Lispy — Short & Sweet Semantically Aware Lisp Editing

Description	Key	Function	Note
Lispy : Context-based modal editing of Lisp code **Ref: Lispy function Reference**	The **lispy** minor mode provides modal-like editing to Emacs for Lisp-like languages with very few keys when point is before **(** or after **)** "***paren***". • On other locations keys self insert, but when point (the cursor) is before the left, opening, paren or after the right, closing paren, **the keys are interpreted as lispy commands**. This table lists the lispy command keys, with links to the **Lispy function Reference** for each one. 📦 This requires the **lispy** external package. PEL downloads, installs and activates lispy when the **pel-use-lispy** user option is set to **t**. 🎛 ☝ To get lispy mode run when Emacs visits a file of a specified mode, include the major mode in the PEL user-option **pel-modes-activating-lispy .** • PEL does not activate lispy for any major mode by default. That's OK to learn lispy by activating it for testing. But once you learn and are comfortable with it you will want to activate when the file is opened automatically by adding the major mode in that list.		
∑ Customize PEL use of Lispy and Lispy itself.	**<f11> <f2> SPC M-L**	(**pel-cfg-pkg-lisp** &optional OTHER-WINDOW)	Prompt to customize: 1. PEL lispy support for Emacs Lisp and Common Lisp 2. lispy itself. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Toggle Lispy mode See also: • **𝔓𝔩 - Common Lisp** • **𝔈𝔩 - Emacs Lisp**	• **<f12> M-L** • **<M-f12> M-L** **<f11> SPC l M-L**	(**pel-lispy-mode** &optional ARG)	Toggle lispy-mode on/off. Lispy is a minor mode for navigating and editing LISP dialects. 📦 Requires lispy external package. PEL downloads, installs and configure it when **pel-use-lispy** user option is set to **t**. Please read the information on lispy web site. 💻 **pel-lispy-mode** calls **lispy-mode** but also prepares hydra, loaded dynamically with PEL. ☝ Set the **pel-modes-activating-lispy** user-option to activate lispy automatically for major modes.
Getting Code Help See also: **∑ Help/Info**	Use the following keys to pop information inside the current window or into a help buffer. See the **∑ Help/Info** table for more help commands • The **<f12> 1** and **<f12> 2** PEL keys are available even when lispy mode is off.		
Describe function at point **See Also: ∑ Help/Info**	**C-1** **<f12> 1**	(**lispy-describe-inline**)	Display documentation of current Lisp function (or variable if marked) as a pop-up overplayed window. • If docstring is too long it is displayed inside a *lispy-help* buffer. The **<f12> 1** key can be used even when lispy mode is not active.
Describe function arguments	**C-2** **<f12> 2**	(**lispy-arglist-inline**)	Show the argument list of current function. The **<f12> 2** key can be used even when lispy mode is not active.
Describe function/ variable	**xh**	(**lispy-describe**)	A shorthand for describe-function or describe-variable, showing help in the *Help* buffer. • If you want to call describe-variable, you should mark the symbol first.
Show top level form	**xw**	(**lispy-show-top-level**)	Show top-level form containing point on mode-line. Eg. inside a defun, show defun name & args.
Numeric Arguments in Lispy	☝ With lispy, numeric arguments can be typed as straight numbers: there's no need to use **M-2** to provide the argument 2, just type **2**. • For example just type two characters **4**, followed by **c** to create 4 clones of the following S-expression (sexp). • This is true only when point is just before **(** or after **)**. You can also type numerical arguments with the Meta key prefix for some commands in other positions, such as the **]** and **[** keys.		
Miscellaneous	Here's a set of commands you might need to use very early when using lispy.		
undo	**u**	(**special-lispy-undo**)	Deactivate region and 'undo'.
View: center current sexp	**v**	(**special-lispy-view**)	Recenter current sexp to be on the first line of the window. **vv** recenters back to the original position.
Multiple Cursors See: **∑ Cursor**.	Lispy supports operations on multiple cursors, allowing concurrent visible operations on several spots in the current window. 📦 Requires the external **multiple-cursors** package. PEL activates it when **pel-use-multiple-cursors** is set to **t**.		
Set multiple cursors • **Add extra cursor(s)**	**xm**	(**lispy-cursor-ace**)	Add a cursor at a visually selected paren using an Avy target. • Only one cursor can be added with local binding. Any amount can be added with a global binding. • Return to single cursor with **C-g**
Add cursors down See: **∑ Cursor**.	**C-7** **<f12> 7**	(**lispy-cursor-down** ARG)	Add ARG cursors using 'lispy-down'. ☝ I found that using the multi-cursor commands directly works well, and often better, than this command.
Insert	The following keys insert and modify whitespace. See other code re-formatting commands in the "Reformat Code" section below.		
Context sensitive space insertion	**<SPACE>**	(**lispy-space** ARG)	Insert one space, with position depending on ARG: If ARG is 2, amend the current list with a space from current side. If it is 3, switch to the different side beforehand. • If jammed between parens, "(|(" unjam: "(| (".
	• If after an opening delimiter and before a space (after wrapping a sexp, for example), do the opposite and delete the extra space, "(| foo)" to "(|foo)".		
	o<SPACE>	(**special-lispy-other-space**)	Alternative to 'lispy-space': leave point on the other side.
Insert a new indented line	• **C-m** • **RET**	(**lispy-newline-and-indent-plain**)	Insert new line and indent next line appropriately
Insert a colon	**:**	(**lispy-colon**)	Insert a colon and precede it by a space in situations where a tag could be written.
Insert a caret	**^**	(**lispy-hat**)	Insert a caret and precede it by a space in required situations. Used for **Clojure metadata marker.**
Commenting	Lispy provides the **;** key that comments the sexp the follows point, as opposed to the standard **M-;**, also available, which creates a comment at the end of the line. When a block is marked both commands comment it.		
Inserting comment	**;**	(**lispy-comment** &optional ARG)	Comment ARG sexps. • **C-u ;** un-comments.
Insert pairs	The following commands insert pairs of delimiters or quotes. They can be typed anywhere.		
insert a paren pair	**(**	(**lispy-parens** ARG)	Insert a **( )** parenthesis pair, leave point inside.
Insert a paren pair after end of current list	**C-8** **<f12> 8**	(**lispy-parens-down**)	Exit the current S-expr and insert a **( )** parenthesis pair , leave point inside.
Insert [ ]	**}**	(**lispy-brackets** ARG)	Insert a **[ ]** pair, leave point inside.
Insert { }	**{**	(**lispy-braces** ARG)	Insert a **{ }** pair, leave point inside.
Insert " "	**"**	(**lispy-quotes** ARG)	Insert a pair of quotes around the point. When the region is active, wrap it in quotes instead. • When inside string, if ARG is nil quotes are quoted, otherwise the whole string is unquoted.
Delete See also: **∑ Cut & Paste**	Lispy provide two context sensitive delete commands, but does not bind the **<deletechar>** key (the ⌦ key, available as **Fn** ⌫ on Apple laptops). • **<f11> DEL x** deletes the S-expression at point. It deletes a complete list when point is on the parens. • **<f11> DEL (** deletes the complete list enclosing point as long as is inside the list and not on a parens. See **∑ Cut & Paste**		
Delete sexp forward	**C-d**	(**lispy-delete** ARG)	Delete ARG chars or sexps depending on context. Delete sexp, string when point is at the beginning of the sexp or string. When point is at end of sexp/string, delete any trailing whitespace and move to beginning of sexp/string to allow using C-d again to delete the sexp/string.
Delete sexp backward	**DEL**	(**lispy-delete-backward** ARG)	From ")|", delete ARG sexps backwards. • Otherwise ('backward-delete-char-untabify' ARG). • Useful to remove all spaces between a paren and the previous one.

