

# Jedi.el - Python auto-completion for Emacs

## What is it?

Jedi.el is a Python auto-completion package for Emacs. It aims at helping your Python coding in a non-destructive way. It also helps you to find information about Python objects, such as docstring, function arguments and code location.

## Quick start

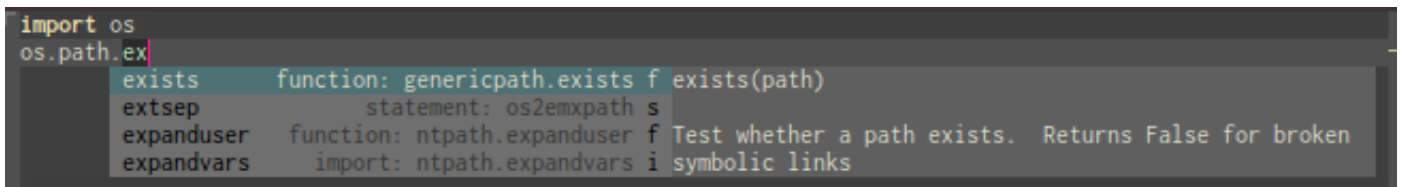
Install Jedi.el via el-get, Marmalade or MELPA (see install for more info) and add this to your Emacs configuration:

```
(add-hook 'python-mode-hook 'jedi:setup)
(setq jedi:setup-keys t)                ; optional
(setq jedi:complete-on-dot t)           ; optional
```

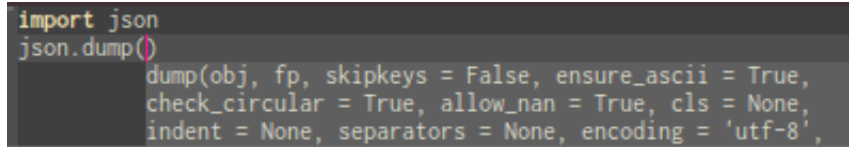
If you install Jedi.el manually (BTW, you shouldn't!), you need to add more stuff to it. See manual install section.

## Screenshots

Jedi.el comes with a set of useful features. Here is a list of screenshots to show some of them.

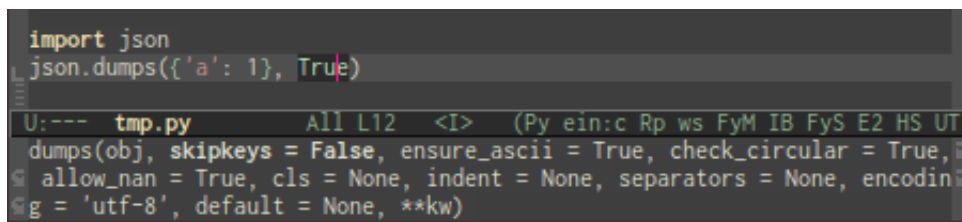


Auto-completion and popup help. This is the main feature of Jedi.el. You don't need to type any special command. Completions and help popup as you type.



