、Y しかない。



(10. X Y) が頂点の数だけ出てくる。

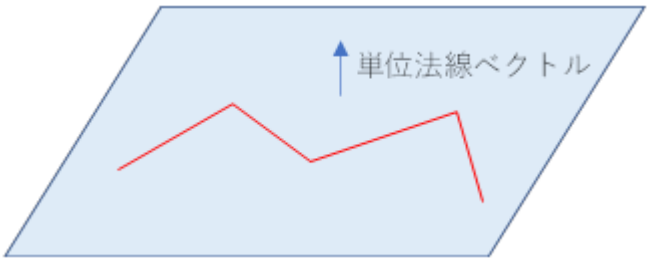
entmake するときに必要なものだけ解説

(cons 0 "LWPOLYLINE")	オブジェクトの種類
(cons 100 "AcDbEntity")	この 100 が 2 つないと entmake できない
(cons 100 "AcDbPolyline")	内容同じでよい
(cons 90 nn)	ポリラインの点の数 nn
(cons 70 1)	閉じてる:1　閉じてない:0
(cons 10(list X Y))	90 で指定しただけ座標を書く
(cons 42 0.0)	円弧のとき必要　書かなければ 0
・・・	座標 10 は書く順番が大事
(cons 38 hh)	ポリラインの高度
(cons 210(list A B C))	ポリラインの UCS による単位法線ベクトル

210 と 38 と 10 の関係について

ポリラインの点はすべて同一面内に存在する。

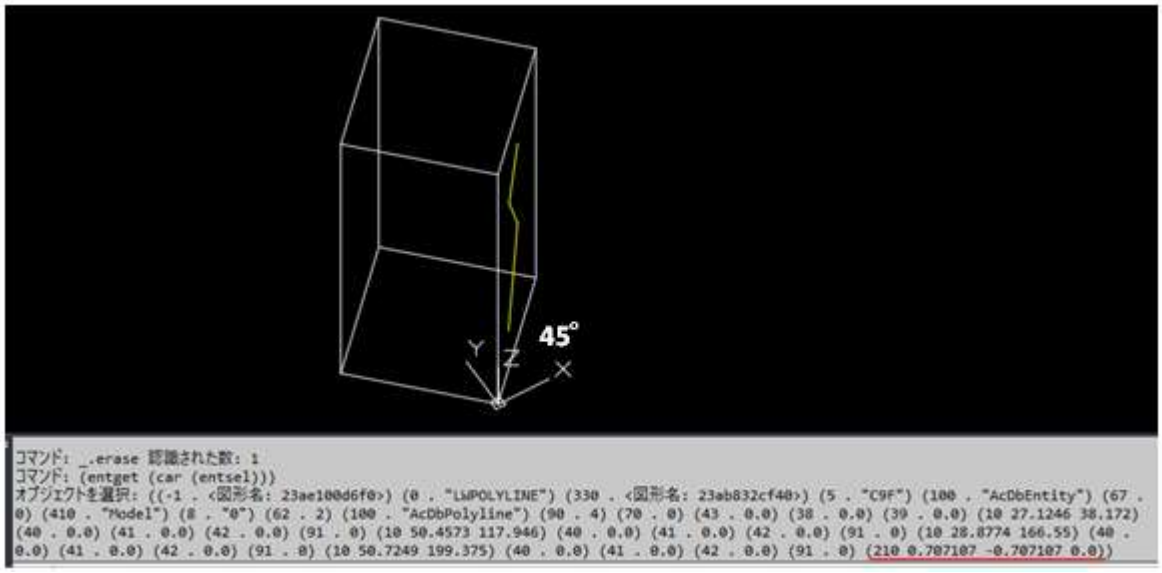
210 はその面の単位法線ベクトルを表す。



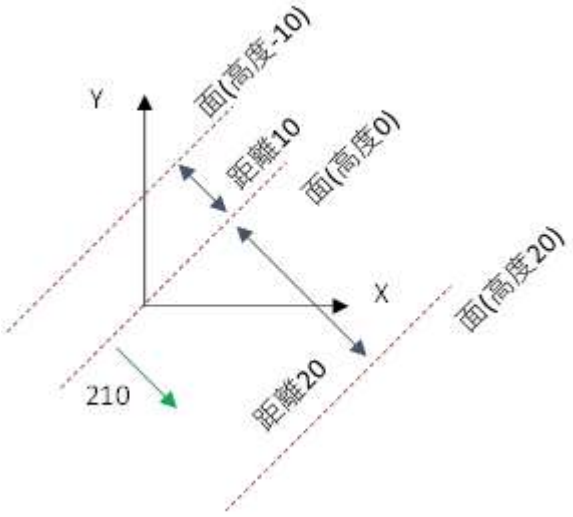
UCS ワールドの場合　(210. 0 0 1)　になる。

図の位置だと(210 0.707107 -0.707107 0.0)になる。

0.707107≒√0.5

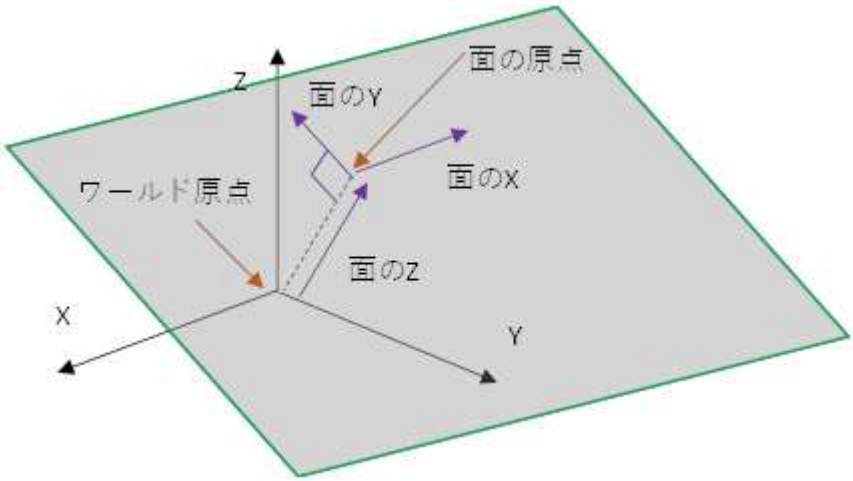


38 は高度を表す。
210 で構成される面が（X,Y,Z）＝（0,0,0）を通るとき高度 0 となる。
上の図を平面的に見ると



円やブロックには 38 が存在しない
代わりに座標 10 に（X,Y,Z）を入力する　この Z が高度になる

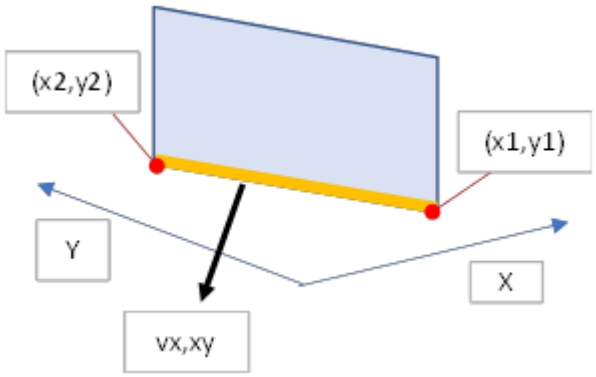
（X,Y）＝（0,0）はどこか？
ワールドにおける原点（0,0,0）からオブジェクトが存在する面に垂直な線を引いたとき
その線と面の交点が（X,Y）＝（0,0）になる
X 方向はワールドにおける XY 平面に平行になる（ワールドにおける Z 差分が 0）



