keyboard

Take full control of your keyboard with this small Python library. Hook global events, register hotkeys, simulate key presses and much more.

Features

- Global event hook on all keyboards (captures keys regardless of focus).
- **Listen** and **sends** keyboard events.
- Works with **Windows** and **Linux** (requires sudo), with experimental **OS X** support (thanks @glitchassassin!).
- **Pure Python**, no C modules to be compiled.
- **Zero dependencies**. Trivial to install and deploy, just copy the files.
- **Python 2 and 3**.
- Complex hotkey support (e.g. `Ctrl+Shift+M, Ctrl+Space`) with controllable timeout.
- Includes **high level API** (e.g. record and play, add_abbreviation).
- Maps keys as they actually are in your layout, with **full internationalization support** (e.g. `Ctrl+ç`).
- Events automatically captured in separate thread, doesn't block main program.
- Tested and documented.
- Doesn't break accented dead keys (I'm looking at you, pyHook).
- Mouse support available via project mouse (`pip install mouse`).

This program makes no attempt to hide itself, so don't use it for keyloggers.

Usage

Install the PyPI package:

```
$ sudo pip install keyboard
```

or clone the repository (no installation required, source files are sufficient):

```
$ git clone https://github.com/boppreh/keyboard
```

Then check the API docs to see what features are available.

Example

```
import keyboard

keyboard.press_and_release('shift+s, space')

keyboard.write('The quick brown fox jumps over the lazy dog.')

# Press PAGE UP then PAGE DOWN to type "foobar".
keyboard.add_hotkey('page up, page down', lambda: keyboard.write('foobar'))

# Blocks until you press esc.
keyboard.wait('esc')

# Record events until 'esc' is pressed.
recorded = keyboard.record(until='esc')
# Then replay back at three times the speed.
keyboard.play(recorded, speed_factor=3)
```

```
# Type @@ then press space to replace with abbreviation.
keyboard.add_abbreviation('@@', 'my.long.email@example.com')
# Block forever.
keyboard.wait()
```

Known limitations:

- Events generated under Windows don't report device id (`event.device == None`). #21
- Linux doesn't seem to report media keys. #20
- Currently no way to suppress keys ('catch' events and block them). #22
- To avoid depending on X the Linux parts reads raw device files (`/dev/input/input*`) but this requries root.
- Other applications, such as some games, may register hooks that swallow all key events. In this case `keyboard` will be unable to report events.

API

Table of Contents

- keyboard.**write**
- keyboard.**to_scan_code**
- keyboard.**send**
- keyboard.**press**
- keyboard.**release**
- keyboard.**press_and_release**
- keyboard.**wait**
- keyboard.**read_key**
- keyboard.**record**
- keyboard.**play**
- keyboard.**replay** *(alias)*
- keyboard.**get_typed_strings**
- keyboard.**start_recording**
- keyboard.**stop_recording**
- keyboard.**get_shortcut_name**
- keyboard.**read_shortcut**
- keyboard.**read_hotkey** *(alias)*

keyboard.KEY_DOWN

```
= 'down'
```

keyboard.KEY_UP

```
= 'up'
```

class keyboard.KeyboardEvent

KeyboardEvent.event_type

KeyboardEvent.name

KeyboardEvent.scan_code

KeyboardEvent.time

keyboard.all_modifiers

```
= ('alt', 'alt gr', 'ctrl', 'shift', 'win')
```

keyboard.queue

A multi-producer, multi-consumer queue.

keyboard.recording

keyboard.is_str(x)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L90)

keyboard.is_number(x)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L91)

keyboard.matches(event, name)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L141)

Returns True if the given event represents the same key as the one given in `name`.

keyboard.is_pressed(key)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L158)

Returns True if the key is pressed.

```
is_pressed(57) -> True
is_pressed('space') -> True
is_pressed('ctrl+space') -> True
```

keyboard.canonicalize(hotkey)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L180)

Splits a user provided hotkey into a list of steps, each one made of a list of scan codes or names. Used to normalize input at the API boundary. When a combo is given (e.g. 'ctrl + a, b') spaces are ignored.

```
canonicalize(57) -> [[57]]
canonicalize([[57]]) -> [[57]]
canonicalize('space') -> [['space']]
canonicalize('ctrl+space') -> [['ctrl', 'space']]
canonicalize('ctrl+space, space') -> [['ctrl', 'space'], ['space']]
```

Note we must not convert names into scan codes because a name may represent more than one physical key (e.g. two 'ctrl' keys).

keyboard.call_later(fn, args=(), delay=0.001)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L214)

Calls the provided function in a new thread after waiting some time. Useful for giving the system some time to process an event, without blocking the current execution flow.

keyboard.clear_all_hotkeys()

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L232)

Removes all hotkey handlers. Note some functions such as 'wait' and 'record' internally use hotkeys and will be affected by this call.