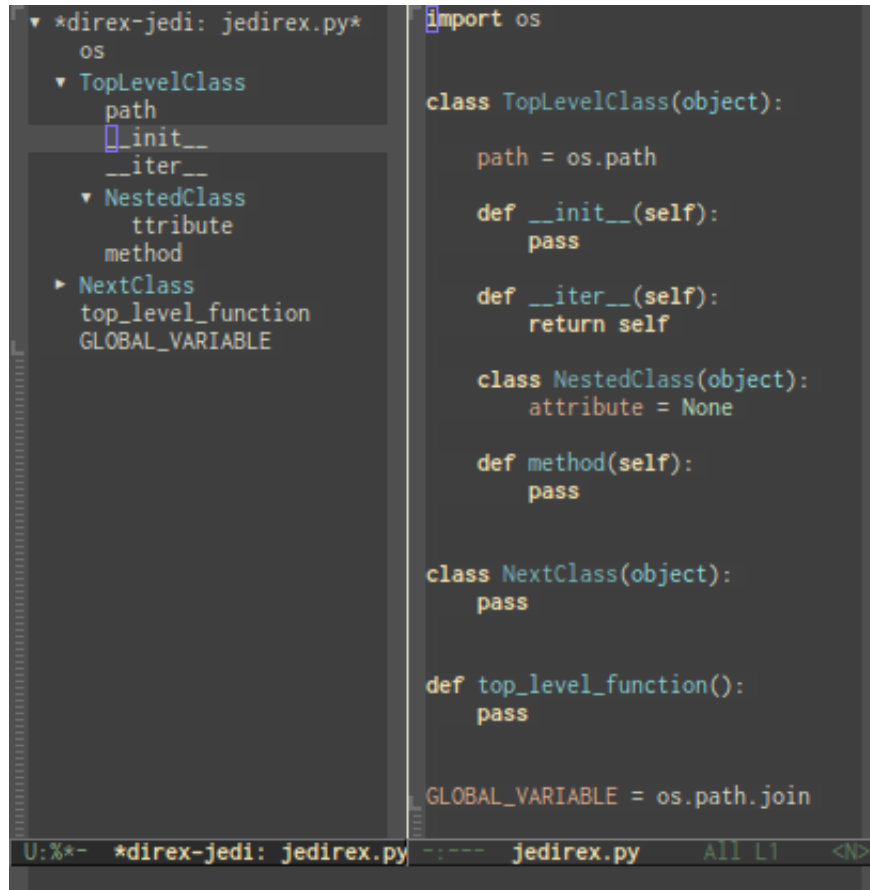
```
import json
json.dump()
    dump(obj, fp, skipkeys = False, ensure_ascii = True,
         check_circular = True, allow_nan = True, cls = None,
         indent = None, separators = None, encoding = 'utf-8',
```

Popup-style call signature help. This is useful when you don't remember what argument to pass.



```
import json
json.dumps({'a': 1}, True)
U:--- tmp.py      All L12  <I>  (Py ein:c Rp ws FyM IB FyS E2 HS UT
dumps(obj, skipkeys = False, ensure_ascii = True, check_circular = True,
allow_nan = True, cls = None, indent = None, separators = None, encodin
g = 'utf-8', default = None, **kw)
```

eldoc-style call signature help. This is another style of showing arguments. Use *jedi:tooltip-method* to configure which style to use.



Source code viewer (need [jedi-direx](#) extension).

## Requirements

### Emacs

- [EPC](#)
- [deferred.el](#) (> v0.3)
- [auto-complete](#)

If your completion popup is broken when width of completion candidates is wide, try the newest version of [popup.el](#).

Jedi.el is currently tested against Emacs 24.3-devel, 24.2 and 23.1.

## Python

- [Jedi](#) ( $\geq 0.6.0$ )
- [python-epc](#)
- `argparse` (for Python 2.6)

Jedi.el is tested against Python 2.6, 2.7 and 3.2.

## Optional dependencies for automatic installation:

- [virtualenv](#)
- `make`

## Install

### el-get

The easiest way to install Jedi.el is to use [el-get](#): just do `M-x el-get-install jedi`. You need to have [virtualenv](#) to automatically install Python module dependencies. If your el-get does not have the recipes for Jedi.el yet, get them from [this pull request](#).

### package.el (Marmalade or MELPA)

You can install Jedi.el using package.el interface from [Marmalade](#) or [MELPA](#). As package.el does not support installing non-elisp packages, you need to install Python part manually (see the next section).

## Manual install

1. Install [EPC](#) and [auto-complete](#).
2. Install Jedi.el. Download the repository of Jedi.el and add it to *load-path*.
3. Install [Jedi](#) and [python-epc](#) by
  - `make requirements` (no need for root privileges [1]) or

- `pip install -r requirements.txt` if you want to determine where to install Python modules. You need root privileges (i.e., `sudo`) to install it in system directory.

#### 4. Add `(autoload 'jedi:setup "jedi" nil t)` in your Emacs configuration.

- [1] You need [virtualenv](#) for make `requirements`. It installs all requirements for Jedi EPC server in an isolated Python environment in `env/` directory under the directory where `jedi.el` locates. Note that you don't need to worry about if you want to use Jedi.el to complete modules in another virtualenv you made. Jedi EPC server recognize the virtualenv it is in (i.e., the environment variable `VIRTUAL_ENV` in your Emacs) and then add modules in that environment to its `sys.path`.