42 カーブについて参考 https://qiita.com/tydesign/items/cab9d9cf6a625204707c

ポリラインの 3 次元配置計算その 1

●法線方向を決定する



面が通る 2 点を（x1,y1)、(x2,y2）とする

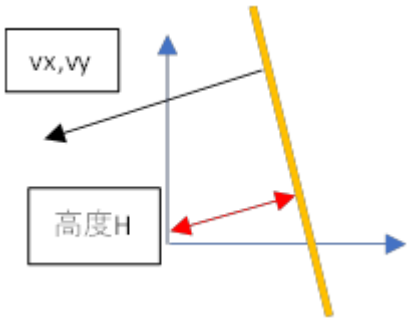
a =x2-x1　　b =y2-y1

法線ベクトルを vx,vy とする

vx = -b　　vy = a

これを長さで割って単位ベクトルにする（割ったことにして同じ文字 vx,vy を使う）

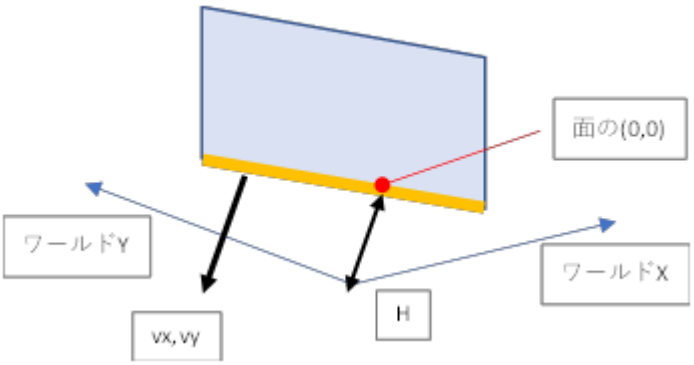
●面の高度を求める



面の高度はワールド座標系における(X,Y,Z)=(0,0,0)から面までの垂線の長さになる
面内の点（どこでもいい）のワールド座標系における座標をひとつ用意する
前述の(x1,y1,0)を使う

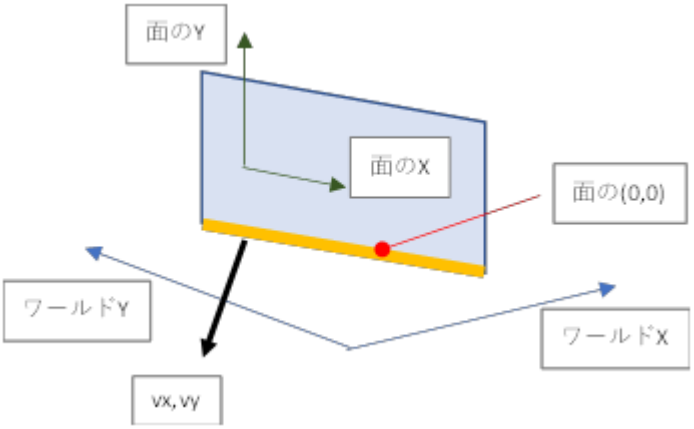
内積を計算すると面の高度になる
H = vx・x1 + vy・y1
x1,y1 は面内のどの座標であっても内積の仕組みにより H の計算結果は同じになる
H = vx・x2 + vy・y2　（これも正しい）
（この図の場合だと H の値は負になる）

●面内座標系における(0,0)の位置を求める



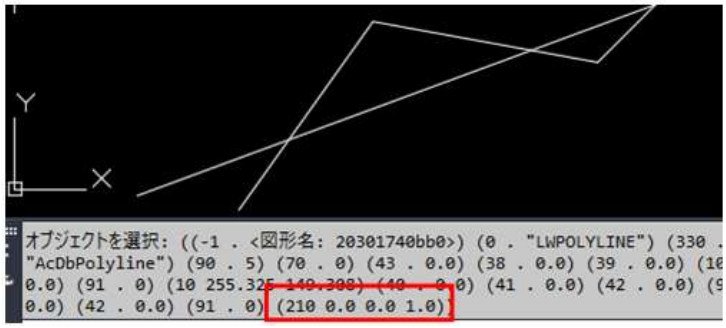
面の原点はワールド座標系における(X,Y,Z)=(0,0,0)から面までの垂線の交点になる
面の原点のワールド座標系における位置は
X0=H・vx　　Y0=H・vy

●面内の X 方向、Y 方向のベクトルを求める



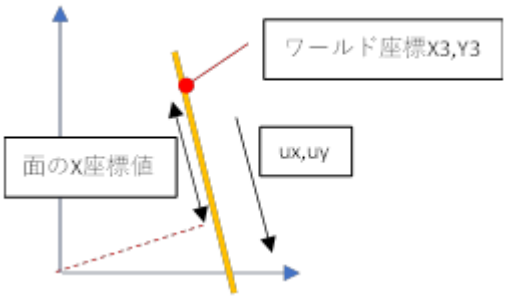
面はワールド XY 平面に対して垂直なので、面の Y 方向はワールドの Z 方向と同じ

XY 平面にポリラインを引いたときの 210 の値は（0 0 1）　⇒　z 軸正方向



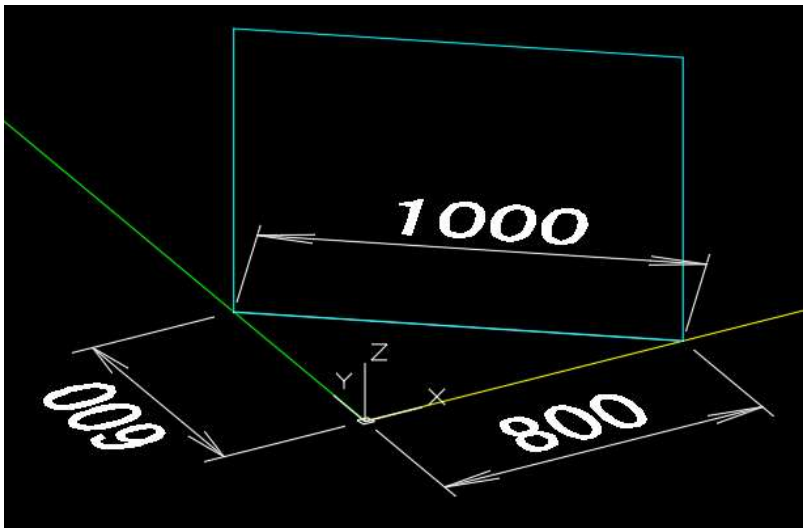
面の X 方向は面の法線ベクトルが手前に向かってくるように見たときの右向きになる
面の X 方向を(ux,uy)とすると
ux = -vy　　uy = vx

