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** • **𝔈𝔓𝔩 - 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**	(**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.
	<f12> 1		The **<f12> 1** key can be used even when lispy mode is not active.
Describe function arguments	**C-2**	(**lispy-arglist-inline**)	Show the argument list of current function.
	<f12> 2		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. When called twice in a row, recenter 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**	(**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.
	<f12> 7		
Insert	The following keys insert and modify whitespace.		
Insert a space	**<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.
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.
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**	(**lispy-parens-down**)	Exit the current S-expr and insert a **( )** parenthesis pair , leave point inside.
	<f12> 8		
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	Lispy provide two context sensitive delete commands, but does not bind the **<deletechar>** key (the ⌦ key, available as **Fn** ⌫ on Apple laptops).		
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).

Description	Key	Function	Note
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 and by-region	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 **move in the direction of 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). • 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**)	Forward to 'move-beginning-of-line'. • Reveal outlines.
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 list ARG times or until error. • Move backward 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.
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 ☮
Cut & Paste, Mark, Hide and Indent			
Indent/hide/show outline	**i**	• With no active region: (**special-lispy-tab**)	• If inside outline: hide/show outline, • otherwise indent and prettify all code of current paren
mark car: select car of marked list ☮		• With active region: (**lispy-mark-car**)	Mark the car of currently active region. Moves point after the first symbol in the list.
Copy region or sexp to kill ring	**n**	(**special-lispy-new-copy**)	Copy marked region or sexp to kill ring.
Mark symbol	**M-m**	(**lispy-mark-symbol**)	Mark current symbol. Can be issued anywhere.
Mark 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.

Description	Key	Function	Note
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.
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			
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 using projectile base directory	• `0g` • `ogp`	(**lispy-goto-projectile**)	Jump to symbol within files in ('projectile-project-root').
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`	(**special-lispy-follow**)	When region is active jump to the definition of marked symbol. Otherwise jump to the definition of the first symbol in current sexp.
	• `M-.`	(**lispy-goto-symbol** SYMBOL)	
Pop tag	• `D`	(**special-pop-tag-mark**)	Go back from where it came with Follow
	• `M-,`	(**pop-tag-mark**)	
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.		
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
	`ogf`	(**special-lispy-follow**)	Follow to 'lispy--current-function'.
	`ogj`	(**special-lispy-goto-def-down**)	Jump to definition of ARGth element of current list.
	`ogl`	(**special-lispy-goto-local**)	Jump to symbol within current file. • When ARG is non-nil, force a reparse.
	`ogq`	(**special-lispy-quit**)	Remove modifiers.
	`ogr`	(**special-lispy-goto-recursive**)	Jump to symbol within files in current directory and its subdirectories. ⚠ Potentially long search process. Stop with **`C-g`**.
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.
• **Move S-expr**	The following operations essentially move a S-expression in various positions.		

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)`

Description	Key	Function	Note

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 their-count))`
			
			`(+ my-count your-count their-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.

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 (> (sum count1 coun2 count3) 30)`
			`  (when verbose`
			`  (message "over 30")))`
			
			`(when verbose`
			`  (if (> (sum count1 coun2 count3) 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.
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.
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
	tt 🎯		Move current sexp to Ace target to any sexp inside current window.

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-again))`
			
			`(progn`
			`  (do-this) (do-that))`
			`(do-it-again)`

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.
Reverse list	**xR**	(**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.
• **Format code**	The following command do not modify the semantics of the code, they just add or remove whitespace.		
Turn current sexp into one line	**O**	(**special-lispy-oneline**)	Turn current sexp into one line. • Move comments ahead of sexp.
Stringify current sexp	**S**	(**special-lispy-stringify** &optional ARG)	Transform current sexp into a string. • Quote newlines if arg isn't 1.
Bind var: current sexp to let bound variable	**xb**	(**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.
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**.
turn nested if into cond	**xc**	(**lispy-to-cond**)	Transform current 'if' expressions to equivalent 'cond' expression.
turn current lambda into a defun	**xd**	(**lispy-to-defun**)	Turn the current lambda or toplevel sexp or block into a defun.

Description	Key	Function	Note
	xD	(lispy-extract-defun)	Extract the marked block as a defun. • For the defun to have arguments, capture them with 'lispy-bind-variable'
Inline current function or macro call	xf	(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.
	xF	(lispy-let-flatten)	Inline a function at the point of its call using 'let'.
turn cond into nested if expressions	xi	(lispy-to-ifs)	Transform current 'cond' expression to equivalent 'if' expressions.
	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 "[".
turn current defun into a lambda	xl	(lispy-to-lambda)	Turn the current function definition into a lambda.

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.

EDegug Support			
	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.
	xj	(lispy-debug-step-in)	Eval current function arguments and jump to definition
ERT test support			
	xt	(lispy-view-test)	View better the test at point.
	xT	(lispy-ert)	Call ('ert' t) : run all ERT tests.
Outline operations	Also see C-a above which moves to beginning of line and reveals outlines.		
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	• With no active region: (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.
Evaluate Code			
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.
	xv	(lispy-eval-expression)	Like 'eval-expression', but for current language (Emacs Lisp, Common Lisp, Clojure, etc..)
Eval current sext & replace it at point	xr		
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.
EDebug current defun	xe		edebug current defun. Or cider-debug-defun-at-point for Clojure.
	2xe		2xe will eval current defun instead.
Debug - step in	xj		• 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.
Execute Tests: run ert	xT		
Buffer/Region operations			
Store current buffer and region for further operation	xB	(lispy-store-region-and-buffer)	Store current buffer and 'lispy--bounds-dwim'.

Description	Key	Function	Note
Ediff regions	`B`	(**special-lispy-ediff-regions**)	Comparable to 'ediff-regions-linewise'. • First region and buffer come from 'lispy-store-region-and-buffer' • Second region and buffer are the current ones.
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 🚧			
	`xC`	(**lispy-cleanup**)	
	• **`xC-h`** • **`x?`**	(**lispy-x-more-verbosity**)	
	`xn`	(**lispy-cd**)	Change the current REPL working directory. Not implemented for Emacs Lisp.
	`xp`	(**lispy-set-python-process**)	