## Setup

All you need to do is to call `jedi:setup` in python buffer. To do that, add the following in your Emacs configuration:

```
(add-hook 'python-mode-hook 'jedi:setup)
```

If auto-completion is all you need, use `jedi:ac-setup` instead:

```
(add-hook 'python-mode-hook 'jedi:ac-setup)
```

To setup recommended keybinds for Jedi.el, add this to your Emacs configuration. Note that you must set `jedi:setup-keys` before loading `jedi.el`. See its docstring (`<f1> v jedi:setup-keys`) for more information.:

```
(setq jedi:setup-keys t)
```

## Extension

## IPython integration

Sometimes it is useful to find completion using Python interpreter. To do that in a seamless manner, you can use IPython and its Emacs binding EIN (Emacs IPython Notebook). See [ein:jedi-setup](#) in the EIN manual. Using this setup, you can run auto-completion command in Jedi.el and EIN simultaneously.

## Configuration

### *function* **jedi:setup**

Fully setup `jedi.el` for current buffer. It setups **ac-sources** (calls **jedi:ac-setup**) and turns **jedi-mode** on.

This function is intended to be called from **python-mode-hook**, like this:

```
(add-hook 'python-mode-hook 'jedi:setup)
```

You can also call this function as a command, to quickly test what Jedi can do.

### *function* **jedi:ac-setup**

Add Jedi AC sources to **ac-sources**.

### *variable* (**jedi:complete-on-dot** *nil*)

Non-**nil** means automatically start completion after inserting a dot. To make this option work, you need to use **jedi:setup** instead of **jedi:ac-setup** to start Jedi.

### *variable* (**jedi:server-command** *("python" "JEDI:SOURCE-*

*DIR/jediepcserver.py"))*

Command used to run Jedi server.

If you setup Jedi requirements using `make requirements` command, **jedi:server-command** should be automatically set to:

```
'("JEDI:SOURCE-DIR/env/bin/python"
  "JEDI:SOURCE-DIR/jediepcserver.py")
```

Otherwise, it should be set to:

```
'("python" "JEDI:SOURCE-DIR/jediepcserver.py")
```

If you want to use your favorite Python executable, set **jedi:server-command** using:

```
(setq jedi:server-command
  (list "YOUR-FAVORITE-PYTHON" jedi:server-script))
```

If you want to pass some arguments to the Jedi server command, use **jedi:server-command**.

*variable (jedi:server-args nil)*

Command line arguments to be appended to **jedi:server-command**.

If you want to add some special **sys.path** when starting Jedi server, do something like this:

```
(setq jedi:server-args
  ('("--sys-path" "MY/SPECIAL/PATH"))
```

```
"--sys-path" "MY/OTHER/SPECIAL/PATH"))
```

If you want to include some virtualenv, do something like this. Note that actual **VIRTUAL\_ENV** is treated automatically. Also, you need to start Jedi EPC server with the same python version that you use for the virtualenv.:

```
(setq jedi:server-args
  '("--virtual-env" "SOME/VIRTUAL_ENV_1"
    "--virtual-env" "SOME/VIRTUAL_ENV_2"))
```

To see what other arguments Jedi server can take, execute the following command:

```
python jediepcserver.py --help
```

## Advanced usage

Sometimes you want to configure how Jedi server is started per buffer. To do that, you should make this variable buffer local in **python-mode-hook** and set it to some buffer specific variable, like this:

```
(defun my-jedi-server-setup ()
  (let ((cmds (GET-SOME-PROJECT-SPECIFIC-COMMAND))
        (args (GET-SOME-PROJECT-SPECIFIC-ARGS)))
    (when cmds (set (make-local-variable 'jedi:server-command) cmds))
    (when args (set (make-local-variable 'jedi:server-args) args))))

(add-hook 'python-mode-hook 'my-jedi-server-setup)
```

Note that Jedi server run by the same command is pooled. So, there is only one Jedi server for the same set of command. If you want to check how many EPC servers are running, use the EPC GUI: M-x



**epc:controller**. You will see a table of EPC connections for Jedi.el and other EPC applications.

If you want to start a new ad-hoc server for the current buffer, use the command **jedi:start-dedicated-server**.