ワールド座標系の座標を面の座標系に変換する
ワールド座標系の座標値と面の X ベクトルの内積を計算することで面の X 座標を求めることができる



面の X= x3・ux + y3・uy

ポリラインの 3 次元配置計算その 1.1
実際の数値を見ながらポリラインについて解説するため下図のように面を作る



3 辺の長さがすべて整数なので解説しやすい

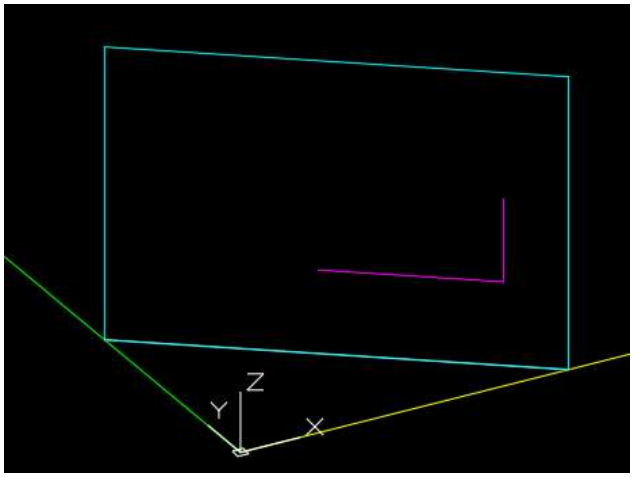
この面にポリラインを作る

1 点目　(X,Y,Z)＝（368,324,200）

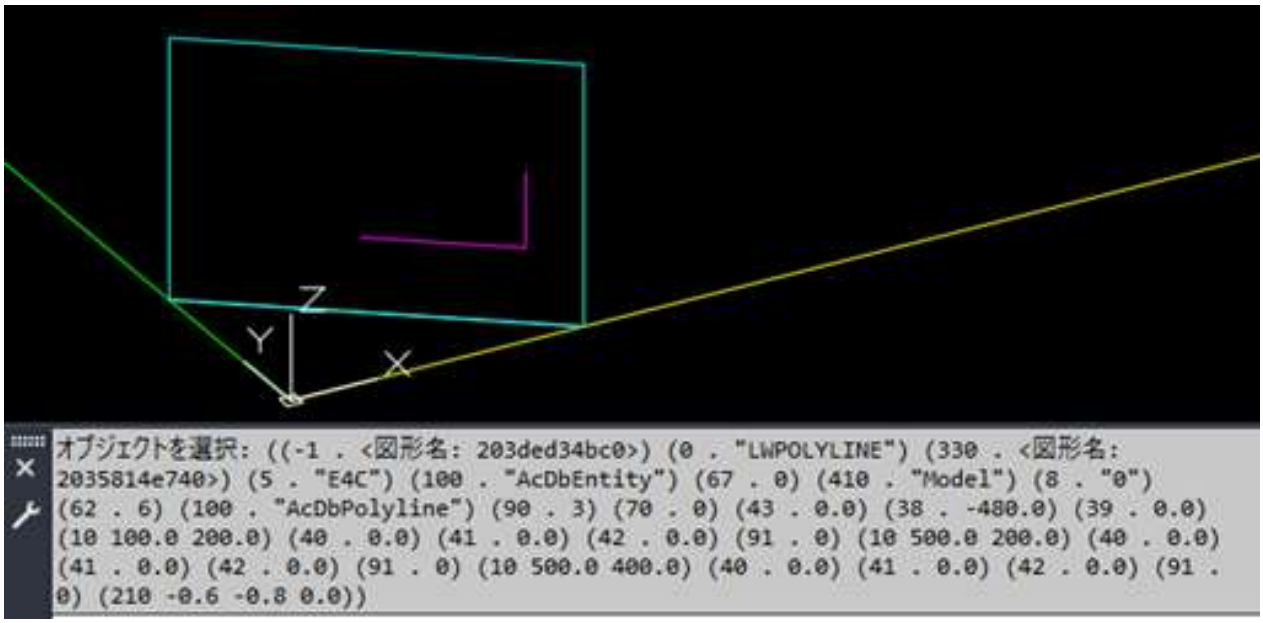
2 点目　(X,Y,Z)＝（688,84,200）

3 点目　(X,Y,Z)＝（688,84,400）

とする



このポリラインのグループコードを取得する



解説が書いてないものはよくわかってないし、あまり重要ではない

(-1 . <図形名: 203ded1f900>)　　図形名(entmake には不要)

(0 . "LWPOLYLINE")　　オブジェクトの種類

(330 . <図形名: 2035814e740>)

(5 . "D30")　　ハンドル(entmake には不要)

(100 . "AcDbEntity")　　ポリラインであることを表す　entmake に必要

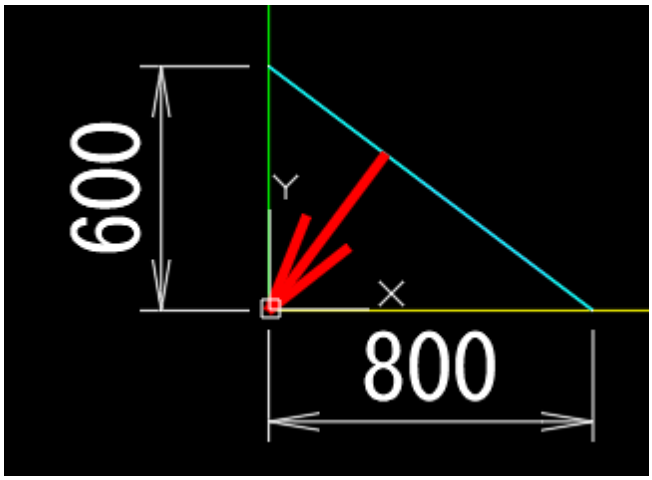
(67 . 0)
(410 . "Model")
(8 . "0")　　画層名(entmake には不要)
(62 . 6)　　インデックスカラー(entmake には不要)
(100 . "AcDbPolyline")　　ポリラインであることを表す　entmake に必要
(90 . 3)　　ポリラインの点の数　3 点
(70 . 0)　　閉じてない⇒0　閉じてる⇒1
(43 . 0.0)　　グローバル幅(entmake には不要)
(38 . -480.0)　　ポリラインの高度
(39 . 0.0)
(10 100.0 200.0)　　1 点目の座標
(40 . 0.0)　　2 点間でグローバル幅が変化するときに使う
(41 . 0.0)　　2 点間でグローバル幅が変化するときに使う
(42 . 0.0)　　円弧のとき必要
(91 . 0)
(10 500.0 200.0)　　2 点目の座標
(40 . 0.0) (41 . 0.0) (42 . 0.0) (91 . 0)　　上に同じ
(10 500.0 400.0)　　3 点目の座標
(40 . 0.0) (41 . 0.0) (42 . 0.0) (91 . 0)　　上に同じ
(210 -0.6 -0.8 0.0)　　ポリラインの法線方向

