↖ Topic Index

Emacs Lisp Topics 🚧

Topic & Link	Description	Note
Last updated on:	2026-03-17	A sparse table holding some info about Emacs Lisp. An early state content not updated often.
Lisp General Learning		
Structure and Interpretation of Computer Programs	General intro to Lisp - must read	
ELisp - Emacs Lisp		
GNU Emacs Manuals Online	Lists all GNU Emacs manuals, reference cards, etc…	
An Introduction To Emacs Lisp Programming - Robert J. Chassell	Read first. This can also be read in Info mode inside emacs. This is best because you can execute the ELisp code directly. Note , however, that this is a slow rate introductory text.	
An Introduction To Emacs Lisp Programming - Book Review		
Gnu Emacs Manual - Evaluating Emacs Lisp Expressions	Gnu Emacs manual section on evaluating expressions: list commands available.	
Emacs manual - 27.9: evaluating Emacs Lisp Expressions		
Emacs Lisp @ wikipedia		
How to run set interactively		
Emacs Lisp Guide	A quick overview, with links to Emacs Lisp manual on various topics.	
Elisp - Symbols & Variables		
GNU Emacs Manual - Ch 8 - Symbols	(**symbolp** OBJECT)	
Emacs keywords: constant variables: symbol whose name start with ':'	(**keywordp** OBJECT)	
Valued constants, defined with (defconst)	- (**defconst** SYMBOL INITVALUE [DOCSTRING]) - defines as global variable not meant to be modified (but can still be changed by code)	
Variables; defined with (defvar)	- (**defvar** SYMBOL &optional INITVALUE DOCSTRING) - defines as global variable	
GNU Emacs Manual - Ch 11 - Variables		
GNU Emacs Manual - Ch 14 - Customization Settings	Making variable customizable	
Gnu Emacs Manual - Defining Customization variables with: defcustom, deface and defgroup macros	- (**defcustom** SYMBOL STANDARD DOC &rest ARGS) - (**defface** FACE SPEC DOC &rest ARGS) - (**defgroup** SYMBOL MEMBERS DOC &rest ARGS)	
Gnu Emacs Manual - Common item keywords (Ch14.1)	- :tag - :group - :link - :load - :require - :version - :package-version	

Comparing values - Equality Predicates	Operator	Description	Examples
• Use **eq** to compare symbols	**eq**	t if both arguments are the exact same **object** in memory, otherwise nil. • Fast comparison of **symbols.** • **make-symbol** returns an unintended symbol, distance from a symbol that is used in a Lisp expression. Distance symbols with the same name are not **eq**. • **Built-in**	• `(eq 'abc 'abc) → t` • `(let ((sa 'abc) (sb 'abc)) (eq sa sb)) → t` • `(eq (make-symbol "foo") 'foo) → nil` • `(eq (intern "foo") 'foo) → t` • `(eq 1.0 1) → nil` • `(eq 1.0 1.0) → t or nil` • `(eq 1 1) → t` • `(let ((a 1) (b 1)) (eq a b)) → t` • `(let ((la '(a b) (lb '(a b)) (eq la lb)) → nil` • `(eq "abc" "abc") → t or nil` • `(eq (make-string 3 ?A) (make-string 3 ?A)) → nil` • `(eq [(1 2) 3] [(1 2) 3]) → nil` • `(eq (point-marker) (point-marker)) → nil`
• To compare numbers of the same type	**eql**	Same as **eq** but specifically compare numeric types. • Mostly used to compare numbers of similar type. • **Built-in**	• `(eql 'abc 'abc) → t` • `(let ((sa 'abc) (sb 'abc)) (eql sa sb)) → t` • `(eql 1.0 1) → nil` • `(eql 1.0 1.0) → t` • `(eql 1 1) → t` • `(let ((a 1) (b 1)) (eql a b)) → t` • `(let ((la '(a b) (lb '(a b)) (eql la lb)) → nil` • `(eql "abc" "abc") → nil`
• Use **equal** for general content comparison.	**equal**	Content-based comparison. Works for most types. • numbers, characters, strings comparison (case sensitively), recursively compares elements of lists and vectors. • **Built-in** • Does not signal an error when comparing a string to a non-string type. • Ignores text properties when comparing strings.	• `(equal 'abc 'abc) → t` • `(let ((sa 'abc) (sb 'abc)) (equal sa sb)) → t` • `(equal 1.0 1) → nil` • `(equal 1 1) → t` • `(let ((a 1) (b 1)) (equal a b)) → t` • `(let ((la '(a b) (lb '(a b)) (equal la lb)) → t` • `(equal "abc" "abc") → t` • `(equal "abc" 'abc) → nil` • `(equal "abc" 2) → nil` • `(equal "2" 2) → nil`
• More flexible comparison from the **cl-lib**	**cl-equalp**	• From the **cl-lib** package. • More flexible comparison, specifically for case-insensitive string comparison and number comparison without regard to type. • It does not signal an error when comparing string and number	• `(cl-equalp 'abc 'abc) → t` • `(let ((sa 'abc) (sb 'abc)) (cl-equalp sa sb)) → t` • `(cl-equalp 1.0 1) → t` • `(cl-equalp 1 1) → t` • `(let ((a 1) (b 1)) (cl-equalp a b)) → t` • `(let ((la '(a b) (lb '(a b)) (cl-equalp la lb)) → t` • `(cl-equalp "abc" "abc") → t` • `(cl-equalp "abc" 'abc) → nil` • `(cl-equalp "abc" 2) → nil` • `(cl-equalp "2" 2) → nil`