*variable (jedi:get-in-function-call-timeout 3000)*

Cancel request to server for call signature after this period specified in in millisecond.

*variable (jedi:get-in-function-call-delay 1000)*

How long Jedi should wait before showing call signature tooltip in millisecond.

*variable (jedi:tooltip-method '(pos-tip popup))*

Configuration for **jedi:tooltip-show**. This is a list which may contain symbol(s) **pos-tip** and/or **popup**. It determines tooltip method to use. Setting this value to nil means to use minibuffer instead of tooltip.

*variable (jedi:goto-definition-config '((nil nil nil) (t nil nil) (nil definition nil) (t definition nil) (nil nil t) (t nil t) (nil definition t) (t definition t)))*

Configure how prefix argument modifies **jedi:goto-definition** behavior.

Each element of the list is arguments (list) passed to **jedi:goto-definition**. Note that this variable has no effect on **jedi:goto-definition** when it is used as a lisp function

The following setting is default (last parts are omitted). Nth element is used as the argument when N universal prefix arguments (C-u) are given.:

```
(setq jedi:goto-definition-config
  '((nil nil nil) ; C-.
    (t nil nil) ; C-u C-.
    (nil definition nil) ; C-u C-u C-.
    (t definition nil) ; C-u C-u C-u C-.
    ...))
```

For example, if you want to follow “substitution path” by default, use the setting like this:

```
(setq jedi:goto-definition-config
  '((nil definition nil)
    (t definition nil)
    (nil nil nil)
    (t nil nil)
    (nil definition t )
    (t definition t )
    (nil nil t )
    (t nil t )))
```

You can rearrange the order to have most useful sets of arguments at the top.

*variable* (**jedi:goto-definition-marker-ring-length** 16)

Length of marker ring to store **jedi:goto-definition** call positions

*variable* (**jedi:doc-mode** 'rst-mode)

Major mode to use when showing document.

*variable* (**jedi:doc-display-buffer** *'display-buffer'*)

A function to be called with a buffer to show document.

*variable* (**jedi:install-imenu** *nil*)

[EXPERIMENTAL] If **t**, use Jedi to create **imenu** index. To use this feature, you need to install the developmental version (“dev” branch) of Jedi.

*variable* (**jedi:imenu-create-index-function** *'jedi:create-nested-imenu-index'*)

**imenu-create-index-function** for Jedi.el. It must be a function that takes no argument and return an object described in **imenu--index-alist**. This can be set to **jedi:create-flat-imenu-index**. Default is **jedi:create-nested-imenu-index**.

## Keybinds

*variable* (**jedi:setup-keys** *nil*)

Setup recommended keybinds.

### Default keybinds

**<C-tab> : = *jedi:key-complete***

Complete code at point. (**jedi:complete**)

**C-. : = *jedi:key-goto-definition***

Goto the definition of the object at point. (**jedi:goto-definition**)

**C-c d : = *jedi:key-show-doc***

Goto the definition of the object at point. (**jedi:show-doc**)

**C-c r := `jedi:key-related-names`**

Find related names of the object at point. (**`helm-jedi-related-names`** / **`anything-jedi-related-names`**)

When **`jedi:setup-keys`** is non-**`nil`**, recommended keybinds are set in **`jedi-mode-map`** when **loading** `jedi.el`. Therefore, you must set this value before `jedi.el` is loaded. As recommended usage of `jedi.el` is to call **`jedi:setup`** via **`python-mode-hook`** where **`jedi:setup`** is autloaded, setting **`jedi:setup-keys`** to **`t`** in you emacs setup (e.g., `.emacs.d/init.el`) works fine.:

```
(setq jedi:setup-keys t)
(add-hook 'python-mode-hook 'jedi:setup)
```

If you want to require `jedi.el` explicitly when loading Emacs, make sure to set **`jedi:setup-keys`** before loading `jedi.el`:

```
(setq jedi:setup-keys t)
(require 'jedi)
```