Abbreviations and word listeners are not hotkeys and therefore not affected.
To remove all hooks use `unhook_all()`.

keyboard.remove_all_hotkeys

Alias for `clear_all_hotkeys`.

keyboard.add_hotkey(hotkey, callback, args=(), suppress=False, timeout=1, trigger_on_release=False)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L249)

Invokes a callback every time a key combination is pressed. The hotkey must be in the format "ctrl+shift+a, s". This would trigger when the user holds ctrl, shift and "a" at once, releases, and then presses "s". To represent literal commas, pluses and spaces use their names ('comma', 'plus', 'space').

- `args` is an optional list of arguments to passed to the callback during each invocation.
- `suppress` defines if the it should block processing other hotkeys after a match is found. Currently Windows-only.
- `timeout` is the amount of seconds allowed to pass between key presses.
- `trigger_on_release` if true, the callback is invoked on key release instead of key press.

The event handler function is returned. To remove a hotkey call `remove_hotkey(hotkey)` or `remove_hotkey(handler)`. before the combination state is reset.

Note: hotkeys are activated when the last key is *pressed*, not released. Note: the callback is executed in a separate thread, asynchronously. For an example of how to use a callback synchronously, see `wait`.

```
add_hotkey(57, print, args=['space was pressed'])
add_hotkey(' ', print, args=['space was pressed'])
add_hotkey('space', print, args=['space was pressed'])
add_hotkey('Space', print, args=['space was pressed'])

add_hotkey('ctrl+q', quit)
add_hotkey('ctrl+alt+enter, space', some_callback)
```

keyboard.register_hotkey

Alias for `add_hotkey`.

keyboard.hook(callback)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L328)

Installs a global listener on all available keyboards, invoking `callback` each time a key is pressed or released.

The event passed to the callback is of type `keyboard.KeyboardEvent`, with the following attributes:

- `name`: an Unicode representation of the character (e.g. "&") or description (e.g. "space"). The name is always lower-case.
- `scan_code`: number representing the physical key, e.g. 55.
- `time`: timestamp of the time the event occurred, with as much precision as given by the OS.

Returns the given callback for easier development.

keyboard.unhook(callback)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L347)

Removes a previously hooked callback.

keyboard.unhook_all()

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L351)

Removes all keyboard hooks in use, including hotkeys, abbreviations, word listeners, `record`ers and `wait`s.

keyboard.hook_key(key, keydown_callback=<lambda>, keyup_callback=<lambda>)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L360)

Hooks key up and key down events for a single key. Returns the event handler created. To remove a hooked key use `unhook_key(key)` or `unhook_key(handler)`.

Note: this function shares state with hotkeys, so `clear_all_hotkeys` affects it aswell.

keyboard.on_press(callback)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L381)

Invokes `callback` for every KEY_DOWN event. For details see `hook`.

keyboard.on_release(callback)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L387)

Invokes `callback` for every KEY_UP event. For details see `hook`.

keyboard.remove_hotkey(hotkey_or_handler)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L416)

Removes a previously registered hotkey. Accepts either the hotkey used during registration (exact string) or the event handler returned by the `add_hotkey` or `hook_key` functions.

keyboard.unhook_key

Alias for `remove_hotkey`.

keyboard.add_word_listener(word, callback, triggers=['space'], match_suffix=False, timeout=2)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L435)

Invokes a callback every time a sequence of characters is typed (e.g. 'pet') and followed by a trigger key (e.g. space). Modifiers (e.g. alt, ctrl, shift) are ignored.

- `word` the typed text to be matched. E.g. 'pet'.
- `callback` is an argument-less function to be invoked each time the word is typed.
- `triggers` is the list of keys that will cause a match to be checked. If the user presses some key that is not a character (len>1) and not in triggers, the characters so far will be discarded. By default only space bar triggers match checks.
- `match_suffix` defines if endings of words should also be checked instead of only whole words. E.g. if true, typing 'carpet'+space will trigger the listener for 'pet'. Defaults to false, only whole words are checked.
- `timeout` is the maximum number of seconds between typed characters before the current word is discarded. Defaults to 2 seconds.

Returns the event handler created. To remove a word listener use `remove_word_listener(word)` or `remove_word_listener(handler)`.

Note: all actions are performed on key down. Key up events are ignored. Note: word mathes are **case sensitive**.

keyboard.register_word_listener

Alias for `add_word_listener`.

keyboard.remove_word_listener(word_or_handler)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L486)

Removes a previously registered word listener. Accepts either the word used during registration (exact string) or the event handler returned by the `add_word_listener` or `add_abbreviation` functions.

keyboard.remove_abbreviation

Alias for `remove_word_listener`.

keyboard.add_abbreviation(source_text, replacement_text, match_suffix=False, timeout=2)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L494)

Registers a hotkey that replaces one typed text with another. For example

```
add_abbreviation('tm', u'™')
```

Replaces every "tm" followed by a space with a ™ symbol (and no space). The replacement is done by sending backspace events.

