初級 AutoLISP 講座 2

エディタがねぇ．．．

エディタって知ってます？

知ってる人は、帰り支度をしていいですよ (^^; 知らない人の為にちょっとだけ説明しますので、聞いてって下さいね。

プログラムを作る第一歩

LISP プログラムを作るには、テキストエディタっていうソフトウェアが必要です。まぁ、無くても作れないわけじゃないんですけど、あったほうがはるかに楽できます。テキストエディタというのは、文章を編集するソフトウェアのこと。種類はなんでもいいですね。メモ帳や、MS-Word その他、市販されているテキストエディタ等、文章が編集できるソフトウェアなら大抵 LIPS プログラムを作ることができます。ただし、ワープロ系のソフトウェアの場合は、保存するときに「テキストのみ」といった方法で保存しないと、動かない LISP プログラムになっちゃいますけどね。

- ワープロ系のソフトウェアは、文章を保存するときに文字の大きさや色 などのデータも文章と一緒に保存してしまうんです。プログラムを 実行するときにじゃまな情報なので、「テキストのみ」にしないとい けないんですよ。

んじゃ、どんなのがいいの？

もし市販のテキストエディタを持っていないのなら、ウィンドウズに付属しているメモ帳がお薦めです。「やるぞっ」って意気込みだけで市販の高価なエディタを買ってしまってから「やっぱ、やーめた」ってなことにならないように、最初の投資は最小限にしましょうね。

ちょっと練習してみましょうか

まず最初に練習用のフォルダを作りましょう。名前は [Prototype] にしておきましょうか。

- フォルダの作り方がわからなかった人、「どこに作ればいいの？」とか 聞いた人。残念ですけど、もうちょっと勉強してから来て下さいね。基本的なことから手取 り足取りだと、ものすごい量になっちゃって、私が説明で きる限界を越えちゃいます.（あぁ、言い訳モード (^^;)

メモ帳を立ち上げて

メモ帳を立ち上げたら、次のように入力してから

```
(princ "hello")
```

HELLO.LSPという名前で先程作ったフォルダに保存しましょう。保存するときに拡張子を LSP にしないと HELLO.TXT とか HELLO.LSP.TXT とかおちゃめなファイル名になってしまうので、気をつけて下さいね。

- 拡張子っていうのは、エクスプローラとかでファイルを見たときにファイル名に xxxx.DWG なんて表示される "." から後ろの部分を拡張子って呼ぶんです。でも、エクスプローラの設定によっては見えないんですねぇ ... これが (^^; エクスプローラのプルダウンメニューにある【表示】【オプション】をクリックして、オプションダイアログボックスの【表示】タブ【登録されているファイルの拡張子は表示しない】という欄のチェックボックスをオフ（印が付いていない状態）にしてあげればちゃんと拡張子まで表示されるようになります。

セーブの確認

ちゃんとセーブされているか、エクスプローラで確認して下さいね。先程作ったフォルダに HELLO.LSP ができていれば成功です。

復習

今日は、復習無しです (^^)。もうちょっと詳しくセーブの方法とか説明したほうが良かったかな？でも、これからプログラムを作るんだもん、このくらいは自分でなんとかできるよね (^^) さて、久しぶりに続きを書き始めたのですが、いつまで続くかなぁ．．．(^^;;

見つからない？

使えてはじめて役に立つ

私たちが LISP プログラムを作るのは、標準に無い機能を AutoCAD に追加 したり、操作の自動化をしたりする為ですよね。せっかく作った LISP プ ログラムも、使えなければハードディスクのゴミになるだけです。今回は、 プログラムの実行方法と、実行できるプログラムの書き方のお話しです。

前回は、途中で帰っちゃった？(^^;;

第一回でやった HELLO.LSP を実行してみましょう。まず AutoCAD を立ち上げます。その後、コマンドラインから

```
(load "HELLO.LSP")
```

と、入力してみましょう。実行できましたか？ここで実行できている方は、既にパスの概念がわかってる人かな？

実行できないのが普通です

AutoCAD が LISP（や、その他）のファイルを探すのに、AutoCAD 検索パス という順路で探すんです。この道順にファイルが無ければ見つけられないんですねぇ。で、HELLO.LSP があるフォルダも、検索するときに寄り道して、 ついでに探してもらいましょう。

やり方は？

プルダウンメニューの【ツール】【基本設定】から（コマンドラインから 打つ場合は preferences です）基本設定ダイアログボックスを開いて、 【ファイル】タブの【サポートファイルの検索パス】の部分に HELLO.LSP があるフォルダを「追加」します。