Byte compiler warns about unbound variable if you set **`jedi:setup-keys`** before loading `jedi.el`. The proper way to suppress this warning is the following:

```
(eval-when-compile (require 'jedi nil t))
(setq jedi:setup-keys t)
```

You can change these keybinds by changing **`jedi:key-complete`**, **`jedi:key-goto-definition`**, **`jedi:key-show-doc`**, and **`jedi:key-related-names`**. For example, default keybind for ropemacs's **`rope-show-doc`** is same as **`jedi:show-doc`**. You can avoid collision by something like

this:

```
(setq jedi:key-show-doc (kbd "C-c D"))
```

*variable* (**jedi:key-complete** (**kbd** "<C-tab>"))

Keybind for command **jedi:complete**.

*variable* (**jedi:key-goto-definition** (**kbd** "C-."))

Keybind for command **jedi:goto-definition**.

*variable* (**jedi:key-show-doc** (**kbd** "C-c d"))

Keybind for command **jedi:show-doc**.

*variable* (**jedi:key-related-names** (**kbd** "C-c r"))

Keybind for command **helm-jedi-related-names** or **anything-jedi-related-names**.

*variable* (**jedi:goto-definition-pop-marker** (**kbd** "C-,"))

Goto the last point where **jedi:goto-definition** was called.

## Command

*function* **jedi:stop-server**

Stop Jedi server. Use this command when you want to restart Jedi server (e.g., when you changed **jedi:server-command** or **jedi:server-args**). Jedi server will be restarted automatically later when it is needed.

*function* (**jedi:start-dedicated-server** *command*)

Start Jedi server dedicated to this buffer. This is useful, for example, when you want to use different **sys.path** for some buffer. When invoked as an interactive command, it asks you how to start the Jedi server. You can edit the command in minibuffer to specify the way Jedi server run.

If you want to setup how Jedi server is started programmatically per-buffer/per-project basis, make **jedi:server-command** and **jedi:server-args** buffer local and set it in **python-mode-hook**. See also: **jedi:server-args**.

*function* (**jedi:complete** *&rest --cl-rest--*)

Complete code at point.

*function* **jedi:get-in-function-call**

Manually show call signature tooltip.

*function* (**jedi:goto-definition** *&optional other-window deftype use-cache index*)

Goto the definition of the object at point.

See **jedi:goto-definition-config** for how this function works when universal prefix arguments (c-u) are given. If *numeric* prefix argument(s) (e.g., M-0) are given, goto point of the INDEX-th result. Note that you cannot mix universal and numeric prefixes. It is Emacs's limitation. If you mix both kinds of prefix, you get numeric prefix.

When used as a lisp function, popup a buffer when OTHER-WINDOW is non-nil. DEFTYPE must be either **assignment** (default) or **definition**.

When `USE-CACHE` is non-nil, use the locations of the last invocation of this command. If `INDEX` is specified, goto `INDEX`-th result.

*function* **jedi:goto-definition-pop-marker**

Goto the last point where **jedi:goto-definition** was called.

*function* **jedi:show-doc**

Show the documentation of the object at point.

*function* **helm-jedi-related-names**

Find related names of the object at point using **helm** interface.

*function* **anything-jedi-related-names**

Find related names of the object at point using **anything** interface.

*function* **jedi:toggle-debug-server**

Setup **jedi:server-command** and **jedi:server-args** to debug server using `pdb` or `ipdb`.

When this command is called, it essentially execute the following code:

```
(jedi:stop-server)
(setq jedi:server-command (list "cat" "jedi-port.log" )
      jedi:server-args nil)
```

It means to pass the port number recorded in the file `jedi-port.log` to EPC client.

To start Jedi server in terminal and record port to the file, use the

following command:

```
python jediepcserver.py --port-file jedi-port.log --pdb
```

This command will be copied in the kill-ring (clipboard) when this command is called. You can use `-ipdb` instead of `-pdb` to use ipdb instead of pdb.

Calling this command again restores the original setting of `jedi:server-command` and `jedi:server-args` then stops the running server.

## Troubleshooting

Before posting question or bug report in the [issue tracker](#), please investigate the problem by yourself. Here is some checklist.