Description	Key	Function	Note
Mark a region	Mark S-expression with the following commands. ☝See the command **a** above: it allows marking any symbol using avy.		
Mark symbol	`M-m`	(**lispy-mark-symbol**)	Mark current symbol. Can be issued anywhere.
Mark/Unmark list	`m`	(**special-lispy-mark-list** ARG)	Mark the current sexp, moving point to the other end. • If mark is already active, deactivate it instead. When ARG is more than 1, mark ARGth element.
mark car: select car of marked list	`i`	(**lispy-mark-car**)	Mark the car of **currently active region**. Moves point after the first symbol in the list.
Grow marked area: include next/prev sexp	`>`	(**special-lispy-slurp** ARG)	Grows marked S-expression to include next.
Shrink marked area: exclude next/prev sexp	`<`	(**special-lispy-barf** ARG)	Shrink marked S-Expression: exclude the one at current end of list of marked S-expressions.
Kill, Copy & Paste See also: **Cut & Paste**	Lispy kill commands below maintain the consistency of the list parens. ☝PEL provides commands to kill and copy S-expressions when point is inside and not on parens: `<f11> -` `(` and `<f11> =` `(` **See Cut & Paste**		
Kill string or list at point	`C-,` `<f12> DEL`	(**lispy-kill-at-point**)	Kill the quoted string or the list that includes the point. ☛ The **C-,** key binding is not available in terminal mode. PEL provides the `<f12> DEL` alternative.
Kill word forward	`M-d`	(**lispy-kill-word** ARG)	Kill ARG words, keeping parens consistent.
Kill word backward	`M-DEL`	(**lispy-backward-kill-word** ARG)	Kill ARG words backward, keeping parens consistent.
Kill line	`C-k`	(**lispy-kill**)	Kill line, keeping parens consistent.
Kill from point to end of list	`M-k`	(**kill-sentence** &optional ARG)	Kill from point to end of list. • With arg, repeat; negative arg -N means kill back to Nth start of list.
Copy region or sexp to kill ring	`n`	(**special-lispy-new-copy**)	Copy marked region or sexp to kill ring.
Paste	`P`	(**special-lispy-paste** ARG)	When region is active, replace it with current kill. Forward to yank otherwise. • When ARG is given, paste at that place in the current list.
Navigate with avy commands 🎯	The following commands use avy-style highlighting to identify a word target to move to. Avy is similar to Ace. Lispy uses Avy internally. • By default the scope is the current list. Use a command numerical prefix to select a larger outer scope. • After hitting the command key, type the letter(s) identifying the target to move to that word and select it. **They all use avy navigation.**		
ace symbol move • **ARG sets target scope** • **ace highlight targets** • **move to selected word** • **and mark it**	`a` 🎯	(**special-lispy-ace-symbol** ARG)	Jump to a symbol within the **current S-exp** and **mark it**. • Each symbol in S-exp is shown with highlight letter: type that letter to move to the symbol. • S-exp scope is obtained by exiting the list ARG times: default is 1: current S-exp. to select a larger scope S-exp, use a numeric argument: • Example: **3a** selects 3 layers of enclosing S-exp to select ace targets.
ace sub-word • **ARG sets target scope** • **ace highlight targets** • **move to selected sub-word and mark it**	`-` 🎯	(**special-lispy-ace-subword** ARG)	Similar to lispy-ace-symbol, but selects a subword instead. • S-exp scope is obtained by exiting the list ARG times: default is 1: current S-exp. to select a larger scope S-exp, use a numeric argument: • Example: **3a** selects 3 layers of enclosing S-exp to select ace targets.
Move to Ace target symbol & erase to replace	`H` 🎯	(**special-lispy-ace-symbol-replace** ARG)	Jump to a symbol within the current sexp and **delete it**, leaving point at location to type the new symbol. • Sexp is obtained by exiting the list ARG times. • Calls lispy-ace-symbol and deletes the selected symbol.
Move to Ace paren target	`q` 🎯	(**special-lispy-ace-paren** &optional ARG)	Highlights each **symbol** in current sexp as ace target and jump to the selected one. • Updates lispy-back history. • S-exp scope is obtained by exiting the list ARG times: default is 1
Move to Ace target char	`Q` 🎯	(**special-lispy-ace-char**)	Prompts for character, highlights each one in current sexp as ace target and jump to the selected one.
Navigate by-list	The following commands move point inside code when point is before left paren or after right paren. Use **d** to switch side to control direction. The **z** key starts the **knight movement hydra** providing access to the **j** knight-down and **k** knight-up. Use **z**, or any key but **j** or **k** to stop the hydra.		
Move left outward	`h` ↖	(**special-lispy-left** ARG)	Move outside list backwards ARG times.
Move down current list • **never exit current list** • **from beginning of top level form to the next**	`j` ↓	(**special-lispy-down** ARG)	Move down ARG times inside current list. • With point at the top level move to the next top-level form. Inside a list move to each • Guaranteed to never exit the list: **99j** moves to the last element of the current list. • Moves downward to next to comment if issued from point at start of comment line (on the **;;**).
Move down left-most parens on each line	• `zj` ↳ • `j`	(**lispy-knight-down**)	Move down left-most paren to the next line (can exit list).
Move up current list • **never exit current list** • **from end of top level form to previous one**	`k` ↑	(**special-lispy-up** ARG)	Move up ARG times inside current list. • Guaranteed to never exit the list: **99k** moves to the first element of the current list. • Moves upward to previous to comment if issued from point at start of comment line (on the **;;**).
Move up left-most parens on each line	• `zk` ↰ • `k`	(**lispy-knight-up**)	Move up left-most paren to the previous line (can exit list)
Move outside list forward .	`l` ↗	(**special-lispy-right** ARG)	Move outside list forwards ARG times. • Parens in strings and comments are ignored.
Flow via current paren • **(➜ down** • **; ➜ down** • **) ➜ up**	`f` ⤷	(**special-lispy-flow** ARG)	Move in the direction of current paren **inside** current list and then to the next/previous list: • At left : move to next left paren (move going down the file or **into** the list). • Move forward **into a list**, then each sub-list, then to beginning of next top-level list. • At right: move to previous right parent (move going up the file). • Don't enter strings or comments.
Move to beginning of current defun	`A`	(**special-lispy-beginning-of-defun** &optional ARG)	Forward to beginning-of-defun. When called twice in a row, restore the previous point and mark positions.
Move to beginning of line. Reveal Outline	`C-a`	(**lispy-move-beginning-of-line**)	Move to beginning of line
Move to end of line. • **In string: to end of string.** • **Again: back to original**	`C-e`	(**lispy-move-end-of-line**)	Forward to 'move-end-of-line' unless already at end of line. • Then return to the point where it was called last, when it was in a string, back to the end paren close to where it was. • If this point is inside string, move outside string. ☝Useful in multi-line strings.
Move forward to end of list • **from beginning of top level form to the next**	`]` →	(**lispy-forward** ARG)	Move forward list ARG times or until error. • ☝Can type it from any location, even when point is not before the beginning or after the end of a list. • ⚠Also active inside strings and comments. Use **}** to insert a **[]** pair.
Move backward to beginning of list • **from end of top level form to previous one**	`[` ←	(**lispy-backward** ARG)	Move backward ARG times to beginning of previous list, up to out of current top-level list and then to previous top level-list. • ☝Can type it from any location, even when point is not before the beginning or after the end of a list. • ⚠Also active inside strings and comments. Use **}** to insert a **[]** pair.

