coolda

だってクールだ newLISP

NEWLISP

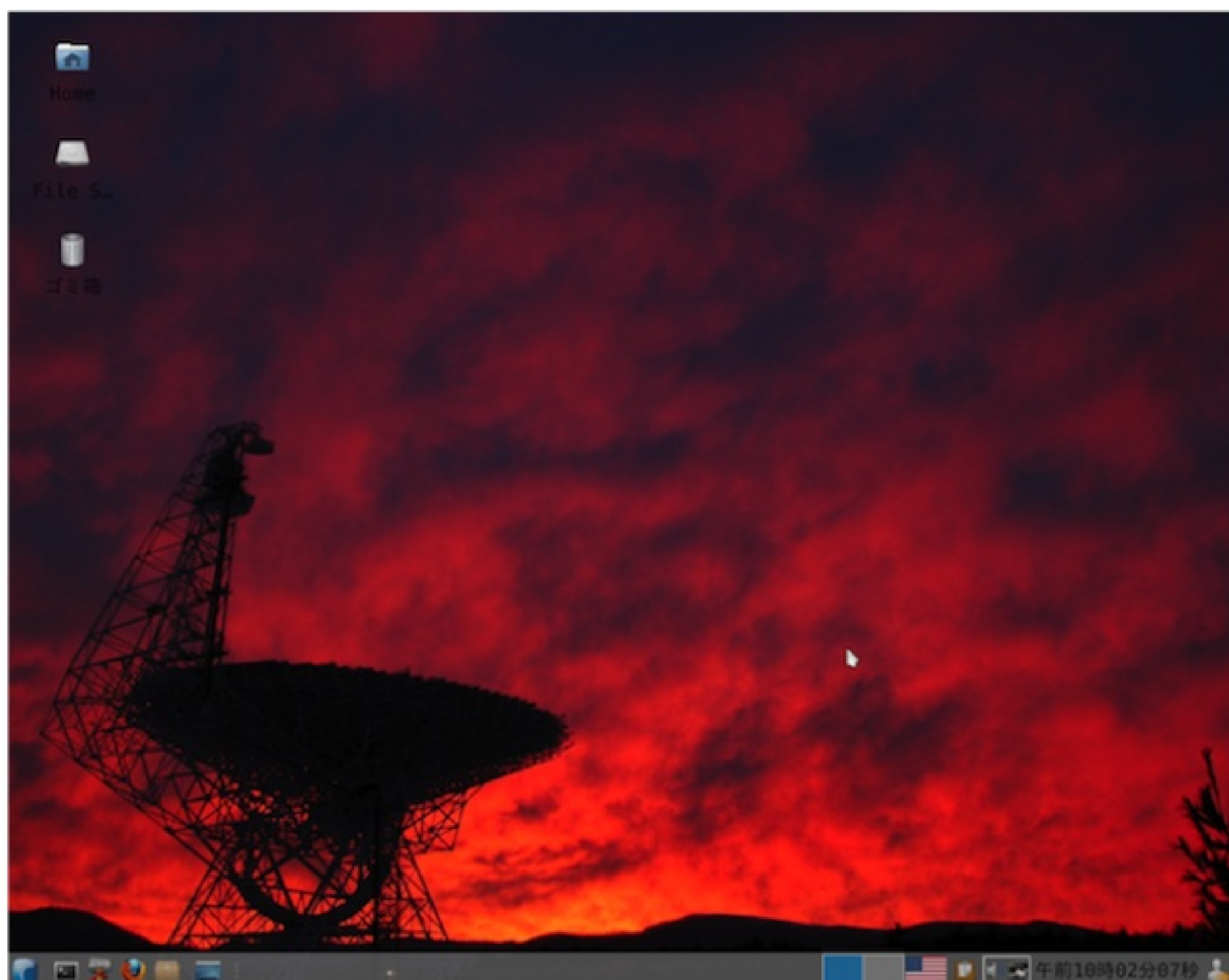


Image courtesy of NRAO/AUI

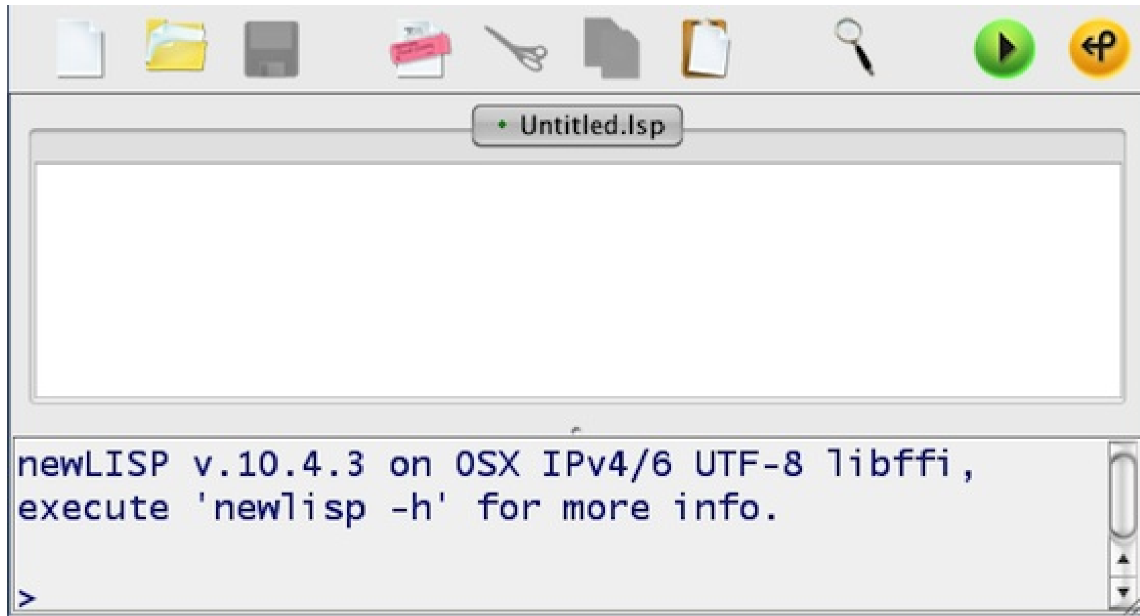
newLISP is a registered trademark of Lutz Mueller

ステージ１ **newLISP**をダウンロード。

newLISP is a registered trademark of Lutz Mueller.

リスプでは、関数を引数にして渡したりする操作は、
オリジナルのリスプが出来た、今から半世紀前に、すでに可能でした。
問題は、リスプは難しいのでは、という迷信の存在です。
ニューリスプは、リスプを出来る限り易しく使えるようにする、
ということに、真っ向から挑戦し、成功しています。
早速、ダウンロードして、立ち上げます。
http://www.newlisp.org/index.cgi?page=Downloads
にて、OS毎にパッケージが用意されていますので、
ご使用のOSに合わせたものをお選び下さい。
なお、GUIが、jreを使用しますので、
インストールが必要なケースがあると思います。この場合、
http://www.java.com/download.
にて、jreを入手して、先にインストールしておきます。
jre6、jre7、どちらでも動作します。
リナックスでは、この他、下記の２本が必要です。
インストールされてなければ、取り込んで下さい。
1. http://sourceware.org/libffi/
　libffi-3.0.11.tar.gz　を入手。
2. http://ftp.gnu.org/gnu/readline/
　readline-6.2.tar.gz　を入手します。
