An Introduction to Programming in Emacs Lisp

The homepage for GNU Emacs is at http://www.gnu.org/software/emacs/.
To view this manual in other formats, click here.

This is an Introduction to Programming in Emacs Lisp, for people who are not programmers.

Distributed with Emacs version 25.2.

Printed copies available from http://shop.fsf.org/. Published by:

```
GNU Press,                          http://www.fsf.org/licensing/gnu-press/
a division of the                   email: sales@fsf.org
Free Software Foundation, Inc.      Tel: +1 (617) 542-5942
51 Franklin Street, Fifth Floor     Fax: +1 (617) 542-2652
Boston, MA 02110-1301 USA
```

ISBN 1-882114-43-4

This master menu first lists each chapter and index; then it lists every node in every chapter.

Preface	What to look for.
List Processing	What is Lisp?
Practicing Evaluation	Running several programs.
Writing Defuns	How to write function definitions.
Buffer Walk Through	Exploring a few buffer-related functions.
More Complex	A few, even more complex functions.
Narrowing & Widening	Restricting your and Emacs attention to a region.
car cdr & cons	Fundamental functions in Lisp.
Cutting & Storing Text	Removing text and saving it.
List Implementation	How lists are implemented in the computer.
Yanking	Pasting stored text.

Loops & Recursion	How to repeat a process.
Regexp Search	Regular expression searches.
Counting Words	A review of repetition and regexps.
Words in a defun	Counting words in a `defun`.
Readying a Graph	A prototype graph printing function.
Emacs Initialization	How to write a `.emacs` file.
Debugging	How to run the Emacs Lisp debuggers.
Conclusion	Now you have the basics.
the-the	An appendix: how to find reduplicated words.
Kill Ring	An appendix: how the kill ring works.
Full Graph	How to create a graph with labeled axes.
Free Software and Free Manuals	
GNU Free Documentation License	
Index	
About the Author	

Detailed Node Listing

Preface

Why	Why learn Emacs Lisp?
On Reading this Text	Read, gain familiarity, pick up habits....
Who You Are	For whom this is written.
Lisp History	
Note for Novices	You can read this as a novice.
Thank You	

List Processing

Lisp Lists	What are lists?
Run a Program	Any list in Lisp is a program ready to run.
Making Errors	Generating an error message.
Names & Definitions	Names of symbols and function definitions.
Lisp Interpreter	What the Lisp interpreter does.
Evaluation	Running a program.
Variables	Returning a value from a variable.
Arguments	Passing information to a function.
set & setq	Setting the value of a variable.
Summary	The major points.
Error Message Exercises	

Lisp Lists

Numbers Lists	List have numbers, other lists, in them.

Lisp Atoms — Elemental entities.
Whitespace in Lists — Formatting lists to be readable.
Typing Lists — How GNU Emacs helps you type lists.

The Lisp Interpreter

Complications — Variables, Special forms, Lists within.
Byte Compiling — Specially processing code for speed.

Evaluation

How the Interpreter Acts — Returns and Side Effects...
Evaluating Inner Lists — Lists within lists...

Variables

fill-column Example
Void Function — The error message for a symbol without a function.
Void Variable — The error message for a symbol without a value.

Arguments

Data types — Types of data passed to a function.
Args as Variable or List — An argument can be the value of a variable or list.
Variable Number of Arguments — Some functions may take a variable number of arguments.
Wrong Type of Argument — Passing an argument of the wrong type to a function.
message — A useful function for sending messages.

Setting the Value of a Variable

Using set — Setting values.
Using setq — Setting a quoted value.
Counting — Using `setq` to count.

Practicing Evaluation

How to Evaluate — Typing editing commands or `C-x C-e` causes evaluation.
Buffer Names — Buffers and files are different.
Getting Buffers — Getting a buffer itself, not merely its name.
Switching Buffers — How to change to another buffer.
Buffer Size & Locations — Where point is located and the size of the buffer.
Evaluation Exercise

How To Write Function Definitions

Primitive Functions
defun — The `defun` macro.
Install — Install a function definition.
Interactive — Making a function interactive.
Interactive Options — Different options for `interactive`.
Permanent Installation — Installing code permanently.
let — Creating and initializing local variables.
if — What if?

else — If--then--else expressions.
Truth & Falsehood — What Lisp considers false and true.
save-excursion — Keeping track of point and buffer.
Review
defun Exercises

Install a Function Definition

Effect of installation
Change a defun — How to change a function definition.

Make a Function Interactive

Interactive multiply-by-seven — An overview.
multiply-by-seven in detail — The interactive version.