Description	Key	Function	Note
Move to different (other) side of sexp	`d`	(**special-lispy-different**)	Switch to the different side of current sexp. • If before '`(`' move after '`)`' and vice-versa.
Move outside list forward	`C-3` `<f12> 3`	(**lispy-right** ARG)	Move outside list forwards (up level and right) ARG times. Ignore parens in strings. • With no argument, or using Meta prefixed numerical arguments, this key can be typed anywhere. • Just outside parens the argument can be typed as strength numbers. ⌨ The `C-3` key sequence is not available in terminal mode. PEL provides `<f12> 3` as an alternative.
Move outside list forward but self-insert inside strings and comments	`)`	(**lispy-right-nostring** ARG)	Same as **lispy-right:** move outside list forwards (up level and right) ARG times. • However self-insert when point is located in a string or a comment.
Navigation History	To restore past positions, type `b` around parens. The commands marked with ☮ update lispy back history.		
Move back	`b`	(**special-lispy-back** ARG)	Move point to ARGth previous position in lisps-back history. • If position isn't special, move to previous or error. • Lispy back history updated by: `f`, `h`, `i`, `j`, `k`, `l`, `m`, and `q`. These commands are identified with ☮
Search	Lispy search operations are restricted to a specific list scope. See ∑ **Search/Replace** for more search operations, including the unbounded occur search.		
Occur search inside the current top-level sexp	`y`	(**special-lispy-occur**)	Do an occur for the current top-level sexp. Go back-to-paren afterwards. This is useful e.g. to see where a particular variable is used within the current defun.
Goto Definition	The following commands take advantage of the cross reference system available to jump to the definition of the specified symbol. • Some of the commands prompt using the ivy completion mechanism. More information on input completion is available in ∑ **Completion/Input** ☝ Once you use one command that starts with the `og` prefix the next letter is interpreted within this group. To get out you must type the letter `q`.		
goto definition using directory tags	`g`	(**special-lispy-goto** &optional ARG)	Jump to symbol within files **in current directory**. Prompt for symbol and jump to it. • When ARG isn't nil, call 'lispy-goto-projectile' instead. • See **lispy goto wiki page**.
goto definition in local file	`G`	(**special-lispy-goto-local** &optional ARG)	Similar to lispy-goto, but only current file's tags are used instead of whole directory's tags.
Follow: jump to definition	`F` `M-.`	(**special-lispy-follow**) (**lispy-goto-symbol** SYMBOL)	When region is active jump to the definition of marked symbol. Otherwise jump to the definition of the first symbol in current sexp. • `M-.` can be issued from any position.
Move back from symbol definition jump	`D` `M-,`	(**special-pop-tag-mark**) (**pop-tag-mark**)	Go back from where it came with Follow. • M-, can be issued from any position.
Move to definition of selected lisp element	`oga`	(**special-lispy-goto-def-ace** ARG)	Jump to definition of selected element of current sexp. • Sexp is obtained by exiting list ARG times.
Move back: pop tag	`ogb`	(**special-pop-tag-mark**)	Pop back to where M-. was last invoked.
Move to symbol within files of current directory	`ogd`	(**special-lispy-goto** ARG)	Jump to symbol within files in current directory. ⚠ Potentially long search process. Stop with **C-g**. • When ARG isn't nil, call 'lispy-goto-projectile' instead.
Move to Elisp command pithing current file	`oge`	(**special-lispy-goto-elisp-commands**)	Jump to Elisp commands **within current file**. Prompts using ivy completion mechanism. • When ARG is non-nil, force a reparse.
Follow to the function definition	`ogf`	(**special-lispy-follow**)	Follow to 'lispy--current-function'.
Jump to definition of ARgth element of current list.	`ogj`	(**special-lispy-goto-def-down**)	Jump to definition of ARGth element of current list. ☝ Use this when an argument is a function call. This moves point to the definition of that function.
Jump to definition of symbol	`ogl`	(**special-lispy-goto-local**)	Jump to symbol within current file. Prompts with ivy. • When ARG is non-nil, force a reparse.
goto definition using projectile base directory	• `0g` • `ogp`	(**special-lispy-goto-projectile**)	Jump to symbol within files in ('projectile-project-root').
Quit the 'og' command	`ogq`	(**special-lispy-quit**)	Remove modifiers.
Jump to definition of symbol at point	`ogr`	(**special-lispy-goto-recursive**)	Jump to symbol within files in current directory and its subdirectories. ⚠ Potentially long search process. • Search tags in complete directory tree. Stop with **C-g**.
Narrow/Widening See also: ∑ **Narrowing**	• Narrowing hides everything in the buffer except the selected region, allowing work on that region alone. • Widen it back to see the complete buffer again.		
Narrow current sexp | region	`N`	(**special-lispy-narrow** ARG)	Narrow current sexp or region.
Widen	`W`	(**special-lispy-widen**)	Widen back to see the complete buffer.