http://p.booklog.jp/book/34047
のステージ１１に説明がありますので、
ご参照下さい。

ステージ２　GUIを立ち上げる。



OSXでは、トンボのアイコンをクリックします。
リナックスでは、コマンドラインから、
$ newlisp-edit とします。
これで、上のようなフレームが出ます。
上の段は、実行可能なエディター、
下は、ランタイムのインタープリターです。
ディバッグ情報も自動的に出てきますので、
Cの時のように、プリントエフを沢山書いておく必要がありません。
上の欄に、何かプログラムを書いたら、ランボタンを押します。
すると、下の欄に、途中の情報が表示され、さらに、結果が表示されます。
上の欄に何か入力があると、フロッピーマークが青に変わります。
これで、コマンド+Sで、ナビゲート画面が出ますので、ファイルをセーブします。
ファイルを呼ぶときは、立ち上げたら、メニューから、
File -> Recent Files で、ファイルをセレクトします。
下の欄をきれいにするには、ランボタンの次にある、
更新ボタンを押してから、新たに実行します。

ステージ３　イフ式を使う。

今、温度を計測するセンサーからは、15という値が来ています。
この数字が、変化したか、それとも同じなのかを、
知らせてくれるプログラムを作りたいと思います。
新しいデータを読み込んだら、その前のデータとくらべます。
同じなら、"same"、変化したら、"not the same"　とプリントされます。
これには、イフ式を使います。

```
(define is_same
   (lambda (x y)
      (if
         (= x y) "same"
      "not the same")))
;;; call
(is_same 15 13)
(is_same 7 7)
```