ワールド座標系における座標値とグループコードの座標値が違う

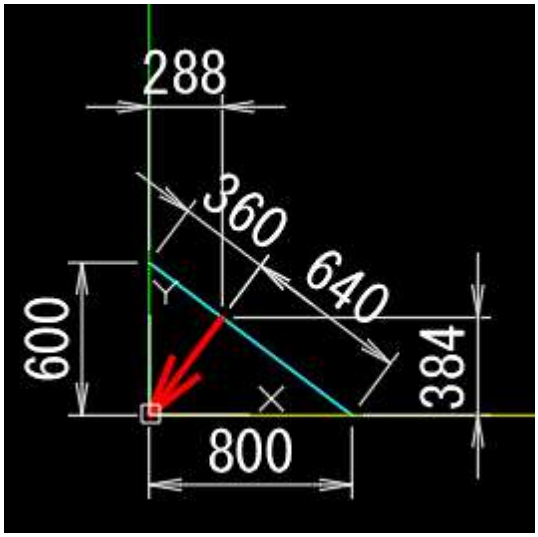
	ワールド座標系			グループコード		
	X	Y	Z	X	Y	Z
1 点目	368	324	200	100	200	なし
2 点目	688	84	200	500	200	なし
3 点目	688	84	400	500	400	なし

ワールド座標系の Z はグループコードの Y になりそうということがわかる

●ポリラインの法線方向とは
ポリラインが描かれている面から垂直な線をワールド座標系の(0,0,0)に向けて引く
このベクトルがポリラインの法線である

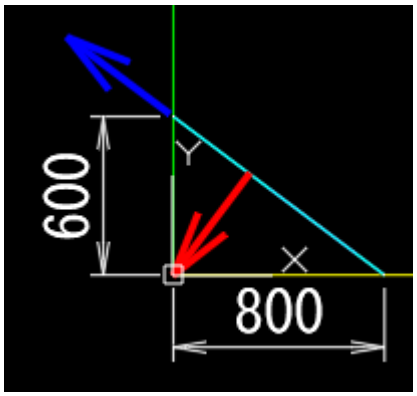


三角形の比例により



10：6 ＝ 600：360
10：8 ＝ 360：288
10：8 ＝ 800：640
10：6 ＝ 640：384

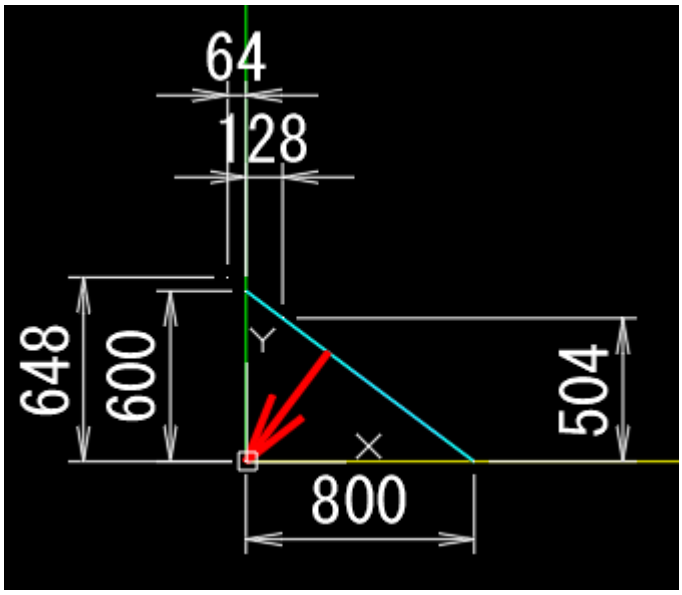
もとめるベクトルの値は（-288 , -384）となる
これを単位ベクトル（長さが 1 のベクトル）にする
10：8：6 ＝ 480：384：288　より
このベクトルの長さは　480　なので　480　で割る
-288/480 ＝ -0.6
-384/480 ＝ -0.8
よって法線ベクトルは（x , y , z）＝（-0.6 , -0.8 , 0）
回りくどいことをしたが結局は面の方向のベクトルを 90°回転させるだけ



面の単位ベクトル（青）は（x1 , y1 , z1）＝（-0.8 , 0.6 , 0）
面の単位法線ベクトル（赤）は（x2 , y2 , z2）＝（-0.6 , -0.8 , 0）
反時計回りに 90° 回転させると x2 = -y1　y2 = x1　になる

●ポリラインの高度とは
高度はワールド座標系(0,0,0)から面の法線ベクトル方向を正として面までの距離
ここでは原点から見て面はベクトルの方向（赤矢印の方向）と反対にあるので、値は-480 になる
面の法線ベクトル（-0.6 , -0.8 , 0）と原点から面内の点までのベクトル（x , y , z）の内積を計算する

z 座標はなんであっても 0 掛けられてなくなるので無視する



例
（x , y）=(128,504)のとき
(-0.6)×128 + (-0.8)×504 = -480
（x , y）=(-64,648)のとき
(-0.6)×(-64) + (-0.8)×648 = -480