- 必ず「追加」にしましょう。ここには、AutoCAD を実行するのに必要なパスが書いてあります。既にあるパスを、勝手に削除したり変更したりすると、AutoCAD が動かなくなっちゃいますよ。あと、「追加」ボタンを押した後に、その上にある「参照」ボタンを押した方が、入力は楽できますね。

※ AutoCAD 2002 の検索パス追加方法ははメニュー・カスタマイズの部分メニューを作るでも説明しています。

仕切り直し

検索パスを変更すると、AutoCAD を終了して、再度立ち上げるようにメッセージが表示されるので、その通りにします。さて、これで AutoCAD は、HELLO.LSP が探せるようになったはずです。もう一度 HELLO.LSP を実行してみましょう。

```
(load "HELLO.LSP")
hello"hello"
```

今度は、ちゃんと見つかったみたいですね (^^)v

なんか変 ... (;_;)

このままだと、この HELLO.LSP は、実行するたびにファイルをロードしな くてはいけません。お世辞にもかっこいいとはいえませんねぇ (^^;;

終業のベルが. . .

えぇ～。時間が来てしまいましたので、今回は、ここまでっ！ 関数の作り方は、次に回しま～す (^^;;

関数って作れるよ

まず関数を作りましょう

今回は、LISP の編集とロード、実行を繰り返しながら話を進めます。エディタの用意は、いいですか？

さーて、関数の作り方ですが。関数は、関数を作る関数で作ります。（なんだか禅寺の押し問答みたいですね (^^;;)
あまり深く考えると頭がパンクしそうなので、シンプルなものから始めましょう。前回作った HELLO.LSP をエディタで修正して、次のようにします。前回やったので、動作は大体わかると思いますが。これは、AutoCAD のコマンドラインに hello と表示する為のプログラムです。

```
(defun sample() (princ "hello"))
```

部分ごとに分けて説明すると、こんな感じになります。

(defun ... () ...)	関数を定義する defun 関数です。
sample	これが、関数の名前になります。
(princ "hello")	hello という文字列をコマンドラインに表示する為に (princ) という関数を使います。

これをロードして、実行してみましょう。まず AutoCAD を立ち上げて、コマンドラインから HELLO.LSP をロードします。

```
(load "HELLO.LSP")
```

次に、コマンドラインから次のように入力します。

```
(sample)
hello"hello"
```

一度ロードしてしまえば、何回でもコマンドラインから実行することができます。

```
(sample)
hello"hello"
```

さて、ここで問題です

もし、(load "HELLO.LSP") が、AutoCAD の実行時に自動的に実行されるようになっているとしたら、あなたは、(sample) という関数が AutoCAD に標準で用意されている関数なのか、それともユーザが定義した関数なのか、見分けがつきますか？
もちろん、今回は自分で (sample) という関数を作ったのでわかりますけど、もし、他の人が作ったのなら、おそらく見分けられないでしょう。つまり、標準で用意されている関数とほぼ同じものが作れてしまうんです。

その他、色々と．．．

リスプの関数には、引数を必要とするものもたくさんあります。今度は、HELLO.LSP が引数を受け取れるようにしてみましょう。HELLO.LSP をエディタで修正して、次のようにします。

```
(defun sample(string) (princ string))
```

部分ごとに分けて説明すると、こんな感じになります。

(defun ...)	関数を定義する defun 関数です。
sample	これが、関数の名前になります。
(string)	受け取った引数は、この string という変数に入れられます。
(princ string)	string を、コマンドラインに表示します。

これもロードして、実行してみましょう。コマンドラインから HELLO.LSP をロードします。

```
(load "HELLO.LSP")
```

●AutoCAD の中には、修正前の LISP がロードされています。再度ロードすることで修正後の LISP に置き換わるので、修正したら修正後の LISP をロードしなおさないといけません。

コマンドラインから、実行します。

```
(sample "My name")
My name"My name"
```

●ここで、前のように (sample) とすると、引数が必要なのに、指定されていません ... となってしまいますよ。

string は、どうなったの？

関数を実行した後に string という変数はどうなったのでしょうか。変数を確認する方法は !string でしたよね。

```
!string
nil
```

試しに、string に値を入れておいて見ましょう。コマンドラインから、変数 string に値を代入して、(sample) を実行してみます。

```
(setq string "Important")
"important"
(sample "Hi student")
Hi Student"Hi student"
```

