

Pye: Small Python Text Editor

Looking for a code editor that would fit onto Pyboard (and now WiPy, ESP8266, ESP32, Teensy, W60X, Circuitpython and the pycom.io devices), I made my way through the Micropython forum and found **pfalcon's** (Paul) Python editor code, which I took and ported it to PyBoard. It's really impressive how few lines of code Paul needed to implement a reasonable amount of functionality. Since the code looked clean, and it seemed so easy to add features, I could not resist adding a little bit, using some ideas of **dhylands** (Dave) for screen and keyboard handling, and yes, it got quite a bit larger. It still contains the code for the Linux/Darwin environment, so you can run it in Linux/Mac Python3 or MicroPython (if you install the os module). I sprayed a few C Preprocessor statements in it, so you can use cpp to remove the stuff which is not needed for a specific target. What did I change and add:

- Use stdin/stdout on the MicroPython boards.
- Changed the read keyboard function to comply with char-by-char input on serial lines.
- Added support for TAB, BACKTAB, SAVE, FIND, REPLACE, GOTO Line, CUT, COPY, PASTE, REDRAW, UNDO, REDO, INDENT, DEDENT, COMMENT, OPEN (new file), scrolling, and more.
- Join lines by Delete Char at the end or Backspace at the beginning, Auto-indent for Enter.
- Moved main into a function with an option for Pipe'ing in on Linux & Python3
- Added a status line and single line prompts for Quit, Save, Find, Replace, Goto, Open file and Flag settings.
- Support the simultaneous editing of multiple files.
- Support of the basic mouse functions scrolling up/down and setting the cursor.

The editor works in Insert mode. Cursor Keys, Home, End, PgUp and PgDn work as you would expect. The other functions are available with Ctrl-Keys. Keyboard Mapping:

Keys	Alternative	Function
Up	Ctrl-Up	Move the cursor up one line.
Down	Ctrl-Down	Move the cursor down one line.
Left		Move the cursor left by one char, skipping over to the previous line .
Right		Move the cursor right by one char, skipping over to the next line.
Ctrl-Left		Move the cursor left to the start of the actual or previous symbol, skipping over to the previous line .
Ctrl-Right		Move the cursor right behind the next symbol or line end, skipping over to the next line.
Shift-Left		Start text highlighting and extend the highlight by moving left.

Shift-Right	Start text highlighting and extend the highlight by moving right.
Shift-Up	Start text highlighting and extend the highlight by moving up.
Shift-Down	Start text highlighting and extend the highlight by moving down.
Shift-Right	Start the text highlight and extend the highlight by moving right.
Ctrl-Shift-Left	Highlight the previous word or extend the highlight.
Ctrl-Shift-Right	Highlight the next word or extend the highlight.
Ctrl-Up	Scroll the window down
Ctrl-Down	Scroll the window up
Alt-Up	Move the current line or highlighted area up and move the cursor up.
Alt-Down	Move the current line or highlighted area down and move the cursor down.
Alt-Left	Move the character under the cursor left and move the Cursor.
Alt-Right	Move the character under the cursor right and move the Cursor.
PgUp	Move the cursor up by one screen height.
PgDn	Move the cursor down by one screen height.
Home	Go to start-of-line , if the Cursor is at start-of-text . Otherwise go to start-of-text .
End	Toggle the position between the end-of-line and end-of-code .
Enter \n	Insert a line break at the cursor position. Auto-indent is supported.
Backspace	Delete the char left hand to the cursor. If the mark is set, just delete the marked line range. (The key must be set to ASCII-Del) At the beginning of the line Backspace joins the previous line.
Del	Delete char under the cursor. At the end of the line join the next line. If auto-indent is enabled, delete also the leading spaces of the joined line. If the mark is set, delete the marked line range.
Ctrl-Del	Delete Word word under the cursor or space up to the next non-space.
Shift-Del	Delete Line. Deletes the line at the cursor.

