↖ Topic Index

Smartparens

Description	Key	Function	Note
Smartparens ○ Help & configuration • Help on smartparens • Toggle smartparens mode • Mark block, Narrow block ○ Navigate ○ Move over space • Copy & clone • Transform ○ Kill ○ Kill/splice • Delete/kill region • Delete char ○ Delete/kill symbol ○ Indentation • Highlight	Smartparens is a minor mode "*that deals with parens pairs and tries to be smart about it*" as per its author. • It has features comparable to **𝔇𝔦𝔯- Lispy** but supports multiple programming languages and text formats. 📦 The **smartparens** external package is activated by PEL downloads via the **pel-use-smartparens** user-option. • With PEL: • Use the **<f11> (** they prefix to get information about **smartparens** or to activate it in the current buffer or globally. • Once **smartparens** mode is active use the **<M-f7>** key prefix to quickly access **smartparens** key bindings.		
Last updated on:	2025-05-12	See **Smartparens manual**	
Open this PDF file. See also: **𝕀 Help/Info**	**<f11> (<f1>**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **𝕀𝔸 Smartparens** local PDF. If the prefix argument (like **C-u** or **M--**) is used, then it opens the remote GitHub hosted raw PDF instead. If the **pel-flip-help-pdf-arg** user-option is set it's the other way around.
𝕀 Customize PEL smart parens control	**<f11> (<f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL text insertion support: rainbow-delimiters, **smartparens**. • If OTHER-WINDOW is non-nil (use **C-u**), display in other window.
𝕀 Customize Emacs smart parens control	**<f11> (<f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs text insertion support: rainbow-delimiters, **smartparens.** If OTHER-WINDOW is non-nil (use **C-u**), display in other window.
Smartparens Mode • **Smartparens manual** See also: **𝕀 Inserting Text**	Simplify insertion of matching pairs with the **smartparens** minor mode. PEL binds a set of keys, described below, to toggle activation of that mode. 📦 This uses the **smartparens** external package. PEL activates it when **pel-use-smartparens** is set to **t**. • Smartparents enhances the behaviour of certain keys, namely those that are part of any pair or tag. • Mode line lighter: smartparens-mode: **SP** smartparens-strict-mode: **SP/s**		
Help on smartparens	**<f11> (?**	(**sp-cheat-sheet** &optional ARG)	Generate a cheat sheet of all the smartparens interactive functions. Shows inside Emacs buffer. • Print only the short documentation and examples. • With non-nil prefix argument ARG (**C-u**), show the **full documentation** for each function. • You can follow the links to the function or variable help page. • To get back to the full list, use **M-x help-go-back**. • You can use 'beginning-of-defun' and 'end-of-defun' to jump to the previous/next entry. • Examples are fontified using the 'font-lock-string-face' for better orientation.
Describe user system	**<f11> (M-?**	(**sp-describe-system** STARTERKIT)	Describe user's system. Prompt for starter kit: Evil, Spacemac, Vanilla. • The output of this function can be used in bug reports.
Print smartparens info	**<f11> (i**	(**pel-smartparens-info**)	Print smartparens setup info in *pel-smartparens-info* buffer. • The information printed includes values of several variables organized by sections. Sections and variable names are clickable buttons. Sections lead to relevant smartparens pages, while variables buttons open help for the variable.
Toggle smartparens mode	**<f11> ((**	(**smartparens-mode** &optional ARG)	Toggle smartparens mode.
Toggle smartparens-strict mode	**<f11> ( )**	(**smartparens-strict-mode** &optional ARG)	Toggle the strict smartparens mode. • When strict mode is active, 'delete-char', 'kill-word' and their backward variants will skip over the pair delimiters in order to keep the structure always valid (the same way as 'paredit-mode' does). This is accomplished by remapping them to 'sp-delete-char' and 'sp-kill-word'. There is also function 'sp-kill-symbol' that deletes symbols instead of words, otherwise working exactly the same (it is not bound to any key by default). • When strict mode is active, this is indicated with "/s" after the smartparens indicator in the mode list
Toggle smartparens mode	**<f11> (M-(**	(**smartparens-global-mode** &optional ARG)	Toggle Smartparens mode in all buffers. • With prefix ARG, enable Smartparens-Global mode if ARG is positive; otherwise, disable it. • Smartparens mode is enabled in all buffers except this identified in **sp-ignore-mode-list**.
Toggle smartparens-strict mode	**<f11> (M-)**	(**smartparens-global-strict-mode** &optional ARG)	Toggle Smartparens-Strict mode in all buffers. • With prefix ARG, enable Smartparens-Global-Strict mode if ARG is positive; otherwise, disable it. • Smartparens-Strict mode is enabled in all buffers where 'turn-on-smartparens-strict-mode' would do it.
Mark	See **𝕀 Marking** for more information on marking.		
Mark current block	**M-<f7> . .**	(**sp-mark-sexp** &optional ARG ALLOW-EXTEND)	Set mark ARG balanced expressions from point. • The place mark goes is the same as **M-x sp-forward-sexp** would move to. • Interactively, if this command is repeated or (in Transient Mark mode) if the mark is active, it marks the next ARG sexps after the ones already marked. • This command assumes point is not in a string or comment.
Mark next	**M-<f7> . n**	(**sp-select-next-thing** &optional ARG POINT)	Set active region over next thing as recognized by 'sp-get-thing'. • If ARG is number 0 (zero), select all the things inside the current expression. • If ARG is positive N, select N expressions forward. If ARG is negative -N, select N expressions backward. • If ARG is a raw prefix **C-u** select all the things up until the end of current expression. • If ARG is a raw prefix **C-u C-u** select the current expression (as if doing 'sp-backward-up-sexp' followed by 'sp-select-next-thing'). • If POINT is non-nil, it's assumed it's a point inside the buffer from which the selection extends, either forward or backward, depending on ARG's value. • If the currently active region contains a balanced expression, following invocation of 'sp-select-next-thing' will select the inside of this expression. Therefore calling this function twice with no active region will select the inside of the next expression. • If the point is right in front of the expression any potential prefix is ignored. For example, '|(foo) would only select (foo) and not include ' in the selection. If you wish to also select the prefix, you have to move the point backwards. • With '**sp-navigate-consider-symbols**' symbols and strings are also considered balanced expressions.
Mark next and exchange	**M-<f7> . N**	(**sp-select-next-thing-exchange** &optional ARG POINT)	Just like 'sp-select-next-thing' but run 'exchange-point-and-mark' afterwards.
Mark previous	**M-<f7> . p**	(**sp-select-previous-thing** &optional ARG POINT)	Set active region over ARG previous things as recognized by 'sp-get-thing'. • If ARG is negative -N, select that many expressions forward. • With 'sp-navigate-consider-symbols' symbols and strings are also considered balanced expressions.
Mark previous and exchange	**M-<f7> . P**	(**sp-select-previous-thing-exchange** &optional ARG POINT	Just like 'sp-select-previous-thing' but run 'exchange-point-and-mark' afterwards.
Narrowing	See **𝕀 Narrowing** for more information on narrowing.		
Narrow to sexp	**M-<f7> M-n**	(**sp-narrow-to-sexp** ARG)	Make text outside current balanced expression invisible. • A numeric arg specifies to move up by that many enclosing expressions. • See also 'narrow-to-region' and 'narrow-to-defun'.