なんでそうなるのか

（128 , 504） ＝ （288 , 384）＋（-8 , 6）×20

（-64 , 648） ＝ （288 , 384）＋（-8 , 6）×44

と書き換える

(-0.6)×128＋(-0.8)×504＝ (-0.6)×288＋(-0.8)×384
＋{(-0.6)×(-8)＋(-0.8)×6}×20

(-0.6)×(-64)＋(-0.8)×648＝ (-0.6)×288＋(-0.8)×384
＋{(-0.6)×(-8)＋(-0.8)×6}×44

{(-0.6)×(-8)＋(-0.8)×6} は常に 0 なので

(-0.6)×288＋(-0.8)×384 だけが残り、面内の点を使うのであれば内積は必ず-480 になる

「面内の点までのベクトルと面の単位法線ベクトルの内積は面の高度」

●面の原点とは

面の原点は面の法線ベクトルがワールド座標系の原点を通るように線を引いたとき

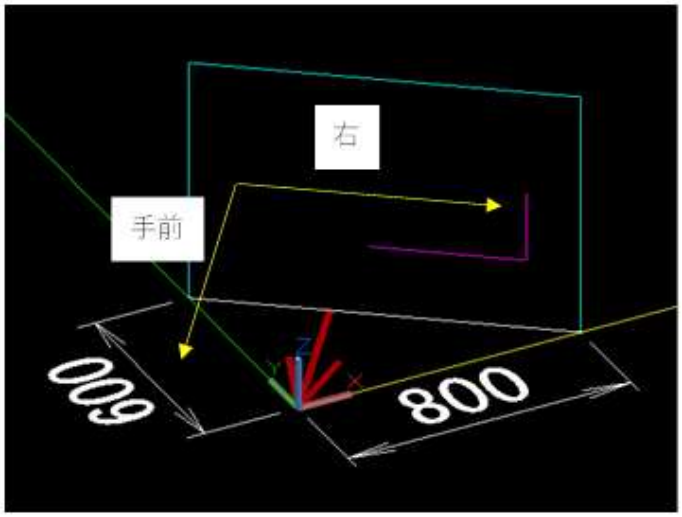
法線ベクトルが面と交差する点が面の原点になる

法線ベクトルに高度を掛けたものが面の原点のワールド座標系における座標になる

（x , y）=((-0.6)×（-480） , (-0.8)×（-480））＝ （288 , 384）

●ワールド座標系の座標値を面の座標系における座標値に変換する

面の座標系の X 方向は「面の法線ベクトルが手前に向かうように面を見て右向き」になる



面の X 方向単位ベクトル （0.8 , -0.6 , 0）

面の X ベクトル（0.8 , -0.6 , 0）と原点から面内の点までのベクトル（x , y , z）の内積を計算すると

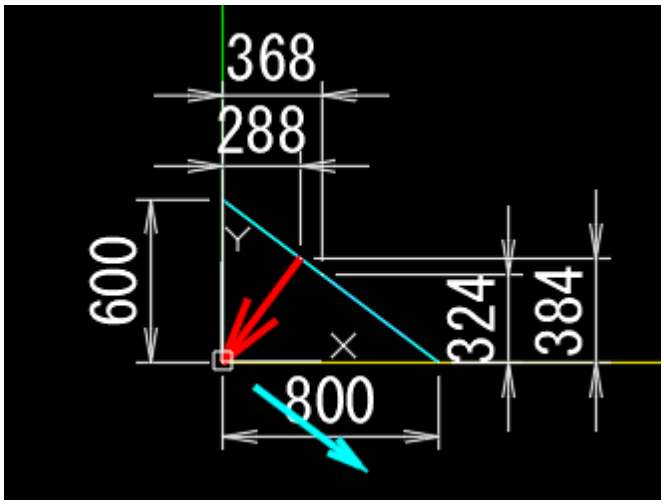
面の X 座標の値が求められる

1 点目 368×0.8＋324×（-0.6）＝100

2 点目 668×0.8＋84×（-0.6）＝500

ここでもベクトルを分解して考える

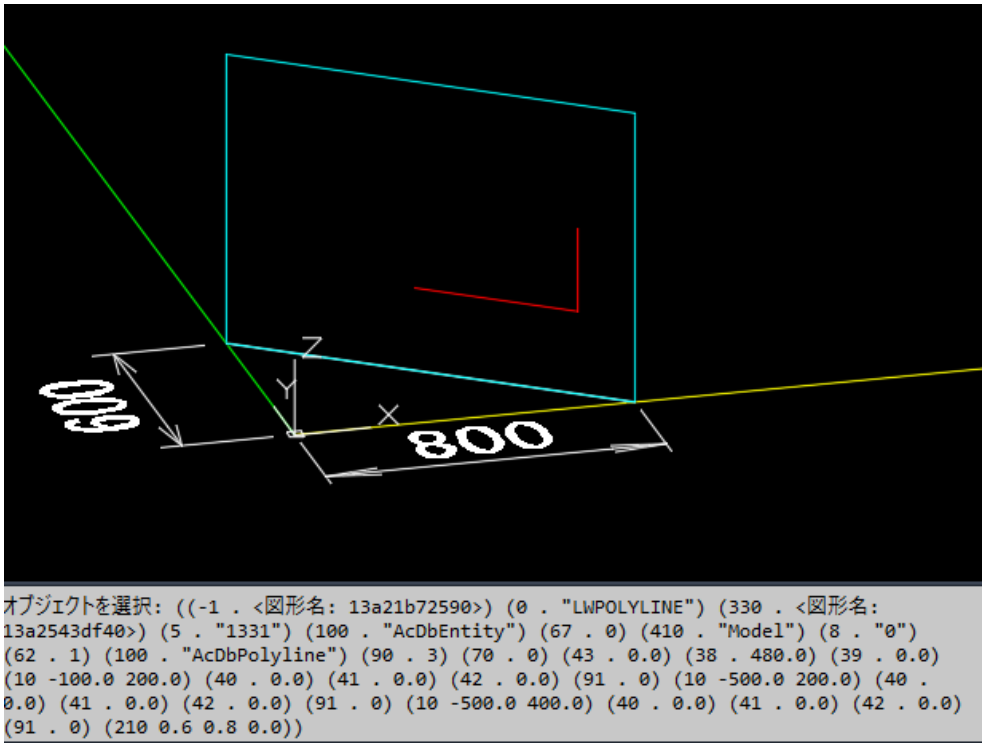
368×0.8 + 324×（-0.6）＝{288×0.8 + 384×（-0.6）}
+ {0.8×0.8 + （-0.6）×（-0.6）}×100
＝0 + 100



「直交するベクトルの内積は 0 になり、単位ベクトルと平行なベクトルの内積はその長さになる」

前述のとおり面のベクトルの Y 方向はワールド座標系の Z 方向
これで面の座標系における座標が求められる

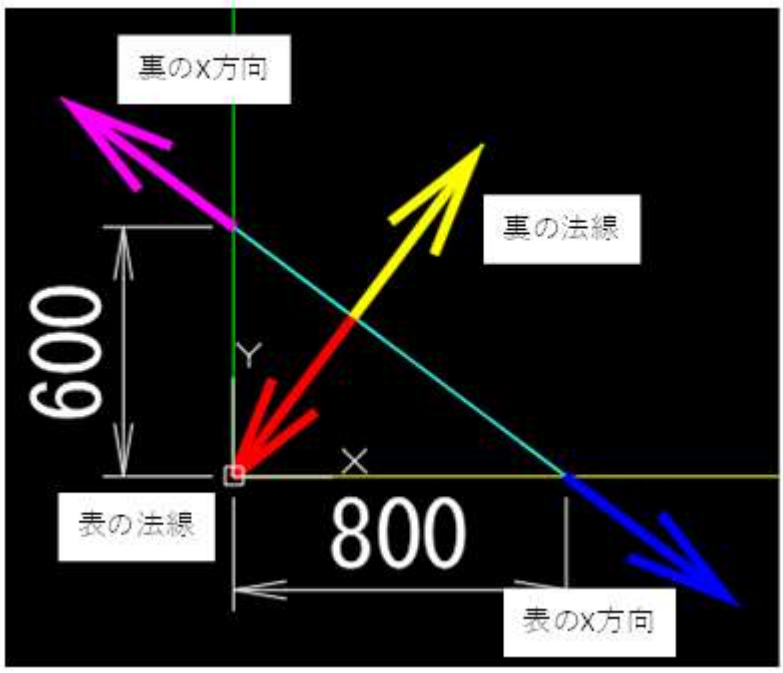
ポリラインの 3 次元配置計算その 1.1 裏
ポリラインを作成する面には裏側が存在する




```
オブジェクトを選択: ((-1 . <図形名: 13a21b72590>) (0 . "LWPOLYLINE") (330 . <図形名:
13a2543df40>) (5 . "1331") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0")
(62 . 1) (100 . "AcDbPolyline") (90 . 3) (70 . 0) (43 . 0.0) (38 . 480.0) (39 . 0.0)
(10 -100.0 200.0) (40 . 0.0) (41 . 0.0) (42 . 0.0) (91 . 0) (10 -500.0 200.0) (40 .
0.0) (41 . 0.0) (42 . 0.0) (91 . 0) (10 -500.0 400.0) (40 . 0.0) (41 . 0.0) (42 . 0.0)
(91 . 0) (210 0.6 0.8 0.0))
```