Operating on Regions	The commands listed above can be used to operate on a marked region of code:
	
	• **Activate a region** first with one of:
	  • `m` To mark a sexp.
	  • `a` To mark a symbol by its ace target letter. Use numeric argument to widen scope out of current list.
	• **Select another sexp within the list** with:
	  • `j` To select the next sexp in the current list.
	  • `k` To select the previous sexp in the current list.
	• First **select the region growing side**. The grow/shrink operations apply to the current side of the region. Move point to the other side of the region with:
	  • `d` to move to the other side of the region.
	• **Grow or shrink the region** with:
	  • `>` Extends the region with another sexp on the current side.
	  • `<` Shrinks the region by one sexp on the current side.
	  • `h` To mark the entire parent list with the point at the beginning.
	  • `l` To mark the entire parent list with the point at the end.
	  • `i` To reduce the mark to only the first child (the car) of the current list
	• **Operate on the region**:
	  • `m` Deactivate the region.
	  • `u` Deactivate the region and undo.
	  • `c` Clone region and keep it active.
	  • `s` Move region on sexp down.
	  • `w` Move region one sexp up.
	  • `t` Move region inside sexp selected with ace target
	  • `C` Convolute: exchange the order of application of two S-exprs that contain region
	  • `n` Copy region in kill ring without de-activating the mark.
	  • `P` Replace region with current kill.		