Description	Key	Function	Note
Navigation	PEL also provides 10 bindings using the **C-M-** modifiers combination for the main navigation commands. 6 of them correspond to the **recommended navigation key bindings**, the other 4 differ to allow valid bindings when Emacs runs in terminal mode and better reflect standard bindings. The changes are: • `sp-backward-down-sexp (&optional arg)` ;; ~~C-M-a~~ --> C-M-z • `sp-up-sexp (&optional arg)` ;; ~~C-M-e~~ --> C-M-] • `sp-beginning-of-sexp (&optional arg)` ;; ~~C-S-d~~ --> C-M-a • `sp-end-of-sexp (&optional arg)` ;; ~~C-S-a~~ --> C-M-e • The smartparens package does not bind any key by default. However, the recommended bindings are shown in **blue** as if they were. PEL binds them. • For bindings that differ from the recommended ones, the recommended binding is shown in ~~crossed out red~~. PEL doe not activate these bindings.		

To end of next element/ block
• **forward**
 	
Behaves as lispy j when point after end parens
	• **M-<f7> f**
	• **C-M-f**
		(**sp-forward-sexp** &optional ARG)	Move forward across one balanced expression. With ARG, do it that many times.
			• A **negative argument** N means move backward across N balanced expressions.
			• If there is no forward expression, jump out of the current one (effectively doing 'sp-up-sexp').
			• With 'sp-navigate-consider-symbols' symbols and strings are also considered balanced expressions.
			`|(foo bar baz) -> (foo bar baz)|`
			`(|foo bar baz) -> (foo| bar baz)`
			`(|foo bar baz) -> (foo bar| baz) ;; with ARG=2`
			`(foo (bar baz|)) -> (foo (bar baz)|)`

To beginning of previous element/block
• **backward**
	• **M-<f7> b**
	• **C-M-b**
		(**sp-backward-sexp** &optional ARG)	Move point backward to beginning of previous block element. With ARG, do it that many times.
			• A **negative argument** N means move forward across N balanced expressions.
			• If there is no previous expression, jump out of the current one (effectively doing 'sp-backward-up-sexp'): moves out of block, then previous block.
			• With 'sp-navigate-consider-symbols' symbols and strings are also considered balanced expressions.
			`(foo bar baz)| -> |(foo bar baz)`
			`(foo| bar baz) -> (|foo bar baz)`
			`(foo bar| baz) -> (|foo bar baz) ;; with ARG=2`
			`((|foo bar) baz) -> (|(foo bar) baz)`

To beginning of next element/block
• **forward/backward**
	• **M-<f7> n**
	• **C-M-n**
		(**sp-next-sexp** &optional ARG)	Move forward to beginning of next block element. At end of block move to beginning of outer block. With ARG, do it that many times. A **negative argument** N means move to the beginning of N-th previous balanced expression.
			• If there is no next expression at current level, jump one level up (effectively doing 'sp-backward-up-sexp').
			• If 'sp-navigate-interactive-always-progress-point' is non-nil, and this is called interactively, the point will move to the first expression in forward direction where it will end up greater than the current location.
			• With '**sp-navigate-consider-symbols**' symbols and strings are also considered balanced expressions.
			`((foo) |bar (baz quux)) -> ((foo) bar |(baz quux))`
			`((foo) bar |(baz quux)) -> |((foo) bar (baz quux))`
			• With non-nil '**sp-navigate-interactive-always-progress-point**':
			`(f|oo bar) -> (foo |bar)`
			`((fo|o) (bar)) -> ((foo) |(bar))`

To end of previous element
• **backward**
	• **M-<f7> p**
	• **C-M-p**
		(**sp-previous-sexp** &optional ARG)	Move backward to end of previous block element.
			• With ARG, do it that many times. If there is no next expression at current level, jump one level up (effectively doing 'sp-up-sexp').
			• A **negative argument** N means move to the end of N-th following balanced expression.
			• With 'sp-navigate-consider-symbols' symbols and strings are also considered balanced expressions.
			• If '**sp-navigate-interactive-always-progress-point**' is non-nil, and this is called interactively, the point will move to the first expression in backward direction where it will end up less than the current location.
			`((foo) bar| (baz quux)) -> ((foo)| bar (baz quux))`
			`((foo)| bar (baz quux)) -> ((foo) bar (baz quux))|`
			• And if '**sp-navigate-interactive-always-progress-point'** is non-nil:
			`(foo b|ar baz) -> (foo| bar baz)`
			`(foo (b|ar baz)) -> (foo| (bar baz))`

• **forward**	**M-<f7> F**	(**sp-forward-parallel-sexp** &optional ARG)	Move forward across one balanced expressions at the same depth. • If calling 'sp-forward-sexp' at point would result in raising a level up, loop back to the first expression at current level, that is the first child of the enclosing sexp as defined by 'sp-get-enclosing-sexp'.
• **backward**	**M-<f7> B**	(**sp-backward-parallel-sexp** &optional ARG)	Move backward across one balanced expressions at the same depth. • If calling 'sp-backward-sexp' at point would result in raising a level up, loop back to the last expression at current level, that is the last child of the enclosing sexp as defined by 'sp-get-enclosing-sexp'.