イフの使い方は、簡単すぎて難しい、という話がありますので、
少し詳しくご説明します。
イフがある場合。
　　コンピューターよ、データを読み込め、という命令が出ます。
　　次に、評価せよ、という命令が発せられます。
　　評価せよ、は、ここでは、比べよ、という意味です。
　　"何を比べるのか"と聞いてくるので、先回りして、括弧の中に書いておきます。
　　先頭の'='で、'x'と'y'が同じかどうか調べよ、となります。
　　それで、もし、となります。
　　もし、比べてみて、結果が当たりなら、すぐ次に書いてあることを実行せよ、です。
　　ここでは、"same" を実行します。
　　ここに、さらに隠れていることがあって、
　　もし、結果が当たりで、すぐ次に書いてあることを実行したら、
　　さっさと脱出せよ、が隠れています。

脱出した場合、最後に実行されるのは。

脱出した場合、最後に実行されるのは、
"same" のディスプレイへのプリントになります。
この場合"not the same"の部分は、もう無視されます。
しかし、比較の結果がはずれの時は、
その下の行に移動し、そこに書いてあるものを実行せよ、が隠れています。
従って、はずれの時は、"not the same"　が実行されます。
この場合の実行は、"not the same"　が、
ディスプレイに自動的にプリントされます。
そしてコンピュータは、次の仕事のために待機します。
スキーム、ニューリスプなどでは、この１連の動きを、
REPL、レプルなどと云います。
Read　　データを読む。
Eval　　イーバル、エバリュエート、評価。
Print　　プリントアウト。
Loop　　次の仕事の為に待機。
隠れている部分は、ニューリスプを作った、ラッツミュラーさんが、
舞台裏で、コードを書いてくれているからで、
私たちは、とても少ないタイプ量で、まかなえてしまうという訳です。

ステージ４　ワイル式を使う。

whileは、英語で、〜をしている間は、という意味で、
イフ文の時と同じように、例えば、リミットを設定し、
そのリミットを超えない間は、ボディ部分に書いた命令を、
繰り返して実行します。
つまり、ループ状態を作る時に使います。
ループ状態を停止させて、脱出する命令を書き損じると、
ループ状態が止まらなくなります。
なので、よく注意して、少し慎重にプログラムを書きます。
イフ文の時と同じように、比較せよ、を使いますが、
今回は、算数で使う、'＜'　を使います。
算数では、これを、少なり、と云っていました。
英語では、'less than'、レスザンと云います。
１つずつ増やしていくインデックスも用意します。
インデックスは、ニューリスプが専用に用意した、$idx を使います。
使う時は、名前をこれと全く同じにして下さい。
$idxは繰り返しを検知して、自動的に値を１つ増加させます。
$idxが５になるまで$idxの値を次々に表示し、5を見て自分が5に
なっていたら終了します。なので、4までが表示される形となります。

```
(while (< $idx 5) (println $idx))
```

これを実行すると、

```
>
0
1
2
3
4
4
>
```

２回目の**4**は。

最後の4は、最後に実行されたものは、リターンされる、
というサービスで、出力されたものです。
それなら、これを使って、終了したら、"done"と
表示するようにして、数字は、改行しないタイプの関数、'print'
を使います、この時、数字と数字の間には、スペースを１文字分とるようにします。
変更を加えた、プログラムは、

```
(while (< $idx 5) (print $idx " ")
   "done")

(while (< $idx 5) (print $idx " ")    "done")
```

のように１行に書くこともできます。

```
;;; 実行
0 1 2 3 4 "done"
```

$idxはループのインデックスとして使っていますので、
安易に変えてしまうと、まずいので、
$idx自体の値は変えないで、$idxに100を加えたものを返してもらい、
今度は、それを表示するように変更します。
この操作は、プラス関数を使って行います。

```
(while (< $idx 5) (print (+ 100 $idx) " ")
   "done")

;;; 実行
100 101 102 103 104 "done"
```

プリント関数をまとめると、

1. プリントしたら改行するのは、(println)
2. 改行しないのは、(print)
   数字の場合、" "を使いスペースを入れないと、数字が繋がってしまいます。

ステージ５　ワイル式の中でイフを使う。

ワイル式を走らせている中で、状況に応じて出力を変える。

```
(while (< $idx 5)
   (if
      (= $idx 0) (println "started")
      (= $idx 1) (println "power: one")
      (= $idx 2) (println "power: two")
      (= $idx 3) (println "power: three")
      (= $idx 4) (println "power: max"))
   "done")
;;; 実行
>
started
power: one
power: two
power: three
power: max
"done"
>
```