Tab	Ctrl-I	Tab. Insert spaces at the cursor position up to the next tab location, moving the cursor. If text is highlighted, indent the highlighted line range.
BackTab	Ctrl-U	Back Tab. Remove spaces left to the cursor position up to the next tab location or the next non-space char, and moves the cursor. If text is highlighted, dedent the highlighted line range.
Ctrl-Q		Quit a file buffer or the line edit mode. If the edited text was changed, ask for confirmation. If the last buffer is closed, the editor will terminate too.
Ctrl-S		Save to file. The file name will be prompted for. If a new name is given and that file already exists, ask for confirmation to overwrite. The content will be written to a temporary file (basename + ".pyetmp") first and then this will be renamed. If the target file name is invalid, the temporary file will have the content. The buffer will after saving have the name of the saved file, which thus acts as Save-as.
Ctrl-E		Redraw the screen according to the actual screen parameters width, height. With MicroPython, as a side effect, garbage collection is performed and the available memory is shown. With Linux/CPython, window size changes result in an automatic redraw.
Ctrl-F		Find text. The last search string is memorized, even across multiple edit windows. Search stops at the end. Whether the search is case sensitive or not, can be set by the Ctrl-A command.
Ctrl-N		Repeat find starting at the column right to the cursor.
Ctrl-H	Ctrl-R	Find and replace. If text is highlighted, it affects only the highlighted region.
Ctrl-G		Go to Line. It prompts for the line number.
Ctrl-B	Ctrl-End	Go to the last line
Ctrl-T	Ctrl-Home	Go to the first line
Ctrl-K		Go to the matching bracket , if any. The cursor has to be on a bracket symbol. Bracket pairs are (), [], {} and <>. Brackets in comments and strings are not discarded .
Ctrl-A		Settings. Sets the state of auto-indent, search case sensitivity, tab size, comment string and write-tabs. Enter 'y' or 'n' or a number in comma separated fields (e.g. n,y,4,n). An empty field leaves the respective value unchanged. The default values are auto-indent: y, case sensitive: n, tab-size: 4, comment string '# ', Write Tabs: n The pye2 variant has an additional flag which affects the

positioning of the cursor during vertical moves (straight vs. following line ends).

Ctrl-L	Ctrl-Space	Start highlighting text and move the cursor to the right. Once highlighting is enabled, it can be extended by moving the cursor. The mark affects Delete, Backspace, Cut lines, Copy lines, Insert lines, Tab, Backtab, Save and Replace. The highlight is cleared by pushing Ctrl-L again or an operation on that area.
Ctrl-X		Cut the highlighted text and keep it in the paste buffer. Together with the Ctrl-V this implements the Cut & Paste feature. The highlight is cleared.
Ctrl-C	Ctrl-D	Copy the area highlighted text to the paste buffer. Together with the Ctrl-V this implements the Copy & Paste feature. The highlight is cleared on copy.
Ctrl-V		Insert the content of the paste buffer at the actual cursor position. If text is highlighted, delete that area first.
Ctrl-W		Switch to the next file .
Ctrl-O		Open a new file. The file name will be prompted for. If the name is left empty, an empty buffer will be opened. If the file cannot be loaded (e.g. because it does not exist), a buffer with that name will still be opened, but will be empty. If the name entered belongs to a directory, the sorted list of file names in that directory will be loaded.
Ctrl-Z		Undo the last change(s). Every char add sequence/deleted char sequence/replaced item/deleted text/inserted text/indent sequence/dedent sequence counts as a single change. The default for the undo stack size per buffer is 50 with PyBoard/WiPy and 500 with Linux/Darwin systems. It can be changed in the call to <code>pye()</code> .
Ctrl-Y		Redo the operation undone by Undo, so it is an Undo of Undo. Redo only works immediately after Undo(s). Any other change to the text will invalidate the Redo history.
Ctrl-P		Comment/Uncomment a line or highlighted area. The string used for commenting can be set through the Ctrl-A command.