Description	Key	Function	Note
• **Reformat code**	The following command do not modify the semantics of the code, they just add or remove whitespace.		
Indent S-expression	`i`	(**special-lispy-tab**)	Update the indentation of all lines in the current S-expression.
Turn current sexp into one line	`O`	(**special-lispy-oneline**)	Turn current sexp into one line. Move comments ahead of sexp. `(progn (one) (two) (three))` → `(progn (one) (two) (three))`
Convert current sexp into multi-line	`M`	(**special-lispy-alt-multiline** &optional SILENT)	Spread current sexp over multiple lines. When SILENT is non-nil, don't issue messages. • Especially useful on results of macroexpand. ☝ The wrapping may not occur for small lists or symbols. `(progn (one) (two) (three))` → `(progn (one) (two) (three))`
Transform code	Lispy provide a large number of code transformation commands. Once you know them they speed up Lisp code editing.		

clone	`c`	(**special-lispy-clone** ARG)	Clone sexp ARG times.
			• When the sexp is top level, insert an additional newline.
			`((one) (two) (three))` → `((one) (two) (three))`
			`((one) (two) (three))`

• **Transform S-expr**	The following operations essentially move or modify S-expressions. Use these to write and refactor code.		

Slurp: grow either current sexp or region	`>`	(**special-lispy-slurp** ARG)	Grow either current sexp or region (if it's active) in appropriate direction. Opposite of lispy-barf.
			• With an arg of 0, grow as far as possible.
			• With an arg of -1, grow until the end of the line where the current sexp ends or as far as possible before that position.
			`(progn (foo) (bar))` → `> →` `(progn (foo) bar))`
			`(progn (foo)_(bar))` → `> →` `(progn (foo (bar))_`

Barf: shrink either current sexp or region	`<`	(**special-lispy-barf** ARG)	Shrink either current sexp or region (if it's active) in appropriate direction. Opposite of lispy-slurp.
			`(progn (foo) (bar))_` → `<  →` `(progn (foo))_(bar)`
			`(progn (foo) (bar))` → `<  →` `(progn (foo) (bar)`

Move current sexp to the left	`oh`	(**special-lispy-move-left**)	Move current sexp (or marked region) to the left, outside current list, ARG times. `(progn (do-something with-this) (do-something with-that))` → `(do-something with-this) (progn (do-something with-that))`
Move current sexp inside first element of list below	`oj`	(**special-lispy-down-slurp**)	Move current sexp or region to become the first element of next sexp. `(100) '((200) (300))` → `'((100) (200) (300))`
Move current sexp to become last element of list above	`ok`	(**special-lispy-up-slurp**)	Move current sexp or region to become the last element of the list above. • If the point is by itself on a line or followed only by right delimiters, slurp the point into the previous list. • This can be of thought as indenting the code to the next level and adjusting the parentheses accordingly. `(progn (do-this) (do-that)) (do-it-again)` → `(progn (do-this) (do-that) (do-it-again))`
Move current sexp to the right, outside current list	`ol`	(**special-lispy-move-right**)	Move current expression (or marked region) to the right, outside the current list. Do it ARG times. `(progn (do-this) (do-that) (do-it-now))` → `(progn (do-this) (do-that)) (do-it-now)`

Join List	`+`	(**special-lispy-join**)	Join next/previous element into current list, as in the next 2 examples.
			`((one)) (two) (three))` → `((one two)) (three))`
			`((one) (two) (three))` → `((one two) (three))`

Split List	`M-j`	(**lispy-split**)	Split S-expressions from character at point as shown in the 4 examples below.
			`(111 222 333)` → `() (111 222 333)`
			`(111 222 333)` → `(111) (222 333)`
			`(111 222 333)` → `(111 2) (22 333)`
			`((one) (two) (three))` → `((one)) ((two) (three))`

Raise: use current sexp as replacement for its parent	`r`	(**special-lispy-raise** ARG)	Use current sexp or region as replacement for its parent. Do so ARG times. `(let ((total 0)) (+ my-count your-count))` → `(+ my-count your-count)`
Raise: current and next| previous sexp as replacement for their parent	`R`	(**special-lispy-raise-some**)	Use current sexp and the following (if called from the left), or the preceeding (if called from the right) sexps, or the active region as replacement for their parent. `(progn (one) (two) (three))` → `(two) (three)`
Convolute: Exchange the order of application of 2 closest outer forms **Example animation**	`C`	(**special-lispy-convolute** ARG)	Exchange the order of application of two closest outer forms, relative to current expression or region. • Replace `(...(,,,|(` with `(,,,(...|(` where `...` and `,,,` is arbitrary code. • When ARG is more than 1, pull ARGth expression to enclose current sexp. • When ARG is nil, convolute only the part above sexp. `(if (> (+ count1 count2) 30) (when verbose (message "over 30")))` → `(when verbose (if (> (+ count1 count2) 30) (message "over 30")))`
Move current sexp up	`w`	(**special-lispy-move-up** ARG)	Move current sexp or region up arg times. Don't exit the parent list. Also works for outlines. `(progn (one) (two) (three))` → `(progn (one) (three) (two))`

