Emacs support for Python 🚧

Description	Keystroke	Function	Note
Python Support	Python support is very basic, it is not yet fully implemented nor fully documented. This is on my todo list. 🎛️ Important aspects of Python source code management are customizable with PEL user option variables. **PEL customization for Python**: • Emacs customization group: **pel-pkg-for-python** (To edit change, use **<f12> <f2>** , see below). • **pel-python-tab-width**: The width of a tab used for c-mode files. Defaults to 4. • This concept differs from indentation: you can have an indentation of 3 and tab width of 8: **M-i** will move point to columns that are multiple of 8 **<tab>** will indent to a column that is a multiple of 3. PEL stores this value inside the **tab-width** variable for python-mode buffers. • The values for those user option variables can also be stored inside directory local files and even as file local variables. You can also modify them for each buffer and view their current settings using the commands listed in the following set of rows. See ∑ File/Directory Variables for more info. • None of the commands below change PEL default; they change the value for the current buffer only. ⌨️ PEL provides the following set of **mode-specific key prefixes**: • **<f11> SPC p** • **<f12>** • **<M-f12>** The first one is always available. The other two prefixes are only available in c-mode buffers. The **<M-f12>** prefix helps the typing flow when the next key is a Meta key. For simplification, the **<f11> SPC p** prefix is normally omitted in the table.		
Open this PDF file. See also: ∑ Help/Info	• **<f11> SPC p <f1>** • **<f12> <f1>**	(**pel-help-pdf**)	Open 𝔓𝔇𝔉 - **Python** local PDF file.
∑ Customize PEL Python support	• **<f11> SPC p <f2>** • **<f12> <f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL Python support: python, python-flymake. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
∑ Customize Emacs Python support	• **<f11> SPC p <f3>** • **<f12> <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs Python support: python, python-flymake. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Python shell			
Start Python shell See also: ∑ Shells	**<f11> x p**	(**run-python** &optional CMD DEDICATED SHOW)	Run an inferior Python process. • Argument CMD defaults to 'python-shell-calculate-command' return value. When called interactively with 'prefix-arg', it allows the user to edit such value and choose whether the interpreter should be DEDICATED for the current buffer. When numeric prefix arg is other than 0 or 4 do not SHOW. • For a given buffer and same values of DEDICATED, if a process is already running for it, it will do nothing. This means that if the current buffer is using a global process, the user is still able to switch it to use a dedicated one.
Highlighting blocks	The following commands can be used to activate or toggle useful modes to highlight blocks of (), {}, and []. • show-paren-mode, which highlights the parens that matches the one before or after point. • rainbow delimiters mode, where matching nested parens are highlighted with the same colour.		
Toggle show-paren mode on/off See also: ∑ Highlight	• **<f12> M-9** • **<M-f12> M-9** • **<f11> b h (**	(**show-paren-mode** &optional ARG)	Toggle visualization of matching parens (Show Paren mode). • With a prefix argument ARG, enable Show Paren mode if ARG is positive, and disable it otherwise. • Show Paren mode is a global minor mode. When enabled, any matching parenthesis is highlighted in 'show-paren-style' after 'show-paren-delay' seconds of Emacs idle time.
Enable/Disable coloured highlight of nested blocks (),{},[] See also: ∑ Highlight	• **<f12> M-r** • **<M-f12> M-r** • **<f11> b h R**	(**rainbow-delimiters-mode** &optional ARG)	Highlight nested parentheses, brackets, and braces with different colours according to their depth. • Customize the depth and colours with **M-x customize-group rainbow-delimiters** 📦 **Requires:** rainbow-delimiters.el 🔄 PEL activates this when the **pel-use-rainbow-delimiters** user option is set to **t**.
Comments			
Toggle display of comments in buffer or active region See also: ∑ Comments	**<f11> ; ;**	(**hide/show-comments-toggle** &optional START END)	Toggle hiding/showing of comments in the active region or whole buffer. • If the region is active then toggle in the region. Otherwise, in the whole buffer. 📦 This requires the **hide-comnt.el** package (see ∑ Comments). 🔄 PEL activates it when the **pel-use-hide-comnt** user option is **t**.
Rendering markup embedded in comments	The following commands are used to create images from specific markup code embedded inside Python source code comments. This can be useful when using these markup languages to describe UML diagrams or finite-state machines for example.		
Preview UML diagram from plantUML source in current plantUML region of commented source code See also: Ⓜ PlantUML	**<f12> u**	(**pel-render-commented-plantuml** PREFIX &optional POS)	Render the PlantUML markup embedded in current mode comment. • Use region if identified otherwise use PlantUML block at point. • Uses prefix (as PREFIX) to choose where to display it: • 4 (when prefixing the command with **C-u**) -> new window • 16 (when prefixing the command with **C-u C-u**) -> new frame. • else -> new buffer • This can be used inside buffer using **any** major mode, when PlantUML markup is embedded inside source code comment. ☝️ Use this in source code to describe your code architecture with PlantUML markup, then generate the UML rendering by moving point inside the PlantUML block and issuing this command. 📦 Requires the **plantuml-mode** external package, 🔄 activated by **pel-use-plantuml** user option being non-nil.
Preview diagram created from Graphviz DOT markup embedded in comments See also: • Ⓜ Graphviz Dot	**<f12> G**	(**pel-render-commented-graphviz-dot** &optional POS)	Render the Graphviz-Dot markup embedded in current mode comment. Search at POS if specified, otherwise search around point. Use region if identified otherwise use Graphviz-Dot block. ☝️ The graphviz DOT code must be located within a block delimited by the following special keywords (that are also in comments): • **@start-gdot** • **@end-gdot** ⚠️ The current implementation leaves the created image file in a temporary directory. You will probably want to move that file or delete it, otherwise the size of this directory will increase with each of these created files. The file names use the pel-gdot- prefix. 📦 Requires the **graphviz-dot-mode package** external package, 🔄 activated by **pel-use-graphviz-dot** user option set to **t**.

Emacs & Python — References

Document	Notes
Emacs - The Best Python Editor?	
emacs-for-python	
Python indentation	
Python code Indentation	

Document	Notes
Elpy - Emacs Python Development Environment	
Python shell prompts not detected @ Github	**Windows-related problem** description, and description of a fix (which I have implemented in my init.el). Fixing that does not solve everything under Windows, and there is another issue, listed in the following lines.
'python-shell-interpreter' doesn't seem to support readline	**Windows-related problem** description, stating that "native" completion does not work on Windows and that we should add "python" to the list of python-shell-completion-native-disabled-interpreters in emacs.
Elpy seems partially incompatible with Emacs 25's 'native completion' feature	**Windows-related problem** description: elpy-issue-887: describes that it cannot be fixed on Windows and that emacs 26 will disable the warning (using the method described above).
GNU bug report logs - #28580 **[w32] python.el: native completion setup failed**	**Windows-related problem** description. Same problem.
PTY - Pseudo terminal @ wikipedia	Description of the PTY/pseudoterminal concept.
pyreadline-ais	Note that by installing pyreadline-ais, the problem remains in emacs.
Python - elpy.el	
company-mode ; Modular in-buffer completion framework for Emacs	