Into block forward
• **forward**
	• **M-<f7> d**
	• **C-M-d**
		(**sp-down-sexp** &optional ARG)	Move forward to the beginning of inner element of a block. With ARG, do this that many times.
			• A **negative argument** N means move backward but still go down a level.
			• If ARG is raw prefix argument **C-u**, descend forward as much as possible. If ARG is raw prefix argument **C-u C-u**, jump to the beginning of current list.
			• If the point is inside sexp and there is no down expression to descend to, jump to the beginning of current one. If moving backwards, jump to end of current one.
			`|foo (bar (baz quux)) -> foo (|bar (baz quux))`
			`|foo (bar (baz quux)) -> foo (bar (|baz quux)) ;; with ARG=2`
			`|foo (bar (baz (quux) blab)) -> foo (bar (baz (|quux) blab)) ;; with C-u`
			`(foo (bar baz) |quux) -> (|foo (bar baz) quux)`
			`(blab foo |(bar baz) quux) -> (|blab foo (bar baz) quux) ;; with C-u C-u`

Into block backward
• **backward**
	• **M-<f7> z**
	• **C-M-z**
	~~**C-M-a**~~
		(**sp-backward-down-sexp** &optional ARG)	Move backward down one level to end of block element.
			• With ARG, do this that many times.
			• A **negative argument** N means move forward but still go down a level. If ARG is raw prefix argument **C-u**, descend backward as much as possible. If ARG is raw prefix argument **C-u C-u**, jump to the end of current list. If the point is inside sexp and there is no down expression to descend to, jump to the end of current one. If moving forward, jump to beginning of current one.
			`foo (bar (baz quux))| -> foo (bar (baz quux)|)`
			`(bar (baz quux)) foo| -> (bar (baz quux|)) foo ;; with ARG=2`
			`foo (bar (baz (quux) blab))| -> foo (bar (baz (quux|) blab)) ;; with C-u`
			`(foo| (bar baz) quux) -> (foo (bar baz) quux|)`
			`(foo (bar baz) |quux blab) -> (foo (bar baz) quux blab|) ;; with C-u C-u`

To beginning of block
• **backward/forward**
	• **M-<f7> a**
	• **C-M-a**
	~~**C-S-d**~~
		(**sp-beginning-of-sexp** &optional ARG)	Jump to beginning of the sexp the point is in. The beginning is the point after the opening delimiter.
			• With no argument, this is the same as **C-u C-u 'sp-down-sexp'**
			• With ARG positive **N > 1**, move forward out of the current expression, move N-2 expressions forward and move down one level into next expression.
			• With ARG negative **N < 1**, move backward out of the current expression, move N-1 expressions backward and move down one level into next expression.
			• With ARG raw prefix argument **C-u** move out of the current expressions and then to the beginning of enclosing expression.
			`(foo (bar baz) quux| (blab glob)) -> (|foo (bar baz) quux (blab glob))`
			`(foo (bar baz|) quux (blab glob)) -> (foo (|bar baz) quux (blab glob))`
			`(|foo) (bar) (baz quux) -> (foo) (bar) (|baz quux) ;; with ARG=3`
			`(foo bar) (baz) (quux|) -> (|foo bar) (baz) (quux) ;; with ARG=-3`
			`((foo bar) (baz |quux) blab) -> (|(foo bar) (baz quux) blab) ;; with C-u`

To end of current block
• **forward**
	• **M-<f7> e**
	• **C-M-e**
	~~**C-S-a**~~
		(**sp-end-of-sexp** &optional ARG)	Jump to end of the current block.
			• With no argument, this is the same as calling **C-u C-u 'sp-backward-down-sexp'**.
			• With ARG positive **N > 1**, move forward out of the current expression, move N-1 expressions forward and move down backward one level into previous expression. With ARG negative **N < 1**, move backward out of the current expression, move N-2 expressions backward and move down backward one level into previous expression.
			• With ARG raw prefix argument **C-u** move out of the current expressions and then to the end of enclosing expression.
			`(foo |(bar baz) quux (blab glob)) -> (foo (bar baz) quux (blab glob)|)`
			`(foo (|bar baz) quux (blab glob)) -> (foo (bar baz|) quux (blab glob))`
			`(|foo) (bar) (baz quux) -> (foo) (bar) (baz quux|) ;; with ARG=3`
			`(foo bar) (baz) (quux|) -> (foo bar|) (baz) (quux) ;; with ARG=-3`
			`((foo |bar) (baz quux) blab) -> ((foo bar) (baz quux) blab|) ;; with C-u`