Mouse support

Mouse left click	Within the text: Click at any position to set the cursor. Double click starts highlighting at the current position, or clears the highlight. The click speed does not matter. If the cursor is on a word, the whole word will be highlighted. If not, just the character under the cursor. The highlight can
------------------	--

then be extended by moving the cursor clicking at different positions. Right click then copies the highlighted text.

- | | |
|--------------------|---|
| Mouse right click | Opens the "find" dialogue in a text file and the "open file" dialogue in a directory listing. |
| Mouse left click | In line edit mode: Clicking on a word copies it to the edit line. Clicking a second time at the same place acts as enter. |
| Mouse scroll wheel | Move the cursor up and down, scrolling the content as needed. |

Instead of Ctrl-letter (e.g. Ctrl-Q), Alt-letter (e.g. Alt-Q) or ESC-letter (e.g. ESC followed by Q) can be used, avoiding conflicts with key binding of some terminal emulators.

The editor is contained in the files `pye.py` and the port specific front-end, e.g. `pye_gen.py`. Start `pye` from the REPL prompt e.g. with

```
from pye_gen import pye
res = pye(object_1, object_2, ..[, tabsize=n][, undo=n])
```

You may also use the combined version `pye_mp.py`:

```
from pye_mp import pye
res = pye(object_1, object_2, ..[, tabsize=n][, undo=n])
```

If `object_n` is a string, it's considered as the name of a file to be edited or a directory to be opened. If it's a file, the content will be loaded, and the name of the file will be returned when `pye` is closed. If the file does not exist, an error is displayed, but the edit window is given that name. If it's a directory, the list of file names will be loaded to the edit window. If `object_n` is a list of other items, these will be converted to strings and edited, and the edited list of strings will be returned. If no object is named, `pye()` will give show the list of files, creating a list of strings, unless you save to a file. In that case, the file name will be returned. It is always the last buffer closed, which determines the return value of `pye()`.

Optional named parameters:

- | | |
|------------------------|---|
| <code>tabsize=n</code> | Tab step (integer). The default is 4 |
| <code>undo=n</code> | Size of the undo stack (integer). The minimum size is 4 |

The Linux/Darwin version can be called from the command line with:

```
python3 pye.py [filename(s)]
```

or using the combined executable version:

```
pye [filename(s)]
```

Obviously, you may use `micropython` too. Using `python3` (not `micropython`), content can also be redirected or pipe'd into the editor.

When reading files, tab characters (\x09) in the text are replaced by spaces, tab size 8, and white space at the end of a line is discarded. When you save the file, you have the option to replace **all** sequences of spaces by tabs, tab size 8. If the initial file contained tab characters, this is the default. However, the original state will NOT be restored. So be careful when editing files with tab characters.

The size of a file that can be edited on the boards is limited by its memory. You may use REDRAW to determine how much space is left. Besides the file itself, both buffer operations and especially undo consume memory. The undo stack can be limited in the call to pye. Up to about 150 lines on ESP8266 and 600 lines on PyBoard V1.1 should be safe to edit. The largest suitable file size is in the same order of what can be handled as source file during import.

When you save a file on PyBoard, these changes may not be visible in the file system of a connected PC until you disconnect and reconnect the Pyboard drive. See also the related discussion in the MicroPython Forum.

