

---

# **Oraide Documentation**

***Release 0.2***

**Daniel D. Beck**

August 06, 2013



# CONTENTS



Oraide is a Python library to help presenters with live coding, demonstrations, and recording screencasts. Oraide uses `tmux` to create the illusion that someone is manually typing in a terminal session. Oraide is free software, provided under a BSD-style license.

Contents:



# TUTORIAL

This tutorial will walk you through the process of installing and using Oraide for the first time.

## 1.1 Prerequisites

Oraide requires tmux. Your system may already have tmux installed, or you may need to use your system's package manager to install it. For example, to install tmux with APT, run `apt-get install tmux`. On Mac OS X, you can install tmux with [Homebrew](#). To install tmux with Homebrew, run: `brew install tmux`.

## 1.2 Installation

To install Oraide, run `pip install oraide`, or download and install the latest release from [PyPI](#).

## 1.3 Setting up

First, get a tmux session up and running:

1. Start a terminal session.
2. Start a new tmux session. Enter `tmux new-session -s 'my_session'` and press `Enter`. You should find yourself in a terminal session, as if you had just started. It's a terminal session in a terminal session (yes, it's a little confusing, but it's easy to leave; run `exit` at the command line, or, by default, press `Ctrl + b` followed by the `&` key).

This is the session we'll be controlling with Oraide.

---

**Note:** Don't crane your neck! If your presentation is going to be on a large screen or projector that would be awkward for you to look at while you give your presentation, attach a second terminal to `my_session` with tmux's `attach-session` command. Put the second terminal on your screen (e.g., your laptop's built-in display) and give your neck a rest.

---

3. Start a separate terminal session. This is the session you will use to send keystrokes to `my_session`.
4. Start the Python interactive interpreter. Enter `python` and press `Enter`. It will look something like this:

```
$ python
Python 2.7.5 (default, Jun  9 2013, 16:41:37)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

## 1.4 Sending keystrokes

At it's simplest, Oraide is a wrapper around `tmux`'s `send-keys` command. Let's give it a try.

1. In the Python interactive session, import Oraide. Enter `import oraide` and press Enter. It'll look like this:

```
>>> import oraide
```

2. Send some keys to the `tmux` session with the `send_keys()` function. It requires two parameters: a session name and a string of keys to send. Enter this:

```
>>> oraide.send_keys('my_session', "echo 'Hello, World!'")
```

and then press Enter. If you look at your `tmux` session, you'll see the second parameter entered at the prompt, like this:

```
$ echo 'Hello, World!'
```

Note that the command is unexecuted, because we haven't sent the Enter key yet.

3. Next, send the Enter key to `my_session`. Type:

```
>>> send_keys('my_session', 'Enter', literal=False)
```

and then press Enter. The Enter key is sent to `my_session`.

The `send_keys()` function accepts an optional keyword argument, `literal`. `tmux` can try to convert the names of keys into the keystrokes themselves (e.g., Escape to the Esc key), but this can be quite surprising. Instead, Oraide defaults to treating the string literally. If you want to look up special keystrokes, set `literal` to `False` (or, at least, something falsy).

### See Also:

Remembering which strings convert to which keystrokes is annoying, so you can use attributes of the `oraide.keys` module instead of literal strings. Then you can substitute `Enter` for `oraide.keys.enter`.

## 1.5 Session management

While `send_keys()` is useful, it's tedious to re-enter the session name every time. To alleviate that frustration, and introduce some more features, Oraide provides the `Session` class. Here's a simple script that demonstrates its use:

```
from oraide import keys, Session

s = Session('my_session')

s.send_keys("echo 'Hello, World!'")
s.send_keys(keys.enter, literal=False)
```

The script is equivalent to the two `send_keys` calls made in the [previous section](#). The `Session.send_keys()` method is just like the `send_keys` function, but the session name is no longer needed.



The `Session` class, by keeping the name of the session, allows for some special behavior, including the `Session.teletype()` and `Session.enter()` methods. Let's take a look:

```
from oraide import keys, Session

s = Session('my_session')

s.teletype("echo 'Hello, World!'")
s.send_keys(keys.enter, literal=False)

s.enter("echo 'Look Ma, no hands!'")
```

The `teletype` method works like `send_keys` with two differences:

1. Before the keys are sent, a prompt appears (in the terminal where the script is running, not the tmux session), so you can control the pacing of your presentation.
2. The keys are sent one at a time, with a short delay between each, to simulate typing. It looks something like this (slowed for dramatic effect):

The `enter` method does the same as `teletype`, except the `Enter` key is sent after the literal keys.

## 1.6 Auto-advancement

By default, the `teletype` and `enter` methods prompt before sending keys to the session. Sometimes this is inconvenient. For example, you may want to narrate a longer sequence of steps without stopping. To suppress prompts, you can use the `Session.auto_advance()` context manager, like this:

```
from oraide import keys, Session

s = Session('my_session')

with s.auto_advance():
    s.teletype("echo 'Hello, World!'")
    s.send_keys(keys.enter, literal=False)

s.enter("echo 'Look Ma, no hands!'")
```

The commands inside the `with` statement are executed without prompting.

If you want to auto-advance all keys sent to a session, you can instantiate a `Session` object with `enable_auto_advance=True`.

---

**Note:** It's unwise to auto-advance your entire presentation. Not only is it easier to practice shorter auto-advancing sequences, but you also give yourself room to respond to questions or repeat an important point.

---

## 1.7 Learning more

Now you're ready to start using Oraide. For more detailed information about the API, see the [API Reference](#). If you'd like to see more examples, try the scripts in Oraide's `examples` directory. If you have problems, see Oraide's [GitHub issues](#).



---

# API REFERENCE

Oraide provides a set of conveniences around tmux's `send-keys` command. For more information about how tmux actually works, please see tmux's [man page](#).

## 2.1 oraide

`oraide.send_keys(session, keys, literal=True)`

Send keys to a tmux session. This function is a wrapper around tmux's `send-keys` command.

If `literal` is `False`, tmux will attempt to convert keynames such as `Escape` or `Space` to their single-key equivalents.

### Parameters

- **session** – name of a tmux session
- **keys** – keystrokes to send to the tmux session
- **literal** – whether to prevent tmux from looking up keynames

`class oraide.Session(session, enable_auto_advance=False, teletype_delay=None)`

A session to which to send keys. This function allows for the deduplication of session names when repeatedly sending keystrokes the same session.

### Parameters

- **session** – the name of a tmux session
- **enable\_auto\_advance** – whether to send keystrokes to the session immediately, or wait for confirmation, on certain methods
- **teletype\_delay** (*int*) – the delay between keystrokes for the `teletype()` method (for overriding the default of 90 milliseconds)

`send_keys(keys, literal=True)`

Send each literal character in `keys` to the session.

### Parameters

- **keys** – literal keystrokes to send to the session
- **literal** – whether to prevent tmux from looking up keynames

### See Also:

`send_keys()`

**teletype** (*keys*, *delay=90*)

Type *keys* character-by-character, as if you were actually typing them by hand.

The *delay* parameter adds time between each keystroke for verisimilitude. The actual time between keystrokes varies up to ten percent more or less than the nominal value. The default, 90 milliseconds, approximates a fast typist.

---

**Note:** If auto-advancing is disabled, then a confirmation prompt appears before keystrokes are sent to the session.

---

#### Parameters

- **keys** – the literal keys to be typed
- **delay** (*int*) – the nominal time between keystrokes in milliseconds.

**enter** (*keys=None*, *teletype=True*, *after='Enter'*)

Type *keys*, then press Enter.

By default, typing character-by-character is enabled with the `teletype` parameter.

---

**Note:** If auto-advancing is disabled, then a confirmation prompt appears before keystrokes are sent to the session.

---

#### Parameters

- **keys** – the keystroke to be sent to the to the session. These keys may only be literal keystrokes, not keynames to be looked up by `tmux`.
- **after** – additional keystrokes to send to the session with `literal` set to `False` (typically for appending a special keys from `oraide.keys`, like the default, Enter)

**auto\_advance** ()

Return a context manager that disables prompts before sending keystrokes to the session. For example:

```
session.enter('vim some_file.txt')      # prompt first
with session.auto_advance():             # disables prompts
    session.teletype('jjji')
    session.enter('Hello, World!', after=keys.escape)
session.enter(':x')                      # prompt first
```

## 2.1.1 Exceptions

If `tmux`'s `send-keys` command ends with an error status code, an exception is raised.

**exception** `oraide.TmuxError` (*returncode*, *cmd*, *output=None*)

The command sent to `tmux` returned a non-zero exit status. This is an unrecognized `tmux` error.

This exception type inherits from `subprocess.CalledProcessError`, which adds `returncode`, `cmd`, and `output` attributes.

**exception** `oraide.ConnectionFailedError` (*returncode*, *cmd*, *output=None*)

Bases: `oraide.TmuxError`

The `tmux` server connection failed (often because the server was not running at the time the command was sent).

**exception** `oraide.SessionNotFoundError` (*returncode, cmd, output=None, session=None*)

Bases: `oraide.TmuxError`

The tmux session was not found (but a connection to tmux server was established).

This exception type adds another attribute, `session`, for your debugging convenience.

## 2.2 `oraide.keys`

This module provides shortcuts for sending special keystrokes to tmux sessions. The functions and constants are typically used with `oraide.Session.send_keys()`, or the `after` parameter on certain `oraide.Session` methods.

The constants of this module are provided as a Pythonic substitute for the strings used in tmux's keyname lookup table. For example, to send a backspace key, the string `'BSpace'` may be replaced with a reference to `oraide.keys.backspace`. The following keys are provided:

- `backspace`
- `end`
- `enter`
- `escape`
- `home`
- `page_down`
- `page_up`
- `space`
- `tab`
- `up`
- `down`
- `left`
- `right`

as well as the function keys `f1` through `f20`.

`oraide.keys.alt` (*key*)

Make a string that tmux will parse as the alt key (Alt) and key pressed at the same time.

---

**Note:** The `key` parameter is case-sensitive. If `key` is uppercase, it's equivalent to entering Shift and the key.

```
>>> from oraide.keys import alt
>>> alt('a') # alt + a
'A-a'
>>> alt('A') # alt + shift + a
'A-A'
```

---

`oraide.keys.command` (*key*)

Make a string that tmux will parse as the command key (also known as the meta, super, cmd, Apple, and Windows key) and key pressed at the same time.

---

**Note:** The `key` parameter is case-sensitive. If `key` is uppercase, it's equivalent to entering Shift and the key.

```
>>> from oraide.keys import command
>>> command('a') # command + a
'M-a'
>>> command('A') # command + shift + a
'M-A'
```

---

`oraide.keys.control` (*key*)

Make a string that tmux will parse as the control key (Ctrl) and key pressed at the same time.

---

**Note:** The `key` parameter is case-sensitive. If `key` is uppercase, it's equivalent to entering Shift and the key.

```
>>> from oraide.keys import control
>>> control('a') # ctrl + a
'C-a'
>>> control('A') # ctrl + shift + a
'C-A'
```

---

# HISTORY

Oraide was created by [Daniel D. Beck](#). The library's name is a portmanteau of *orate* and *aide*.

## 3.1 Release 0.2

Complete rewrite, with all new API and documentation.

## 3.2 Release 0.1

Initial version.

If you'd like to review the source, contribute changes, or file a bug report, please see [Oraide on GitHub](#).





# PYTHON MODULE INDEX

## O

`oraide, ??`  
`oraide.keys, ??`