Topic & Link	Description	Note	
	string=	Compare strings and symbols, character by character, case sensitively. • Ignores text properties • **Signals an error** if one of the argument is not a string or a symbol.	
• Emacs >= 29	**string-equal-ignore-case**		
	equal-including-properties		
	compare-strings		
	compare-buffer-substrings		
Elisp - Defining Functions/Lambdas			
GNU Emacs Manual - Ch 12 - Functions			
Elisp - Macros	**GNU Emacs Manual - Macros**		• **Edebug and macros** • Instrumenting macro calls • Specification list
The pcase macro	• The case of pcase - by Andrew Favia . Provides examples. of how to use pcase.		
Elisp Libraries			
Learning GNU Emacs - Ch 11.7 - Building your own Lisp Library			
GNU Emacs Manual - Libraries of Lisp Code for Emacs	Describe the **load-file** and **load-library** functions, the impact of **load-prefer-newer** and **load-**path variables. It also describes the lower-level **load** function.		
GNU Emacs Manual - How Programs Do Loading			
Elisp - Byte-Compile			
GNU Emacs Manual - Ch 16 - Byte Compilation	Top level chapter on byte-compilation		
Where and when should emacs lisp files be byte-compiled? @ Stack Exchange			
Command to remove all .elc files in directory tree:	find ~/.emacs.d/ -type f -name "*.elc" -delete		Example for inside ~/.emacs.d

Topic	URL	Note
Emacs Lisp - List/Array/Hash Operations		
GNU Emacs Manual - Ch 5 - Lists	https://www.gnu.org/software/emacs/manual/html_node/elisp/Lists.html#Lists	
GNU Emacs Manual - Ch 6 - Sequences, Arrays & Vector	https://www.gnu.org/software/emacs/manual/html_node/elisp/Sequences-Arrays-Vectors.html#Sequences-Arrays-Vectors	
GNU Emacs Manual - Ch 7 - Hash Tables	https://www.gnu.org/software/emacs/manual/html_node/elisp/Hash-Tables.html#Hash-Tables	
List Modifications @ EmacsWiki	https://www.emacswiki.org/emacs/ListModification	
Emacs Lisp - Strings Operations		
GNU Emacs Manual - Ch 4 - Strings and Characters	https://www.gnu.org/software/emacs/manual/html_node/elisp/Strings-and-Characters.html#Strings-and-Characters	
trim string	https://stackoverflow.com/questions/21357537/how-to-trim-whitespace-from-the-end-of-a-variable#21357675	- Module: subr-x - Functions: string-trim, string-trim-left, string-trim-left
concatenating paths to create file path	https://stackoverflow.com/questions/3964715/what-is-the-correct-way-to-join-multiple-path-components-into-a-single-complete	Use: file-name-as-directory
Emacs Lisp - Control Structures		
GNU Emacs Manual - Ch 10 - Control Structures	https://www.gnu.org/software/emacs/manual/html_node/elisp/Control-Structures.html#Control-Structures	- Sequencing, conditionals (if, cond, when, unless), combining conditions (and, or, not), iterations (while loops), generators (generic sequences & coroutines), nonlocal exits.
References - ELisp Code Techniques		
Elisp Cookbook @ EmacsWiki	https://www.emacswiki.org/emacs/ElispCookbook	several easy to use code snippets
Elisp: Check If a {function, variable, feature} is Defined/ Loaded	http://ergoemacs.org/emacs/elisp_check_defined.html	
ELisp - Inserting text		
How do I get emacs to insert text into an arbitrary file?	https://superuser.com/questions/297774/how-do-i-get-emacs-to-insert-text-into-an-arbitrary-file	- Use the function: (insert “text”)
ELisp - Lack of Tail Call Optimization		
recur - package that implement TCO	https://github.com/VincentToups/recur	Implements a lisp package (recur) that provides TCO. Good description. Author also uses Clojure.
tco.el	https://github.com/Wilfred/tco.el	Another implementation
Stack Overflow - Why is there no tail recursion optimization in Emacs lisp, not but like other scheme?	https://stackoverflow.com/questions/38493904/why-is-there-no-tail-recursion-optimization-in-emacs-lisp-not-but-like-other-sc	
Reference - Speeding Up Emacs		
Optimizing Emacs Startup @ Emacs Wiki	https://www.emacswiki.org/emacs/OptimizingEmacsStartup	
What can I do to speed up my start-up? @ StackExchange	https://emacs.stackexchange.com/questions/2286/what-can-i-do-to-speed-up-my-start-up	Has a list of do and don’t .-
How can I make Emacs start-up faster? @ StackOverflow	https://stackoverflow.com/questions/778716/how-can-i-make-emacs-start-up-faster	Another set of ideas listed.
Emacs speed up 1000%	http://blog.binchen.org/posts/emacs-speed-up-1000.html	Speed up by copying to RAM
Speeding Up Emacs	https://anuragpeshne.github.io/essays/emacsSpeed.html	Describes using the **use-package** as well as esup package
To display Emacs init time	(emacs-init-time)	Evaluate this expression to see time it takes to emacs to startup.

Topic	URL	Note
Profile your Emacs start-up time	https://oremacs.com/2015/02/24/emacs-speed-test/	Also use the **use-package** . Use profile-dotemacs.el
Speedup Emacs startup using byte-compiled Elisp files	https://coderwall.com/p/_4irmw/speedup-emacs-startup-using-byte-compiled-elisp-files	Just a reminder of the techniques described in the GNU Emacs manual.
Reference - Emacs (generic)		
What are the best resources for learning Emacs Lisp?	Description of best resources to learn lisp. Worth reading.	
An Introduction to Programming in Emacs Lisp	The introduction manual part of Emacs.	
Emacs blog - emacshorror		
Emacs blog - emacsninja		