Description	Key	Function	Note
To beginning of next block • **forward**	`M-<f7> j`	(**sp-beginning-of-next-sexp** &optional ARG)	`(f|oo) (bar) (baz) -> (foo) (|bar) (baz)` `(f|oo) (bar) (baz) -> (foo) (bar) (|baz)  ;; with ARG=2`
To beginning of previous block • **backward**	`M-<f7> k`	(**sp-beginning-of-previous-sexp** &optional ARG)	`(foo) (b|ar) (baz) -> (|foo) (bar) (baz)` `(foo) (bar) (b|az) -> (|foo) (bar) (baz)  ;; with ARG=2`
To end of next block • **forward**	`M-<f7> N`	(**sp-end-of-next-sexp** &optional ARG)	`(f|oo) (bar) (baz) -> (foo) (bar|) (baz)` `(f|oo) (bar) (baz) -> (foo) (bar) (baz|)  ;; with ARG=2`
To end of previous block • **backward**	`M-<f7> K`	(**sp-end-of-previous-sexp** &optional ARG)	`(foo) (b|ar) (baz) -> (foo|) (bar) (baz)` `(foo) (bar) (b|az) -> (foo|) (bar) (baz)  ;; with ARG=2`
Out block forward • **forward**	• `M-<f7> ]` • `C-M-]`	(**sp-up-sexp** &optional ARG INTERACTIVE)	Move forward out of one level of parentheses. With ARG, do this that many times. • A **negative argument** means move backward but still to a less deep spot.

	• The argument INTERACTIVE is for internal use only.
	• If called interactively and '**sp-navigate-reindent-after-up**' is enabled for current major-mode, remove the whitespace between end of the expression and the last "thing" inside the expression.
	• This behaviour can be suppressed for syntactic string sexps by setting '**sp-navigate-reindent-after-up-in-string**' to nil.
	• If '**sp-navigate-close-if-unbalanced**' is non-nil, close the unbalanced expressions automatically.
	`(foo |(bar baz) quux blab) -> (foo (bar baz) quux blab)|`
	`(foo (bar |baz) quux blab) -> (foo (bar baz) quux blab)| ;; with ARG=2`
	• re-indent the expression
	`(foo bar |baz -> (foo bar baz)|`
	`_  )`
	• close unbalanced expression
	`(foo  |(bar baz) -> (foo)| (bar baz)`		

Out block backward • **backward**	• `M-<f7> u` • `C-M-u`	(**sp-backward-up-sexp** &optional ARG INTERACTIVE)	Move backward out of one level of parentheses. With ARG, do this that many times. • A **negative argument** means move forward but still to a less deep spot. • The argument INTERACTIVE is for internal use only.

	• If called interactively and '**sp-navigate-reindent-after-up**' is enabled for current major-mode, remove the whitespace between beginning of the expression and the first "thing" inside the expression.
	`(foo (bar baz) quux| blab) -> |(foo (bar baz) quux blab)`
	`(foo (bar |baz) quux blab) -> |(foo (bar baz) quux blab) ;; with ARG=2`
	`(  -> |(foo bar baz)`
	`_  foo |bar baz)`		

Move over space			
To beginning of next symbol/ block	`M-<f7> SPC n`	(**sp-skip-forward-to-symbol** &optional STOP-AT-STRING STOP-AFTER-STRING STOP-INSIDE-STRING)	`foo|  bar  -> foo  |bar` `foo|  [bar baz] -> foo  |[bar baz]`

To end of next symbol or block	`M-<f7> SPC m`	(**sp-forward-symbol** &optional ARG)	`|foo bar baz  -> foo| bar baz`
			`|foo (bar (baz))  -> foo (bar| (baz))  ;; 2` 🚧 **check this**
			`|foo (bar (baz) quux) -> foo (bar (baz) quux|) ;; 4`

To beginning of previous	`M-<f7> SPC p`	(**sp-backward-symbol** &optional ARG)	`foo bar| baz  -> foo |bar baz`
			`((foo bar) baz)|  -> ((foo |bar) baz) ;; 2`
			`(quux ((foo) bar) baz)| -> (|quux ((foo) bar) baz) ;; 4`

Skip forward past whitespace	`M-<f7> SPC .`	(**sp-forward-whitespace** &optional ARG)	Skip forward past the whitespace characters. With non-nil ARG return number of characters skipped.
Skip backward past whitespace	`M-<f7> SPC ,`	(**sp-backward-whitespace** &optional ARG)	Skip backward past the whitespace characters. With non-nil ARG return number of characters skipped.
Copy and Clone	With PEL, the commands that are marked with 🍀👁 display the copied string when **pel-show-copy-cut-text** is **t**. Toggle this display with `<f11> M-=`		
Copy current & forward block(s) 🍀👁	`M-<f7> =` `C-M-w`	(**sp-copy-sexp** &optional ARG)	Copy the following ARG expressions to the kill-ring. This is exactly like calling 'sp-kill-sexp' with second argument t. All the special prefix arguments work the same way.
Copy previous block(s) 🍀👁	`M-<f7> M-=`	(**sp-backward-copy-sexp** &optional ARG)	Copy the previous ARG expressions to the kill-ring. This is exactly like calling 'sp-backward-kill-sexp' with second argument t. All the special prefix arguments work the same way.
clone current block	`M-<f7> c`	(**sp-clone-sexp**)	Clone sexp after or around point. • If the form immediately after point is a sexp, clone it below the current one and put the point in front of it. • Otherwise get the enclosing sexp and clone it below the current enclosing sexp.
Transform			

Transpose block elements	`M-<f7> t`	(**sp-transpose-sexp** &optional ARG)	`foo |bar baz  -> bar foo| baz`
			`foo |bar baz  -> bar baz foo| ;; 2`
			`(foo) |(bar baz) -> (bar baz) (foo)|`
			`(foo bar)  ->  (baz quux)  ;; keeps the formatting`
			`_  |(baz quux)  |(foo bar)`
			`foo bar baz|  -> foo baz| bar ;; -1`

Transpose block elements 🚧	`M-<f7> T`	(**sp-transpose-hybrid-sexp** &optional ARG)	`foo bar  baz (quux`
			`|baz (quux  ->  quack)`
			`quack)  foo bar\n|`
			`[(foo) (bar) -> [(baz)`
			`|(baz)]  (foo) (bar)|]`
			`foo bar baz  -> quux flux`
			`|quux flux  foo bar baz\n|`