もう一度、変数 string を確認してみましょう。

```
!string
"important"
```

さて、いつものお掃除を

```
(steq string nil)
nil
```

実行後も、string の中は変わりませんでした。これは、関数の中の string は、関数の外の string とは別の物として扱われるからです。さらに、string が生きていられる範囲はかぎられていて、この範囲を越えると、string は天へと召されてしまうんです (T_T)
この例でいくと (defun sample(string) ..... この範囲 .....) ですね。でもねぇ、こんなことをすると ... あ、これもエディタで HELLO.LSP を修正して、実行します。

```
(defun sample(string) (setq var string)(princ var))
```

これ、わかります？ string という変数で受け取った引数を、変数 var にコピーして、var を表示しています。これを実行すると、

```
(load "HELLO.LSP")
(sample "My name")
My name"My name"
```

結果はあまり変わっていないようですが。変数 var の値を確認すると

```
!var
"My name"
```

var という変数の中には、実行後も "My name" という値が入ったままになってしまいます。「var なんて変数使わなきゃいいんじゃないの？」という鋭いつっこみは .... やめてね (^^;; だって、うまく説明できないんだもん。
自分のプログラムではやらなくても、他の人が作ったプログラムで var という変数を使っていたとします。そして、var には、そのプログラムにとって非常に重要な情報が入っていたら .... おそらく、その人が作ったプログラムは正常に動かなくなってしうでしょう。なるべく他人に迷惑をかけない（たぶん自分にも）ためには var を、「局所変数」というものにしなくてはいけません。

関数の中でだけ使うのね

var を局所変数にするには、HELLO.LSP を、次のように編集します。

```
(defun sample(string / var) (setq var string)(princ var))
```

(string) を、(string / var) に書け変えただけなんですけど (^^;

- / の前後には、必ずスペースを入れましょう。そうでないと、string/ という変数なのか /var という変数なのか、はたまた string/var という変数なのかわかりませんよね。

先程実行したときに var には "My name" という値が入っています。もし、var がこのプログラムで変更されるとすれば、var の値は違うものになってしまいますよね。では、これを実行すると

```
(load "HELLO.LSP")
(sample "Your name")
Your name"Your name"
!var
"My name"
```

これで、一安心。他の人に迷惑をかける可能性は、ぐんと減りました (^^)v
最後に、いつものお掃除をしておきましょう。

```
(steq var nil)
nil
```

φ(..)メモメモ

補足説明

前回の説明は、大急ぎでやってしまったので少し補足しておきますね。前回でのポイントは、

1. 数を定義するには、(defun) という、関数を定義するための関数を使う。
2. 関数の中で使う変数は、特別な理由がない限り局所変数（ローカル変数とも言います）で定義する。

という、二点です。それと、(defun) 関数での変数の書き方は、例えば、sample という関数名で、関数を定義する場合。

1. 引数がいらない関数
   (defun sample () ……)
2. 引数が一つ必要な関数
   (defun sample (var) ….)
3. 引数が二つ必要な関数
   (defun sample (var1 var2) ….)
4. 一つの引数と、一つの局所変数が必要な関数
   (defun sample (var / local) ….)
5. 一つの引数と、二つの局所変数が必要な関数
   (defun sample (var / local1 local2) ….)
6. 局所変数が、一つだけ必要な関数
   (defun sample (/ local) ….)
7. 局所変数が二つ必要な関数
   (defun sample (/ local1 local2) ….)

といった具合になります。後は、同じ要領で必要な数だけ引数とローカル変数を書いていけば OK ですよ。少しは、補足になったかなぁ？ (^^;;

ヘビのようなプログラム

前回使ったプログラムくらいなら、一行で書いてしまってもあまり見苦しくありませんよね。（非難を浴びそうな発言かな (^^;）でも、これがもっと長くなったら .....

```
(defun pick_circle (/ ax bx dx)(setq ax nil)(while (not ax)(if (setq ax (entsel "¥n円 又は、円弧を指示: "))(setq bx (entget (car ax)) dx (cdr (assoc 0 bx))))(if (not (member dx '("CIRCLE" "ARC")))(progn (princ "¥nこれは 円, 円弧ではありません¥n")(setq ax nil)))))
```

ヘビみたいでしょ。

●プログラムの中身までは気にしなくていいですよ。「こんなもの読む気にならないよなぁ」と思ってもらえればいいんです。(^^)