ステージ６　ビット演算を試す。

アセンブリー言語で、レジスターを初期化したりする時、

ビット演算を使うことがあります。

ニューリスプでは、ビット演算をサポートしていますので、これを試してみます。

例えば、Cで、

'a'という名前の変数をゼロで初期化するには、a=0; とします。

ニューリスプでは、このような代入文は使いません。

かわりに、セット関数を使います。

(set 'a 0)

ここで、ビット演算を使って'a'の初期化はできるでしょうか。

ビット演算では、数字を、ビットが並んだもの、と捉えます。

ビットとは、実際の部品の名前で、メモリデバイスの中に装備されています。

ビットの状態は、1か0で表わします。

0 --> 0000

1 --> 0001

2 --> 0010

3 --> 0011

4 --> 0100

4と0100は、同じ値ですが、表現の仕方が異なるということです。

ニュ−リスプでのビット演算は、

演算子を名前にした、関数を使います。

最初はアンディングです。

アンディングには入力が２つ必要。

アンディングには入力データが２つ必要です。
(& exp1 exp2)
結果を入れる変数も用意しておきます。
入力される２本の値の、ビット１つ１つを比べ、
両方１であれば、結果は１になり、それ以外はゼロになります。
(set 'a 4)
(set 'b 0)
4は、ビットを見ると、0100 です。
これに'b'、ゼロをあてます。
ゼロのビット列は、0000 です。
a 0100
b 0000
c 0000
'a'と'b'を縦列で見て行くと、１１という組み合わせがないので、
結果は、すべてゼロ、即ち、cは、ゼロで初期化されます。
(set 'a 4)
(set 'b 0)
c
(setf c (& a b))

初期化なら、エクスオーアールも使える。
1. aとbのビットが異なる場合、結果は、１。
2. aとbのビットが同じなら、結果はゼロ。

2. を使って、同じ変数で、エクスオーアールすれば、全データはクリアされます。
エクスオーアールには、'^'を名前にした関数を使います。
(set 'a 0xa)　 ;;;; 1010 16進数a デシマル10
(setf a (^ a a))
これで、(setf a 0)　と同じ結果が得られます。
ニューリスプでは、フィールドに名前だけボンと置くことが出来ます。
先ほどの'c'は、nilで初期化され、登録されています。

ステージ７　フォー式を使う。

フォー式を使い、1から10を１つおきにプリントします。

```
(for (i 1 10 2) (print i " ")
   "done")
--> 1 3 5 7 9 "done"
```

1. 左括弧に'for'
2. 左括弧に'i' とか 'cnt'など、
      カウント用の変数の名前を入れる。これは、新規のものでかまわない。
3. 開始の数字を入れる。ゼロとか1など。
4. 終了の数字を入れる。10とか。
5. オプションでステップ、これを2にすれば１つおきになる。書かなければ１。
6. さらにオプションで、途中でブレークする条件を書ける。
7. 　右括弧
8. ボディー。
9. 終了した時、最後の部分が返るので、これを利用して"done"。
9. 右括弧。

ボディーを複数書くときは、

```
(for (i 1 10 2)
   (print i " ")
   (println "o'clock")
   "done")
-->
1 o'clock
3 o'clock
5 o'clock
7 o'clock
9 o'clock
"done"
```

ステージ８　リストを作る。

リストをつくるときは、list関数を使います。
(list 10 20 30)
リストが作られ、返ってきましたが、このままでは、
操作がしにくいので、名前をつけます。
ニューリスプでは、セットを使って名前をつけるとき、
クオト関数も一緒に使います。
クオト関数の役目は、式を評価しない、ということです。
リストは、データ構造なので、
これはデータですよ、ということを、はっきりしておく必要があります。
丁寧な形は、
(set (quote a) 100)
これを短くすると、
(set 'a 100)

(set (quote b) (list 10 20 30))
--> (10 20 30)
これを短くすると,
(set 'b '(10 20 30))
リストbの先頭にアクセスするには、
(b 0)
--> 10
関数(first)を使って、
(first b)
--> 10
ワイルを使って１つずつプリントしたり、そこで何かする場合は、

```
(while (< $idx 3)
   (print (b $idx) " ")
   "done")
```

ステージ９　リストを操作する。

リストを'lis'という名前で作ります。

(set 'lis (list 10 20 30 40))

--> (10 20 30 40)

これを'lis2'の中に組み込みます。

(set 'lis2 (list "hello" 100 lis))

--> ("hello" 100 (10 20 30 40))