Push current block after next
Like lispy s
	`M-<f7> s`	(**sp-push-hybrid-sexp**)	`|x = big_function_call(a,  |(a,`
			`b)  b) = read_user_input()`
			`->`
			`(a,  x = big_function_call(a,`
			`b) = read_user_input()  b)`

Transform - slurp			

Enclose next outside element into current block	`M-<f7> >`	(**sp-forward-slurp-sexp** &optional ARG)	`(foo |bar) baz  -> (foo |bar baz)`
			`[(foo |bar)] baz  -> [(foo |bar) baz]`
			`[(foo |bar) baz]  -> [(foo |bar baz)]`
			`((|foo) bar baz quux) -> ((|foo bar baz quux)) ;; with C-u`
			`"foo| bar" "baz quux" -> "foo| bar baz quux"`

Enclose next outside element into current block	`M-<f7> M->`	(**sp-slurp-hybrid-sexp**)	Add hybrid sexp following the current list in it by moving the closing delimiter. • This is conceptually similar to 'sp-forward-slurp-sexp' but works better in "line-based" languages like C or Java. • Because the structure is much looser in these languages, this command currently does not support all the prefix argument triggers that 'sp-forward-slurp-sexp' does.

Description	Key	Function	Note

Enclose previous outside element(s) into next block	`M-<f7> <`	(**sp-backward-slurp-sexp** &optional ARG)	`foo (bar| baz)  -> (foo bar| baz)`
			`foo [(bar| baz)]  -> [foo (bar| baz)]`
			`[foo (bar| baz)]  -> [(foo bar| baz)]`
			`(foo bar baz (|quux)) -> ((foo bar baz |quux)) ;; with C-u`
			`"foo bar" "baz |quux" -> "foo bar baz |quux"`

Enclose next outside element(s) into previous block	`M-<f7> }`	(**sp-add-to-previous-sexp** &optional ARG)	`(foo bar) |baz quux  -> (foo bar |baz) quux`
			`(foo bar) |baz quux  -> (foo bar |baz quux) ;; 2`
			`(blab (foo bar) |baz quux) -> (blab (foo bar |baz quux)) ;; C-u`
			`(foo bar) (baz |quux)  -> (foo bar (baz |quux)) ;; C-u C-u`

Enclose previous outside element(s) into next block	`M-<f7> {`	(**sp-add-to-next-sexp** &optional ARG)	`foo bar| (baz quux)  -> foo (bar| baz quux)`
			`foo bar| (baz quux)  -> (foo bar| baz quux) ;; 2`
			`(foo bar |(bar quux) blab) -> ((foo bar |bar quux) blab) ;; C-u`
			`(foo |bar) (baz quux)  -> ((foo |bar) baz quux) ;; C-u C-u`

Transform - barf			

Eject next element(s) out of current block	`M-<f7> /`	(**sp-forward-barf-sexp** &optional ARG)	`(foo bar| baz)  -> (foo bar|) baz  ;; nil (defaults to 1)`
			`(foo| [bar baz]) -> (foo|) [bar baz] ;; 1`
			`(1 2 3| 4 5 6)  -> (1 2 3|) 4 5 6  ;; C-u (or numeric prefix 3)`
			`(foo bar| baz)  -> foo (bar| baz)  ;; -1`

Eject previous element(s) out of current block	`M-<f7> M-/`	(**sp-backward-barf-sexp** &optional ARG)	`(foo bar| baz)  -> foo (bar| baz)`
			`([foo bar] |baz) -> [foo bar] (|baz)`
			`(1 2 3 |4 5 6)  -> 1 2 3 (|4 5 6) ;; C-u (or 3)`

Re-wrap block			

Re-wrap current block	`M-<f7> r`	(**sp-rewrap-sexp** PAIR &optional KEEP-OLD)	Re-wrap current block using another block character.
			`(foo |bar baz) -> [foo |bar baz]  ;; [`
			`(foo |bar baz) -> [(foo |bar baz)] ;; C-u [`

Swap wrapping characters between current block and parent block	`M-<f7> w`	(**sp-swap-enclosing-sexp** &optional ARG)	Swap the wrapping of blocks
			`(foo [|bar] baz)  -> [foo (|bar) baz] ;; 1`
			`(foo {bar [|baz] quux} quack) -> [foo {bar (|baz) quux} quack] ;; 2`

Un-wrap block			

Extract all elements from current/next block	`M-<f7> U`	(**sp-unwrap-sexp** &optional ARG)	Un-wrap current or next block.
			`|(foo bar baz)  -> |foo bar baz`
			`(foo bar| baz)  -> foo bar| baz`
			`|(foo) (bar) (baz) -> |(foo) bar (baz) ;; 2`

Extract all elements from previous block	`M-<f7> W`	(**sp-backward-unwrap-sexp** &optional ARG)	Un-wrap previous block.
			`(foo bar baz)|  -> foo bar baz|`
			`(foo bar)| (baz)  -> foo bar| (baz)`
			`(foo) (bar) (baz)| -> foo (bar) (baz) ;; 3`

Transformation			

Convolute	`M-<f7> C`	(**sp-convolute-sexp** &optional ARG)	Exchange the order of application of the two closest outer forms.
			In the following, we want to move the 'while' before the 'let'.
			`_ (let ((stuff 1)  (while (we-are-good)`
			`_  (other 2))  (let ((stuff 1)`
			`_  (while (we-are-good)  ->  (other 2))`
			`_  |(do-thing 1)  |(do-thing 1)`
			`_  (do-thing 2)  (do-thing 2)`
			`_  (do-thing 3)))  (do-thing 3)))`
			`(forward-char (sp-get env |:op-l)) -> (sp-get env (forward-char |:op-l))`