これを、もうちょっとかっこ良くすると。

```
(defun pick_circle (/ ax bx dx)
  (setq ax nil)
  (while (not ax)
    (if (setq ax (entsel "¥n円 又は、円弧を指示: "))
      (setq bx (entget (car ax))
        dx (cdr (assoc 0 bx)))
    )
    (if (not (member dx '("CIRCLE" "ARC")))
      (progn
        (princ "¥nこれは 円, 円弧ではありません¥n")
        (setq ax nil)
      )
    )
  )
)
```

これなら、少しは「読んでみようかな」って気になるでしょ。プログラムとしては、どちらの書き方でもちゃんと動きますけど、ヘビみたいに書いちゃうと、後でプログラムを修正するのがすごく大変になっちゃいます。そこで、こうやって書き出し位置を少しずつズラして書いていくのが一般的で、こうやって書き出し位置をズラして書いていく方法を、「字下げを行う」とか、「インデントを付けて書く」と言います。字下げを付けて書くコツは、「もし～だったら～という処理をする」という部分と、「ここは一つの固まり」という部分の、始まりの括弧と終わりの括弧の字下げの位置を揃えることです。それぞれの関数のことをまだ説明していないので詳しい説明は省きますが、頭の隅っこにでも置いておいて下さい。

●字下げをするときに、どのくらいズラせば良いのかですが。推奨されているのは、スペース（空白文字）二つ分、右にズラしていくというのが正しいお作法のようです。タブを使って字下げするのはあまり推奨されていないのですが、私はよくやってます (^^;; 問題になりそうなら、エディタの「置換」を使ってタブ文字を二つのスペースに置き換えてもいいし、あまりこだわらなくてもいいみたいですよ。（ちょっと無責任かなぁ？）

これが本題

先程のプログラムがすらすら読める人でも、「ここはこういう動作をしています」と、プログラム中にコメントを書いておくと、とっても親切。コメントを書いておけば、他の人がプログラムを改造しようとしたときや、ここはどういう動作をしているんだろう？と思ってプログラムを見たときの助けになります。なにより一番のメリットは自分がわからなくならないってこと。出来上がってから、何年かたつと、自分が書いたプログラムでもすぐ思い出せる人は少ないんです。どんなに記憶力のいい人でもそうみたいで、「私は大丈夫！」なんて思ってると ... ダメですよ。

```
;; 円 又は、円弧が指示されるまでくるくるまわります
;;
(defun pick_circle (/ ax bx dx)
  (setq ax nil) ; 局所変数なので必要ありませんが、念のため
  (while (not ax) ; ax が nil の間は繰り返します
    (if (setq ax (entsel "¥n円 又は、円弧を指示: ")) ; 図形の指示
    (setq bx (entget (car ax)) ; エンティティリストを bx にセットします
      dx (cdr (assoc 0 bx))) ; 図形のタイプを dx にセットします
)
    (if (not (member dx '("CIRCLE" "ARC"))) ; もし、円か円弧でなければ
      (progn
        (princ "¥nこれは 円, 円弧ではありません¥n") ; メッセージを表示して
        (setq ax nil) ; もう一度実行する為に ax に nil をセットします
      )
    )
  )
)
```

だいぶ良くなったでしょ。コメントの書き方は、コメントの最初に ; （セミコロン）を書くだけです。セミコロンを書くと、行の後ろ（改行）までがコメントになります。

●他にも、複数行にまたがったり、行の途中にコメントを付ける方法として、;| これにはさまれた範囲がコメントになります |; なんて方法もありますよ。

コメントを付けるコツ

LISP ファイルの最初には、何をする為のプログラムなのか、どうやって使うのか、使う場合の注意点、作った人の名前などを書いておくといいですね。（自分がやっていないものだから、話していて赤面してしまいそう (^^;; ） 関数の最初には、何をする関数なのか、値を返す場合は返す値についてなどを書いておきましょう。コメントがいっぱいあると読みにくいとか、日本語で書いてあると見苦しいとか、いろいろ言う人がいますけど、一切気にする必要はありません。どかどかコメント付けましょうね (^^)v

次回は. . .