リストのエレメントを返す時は、Cの配列のように、

ゼロから始まるインデックスで指定できます。

(lis2 1)

--> 100

(lis2 2 1)

--> 20

リストからデータを削除するときは、(pop)が使えます。

インデックスの指定をしなければ、先頭が消えます。

(pop lis2)

--> "hello"

ポップする時、-1を指定すると、尻尾のデータ(last)がポップします。

(pop lis2　-1)

lis2

--> (100)

(push)を使ってデータを追加すれば、ディストラクティブな操作になります。

(push "bye" lis2 -1)

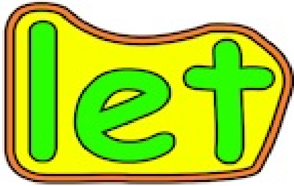
--> (100 "bye")

ディストラクティブの場合、'lis2'自体が変更されます。

ディストラクティブでない場合は、コピーしたデータのみに

適用されますので、本体は、変更されません。

ステージ１０　レットでローカル変数を作る。



(set 'a 100)

にて、作成された変数はグローバル変数です。

グローバル変数は、銀座４丁目の和光の時計のように、

誰もが見ることができ、そして利用できます。

グローバル変数はさらに、誰もが値の変更をすることが出来ます。

変更には、setfが使えます。

(setf a 120)

関数の中では、その関数の中のみで使えるローカル変数を作ることが出来ます。

これには、レット関数を使います。

```
;;;; let1.lsp
(define foo
   (lambda ()
      (let (a 100 b 200)
         (print "a: " a " b: " b " " )
   "done")))

(foo)
--> a: 100 b: 200 "done"
```

ステージ１１　ラムダ式を使う。

ラムダ式は、関数を作成する時に使います。

```
((lambda (x) (* x x)) 5)
--> 25
```

1. ラムダの後の括弧の中の'x'は、引数で、ボディーの中で使えます。
2. 次がボディーで、引数を乗算して結果を返します。
3. 最後の即値５は、これで、'x'を置き換えるのに使います。なので、このまま実行できます。
4. このまま実行できるので、関数に名前はつけません。

即値を渡さないと、関数はできあがりますが、実行はできません。

```
(lambda (x) (* x x))
```

そこで、関数に名前をつけます。

```
(define foo
   (lambda (x) (* x x)))
```

これで、呼びだせます。

```
(foo 7)
--> 49
```

ラムダのスペルが長いと思うときは、'fn'
を代わりに使えます。

```
(define bar
   (fn (x) (* x x)))

(bar 8)
--> 64
```

ラムダをいっそ、省略してしまうこともできます。

```
(define (baz x) (* x x))
(baz 9)
--> 81
```

ステージ１２　リカーションでループする。

リストの中身を順番に見ていく場合、
リカーション、再帰呼び出しを使うと、便利です。
Scheme、ニューリスプなどでは、再帰を使うための準備がすっかり整っています。
最初に、読み込みに使うリストを作ります。
(set 'lat '(ice cream lemon soda))
このリストを操作するのには、イフ式を使いますが、
ここでは、伝統のコンド式を利用します。
コンドは、コンディションを短くしたものです。
リスト全体のうち、先頭部分のみを管理する関数(first) と
残り全部を管理する(rest)も使います。
再帰のスタイルは、'first'部分を操作したら、
自分を呼び出す時、'rest'を引数にします。
すると、次は、'rest'の先頭が'first'になります。
リストがゼロになったら脱出する部分は重要なので、コンドの先頭に書きます。
プログラム全体では、

```
;;; recur.lsp
(set 'lat '(ice cream lemon soda))
(define recur
   (lambda (lat)
      (cond
         (null? lat) 0
         (print (first lat) " ")))
   (recur (rest lat)))

;;; 呼び出し
(recur lat)
--> (ice cream lemon soda)
```

関数の中からの呼び出しでは、先頭部分の'first'
を取り外し、recurを呼ぶ時に、'rest'を渡している所に要注意です。
何気なく、'rest'を渡していますが、'rest'は関数だ、という点にも注目してください。

ステージ１３　ラップアップ。

http://www.newlisp.org/index.cgi?Documentation　には、
Johuさんの作成による、対訳形式の日本語マニュアルも用意されています。
関数は、400程度ありますので、どんなものがあるのか、
マニュアルで、見ておくことおすすめです。
http://en.wikibooks.org/wiki/Introduction_to_newLISP
ウィキブックは英語版ですが、とても分かりやすいので、
全然負担になりません。併せてご利用下さい。