Absorb previous element into current block	`M-<f7> A`	(**sp-absorb-sexp** &optional ARG)	Absorb the outer item into the current block and move point before the absorbed item(s).
			`_ (do-stuff 1)  (save-excursion`
			`_ (save-excursion  ->  |(do-stuff 1)`
			`_  |(do-stuff 2))  (do-stuff 2))`
			`foo bar (concat |baz quux) -> (concat |foo bar baz quux) ;; 2`

Expel previous items from block	`M-<f7> E`	(**sp-emit-sexp** &optional ARG)	Expel previous items from current block out of the block.
			`_ (save-excursion  _(do-stuff 1)`
			`_  (do-stuff 1)  (do-stuff 2)`
			`_  (do-stuff 2)  ->  (save-excursion`
			`_  |(do-stuff 3))  |(do-stuff 3))`
			`_ (while not-done-yet  (execute-only-once)`
			`_  (execute-only-once) ->  (while not-done-yet  ;; arg = 2`
			`_ |(execute-in-loop)) |(execute-in-loop))

	`M-<f7>`	(**sp-extract-before-sexp** &optional ARG	Move the expression after point before the enclosing balanced expression. • The point moves with the extracted expression. • With ARG positive N, extract N expressions after point. • With ARG negative -N, extract N expressions before point. • With ARG being raw prefix argument C-u, extract all the expressions up until the end of enclosing list. • If the raw prefix is negative, this behaves as C-u 'sp-backward-barf-sexp'.

Description	Key	Function	Note
	M-<f7>	(**sp-extract-after-sexp** &optional ARG)	Move the expression after point after the enclosing balanced expression. • The point moves with the extracted expression. • With ARG positive N, extract N expressions after point. • With ARG negative -N, extract N expressions before point. • With ARG being raw prefix argument C-u, extract all the expressions up until the end of enclosing list. • With ARG being negative raw prefix argument - C-u, extract all the expressions up until the start of enclosing list.

Split block	M-<f7> |	(**sp-split-sexp** ARG)	`(foo bar |baz quux)  -> (foo bar) |(baz quux)`
			`"foo bar |baz quux"  -> "foo bar" |"baz quux"`
			`([foo |bar baz] quux) -> ([foo] |[bar baz] quux)`
			`(foo bar| baz quux) -> (foo) (bar|) (baz) (quux) ;; C-u`

Join blocks	M-<f7> J	(**sp-join-sexp** &optional ARG)	`(foo bar) |(baz)  -> (foo bar |baz)`
			`(foo) |(bar) (baz)  -> (foo |bar baz) ;; 2`
			`[foo] [bar] |[baz]  -> [foo bar |baz] ;; -2`
			`(foo bar (baz)| (quux) (blob bluq)) -> (foo bar (baz| quux blob bluq)) ;; C-u`

Kill Commands	The table uses the ⌦ and ⌫ symbols to represent these 2 keys: ⌦ := "forward delete" := **<deletechar>** := **Fn** ⌫ on keyboards. ⌫ := "backward delete" := **<backspace>** Often labelled "delete" on keyboards. ⚠ **C-⌦** and **C-⌫** are not accessible in terminal mode. With PEL, the commands that are marked with 🍀👁 display the killed string when **pel-show-copy-cut-text** is **t**. Toggle this display with **<f11> M-=**		
• **kill block elements**	The following commands kill the element(s) of a block.		

Kill content of next block 🍀👁	• M-<f7> ⌦
	• M-<f7> - n
		(**sp-change-inner**)	Change the inside of the next (or current) expression.
			• First, kill the inside of the next (or current) balanced expression, then move point just after the opening delimiter.
			`(f|oo  [bar] baz) -> (foo [|] baz)`
			`([f|oo] [bar] baz) -> ([|] [bar] baz)`
			`{|'foo': 'bar'}  -> {'|': 'bar'}`

Kill content of current block 🍀👁	M-<f7> - .	(**sp-change-enclosing**)	Change the inside of the enclosing expression.
			• Whitespace on both sides of the inner items is preserved if it contains newlines.
			• Invoking this function on a blank sexp will wipe out remaining whitespace (see 'sp-point-in-blank-sexp').
			• Move the point to the beginning of the original content.
			`(f|oo [bar] baz) -> (|)`
			`{'f|oo': 'bar'}  -> {'|': 'bar'}`

Kill block elements forward 🍀👁	M-<f7> -]
	C-M-k
		(**sp-kill-sexp** &optional ARG DONT-KILL)	Kill the balanced expression following point.
			• If point is inside an expression and there is no following expression, kill the topmost enclosing expression.
			• With ARG being positive number N, repeat that many times.
			• With ARG being Negative number -N, repeat that many times in backward direction.
			• With ARG being raw prefix C-u, kill all the expressions from point up until the end of current list. With raw prefix C-_ C-u, kill all the expressions from beginning of current list up until
			• point. If point is inside a symbol, this is also killed. If there is no expression after/before the point, just delete the whitespace up until the closing/opening delimiter.
			• With ARG being raw prefix C-u C-u, kill current list (the list point is inside).
			• With ARG numeric prefix 0 (zero) kill the insides of the current list, that is everything from after the opening delimiter to before the closing delimiter.
			• If ARG is nil, default to 1 (kill single expression forward)
			• If second optional argument DONT-KILL is non-nil, save the to be killed region in the kill ring, but do not kill the region from buffer.
			• With 'sp-navigate-consider-symbols', symbols and strings are also considered balanced expressions.
			`(foo |(abc) bar)  -> (foo | bar) ;; nil, defaults to 1`
			`(foo (bar) | baz) -> (foo |) ;; 2`
			`(foo |(bar) baz)  -> (foo |) ;; C-u C-u`
			`(1 |2 3 4 5 6)  -> (1|) ;; C-u`
			`(1 |2 3 4 5 6)  -> (1 | 5 6) ;; 3`
			`(1 2 3 4 5| 6)  -> (1 2 3 | 6) ;; -2`
			`(1 2 3 4| 5 6)  -> (|5 6) ;; - C-u`
			`(1 2 |  )  -> (1 2|) ;; C-u, kill useless whitespace`
			`(1 2 3 |4 5 6)  -> (|) ;; 0`