その 1.1 で紹介したポリラインと比較

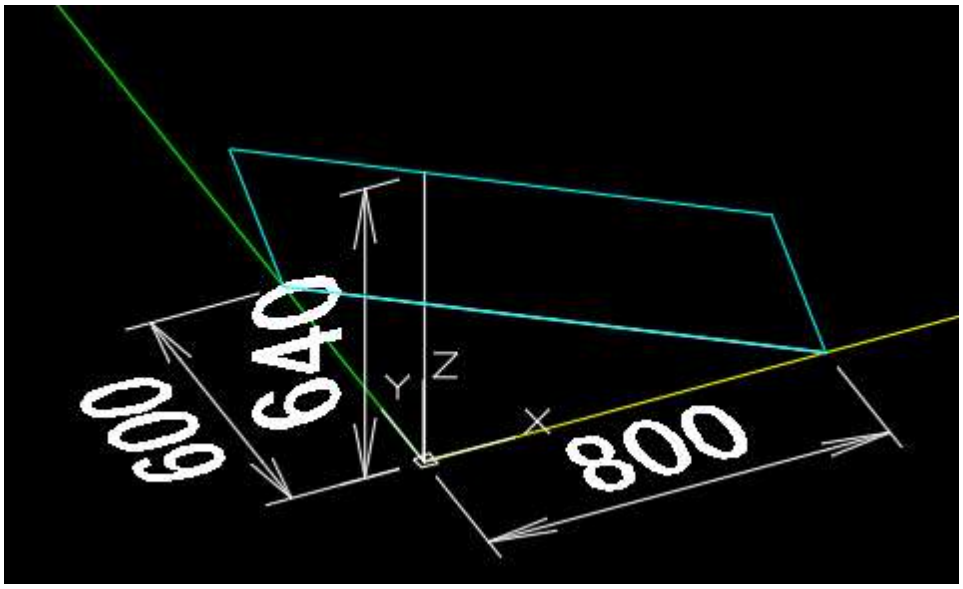
	その 1.1（高度 -480）			裏（高度 480）		
	X	Y	Z	X	Y	Z
1 点目	100	200		-100	200	
2 点目	500	200		-500	200	
3 点目	500	400		-500	400	
法線	-0.6	-0.8	0	0.6	0.8	0



Y 方向は変わらない

どちらで作っても見た目は同じ
たまに受領図面などでプロパティから高度の値を大きくすると逆に下がってしまう現象がみられるがこれが原因だと思われる
表裏はここで勝手に言っているだけで、原点に向かっているほうが表とかそういうのはない（むしろ高度の値が正になるほうが表っぽく感じる）

ポリラインの 3 次元配置計算その 2
XY 平面に直行しない面を考える

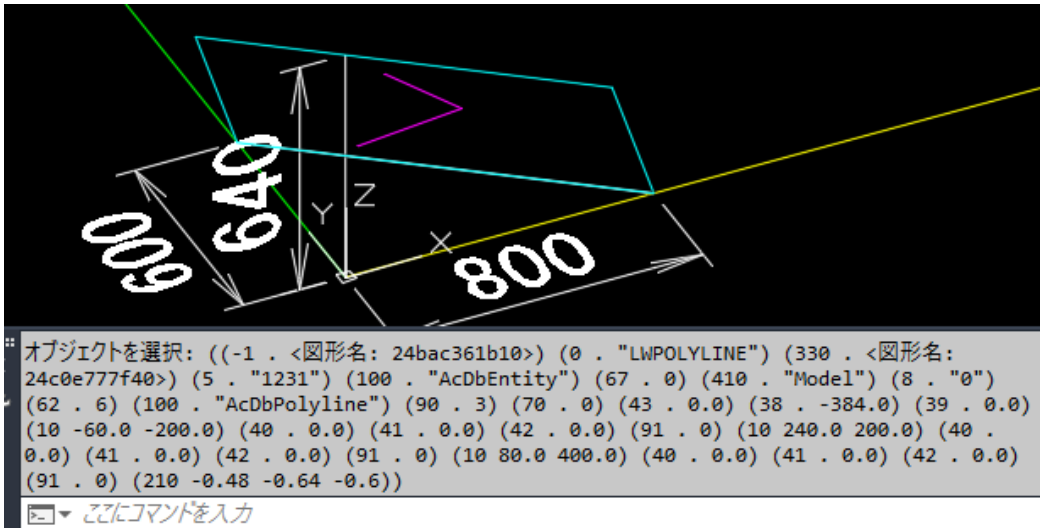


この面にポリラインを作る（グループコードが整数になるようにした）

1 点目　(X,Y,Z)＝（208.32 , 377.76 , 70.4）

2 点目　(X,Y,Z)＝（304.32 , 5.76 , 390.4）

3 点目　(X,Y,Z)＝（104.32 , 5.76 , 550.4）

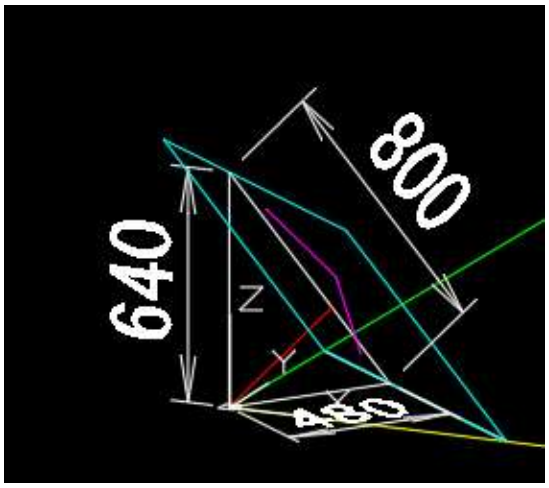


	ワールド座標系			グループコード		
	X	Y	Z	X	Y	Z
1 点目	208.32	377.76	70.4	-60	-200	なし
2 点目	304.32	5.76	390.4	240	200	なし
3 点目	104.32	5.76	550.4	80	400	なし

高度と法線のグループコード

(38 .　-384)
(210 .　-0.48　-0.64　-0.6)

面の法線の長さと交点の座標



計算は省略するが
法線（赤線）の長さは　384
面との交点の座標は（184.32 , 245.76 , 230.4）
になる

いつでも X,Y,Z 軸と面の交点がわかっているわけではないのでこんな方法では計算できない
⇒　外積を使う

面内の 3 点の座標のうち任意の 2 点を使ってベクトルにする
この 2 点はどれを選んでもいい　1 点目⇒2 点目、1 点目⇒3 点目　にする

	ワールド座標系			1 点目からのベクトル		
	X	Y	Z	X	Y	Z
1 点目	208.32	377.76	70.4			
2 点目	304.32	5.76	390.4	96	-372	320
3 点目	104.32	5.76	550.4	-104	-372	480