`let`

Prevent confusion
Parts of let Expression
Sample let Expression
Uninitialized let Variables

The `if` Special Form

if in more detail
type-of-animal in detail — An example of an `if` expression.

Truth and Falsehood in Emacs Lisp

nil explained — `nil` has two meanings.

`save-excursion`

Point and mark — A review of various locations.
Template for save-excursion

A Few Buffer-Related Functions

Finding More — How to find more information.
simplified-beginning-of-buffer — Shows `goto-char`, `point-min`, and `push-mark`.
mark-whole-buffer — Almost the same as `beginning-of-buffer`.
append-to-buffer — Uses `save-excursion` and `insert-buffer-substring`.
Buffer Related Review — Review.
Buffer Exercises

The Definition of `mark-whole-buffer`

mark-whole-buffer overview
Body of mark-whole-buffer — Only three lines of code.

The Definition of `append-to-buffer`

append-to-buffer overview
append interactive — A two part interactive expression.
append-to-buffer body — Incorporates a `let` expression.
append save-excursion — How the `save-excursion` works.

A Few More Complex Functions

copy-to-buffer With `set-buffer`, `get-buffer-create`.
insert-buffer Read-only, and with `or`.
beginning-of-buffer Shows `goto-char`, `point-min`, and `push-mark`.
Second Buffer Related Review
optional Exercise

The Definition of `insert-buffer`

insert-buffer code
insert-buffer interactive When you can read, but not write.
insert-buffer body The body has an `or` and a `let`.
if & or Using an `if` instead of an `or`.
Insert or How the `or` expression works.
Insert let Two `save-excursion` expressions.
New insert-buffer

The Interactive Expression in `insert-buffer`

Read-only buffer When a buffer cannot be modified.
b for interactive An existing buffer or else its name.

Complete Definition of `beginning-of-buffer`

Optional Arguments
beginning-of-buffer opt arg Example with optional argument.
beginning-of-buffer complete

`beginning-of-buffer` with an Argument

Disentangle beginning-of-buffer
Large buffer case
Small buffer case

Narrowing and Widening

Narrowing advantages The advantages of narrowing
save-restriction The `save-restriction` special form.
what-line The number of the line that point is on.
narrow Exercise

`car`, `cdr`, `cons`: Fundamental Functions

Strange Names An historical aside: why the strange names?
car & cdr Functions for extracting part of a list.
cons Constructing a list.
nthcdr Calling `cdr` repeatedly.
nth
setcar Changing the first element of a list.
setcdr Changing the rest of a list.
cons Exercise

`cons`

Build a list
length — How to find the length of a list.

Cutting and Storing Text

Storing Text — Text is stored in a list.
zap-to-char — Cutting out text up to a character.
kill-region — Cutting text out of a region.
copy-region-as-kill — A definition for copying text.
Digression into C — Minor note on C programming language macros.
defvar — How to give a variable an initial value.
cons & search-fwd Review
search Exercises

`zap-to-char`

Complete zap-to-char — The complete implementation.
zap-to-char interactive — A three part interactive expression.
zap-to-char body — A short overview.
search-forward — How to search for a string.
progn — The `progn` special form.
Summing up zap-to-char — Using `point` and `search-forward`.

`kill-region`

Complete kill-region — The function definition.
condition-case — Dealing with a problem.
Lisp macro

`copy-region-as-kill`

Complete copy-region-as-kill — The complete function definition.
copy-region-as-kill body — The body of `copy-region-as-kill`.

The Body of `copy-region-as-kill`

last-command & this-command
kill-append function
kill-new function

Initializing a Variable with `defvar`

See variable current value
defvar and asterisk

How Lists are Implemented

Lists diagrammed
Symbols as Chest — Exploring a powerful metaphor.
List Exercise

Yanking Text Back

Kill Ring Overview
kill-ring-yank-pointer — The kill ring is a list.
yank nthcdr Exercises — The `kill-ring-yank-pointer` variable.

Loops and Recursion

while	Causing a stretch of code to repeat.
dolist dotimes	
Recursion	Causing a function to call itself.
Looping exercise	

`while`

Looping with while	Repeat so long as test returns true.
Loop Example	A `while` loop that uses a list.
print-elements-of-list	Uses `while`, `car`, `cdr`.
Incrementing Loop	A loop with an incrementing counter.
Incrementing Loop Details	
Decrementing Loop	A loop with a decrementing counter.

Details of an Incrementing Loop

Incrementing Example	Counting pebbles in a triangle.
Inc Example parts	The parts of the function definition.
Inc Example altogether	Putting the function definition together.

Loop with a Decrementing Counter

Decrementing Example	More pebbles on the beach.
Dec Example parts	The parts of the function definition.
Dec Example altogether	Putting the function definition together.