Kill block elements backward 🍀👁	M-<f7> - [	(**sp-backward-kill-sexp** &optional ARG DONT-KILL)	Kill the balanced expression preceding point.
			• This is exactly like calling 'sp-kill-sexp' with minus ARG.
			• In other words, the direction of all commands is reversed. For more information, see the documentation of 'sp-kill-sexp'.
			`(foo (abc)| bar)  -> (foo | bar)`
			`blab (foo (bar baz) quux)| -> blab |`
			`(1 2 3 |4 5 6)  -> (|4 5 6) ;; C-u`

Kill element after current 🍀👁	M-<f7> - }	(**sp-kill-hybrid-sexp** ARG)	Kill a line as if with 'kill-line', but respecting delimiters.
			• With ARG being raw prefix **C-u C-u**, kill the hybrid sexp the point is in (see 'sp-get-hybrid-sexp').
			• With ARG numeric prefix 0 (zero) just call 'kill-line'.
			• You can customize the behaviour of this command by toggling 'sp-hybrid-kill-excessive-whitespace'.
			`foo | bar baz  -> foo |  ;; nil`
			`foo (bar | baz) quux  -> foo (bar |) quux  ;; nil`
			`foo | bar (baz quux)  -> foo |  ;; nil`
			`foo "bar |baz quux" quack  -> foo "bar |" quack  ;; nil`
			`foo (bar baz) qu|ux (quack zaq) hoo  ->  foo | hoo  ;; C-u C-u`
			`foo | (bar baz)  -> foo | baz)  ;; C-0`

Description	Key	Function	Note
Kill whole line	`M-<f7> - l`	(**sp-kill-whole-line**)	Kill current line in sexp-aware manner. • First, go to the beginning of current line and then try to kill as much as possible on the current line but without breaking balance. • If there is a hanging sexp at the end of line the it is killed as well. • If there is a closing delimiter for a sexp "up" current sexp, the kill is not extended after it. For more details see 'sp-kill-hybrid-sexp'. `(progn (some |long sexp))  ->  (progn |)`
• **Kill/splice**			

Un-wrap current block, splicing its elements in enclosing block	`M-<f7> l l`	(**sp-splice-sexp** &optional ARG)	Un-wrap current block, splicing its content in enclosing block (if any).
			`(foo (bar| baz) quux) -> (foo bar| baz quux)`
			`(foo |(bar baz) quux) -> foo |(bar baz) quux`
			`(foo (bar| baz) quux) -> foo (bar| baz) quux ;; 2`

Kill block element(s) before point and splice remaining into outer block	`M-<f7> l [`
	`C-M-<backspace>`
		(**sp-splice-sexp-killing-backward** &optional ARG)	Note that to kill only the content and not the enclosing delimiters you can use `C-u M-x sp-backward-kill-sexp`.
			• See 'sp-backward-kill-sexp' for more information.
			`(foo (let ((x 5)) |(sqrt n)) bar)  -> (foo |(sqrt n) bar)`
			`_ (when ok|  |(perform-operation-1)`
			`_  (perform-operation-1)  ->  (perform-operation-2)`
			`_  (perform-operation-2))`
			`_ (save-excursion  -> |(awesome-stuff-happens) ;; 2`
			`_  (unless (test)`
			`_  |(awesome-stuff-happens)))`

Kill block element(s) forward and splice remaining into outer block	`M-<f7> l ]`
	`C-M-<delete>`
		(**sp-splice-sexp-killing-forward** &optional ARG)	Note that to kill only the content and not the enclosing delimiters you can use C-u M-x sp-kill-sexp.
			• See 'sp-kill-sexp' for more information.
			`(a (b c| d e) f) -> (a b c| f)`
			`(+ (x |y z) w)  -> (+ x| w)`

Kill around element	`M-<f7> l o`
	`C-S-<backspace>`
		(**sp-splice-sexp-killing-around** &optional ARG)	`(a b |(c d) e f)  -> |(c d)  ;; with arg = 1`
			`(a b |c d e f)  -> |c d  ;; with arg = 2`
			`(- (car x) |a 3)  -> (car x)|  ;; with arg = -1`
			`(foo (bar |baz) quux) -> |(bar baz) ;; with arg = C-u C-u`

• **Delete/Kill region**	The following commands delete marked regions as long as the deletion would not create unbalanced blocks. • These may be useful inside keyboard macros when deleting text in area where several balanced and nested blocks are present.		
Delete region	`M-<f7> DEL -`	(**sp-delete-region** BEG END)	Delete the text between point and mark, like 'delete-region'. • BEG and END are the bounds of region to be deleted. • If that text is unbalanced, signal an error instead. • With a prefix argument, skip the balance check.
Kill region	`M-<f7> - -`	(**sp-kill-region** BEG END)	Kill the text between point and mark, like 'kill-region'. • BEG and END are the bounds of region to be killed. • If that text is unbalanced, signal an error instead. • With a prefix argument, skip the balance check.
• **Delete char**	The following commands are long-winded key sequences that delete forward and backward without breaking blocks. • The forward and backward delete keys do the same when smartparens-mode is active.		