HELLO.LSP で使った (sample) という関数を AutoCAD のコマンドとして定義します。そこで、次回の予習をしておきましょう。HELLP.LSP を次のように修正して、保存しておいて下さいね。

```
;; 入力された文字に、[こんにちは]と[さん]を連結して表示します
;;
(defun C:SAMPLE (/ string)
  (setq string (getstring "¥nあなたの名前は？: ")) ; 文字を入力
  (princ (strcat "こんにちは、" string " さん")) ; 文字をつなげて表示
)
```

不審な文字が．．．

最初は、プログラムの説明を

```
;; 入力された文字に、[こんにちは]と[さん]を連結して表示します
;;
(defun C:SAMPLE (/ string)
  (setq string (getstring "¥nあなたの名前は？："))；文字を入力
  (princ (strcat "こんにちは、" string " さん"))；文字をつなげて表示
)
```

1. **(defun C:SAMPLE (/ string)** というのは、
   sample という名前の関数を定義します。 string という局所変数を一つだけ使うので、関数名の後が (/ string) となってますね。前回と違うのは C:SAMPLE となっているところですが、これは実行してみればすぐわかります (^^;;
2. **"¥nあなたの名前は？："** というのは、
   ここは、AutoCAD のコマンドラインに表示される部分です。¥n は、改行（リターン）を現す記号で、この部分を表示するときに、最初に改行してから次の新しい行に"あなたの ... " を表示するためにこの記号が書かれています。どうしてこんなことをするのかというと。例えば 他の処理が何かを既に表示していたりする場合に、その表示に続けて"あなたの ... "が表示されないようにする為です。実際にはこんな感じ
   続けて表示されると ...

   ```
   デルタ Y = -115.00000000, デルタ Z =0.00000000 あなたの名前は？：
   ```

   改行してから表示すると ...

   ```
   デルタ Y = -115.00000000, デルタ Z =0.00000000
   あなたの名前は？：
   ```

   今回のサンプルがこうなるというのではありませんが、通常こうやって改行を入 れることが多いので覚えておいて下さいね。それから、こういう入力を求めるメ ッセージを表示することを、「プロンプトを表示する」と言ったりします。良く使う 言葉なので、これも覚えておくといいですよ。
3. **(getstring "¥nあなたの名前は？：")** というのは、
   AutoCAD のコマンドラインに "¥nあなたの名前は？：" の部分を表示してから、ユーザの文字列入力を待つ部分です。
4. **(setq string (getstring "¥nあなたの名前は？："))** というのは、
   これは、もうおなじみ。入力された文字列を、string という変数に代入しています。
5. **(strcat "こんにちは、" string " さん")** というのは、
   "こんにちは、" という文字列の後に、入力された文字列を、さらにその後に "さん" という文字列をくっつくます。結果は "こんにちは xxx さん" という一つの文字列になりますよ。
6. **(princ (strcat "こんにちは、" string " さん"))** というのは、
   これは何回か出てきましたね。AutoCAD のコマンドラインに (strcat .....) の戻り値を表示します。

後は、インデントの付け方ですが

```
(defun
   ........
   ........
   ........
)
```

最初の括弧と最後の括弧が揃えて書いてあります。これで、この範囲が「一つの固まり」というのが視覚的にわかりやすくなりますよね。

これを実行すると

プログラムの動作がわかったところで、実行してみましょう。

```
(load "HELLO.LSP")
sample
あなたの名前は？：はるか
こんにちは、はるか さん"こんにちは、はるか さん"
```

今回は、関数の実行方法が前とは違います。いままでは (sample) としていたのに、今回は sample だけですよね。同じ関数の定義でも、関数名の最初に C: を付けると、関数名だけで実行することが出来るようになります。これを見て、なにか気がついたことはありませんか？そう、同じ関数の定義でも、関数名の最初に C: を付けると、まるで AutoCAD のコマンドのようになっちゃうんです。

●意図的に引数を必要とする関数を C: で定義するなら別ですけど、C: を付けて定義する場合は、引数を付けないで定義するのが一般的です。ただし、局所変数は OK

いよいよ

どうして関数を実行すると、二回も同じ値を表示するのか気になっていた人は多いんじゃないかと思います。ここで思い出して欲しいのが、「関数は、値を返す」です。関数を定義したんですから、当然この関数も値を返します。これを確かめる為に、sample が返す値を変数に代入してみましょう。

```
(setq var (c:sample))
あなたの名前は？：くま
こんにちは、くま さん"こんにちは、くま さん"
!var
"こんにちは、くま さん"
(setq var nil)
```

ポイントは、C:SAMPLE で定義した関数なので、LISP から呼び出すときは (c:sample) としてあげることと、関数を実行したときの二回目に表示される文字列と、変数 var の値を確認したときの文字列が、どちらも "（ダブルクォーテーションマーク）で囲まれているってことです。
うまく説明できないけど、関数を実行したときの最初に表示されている方は、(princ)関数で表示されたもの、二番目に表示されているのは関数が戻した値を表示しているものなんです。