- `match_suffix` defines if endings of words should also be checked instead of only whole words. E.g. if true, typing 'carpet'+space will trigger the listener for 'pet'. Defaults to false, only whole words are checked.
- `timeout` is the maximum number of seconds between typed characters before the current word is discarded. Defaults to 2 seconds.

For more details see `add_word_listener`.

keyboard.register_abbreviation

Alias for `add_abbreviation`.

keyboard.stash_state()

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L520)

Builds a list of all currently pressed scan codes, releases them and returns the list. Pairs well with `restore_state`.

keyboard.restore_state(scan_codes)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L530)

Given a list of scan_codes ensures these keys, and only these keys, are pressed. Pairs well with `stash_state`.

keyboard.write(text, delay=0, restore_state_after=True, exact=False)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L542)

Sends artificial keyboard events to the OS, simulating the typing of a given text. Characters not available on the keyboard are typed as explicit unicode characters using OS-specific functionality, such as alt+codepoint.

To ensure text integrity all currently pressed keys are released before the text is typed.

- `delay` is the number of seconds to wait between keypresses, defaults to no delay.
- `restore_state_after` can be used to restore the state of pressed keys after the text is typed, i.e. presses the keys that were released at the beginning. Defaults to True.
- `exact` forces typing all characters as explicit unicode (e.g. alt+codepoint)

keyboard.to_scan_code(key)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L592)

Returns the scan code for a given key name (or scan code, i.e. do nothing). Note that a name may belong to more than one physical key, in which case one of the scan codes will be chosen.

keyboard.send(combination, do_press=True, do_release=True)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L604)

Sends OS events that perform the given hotkey combination.

- `combination` can be either a scan code (e.g. 57 for space), single key (e.g. 'space') or multi-key, multi-step combination (e.g. 'alt+F4, enter').
- `do_press` if true then press events are sent. Defaults to True.
- `do_release` if true then release events are sent. Defaults to True.

  send(57) send('ctrl+alt+del') send('alt+F4, enter') send('shift+s')

Note: keys are released in the opposite order they were pressed.

keyboard.press(combination)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L629)

Presses and holds down a key combination (see `send`).

keyboard.release(combination)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L633)

Releases a key combination (see `send`).

keyboard.press_and_release(combination)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L637)

Presses and releases the key combination (see `send`).

keyboard.wait(combination=None)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L655)

Blocks the program execution until the given key combination is pressed or, if given no parameters, blocks forever.

keyboard.read_key(filter=<lambda>)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L666)

Blocks until a keyboard event happens, then returns that event.

keyboard.record(until='escape')

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L678)

Records all keyboard events from all keyboards until the user presses the given key combination. Then returns the list of events recorded, of type `keyboard.KeyboardEvent`. Pairs well with `play(events)`.

Note: this is a blocking function. Note: for more details on the keyboard hook and events see `hook`.

keyboard.play(events, speed_factor=1.0)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L694)

Plays a sequence of recorded events, maintaining the relative time intervals. If speed_factor is <= 0 then the actions are replayed as fast as the OS allows. Pairs well with `record()`.

Note: the current keyboard state is cleared at the beginning and restored at the end of the function.

keyboard.replay

Alias for `play`.

keyboard.get_typed_strings(events, allow_backspace=True)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/__init__.py#L722)

Given a sequence of events, tries to deduce what strings were typed. Strings are separated when a non-textual key is pressed (such as tab or enter). Characters are converted to uppercase according to shift and capslock status. If `allow_backspace` is True, backspaces remove the last character typed.

This function is a generator, so you can pass an infinite stream of events and convert them to strings in real time.

Note this functions is merely an heuristic. Windows for example keeps per- process keyboard state such as keyboard layout, and this information is not available for our hooks.

```
get_type_strings(record()) -> ['This is what', 'I recorded', '']
```

keyboard.start_recording(recorded_events_queue=None)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L769)

Starts recording all keyboard events into a global variable, or the given queue if any. Returns the queue of events and the hooked function.

Use `stop_recording()` or `unhook(hooked_function)` to stop.

keyboard.stop_recording()

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L781)

Stops the global recording of events and returns a list of the events captured.

keyboard.get_shortcut_name(names=None)

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L795)

Returns a string representation of shortcut from the given key names, or the currently pressed keys if not given. This function:

- normalizes names;
- removes "left" and "right" prefixes;
- replaces the "+" key name with "plus" to avoid ambiguity;
- puts modifier keys first, in a standardized order;
- sort remaining keys;
- finally, joins everything with "+".

Example:

```
get_shortcut_name(['+', 'left ctrl', 'shift'])
# "ctrl+shift+plus"
```

keyboard.read_shortcut()

[[source]](https://github.com/boppreh/keyboard/blob/master/keyboard/**init**.py#L825)

Similar to `read_key()`, but blocks until the user presses and releases a key combination (or single key), then returns a string representing the shortcut pressed.

Example:

```
read_shortcut()
# "ctrl+shift+p"
```

keyboard.read_hotkey

Alias for `read_shortcut`.