Description	Key	Function	Note
Stringify current sexp	• **S** • **C-u "**	(**special-lispy-stringify** &optional ARG)	Transform current sexp into a string. Quote newlines if arg isn't 1. `(progn (one) (two) (three))` → `"(progn (one) (two) (three))"`
Move sexp down in list	**s**	(**special-lispy-move-down** ARG)	Move current sexp or region down arg times. Don't exit the parent list. Also works for outlines. `(progn (one) (three) (two))` → `(progn (one) (two) (three))`
Splice the current list into the parent list	**/**	(**special-lispy-splice** ARG)	Splice ARG sexp into the containing (parent) list. Move the point to the next list to splice in appropriate direction. If there are none within the parent list, move to the parent list in appropriate direction. `((a) (b) (c))` → `(a (b) (c))`
Teleport: move current sexp to Ace target	**t**	(**special-lispy-teleport** ARG)	Move current sexp to Ace target inside current function. • Use numerical argument to move that many sexp In the example below, after typing **t**, the ace target letters show up. Typing **b** gives the result on the right. `aprogn bone) cto) dthree))` → `(progn (one (three)) (two) )`
	tt		Move current sexp to Ace target to any sexp inside current window. Same as above with a wider scope.

Reverse list	• **xR**
	• **x?R**
	• **x C-h R**
		(**lispy-reverse**)	Reverse the current list or region selection.
			⚠ does not handle quoted list properly: it inserts **quote ly-raw** as the last element of the new list. One way to deal with this is to remove the quote, then reverse and put the quote back.
			`(progn (one) (two) (three))` → `((three) (two) (one) progn)`
			`'(111 222 333)` → `((111 222 333) quote ly-raw)`
			`(111 222 333)` → `(333 222 111)`

• **Refactoring**	The following commands provide refactoring facilities.		
turn nested if into cond	• **xc** • **x?c** • **x C-h c**	(**lispy-to-cond**)	Transform current 'if' expressions to equivalent 'cond' expression. `(if is-one (one) (if is-two (two) (if is-three (three))))` → `(cond (is-one (one)) (is-two (two)) (is-three (three)))`
turn cond into nested if expressions	• **xi** • **x?i** • **x C-h i**	(**lispy-to-ifs**)	Transform current 'cond' expression to equivalent 'if' expressions. `(cond (is-one (one)) (is-two (two)) (is-three (three)))` → `(if is-one (one) (if is-two (two) (if is-three (three))))`
Bind var: current sexp to let bound variable	• **xb** • **x?b** • **x C-h b**	(**lispy-bind-variable**)	Transform the current list expression into a let-bound variable; iedit-mode is used to name the new variable. Use **M-m** to finish naming the variable. • Bind current expression as variable. • 'lispy-map-done' is used to finish entering the variable name. The bindings of 'lispy-backward' or 'lispy-mark-symbol' can also be used. `((one) (two) (three))` → `(let (( ((one) (two) (three)))) _)` After issuing the **xb** command type the name of the variable (like **new-var**) here. It shows inside the definition block and just outside. → `(let ((new-var ((one) (two) (three)))) new-var)`
Unbind a let bound variable	**xu**	(**lispy-unbind-variable**)	Substitute let-bound variable • Unbind a let-bound variable. Also works for Clojure. ⚠ Current version fails to update the values of the unbound variable. **See bug report**. `(defun foobar () (let ((x 10) (y 20) (z 30)) (foo1 x y z) (foo2 x z y) (foo3 y x z) (foo4 y z x) (foo5 z x y) (foo6 z y x)))` → `(defun foobar () (let ((y 20) (z 30)) (foo1 10 y z) (foo2 10 z y) (foo3 y 10 z) (foo4 y z 10) (foo5 z 10 y) (foo6 z y 10)))`
Inline current function or macro call	• **xf** • **x?f** • **x C-h f**	(**lispy-flatten** ARG)	Inline current function or macro call, i.e. replace it with function body. • The function should be interned and its body find-able. • Pass the ARG along. `(setq-local foo 10)` → `(set (make-local-variable 'foo) 10)`

Inline current function/ macro call with a let	• **xF**
	• **x?F**
	• **x C-h F**
		(**lispy-let-flatten**)	Inline a function at the point of its call using 'let'.
			Given the following defun on the right: Typing **xF** to the code below transforms it in the code to the right below.
			`(defun add (a b) "Add A and B." (+ a b))`
			`(defun sum-squared (a b) "Sum of A squared + B squared." (add (* a a) (* b b)))` → `(defun sum-squared (a b) "Sum of A squared + B squared." (let ((a (* a a)) (b (* b b))) (+ a b)))`

turn current lambda into a defun	**xd**	(**lispy-to-defun**)	Turn the current lambda or toplevel sexp or block into a defun. • Prompts for the name of the new defun. Replace the lambda S-expression by the function quoted name and keep the defun S-expression in the kill ring. Use **C-y** later to insert the defun inside a buffer. `(mapcar (lambda (x) (* x x)) (number-sequence 1 10))` → `(mapcar #'square (number-sequence 1 10))`