美しくないなぁ

実際に使える LISP プログラムを作ろうとすると、こうやって意味不明な文字が出てきたのでは美しくありません。（もちろん、意図的に値を返さなければならない場合は別ですが）優雅に終了するためには、値を返さないようにしなくてはいけません。一般的なやり方はプログラムの最後に (princ)と書くだけです。でも、関数が値を返す所を先に説明しなくちゃいけませんね (^^;;

●nil を返せば .... と思った人はかなり近い！いい線いってます。でもね、nil っていうのも値の一種だから 「nil っていう値を返す」ことになっちゃうので、だめなんです。

ちょっとだけ戻り値の説明

関数から返る値は、一番最後に評価された値が返るんです。基本はこれ、

プログラム	戻り値
(defun test() nil)	nil
(defun test(/ var) (setq var 12.5))	12.5

先程の HELLO.LSP で定義した c:sample という関数だと

```
;; 入力された文字に、[こんにちは]と[さん]を連結して表示します
;;
(defun C:SAMPLE (/ string)
  (setq string (getstring "¥nあなたの名前は？: ")) ; 文字を入力
  (princ (strcat "こんにちは、" string " さん")) ; 文字をつなげて表示
)
```

この、(princ ...)という関数が値を返していることになります。 このくらいなら簡単なんですけど、だんだん複雑なプログラムになると、どこが値を返しているのかわからなくなっちゃうんですよね。でも、基本はこれです。

お静かに

静かに終了できるように、プログラムの最後に (princ) を追加します。

```
;; 入力された文字に、[こんにちは]と[さん]を連結して表示します
;;
(defun C:SAMPLE (/ string)
(setq string (getstring "¥nあなたの名前は？: ")) ; 文字を入力
  (princ (strcat "こんにちは、" string " さん")) ; 文字をつなげて表示
  (princ) ; 静かに終了します。
)
```

さっそく実行してみましょう

```
(load "HELLO.LSP")
sample
あなたの名前は？: いるか
こんにちは、いるか さん
```

これで、ここはすっきり。

さらに

リスプファイルをロードしたときに

```
(load "HELLO.LSP")
C:SAMPLE
```

というのが表示されますよね。これは、リスプファイルがロードされたときの (defun C:SAMPLE....) という(defun) 関数の戻り値が表示されているんです。これもイヤ！という場合は、

```
;; 入力された文字に、[こんにちは]と[さん]を連結して表示します
;;
(defun C:SAMPLE (/ string)
(setq string (getstring "¥nあなたの名前は？："))；文字を入力
(princ (strcat "こんにちは、" string " さん"))；文字をつなげて表示
(princ)；静かに終了します。
)
(princ)
```

と、ロードされたきに (princ) が一番最後に評価されるようにしておくと、ロードしたときに関数名が表示されないようにすることもできますよ。

不審な文字が登場しないまま．．．

不審な文字がまだ登場していませんが、このお話しは次回にしましょう。
では、おやすみなさ～い (_._)zzZZZ

さらに不審な文字が

これでもいいか．．．

前回作った HELLO.LSP は思った通りの動きをしているし、特に問題も無いようなので「これでいいや」と思ったら、 このプログラムはここで完成です。でも、こんなときはどうなるでしょう？このプログラムは、ユーザの入力をプログラム実行中に受け取っていますよね。入力中にユーザが（[ESC]キーを押すな どして）キャンセルしてしまった場合のことを考えてみましょう。

```
(load "HELLO.LSP")
sample
あなたの名前は？：にゃ*キャンセル*
エラー: 関数がキャンセルされました
(GETSTRING "¥nあなたの名前は？：")
(SETQ STRING (GETSTRING "¥nあなたの名前は？："))
(C:SAMPLE)
*キャンセル*
```

まさしく不審な文字が .... (^^;;
これは、プログラム実行途中に異常終了してしまったときの症状です。ユーザ入力に限らず他の原因で異常終了してしま った場合も同じような症状に見舞われます。例えば 5 ÷ 0 なんて計算をしたりすると（昔、やっちゃだめって言われま せんでした？）同じようになりますよ。