Delete char forward	`M-<f7> DEL n`	(**sp-delete-char** &optional ARG)	Delete a character forward or move forward over a delimiter.
			• If on an opening delimiter, move forward into balanced expression.
			• If on a closing delimiter, refuse to delete unless the balanced expression is empty, in which case delete the entire expression.
			• If the delimiter does not form a balanced expression, it will be deleted normally.
			• With a numeric prefix argument N > 0, delete N characters forward.
			• With a numeric prefix argument N < 0, delete N characters backward.
			• With a numeric prefix argument N = 0, simply delete a character forward, without regard for delimiter balancing.
			• If ARG is raw prefix argument C-u, delete characters forward until a closing delimiter whose deletion would break the proper pairing is hit.
			`(quu|x "zot") -> (quu| "zot")`
			`(quux |"zot") -> (quux "|zot") -> (quux "|ot")`
			`(foo (|) bar) -> (foo | bar)`
			`|(foo bar) -> (|foo bar)`

Delete char backward	`M-<f7> DEL p`	(**sp-backward-delete-char** &optional ARG)	Delete a character backward or move backward over a delimiter.
			• It has the same description as the above command but goes backward instead of forward.
			`("zot" q|uux) -> ("zot" |uux)`
			`("zot"| quux) -> ("zot|" quux) -> ("zo|" quux)`
			`(foo (|) bar) -> (foo | bar)`
			`(foo bar)| -> (foo bar|)`

• **Delete/Kill word**			
Delete word backward	`M-<f7> DEL v`	(**sp-backward-delete-word** &optional ARG)	(sp-backward-delete-word &optional ARG) • Delete a word backward, skipping over intervening delimiters. • Deleted word does not go to the clipboard or kill ring. • With ARG being positive number N, repeat that many times. • With ARG being Negative number -N, repeat that many times in backward direction.
Delete word forward	`M-<f7> DEL w`	(**sp-delete-word** &optional ARG)	Delete a word forward, skipping over intervening delimiters. • Deleted word does not go to the clipboard or kill ring. • With ARG being positive number N, repeat that many times. • With ARG being Negative number -N, repeat that many times in backward direction.
Kill word backward	`M-<f7> - v`	(**sp-backward-kill-word** &optional ARG)	Kill a word backward, skipping over intervening delimiters. • With ARG being positive number N, repeat that many times. • With ARG being Negative number -N, repeat that many times in backward direction. 🚧
Kill word forward	`M-<f7> - w`	(**sp-kill-word** &optional ARG)	Kill a word forward, skipping over intervening delimiters. • With ARG being positive number N, repeat that many times. • With ARG being Negative number -N, repeat that many times in backward direction.

Description	Key	Function	Note
• **Delete/Kill symbol**	See '**sp-backward-symbol**' and '**sp-forward-symbol**' for what constitutes a symbol for the backward and forward commands respectively.		
Delete symbol backward	`M-<f7> DEL a`	(**sp-backward-delete-symbol** &optional ARG WORD)	Delete a symbol backward, skipping over any intervening delimiters. • Deleted symbol does not go to the clipboard or kill ring. • With ARG being positive number N, repeat that many times. • With ARG being Negative number -N, repeat that many times in forward direction.
Delete symbol forward	`M-<f7> DEL s`	(**sp-delete-symbol** &optional ARG WORD)	Delete a symbol forward, skipping over any intervening delimiters. • Deleted symbol does not go to the clipboard or kill ring. • With ARG being positive number N, repeat that many times. • With ARG being Negative number -N, repeat that many times in backward direction.
Kill symbol backward	`M-<f7> - a`	(**sp-backward-kill-symbol** &optional ARG WORD)	Kill a symbol backward, skipping over any intervening delimiters. • With ARG being positive number N, repeat that many times. • With ARG being Negative number -N, repeat that many times in forward direction.
Kill symbol forward	`M-<f7> - s`	(**sp-kill-symbol** &optional ARG WORD)	Kill a symbol forward, skipping over any intervening delimiters. • With ARG being positive number N, repeat that many times. • With ARG being Negative number -N, repeat that many times in backward direction.
Indentation 🚧			
	`<f11> p <TAB>` `M-<f7> <TAB>`	(**sp-indent-adjust-sexp**)	Add the hybrid sexp at line into previous sexp. All forms between the two are also inserted. • Specifically, if the point is on empty line, move the closing delimiter there, so the next typed text will become the last item of the previous sexp. • This acts similarly to 'sp-add-to-previous-sexp' but with special handling of empty lines.
	`M-<f7> S-<TAB>`	(**sp-dedent-adjust-sexp**)	Remove the hybrid sexp at line from previous sexp. • All sibling forms after it are also removed (not deleted, just placed outside of the enclosing list). Specifically, if the point is on empty line followed by closing delimiter of enclosing list, move the closing delimiter after the last item in the list. • This acts similarly to 'sp-forward-barf-sexp' but with special handling of empty lines.
Re-indent current defun **??in non lisp??**		(**sp-indent-defun** &optional ARG)	Reindent the current defun. • If point is inside a string or comment, fill the current paragraph instead, and with ARG, justify as well. • Otherwise, reindent the current defun, and adjust the position of the point.
Validation 🚧			
		(**sp-region-ok-p** START END)	Test if region between START and END is balanced. • A balanced region is one where all opening delimiters are matched by closing delimiters. • This function does *not* check that the delimiters are correctly ordered, that is `[()]` is considered correct even though it is not logically properly balanced.
		(**sp-newline**)	Insert a newline and indent it. • This is like 'newline-and-indent', but it not only indents the line that the point is on but also the S-expression following the point, if there is one. • If in a string, just insert a literal newline. • If in a comment and if followed by invalid structure, call 'indent-new-comment-line' to keep the invalid structure in a comment.
		(**sp-comment**)	Insert the comment character and adjust hanging sexps such that it doesn't break structure.
		(**sp-wrap-round**)	Wrap following sexp in round parentheses.
		(**sp-wrap-square**)	Wrap following sexp in square brackets.
		(**sp-wrap-curly**)	Wrap following sexp in curly braces.
Highlight 🚧			
		(**sp-show-enclosing-pair**)	Highlight the enclosing pair around point.
		(**sp-highlight-current-sexp** ARG)	Highlight the expression returned by the next command, preserving point position.