Save your time: `dolist` and `dotimes`

dolist

dotimes

Recursion

Building Robots	Same model, different serial number ...
Recursive Definition Parts	Walk until you stop ...
Recursion with list	Using a list as the test whether to recurse.
Recursive triangle function	
Recursion with cond	
Recursive Patterns	Often used templates.
No Deferment	Don't store up work ...
No deferment solution	

Recursion in Place of a Counter

Recursive Example arg of 1 or 2

Recursive Example arg of 3 or 4

Recursive Patterns

Every

Accumulate

Keep

Regular Expression Searches

sentence-end The regular expression for `sentence-end`.
re-search-forward Very similar to `search-forward`.
forward-sentence A straightforward example of regexp search.
forward-paragraph A somewhat complex example.
Regexp Review
re-search Exercises

`forward-sentence`

Complete forward-sentence
fwd-sentence while loops Two `while` loops.
fwd-sentence re-search A regular expression search.

`forward-paragraph`: a Goldmine of Functions

forward-paragraph in brief Key parts of the function definition.
fwd-para let The `let*` expression.
fwd-para while The forward motion `while` loop.

Counting: Repetition and Regexps

Why Count Words
count-words-example Use a regexp, but find a problem.
recursive-count-words Start with case of no words in region.
Counting Exercise

The `count-words-example` Function

Design count-words-example The definition using a `while` loop.
Whitespace Bug The Whitespace Bug in `count-words-example`.

Counting Words in a `defun`

Divide and Conquer
Words and Symbols What to count?
Syntax What constitutes a word or symbol?
count-words-in-defun Very like `count-words-example`.
Several defuns Counting several defuns in a file.
Find a File Do you want to look at a file?
lengths-list-file A list of the lengths of many definitions.
Several files Counting in definitions in different files.
Several files recursively Recursively counting in different files.
Prepare the data Prepare the data for display in a graph.

Count Words in `defuns` in Different Files

lengths-list-many-files Return a list of the lengths of defuns.
append Attach one list to another.

Prepare the Data for Display in a Graph

Data for Display in Detail
Sorting Sorting lists.

Files List Making a list of files.
Counting function definitions
Readying a Graph
Columns of a graph
graph-body-print How to print the body of a graph.
recursive-graph-body-print
Printed Axes
Line Graph Exercise
Your `.emacs` File
Default Configuration
Site-wide Init You can write site-wide init files.
defcustom Emacs will write code for you.
Beginning init File How to write a `.emacs` init file.
Text and Auto-fill Automatically wrap lines.
Mail Aliases Use abbreviations for email addresses.
Indent Tabs Mode Don't use tabs with TeX
Keybindings Create some personal keybindings.
Keymaps More about key binding.
Loading Files Load (i.e., evaluate) files automatically.
Autoload Make functions available.
Simple Extension Define a function; bind it to a key.
X11 Colors Colors in X.
Miscellaneous
Mode Line How to customize your mode line.
Debugging
debug How to use the built-in debugger.
debug-on-entry Start debugging when you call a function.
debug-on-quit Start debugging when you quit with `C-g`.
edebug How to use Edebug, a source level debugger.
Debugging Exercises
Handling the Kill Ring
What the Kill Ring Does
current-kill
yank Paste a copy of a clipped element.
yank-pop Insert element pointed to.
ring file
The `current-kill` Function
Code for current-kill
Understanding current-kill

`current-kill` in Outline

Body of current-kill

Digression concerning error — How to mislead humans, but not computers.

Determining the Element

A Graph with Labeled Axes

Labeled Example

print-graph Varlist — `let` expression in `print-graph`.

print-Y-axis — Print a label for the vertical axis.

print-X-axis — Print a horizontal label.

Print Whole Graph — The function to print a complete graph.

The `print-Y-axis` Function

print-Y-axis in Detail

Height of label — What height for the Y axis?

Compute a Remainder — How to compute the remainder of a division.

Y Axis Element — Construct a line for the Y axis.

Y-axis-column — Generate a list of Y axis labels.

print-Y-axis Penultimate — A not quite final version.

The `print-X-axis` Function

Similarities differences — Much like `print-Y-axis`, but not exactly.

X Axis Tic Marks — Create tic marks for the horizontal axis.

Printing the Whole Graph

The final version — A few changes.

Test print-graph — Run a short test.

Graphing words in defuns — Executing the final code.

lambda — How to write an anonymous function.

mapcar — Apply a function to elements of a list.

Another Bug — Yet another bug ... most insidious.

Final printed graph — The graph itself!