Description	Key	Function	Note
			Type **xd** to extract the lambda: Lispy prompts for the name of the defun and replace it as above right. Then use **C-y** to insert the defun form as shown at right. `(defun square (x) (* x x))`
Create defun out of marked block	**xD**	(**lispy-extract-defun**)	Extract the marked block as a defun. • Prompts for the name of the new defun, turn the block into the defun and insert a call to the defun below. • For the defun to have arguments, capture them with 'lispy-bind-variable'

		Starting with the following code, issue the xD command at the beginning of the form to extract	`(defun some-func (&optional count)`
			`  "Do something."`
			`  (if count`
			`  (while (progn`
			`  (insert (format  "Countdown: %d\n" count))`
			`  (setq count (1- count))`
			`  (> count 0)))`
			`  (insert "Nothing to do!\n")))`

		Lispy extracts the code, prompts for a new function name and insert the new function above the existing one.	`(defun insert-countdown ()`
			`  (while (progn`
			`  (insert (format "Countdown: %d\n" count))`
			`  (setq count (1- count))`
			`  (> count 0))))`
			
			`(defun some-func (&optional count)`
			`  "Do something."`
			`  (if count`
			`  (insert-countdown)`
			`  (insert "Nothing to do!\n")))`

Transform current sexp/ region into a function call	**xk**	(**lispy-extract-block**)	Transform the current sexp or region into a function call. • The newly generated function will be placed above the current function. • Starts the input for the new function name and arguments. To finalize this input, press **[** .

			`(cond (is-one`
			`  (one))`
			`  (is-two`
			`  (two))`
			`  (is-three`
			`  (three)))`
			
			`(defun  ()`
			`  (cond (is-one`
			`  (one))`
			`  (is-two`
			`  (two))`
			`  (is-three`
			`  (three))))`
			
			`()`

			After typing **xk** a name-less defun form is created above an empty form. Then type the name of the defun followed by the name of the arguments which will be populated in both forms as shown to the right.
			
			To complete, type **[** and point will move to the right of the coming parens.
			
			`(defun new-func (is-one is-two is-three)`
			`  (cond (is-one`
			`  (one))`
			`  (is-two`
			`  (two))`
			`  (is-three`
			`  (three))))`
			
			`(new-func is-one is-two is-three)`

turn current defun into a lambda	**xl**	(**lispy-to-lambda**)	Turn the current function definition into a lambda.
			`(defun add (a b)`
			`  "Add 2 numbers."`
			`  (+ a b))`
			
			`(lambda (a b)`
			`  "Add 2 numbers."`
			`  (+ a b))`

Eval sexp and replace it with its result	**xr**	(**lispy-eval-and-replace**)	Eval current expression and replace it with the result.
			`(delete-dups (sort '(3 1 7 5 3 4 2 1 4 7) #'<))`
			
			`(1 2 3 4 5 7)`

Toggle between last threaded macro form and unthreaded form	**x>**	(**lispy-toggle-thread-last**)	Toggle current expression between the **last-threaded macro** form and the unthreaded forms.
			`(+ 40 (- (/ 25 (+ 20 5))))`
			
			`(thread-last (+ 20 5) (/ 25) (-) (+ 40))`
			⚠ As the example shows, the thread-last code is not always created in the nicest-looking way. Emacs help proposes this instead:
			`(thread-last`
			`  5`
			`  (+ 20)`
			`  (/ 25)`
			`  -`
			`  (+ 40))`
			• The macro used used may be customized in '**lispy-thread-last-macro**' user-option. It default to the Emacs Lisp thread-last macro.

Evaluate Code	The following commands allow deep introspection into Emacs Lisp code and to some extent code written in other language. See **Lispy demo 2** showing **the substitution model for procedure** described in classic **Structure and Interpretation of Computer Programs** in action.		
Eval last sexp	**e**	(**special-lispy-eval** ARG)	Eval last sexp. Display result in echo area. • When ARG is 2, insert the result as a comment.
Eval current region|sexp. Insert result.	**E**	(**special-lispy-eval-and-insert**)	Eval current region or sexp. The result will be inserted in the current buffer after the evaluated expression.
Eval current sext & replace it at point	**xr**	(**lispy-eval-and-replace**)	Eval last sexp and replace it with the result.
Eval current sexp in the content of the of the other window	**p**	(**special-lispy-eval-other-window** &optional ARG)	Eval current expression in the context of other window. • In case the point is on a let-bound variable, add a 'setq'. • When ARG is non-nil, force select the window.
Evaluate current expression for current language	**xv**	(**lispy-eval-expression**)	Like 'eval-expression', but for current language (Emacs Lisp, Common Lisp, Clojure, etc..)
EDegug Support	The following commands can be used to start, use and stop an Emacs Lisp edebug session or Clojure cider debug session. The documentation below assumes Emacs Lisp. 🚧 More info should be added for Clojure.		
EDebug current defun See also: **𝔏𝔦𝔰𝔭𝔶 - Emacs Lisp**	**xe**	(**lispy-edebug** ARG)	Start/stop edebug of current thing depending on ARG. • ARG is 1: 'edebug-defun' on this function. • ARG is 2: 'eval-defun' on this function. • ARG is 3: 'edebug-defun' on the function from this sexp. • ARG is 4: 'eval-defun' on the function from this sexp.