不審な物を観察

ぞろぞろと表示されていますが、なんの意味もなく表示されているわけではありません。ただ、読み方にコツがある んです。
これは、行を逆順に読んでいくのが正解

```
(load "HELLO.LSP")
sample
あなたの名前は？：にゃ*キャンセル*
エラー: 関数がキャンセルされました
(GETSTRING "¥nあなたの名前は？：")
(SETQ STRING (GETSTRING "¥nあなたの名前は？："))
(C:SAMPLE)
*キャンセル*
```

(C:SAMPLE)の中の
(SETQ STRING (GETSTRING "¥nあなたの名前は？："))の中の
(GETSTRING "¥nあなたの名前は？：")で、エラーがでてますよ。
エラーの内容は **エラー: 関数がキャンセルされました** です。

今回はプログラムが短いのでこんなものですが、長いプログラムがエラーになると、「xxx の中の」が最大 100 回出てきて、壊れたソフトクリームの機械みたいな感じになります (^O^)
プログラムを作っている最中は、修正する箇所の目星がついていいんですけど、実際に使うときは出ないようにしたいですよね。

どれがベストかはわかりません

結論から言うと (*error*) という関数を定義してあげればいいのですが。問題は、どこで定義するかです。 どれが最良の方法かは、私にもわかりません (^^;; ただ、メニューなどに付属しているエラー処理関数を（将来もこの関数名が変わらなくて、さらに必ず読み込まれて いるとして）そのまま使わせていただくのがよさそうな気がするのですが．．．あ、これじゃよくわかりませんよね (^^;; で、今回は、関数の中で定義してみます。

```
;; 入力された文字に、[こんにちは]と[さん]を連結して表示します
;;
(defun C:SAMPLE (/ string)
  (defun m_err(mes) ; m_err という関数を定義します [A]
    (setq *error* old_error) ; *error* を old_error で置き換えます
    (princ mes) ; エラーの原因を表示します
    (princ)
  )
  (setq old_error *error* *error* m_err) ; [B]
  (setq string (getstring "¥nあなたの名前は？: ")) ; 文字を入力
  (princ (strcat "こんにちは、" string " さん")) ; 文字をつなげて表示
  (setq *error* old_error) ; [C]
  (princ) ; 静かに終了します。
)
(princ)
```

1. sample が実行されると、まず最初に (defun m_err() ....) で (m_err) という関数が定義されます。 （実行されるわけじゃないですよ）この関数は、エラーが発生したときにしか実行されません。
2. その後に、old_error という変数に (*error*) という関数を一時的に待避しておきます。
   それから (*error*) という関数を (m_err) という関数に置き換えて
3. で、最後に 置き換えられていた (*error*) を、一時待避しておいた old_error に置き換えて元どおりにしておきます。

●本当は、エラーが発生すると (*error*) という関数が実行されるのですが、[B] から[C] までの間は (*error*) とすり替えられている (m_err) が実行されるという仕組みです。

こうやって関数の中で定義するのと、関数の外で定義するのでは何が違うのかというと、関数の中で定義しておけば「必ず」(m_err) という関数が使えますけど、関数の外で定義すると(m_err) という関数は「おそらくある だろう」になります。例えば (m_err) を関数の外で定義したプログラムをロードします。その後、sample を実行 する前に (setq m_err nil) とすると .... 「(m_err) はどこに行ったの？」になりますね。

関数の外で定義するのが悪いというのではありませんよ。運用の方法一つなんです。たとえば、「私の会社の LISP プログラムは、みんなが使えるユーティリティがそろっていて、その関数は必ず bx_ で始まるから、この関数を変更 してはいけない」なんていう取り決めをしておけば、問題にはなりません。先程話していた AutoCAD に付属しているメニューファイルなんかはこの典型です。よぉ～く見ると使えそうなのがいっぱい (^^)

一見よさそうなんだけど

このプログラムをロードして実行すると、ちゃんと動くし、実行後は m_err old_string どちらの局所変数も礼儀 正しくしていて問題なさそうです、でもキャンセルした場合はどうでしょう .... あれれ？

```
(load "HELLO.LSP")
sample
あなたの名前は？：にゃ*キャンセル*
関数がキャンセルされました*キャンセル*
!old_error
<Subr: #4021b52>
!m_err
((MES) (SETQ *ERROR* OLD_ERROR) (PRINC MES) (PRINC))
```

一難去ってまた一難 (T-T)　これを回避する良い方法は ..... 今のところ思いつきません。本格的に潔癖症の人は、 エラー処理関数（この場合は (m_err) ですね）の中ですべての局所変数に nil を代入するくらいでしょうか。 もしそうしたとしても、他の人が使っていた変数を書き潰してしまう危険性はありますねぇ
良い方法を知っている人は、手を上げて下さいね。