外積を計算する
x =　(-372) ×480 -　(-372) ×320 = -59520
y = 320×（-104）　- 480×96 = -79360
z = 96×（-372）　-　(-104) ×（-372）　= -74400

●法線ベクトルを求める

外積ベクトルの長さを求めて単位ベクトルにする

$\sqrt{59520^2 + 79360^2 + 74400^2} = 12400$

-59520/124000 = -0.48

-79360/124000 = -0.64

-74400/124000 = -0.6

法線ベクトル：(-0.48 , -0.64 , -0.6)

●高度を求める

法線ベクトルと原点から面内の点に向かうベクトルの内積を求める

（-0.48）×208.32 +　（-0.64）×377.76 +　（-0.6）×70.4 = -384

高度：-384

その 1 でやったときと同様にどの点を使ってもよい

（-0.48）×304.32 +　（-0.64）×5.76 +　（-0.6）×390.4 = -384

●面の原点を求める

法線ベクトルに高度をかけると面の原点のワールド座標系での座標値が求まる

xo =　（-0.48）×（-384）　= 184.32

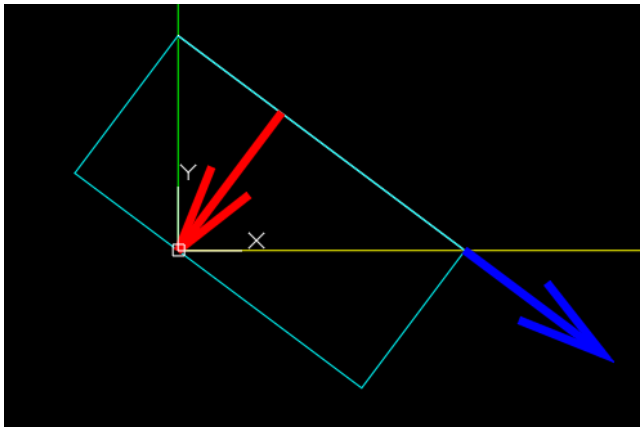
yo =　（-0.64）×（-384）　= 245.76

zo =　（-0.6）×（-384）　= 230.4

●面の X 方向を求める

面の X 方向は法線ベクトルが手前に向かってくるように面を見て右向きで

かつ、**XY 平面と平行**



結局 XY に直交する平面のときと同じである

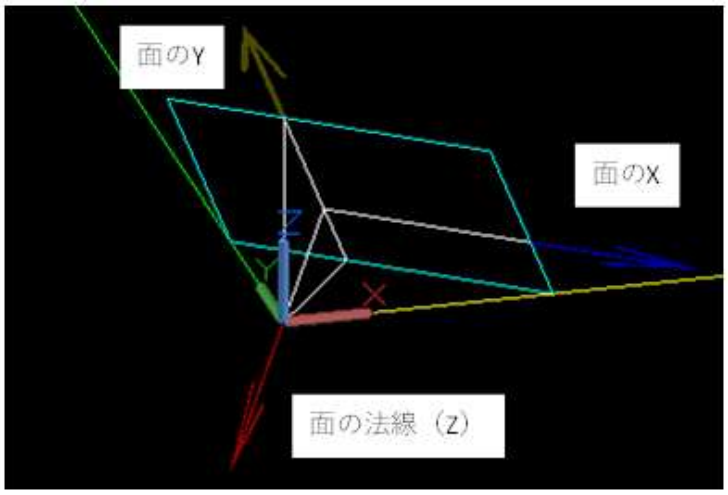
法線ベクトルの Z 成分を無視して単位ベクトルにする

$\sqrt{0.48^2 + 0.64^2} = 0.8$

-0.48/0.8 = -0.6

-0.64/0.8 = -0.8
X 方向ベクトル：(0.8 , -0.6 , 0)

●面の Y 方向ベクトルを求める



当然、面の Y 方向ベクトルは面の Z と面の X に直交するので、
Z と X の外積を求める
面の法線（Z）ベクトル　　：(-0.48 , -0.64 , -0.6)
面の X ベクトル　　：(0.8 , -0.6 , 0)
（-0.64）×0 - （-0.6）×（-0.6） = -0.36
（-0.6）×0.8 - 0×（-0.48） = -0.48
（-0.48）×（-0.6） - 0.8×（-0.64） = 0.8
面の Y ベクトル　　：(-0.36 , -0.48 , 0.8)
面の Y ベクトルはワールド座標系であらわして Z 成分が負になることはない
（経験上一度もない　もしかしたらあるのかもしれない）

●面内の点をワールド座標から面の座標に変換する
面の座標からそれぞれの点までのワールド座標系における位置関係を求める

	ワールド座標系			原点からのベクトル		
	X	Y	Z	X	Y	Z
原点	184.32	245.76	230.4			
1 点目	208.32	377.76	70.4	24	132	-160
2 点目	304.32	5.76	390.4	120	-240	160
3 点目	104.32	5.76	550.4	-80	-240	320

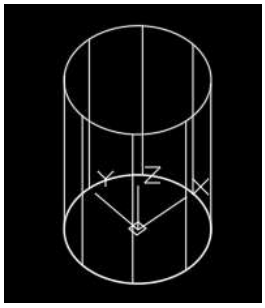
原点からのベクトルと面のＸベクトルの内積を計算すれば面のＸ座標になり、
原点からのベクトルと面のＹベクトルの内積を計算すれば面のＹ座標になる。

1点目
X＝24×0.8＋132×（-0.6）　＋　（-160）×0＝-60
Y＝24×（-0.36）　＋132×（-0.48）　＋　（-160）×0.8＝-200
2点目
X＝120×0.8＋　（-240）×（-0.6）　＋160×0＝240
Y＝120×（-0.36）　＋　（-240）×（-0.48）　＋160×0.8＝200
3点目
X＝　（-80）×0.8＋　（-240）×（-0.6）　＋320×0＝80
Y＝　（-80）×（-0.36）　＋　（-240）×（-0.48）　＋320×0.8＝400