Description	Key	Function	Note
	`1xe`	(**edebug-defun**)	Evaluate the top level form point is in, stepping through with Edebug.
	`2xe`	(**eval-defun** EDEBUG-IT)	Evaluate the top-level form containing point, or after point.
	`3xe`	(**edebug-defun**)	Evaluate the top level form point is in, stepping through with Edebug. On the function from this sexp.
	`4xe`	(**eval-defun** EDEBUG-IT)	Evaluate the function from this sexp.
Debug - step in	`xj`	(**lispy-debug-step-in**)	• Evaluate the arguments at the current function's call • Jump to the function's definition • Set the result of evaluation to the function's arguments
EDebug stop	`Z`	(**special-lispy-edebug-stop**)	Does the same as q in edebug, except current function's arguments will be saved to their current values. • This allows to continue debugging with lispy-eval (e) from edebug's current context. • The advantage is that you can edit the code as you debug, as edebug puts your code in read-only mode.
ERT test support	More information about the Emacs Lisp Regression Testing system in **ERT**		
Execute Tests: run ert	`xT`	(**lispy-ert**)	Call ('ert' t) : run all ERT tests.
View test at point	`xt`	(**lispy-view-test**)	View better the test at point.
Outline operations	Also see `C-a` above which moves to beginning of line and reveals outlines.		
Insert a new heading	`M-RET`	(**lispy-meta-return**)	Insert a new line followed by a comment for a new heading. Something that starts with: **;;*** ⚠ Unfortunately, by default, this key is active all the time, even when not using Lispy inside org-mode. This conflicts with PEL's global binding for this key. PEL provides the **pel-enable-lispy-meta-return** user option, set off (**nil**) by default, which disables this key. If you want to use it, set this user-option to on (**t**).
Toggles on|off org-mode-like outline	`I`	(**special-lispy-shifttab** ARG)	Toggles on/off an org-mode-like outline. • To make this work, lispy-mode will modify outline-regexp and outline-level-function for the current buffer while it's on.
Indent / hide/show outline	`i`	(**special-lispy-tab**)	If in outline: hide/show outline, otherwise indent all code of current paren • When region is active, call 'lispy-mark-car'.
Next outline level	`J`	(**special-lispy-outline-next** ARG	Takes a numeric prefix arg and calls outline-next-visible-heading arg times or until past the last outline-regexp.
Previous outline level	`K`	(**special-lispy-outline-prev** ARG)	Takes a numeric prefix arg and calls outline-previous-visible-heading arg times or until past the first outline-regexp.
Ediff Operations See: **Diff & Merge**	• Use the `xB` command to identify one S-expression or region. • Then use the `B` command to select another S-expression or region and open an Ediff session comparing these 2 sections of code. • You can use the Ediff features to copy code from one section to the other, etc…		
Store current buffer and region for further operation	• `xB` • `x?B` • `x C-h B`	(**lispy-store-region-and-buffer**)	Select S-expression or region, the side A of a diff session started by executing the command `B` below.

Ediff regions
★★
	`B`	(**special-lispy-ediff-regions**)	Select the S-expression or region, the side B of an Ediff session and start that Ediff session.
			• Comparable to 'ediff-regions-linewise'.
			  • First region and buffer come from 'lispy-store-region-and-buffer'
			  • Second region and buffer are the current ones.

Buffer operations			
Save buffer	`xs`	(**save-buffer** &optional ARG)	Save current buffer in visited file if modified. Same as `C-x C-s`
Visit another file See: **Projectile**	`V`	(**special-lispy-visit** ARG)	Visit another file within this project using projectile or find-file-in-project. • Use `V` to open the file in the current window. Use `2V` to open the file in another window.
	Customize **lispy-visit-method** to select what function to use. • PEL supports both of these external packages, and use the **pel-use-projectile** and **pel-use-find-file-in-project** user-options to download and activate each one. Unless you are familiar with find-file-in-project you may find projectile more useful and faster.		
Others			
Perform cleanup	`xC`	(**lispy-cleanup**)	Perform cleanup. Remove all comments in buffer after current point position that start with **;; =>**

Execute specified command	• `x C-h`
	• `x?`
		(**lispy-x-more-verbosity**)	A Hydra that provides access to several other commands accessible with the following **blue** letters.
			`bnd : Bind variable`
			`cnd : lispy-to-cond`
			`def : lispy-to-defun`
			`ede : Execute edebug-defun`
			`fla :`
			`help : lispy-describe: Open help buffer on the specified function symbol`
			`if : lispy-to-ifs`
			`jmp :`
			`blk :`
			`lmb :`
			`mul :`
			`rep :`
			`sav :`
			`unb :`
			`vt :`
			`Bnd : lispy-store-region-and-buffer`
			`Rev : lispy-reverse`
			`erT : Run ERT test.`

Python Support	The following commands are only available for spy, lispy for Python, in a Python source code file.		
Set Python Process	`xp`	(**lispy-set-python-process**)	
Change current directory	`xn`	(**lispy-cd**)	Change the current Python REPL working directory.