Notes:

- **The keyboard mapping assumes VT100. For those interested, I collected the key codes issue by terminal emulators, all claiming VT100 compatible. Picocom seems sometimes to send the Linux Terminal codes. If the KEYMAP is too large, and you know which terminal you are working on, delete or comment out the obsolete lines. If your terminal is different, just change the control codes.**

Key	Putty VT100 & Xterm	Putty esc-[~	Putty Linux	Minicom	GtkTerm	Picocom	Linux Terminal
Up	\e[A	\e[A	\e[A	\e[A	\e[A	\e[A	\e[A
Down	\e[B	\e[B	\e[B	\e[B	\e[B	\e[B	\e[B
Left	\e[D	\e[D	\e[D	\e[D	\e[D	\e[D	\e[D
Right	\e[C	\e[C	\e[C	\e[C	\e[C	\e[C	\e[C
Home	\e[1~	\e[1~	\e[1~	\e[1~	\eOH	\eOH	\e[H
End	\e[4~	\e[4~	\e[4~	\eOF	\eOF	\eOF	\e[F
Ins	\e[2~	\e[2~	\e[2~	\e[2~	\e[2~	\e[2~	\e[2~
Del	\e[3~	\e[3~	\e[3~	\e[3~	\e[3~	\e[3~	\e[3~
PgUp	\e[5~	\e[5~	\e[5~	\e[5~	\e[5~	\e[5~	\e[5~
PgDn	\e[6~	\e[6~	\e[6~	\e[6~	\e[6~	\e[6~	\e[6~
Backspace	\x7f	\x7f	\x7f	\x7f	\x08	\x7f	\x7f
Ctrl-Home				\e[1;5H		\e[1;5H	\e[1;5H
Ctrl-End				\e[1;5F		\e[1;5F	\e[1;5F
Ctrl-Del				\e[3;5~	\e[3;5~	\e[3;5~	\e[3;5~
Tab	\x09	\x09	\x09	\x09	\x09	\x09	\x09
BackTab	\e[Z	\e[Z	\e[Z	\e[Z	\e[Z	\e[Z	\e[Z

- Since all these key sequences start with the Escape char, typing Escape on the keyboard seems to lock the keyboard. You can unlock it by typing any alpha character.
- Windows terminal emulators behave inconsistent. Putty for instance does not report the mouse actions at all, and also does not return code for shift-

arrow keys. TeraTerm, IVT terminal and Xsh20 just report the mouse click, but not the scroll wheel actions. ZOC reports mouse positions constantly, and sends no key codes for Home, End, PgUp, PGDn and Del. The latter holds also for PowerVT. I could not get Qodem working. Hyperterminal's VT100 emulation is crap. So, after all, I consider TeraTerm, Xsh20 or Putty as the best choices for Windows.

- Gnome terminal sometimes does not send the first mouse wheel code, after the pointer was moved into the window. Mate and XFCE4 terminal do, but have slightly different keyboard mappings.
- Saving to the internal flash of PyBoard is really slow, so don't get nervous. Watch the red LED.
- For those who do complain about the enormous long `handle_edit_key()` function: I tried a variant where every if-elif-case of the function was replaced by a little function, and KEYMAP contained the names of the functions as pointers, which then could be called directly. Thus, almost every key had the same handling time. That worked, and the source file was not much longer, but the compiled code size grew by 50% w/o a useful advantage. So I dropped this approach.
- Putting all functions into the single class is a little bit messy. I made a variant of `pye.py`, where all tty related functions are placed in a separate class. That looks much better, but uses about 600 bytes more RAM. For WiPy, I'm still fighting for every byte, so I left it aside. And even then there are some functions (`scroll_xx()`) where it's not clear in which module to place them. Finally, I would have to split them.
- The Linux version runs on Android too, at least in a setup I tried. That consists of the termux terminal emulator app, which includes an 'apt' command, much like apt-get or aptitude. It allows installing a command line version of python 3.5.1, which will run `pye`. The terminal window emulates VT100. The keyboard app I'm using is hacker's keyboard, which provides Escape, Ctrl- and cursor keys.
- Using frozen bytecode (Pyboard, ESP8266, ESP32, W60X, PyCom devices, Maixpy..) or at least compiled byte code is highly recommended. That reduces start-up time and saves RAM in case of frozen bytecode.
- For WiPy1 only the Pre-Compiled version fits into memory, and even that runs only using Telnet with the UART session closed.