毎回ロードするのは面倒だから

日常的にこのプログラム使おうとすると、毎回ロードするのは面倒ですよね。
ですから、とってもお気楽な方法でこのプログラムを自動的にロードしちゃうようにしてみましょう。これは、ツールチップに登録してしまう方法です。ツールチップをカスタマイズするときにコマンドを書く所（「マクロ」って名前になってるのが多いですね）があるでしょ？そこに

```
C^C(if (= sample nil) (load "hello.lsp")); sample;
```

と言う風に書いておきます。sample は実行したい関数名。hello.lsp がロードしたい LISP ファイルの名前ですね。
こうしておけば、自分でロードしなくてもツールチップを叩けばいつでもこのプログラムが使えるようになります。（このままでは役に立たないプログラムですが (^^;)

- この意味は、とっても簡単。もし sample が nil だったら、hello.lsp をロードしてから sample っていうコマンドを実行しなさいってことです。

なんだかすっきりしませんが

この HELLO.LSP は、これで完成です。今日説明したことが大体理解できれば、もう初級じゃないですね (^^)

初級講座のおさらい

いきなりですが、これが題材です

```
;; 指示された直線の端点（どちらかは、わかりません）の位置を変更します
;;
(defun C:PICKS (/ ax bx var ent mod)
  ; 直線を指示してもらいます
  (setq var (entsel "¥n直線を指示して下さい："))
  ; もし、図形が指示されたのなら（var が nil でなければ）
  ; 次の (progn .... ) の部分を実行します
  ; 余談ですが、(if var と書いても同じように動作しますよ
  (if (/= var nil)
    (progn
      ; 指示された図形から情報を取得します
      (setq ent (entget (car var)))
      ; もし図形が直線ならば、次の (progn ... ) の部分を実行します
      (if (= (cdr (assoc 0 ent)) "LINE")
        (progn
          (setq bx (getpoint "¥n新しい点は? : "))
          ; もし、点が指示されたのなら（bx が nil でなければ）
          ; 次の (progn .... ) の部分を実行します
          (if (/= bx nil)
            (progn
              (setq ax (assoc 10 ent))
              (setq mod (subst (cons 10 bx) ax ent))
              (entmod mod)
            )
          )
        )
      )
    )
  )
  (princ)
)
```

もう、マニュアルと格闘すればこれが読めるハズ（うぅ～ん、どうでしょう。顔が見えないのがつらいところですねぇ）
「あ、エラー処理が無い！」なんてツッコミをひそかに期待してるんですけど (^^;;

ちょっと長いけど

プログラムが長くなりましたけど、よぉ～く見ると見覚えがありませんか？そう、最初にやった（その１のほうの）初級 LISP 講座の内容にちょっと手を加えただけです。最初からこれを見たら「 LISP って難しそう ... 」と思うかもしれませんが、もうそんなことないですよね。

初めての自習

このプログラムを実行すると、きっと変更したい部分が出てくると思います。私が思いつくのは

1. エラー処理を付ける。
2. このままでは直線のどちら側の端点が変更されるのかわからないので、ユーザが指示した点に近いほうの端点の位置を変更する。

くらいですけど。皆さんは、どうですか？
題材としては、少し難しいかもしれませんけど、練習にはなりますよ。それから、この自習の解答は出しません。いままでの説明とマニュアルがあれば、（自分の思い通りとはいかないまでも）改造して遊ぶくらいは十分できると思います。

残念ですが、最終回

どうも

書きたいことが多いわりには文章にならなくて、説明が抜けてしまった所もたくさんあります。（句読点や誤字脱字もかな (^^;;） あまり読み返したりしないで、どんどん掲載してしまうのが原因なんですけど。ま、恥をかくのは私ですから、いいですよね。

今のレベルは

LISP ファイルを作って、その中で関数（コマンド）が定義できて、ロードと実行ができれば立派なもの。後は、たくさんある関数をどうやって使っていくかだけです。前に言ったかもしれませんが、すべての関数を一気に覚えるのはかなり難しいことです。自分で作ったプログラムの中で関数を使っていくうちに、少しずつ覚えてしまうので、あせらなくても大丈夫。

最後に

長いことヘタッピな説明にお付き合いいただきまして、本当にありがとうございました。
「関数の定義」いろいろありましたけど、簡単なコマンドなら作れそうな気がしてきましたか？

それでは、またどこかでお会いしましょう (^^)/~~~~