＜とても上級編＞entmake でソリッドの作り方

ソリッドのグループコードはとても複雑で読みづらい。

中心(0,0,0)、半径 4 の円を作成し Z 方向に 12 押し出してグループコードを取得する。



((-1 . <図形名: 28202b66870>) (0 . "3DSOLID") (330 . <図形名: 279e81fc740>) (5 . "D27") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbModelerGeometry") (70 . 1) (1 . "mmloo mf m k ") (1 . "ni ^*+0;:,4 ^*+0¥¥^[mo ^LR mmjqjqoqijjlj QK o ") (1 . "n fqfffffffffffffffjkh:roh nqooooooooooooooooli:rno ") (1 . ">,27:>;:- {rn rn _nl mmjqjqoqijjlj |") (1 . "=0;& {rn rn {rn {m {rn {rn |") (1 . "3*2/ {rn rn {rn {rn {l {n |") (1 . ",7:33 {rn rn {rn {rn {k {rn {m |") (1 . "9><: {j rn {rn {i {h {l {rn {g 90-(>-; ,6183: |") (1 . "/:-,*=:1+r><>;L036;W6,+0-&r>++-6= {rn rn {rn {rn {k n n n o |") (1 . "9><: {f rn {rn {no {nn {l {rn {nm 90-(>-; ,6183: |") (1 . "300/ {rn rn {rn {nl {nk {k |") (1 . "<01:r,*-9><: {rn rn {rn o o o o o n rk ro o n V V o n k 90-(>-; V V V V |") (1 . "/:-,*=:1+r><>;L036;W6,+0-&r>++-6= {rn rn {rn {rn {i n n nooooooooo m |") (1 . "9><: {nj rn {rn {rn {ni {l {rn {nh -:):-,:; ,6183: |") (1 . "300/ {rn rn {rn {rn {ng {i |") (1 . "/3>1:r,*-9><: {rn rn {rn rnqgggjnmfijggmijfimj:rni o nm o o n n o o 90-(>-;@) V V V V |") (1 . "300/ {rn rn {rn {rn {nf {k |") (1 . "<0:;8: {rn rn {rn {nk {nk {ng {mo -:):-,:; {h o {rn |") (1 . "/:-,*=:1+r><>;L036;W6,+0-&r>++-6= {rn rn {rn {rn {no n n nooooooooo n |") (1 . "300/ {rn rn {rn {rn {mn {no |") (1 . "/3>1:r,*-9><: {rn rn {rn rnqgggjnmfijggmijfimj:rni o o o o n n o o 90-(>-;@) V V V V |") (1 . "<0:;8: {rn rn {rn {ng {ng {nk {mo 90-(>-; {nn o {rn |") (1 . "<0:;8: {rn rn {rn {nf {nf {mn {mm 90-(>-; {nl o {rn |") (1 . ":;8: {rn rn {rn {ml rlqnknjfmijljgfhflnni {ml lqnknjfmijljgfhflnni {nk {mk 90-(>-; _h *1410(1 |") (1 . "<0:;8: {rn rn {rn {mn {mn {nf {mm -:):-,:; {ni o {rn |") (1 . ":;8: {rn rn {rn {mj rlqnknjfmijljgfhflnni {mj lqnknjfmijljgfhflnni {nf {mi 90-(>-; _h *1410(1 |") (1 . "):-+:' {rn rn {rn {mo m {mh |") (1 . ":336/,:r<*-): {rn rn {rn o o nm o o n rk o o n V V |") (1 . "):-+:' {rn rn {rn {mm m {mg |") (1 . ":336/,:r<*-): {rn rn {rn o o o o o n rk ro o n V V |") (1 . "/061+ {rn rn {rn k kqgfgjghnfijgfknmgmf:rni nm |") (1 . "/061+ {rn rn {rn k kqgfgjghnfijgfknmgmf:rni o |") (290 . 1) (2 . "{541a93ca-0f99-6948-b1e6-8c0069961a7d}") (100 . "AcDb3dSolid") (350 . <図形名: 28202b668a0>))

AutoCAD でこの操作を行うと数値的なデータがなく、グループコード「1」がたくさんあり、その中に一見よくわからない文字列で暗号化されている。ARESS、IJCAD、BricsCAD などの互換 CAD ではなぜか暗号化されておらず、本家 AutoCAD では暗号化されている。互換 CAD では ZWCAD だけ暗号化を確認。

変換方法は次の通り。
（文字をアスキーコードに変換）→159 からアスキーコードの値を引く→それをアスキーコードとして文字に変換しなおす。LISP で書くと(chr(- 159(ascii "a")))　「a」に任意の文字を入力。
例えば「9><:」は「face」、「/061+」は「point」と読む。

長い文字列を一括変換する LISP
((lambda(tt / t1)(while(/= tt "")(setq t1(substr tt 1 1)tt(substr tt 2))(princ(chr(- 159(ascii t1))))(princ)))"9><:")

これを実行すると「face」と出るので「9><:」の部分を任意の文字列に書き換える。

小ネタ：変換の前後で似ている文字がある。どれか 1 対だけ覚えておけば変換用の数値 159 を覚えられなくてもコード変換して足し算すればすぐ思い出せる。似ているのは完全にたまたま。
アスキーコード表を見てみれば 32 までと 127 以降は文字列として表示されない制御文字が含まれていることがわかる。つまり、表記できる文字列が 33～126 なので 33+126=159 でありこれを逆にしている。こっちのほうが覚えやすいかも。

文字 1	文字 2	1 のアスキー	2 のアスキー
+（プラス）	t　（小文字）	43	116
0（ゼロ）	o　（オー小文字）	48	111
8	g（小文字）	56	103
く（小なり）	c（小文字）	60	99
A（大文字）	^（キャレット）	65	94

「(1. ○○)」は順番に番号が振られている（そのデータの番号はコードには書いてはいない。並んでいる順番になっている。一番最初が「-3」から始まっていると思われる。)。「{○○」は例えば「{mm」⇒「$22」のように変換でき、22 番目のデータを参照している。これを組み合わせて B-rep 法によってモデル作成されている。「face」を構成する「edge」として何番目を使用する～みたいな感じ。
これ以上の話は難しすぎるのでここではしないが、この通りに文字列を作成できれば entmake でソリッドを作成することができる。この方法について言及している WEB サイトや書籍はいまのところ見つかっていないのでほかにやっている人はいてもごくわずかかもしれない。
B-rep 法の歴史は古く 1980 年代からある考え方である。鋭意勉強中。

AutoLisp 参考サイト

エリプス(日本で唯一の LISP 教科書　昔書籍であったものを再編集して無料化)
https://www.ellipse.ne.jp/autolisp.html

図描きの備忘録（入門編）
https://ekakio.hatenablog.jp/archive/category/Lisp%20%E5%85%A5%E9%96%80%E7%B7%A8

Jagaimo LISP
https://jagaimolisp.com/

LEE MAC PROGRAMMING
http://www.lee-mac.com/programs.html
コード多数。全部英語。GIF で解説しているものもある。英語読めたところでプログラムの書き方を解説しているわけではないので何書いてるかわからない可能性あり。いろいろなギミックが使われており、読み解くのが大変だがいろいろ面白いことができそう。

Gizmo Labs Wiki（VisualLisp 関連が詳しく載っている）
https://wiki.gz-labs.net/index.php/%E3%83%A1%E3%82%A4%E3%83%B3%E3%83%9A%E3%83%BC%E3%82%B8

Autodesk グループコード一覧
https://help.autodesk.com/view/OARX/2020/JPN/?guid=GUID-3610039E-27D1-4E23-B6D3-7E60B22BB5BD

関数
http://docs.autodesk.com/ACD/2011/JPN/filesALR/WSfacf1429558a55de1a7524c1004e616f8b-5913.htm

Lisp　メモ
https://studio-jo.net/cad/lisp_memo.htm

LISP で楽になる
https://alraku.com/category/autolisp-introduction/

emacs ショートカット
https://www.creatology.jp/unix/emacs-sc.html