1. You can try Jedi.el without installing it, by running `make tryout` if you have [carton](#) installed. This will install requirements for Jedi.el separated from your local setup in `.emacs.d`. You can also check the configuration file [tryout-jedi.el](#) to see a minimum working configuration. This is the configuration file loaded by `make tryout`. If you have trouble setting up Jedi.el, compare your configuration file and `tryout-jedi.el`.

If you get some error during `make tryout` or any other `make` tasks, checking `elpa/install.log` may help you finding the problem.

If you install [carton](#) in a different place or you don't add it to the `$PATH`, you can call `make` like this: `make CARTON=PATH/TO/bin/carton tryout`. Typically, `PATH/TO/bin/carton` is `~/.carton/bin/carton`.

If you are too lazy to go to [carton](#) site to checkout how to install it, here is what you need to do:



```
curl -fsSkL https://raw.githubusercontent.com/rejeep/carton/master/go | sh
make CARTON=$HOME/.carton/bin/carton tryout
```

Note that this [carton](#) is different from [the one for Perl](#).

2. To make sure that `jedi.el` is running correctly, you can do `M-x jedi:show-jedi-version`. It will show the versions of the Python libraries you are using.

This is least complex way to communicate with the Jedi server. If it doesn't work, rest of Jedi.el functions will not work.

3. To check that `jedi:setup` is called properly via `python-mode-hook`, run `M-: jedi-mode RET` in some Python file. It should return `t`.
4. If you see message “auto-complete-mode is not enabled”, you might forget to setup auto-complete properly. Calling `(global-auto-complete-mode t)` in your Emacs configuration after *loading* auto-complete should solve this problem.

“After loading” means you need to call `(require 'auto-complete)` (or `(require 'auto-complete-config)` if you need) before calling `(global-auto-complete-mode t)`.

5. It is possible that Jedi's keybind conflicts with keybinds of other libraries. You can check the definition of keybind by `<f1> k c-c d` (or `c-h` instead of `<f1>`), for example. This one should show the help for `jedi:show-doc`.

If you find other command using the keybind, you can change it by using `jedi:key-show-doc`, etc. See: `jedi:setup-keys`.

6. If you have trouble setting keybinds, make sure that `(require 'jedi)` is *not* called anywhere before the line `(setq jedi:setup-keys t)`.

To check that keybind setup works, do `M-: jedi-mode-map RET`. It should return something like the following (formatted):

```
(keymap (3 keymap (100 . jedi:show-doc))
        (67108910 . jedi:goto-definition)
        (C-tab . jedi:complete))
```

Note that using `(require 'jedi)` is *not* recommended way to configure Jedi.el because it slows down your Emacs startup [2]. If you use `package.el` or `el-get` to install Jedi.el, there should be no need for adding `(require 'jedi)` to your configuration. See: **`jedi:setup-keys`**.

[2] But if you know what you are doing `(require 'jedi)` can be a good choice. For example, this way is faster to open the first Python file in an Emacs session.

## How it works

Jedi.el uses [jedi](#) (an awesome Python auto-completion library) and [EPC](#) (an RPC stack for Emacs Lisp) and its [Python binding](#) to communicate with Python process. It also uses excellent Emacs [auto-complete](#) module to start completion automatically. As Jedi.el always calls Python function asynchronously (thanks to [EPC](#)), it will not block your Emacs while your are editing.

## Changelog

### v0.1.2 (2013-05-26)

Highlights:

- Package is available from [Marmalade](#).
- Add imenu support (see `jedi:install-imenu` and `jedi:imenu-create-index-function`). Currently it is not on by default as it needs developmental version of [Jedi](#).
- Add `jedi:goto-definition-config` configurable option.
- Jedi.el now pools server instances. So, for example, you can create buffer-local `jedi:server-args` to setup project-specific Jedi server ([issue-28](#)).
- Do not expand common part when completing on inserting dot using `jedi:dot-complete`.
- Strip off newlines from candidate summary. This prevents popup to be disrupted when showing candidates summary containing newlines (e.g., `json.__all__`).

### Contributions from:

- [Aaron Meurer](#)
- [Danilo Barga](#)
- [Fabián Ezequiel Gallina](#)
- [immerrr](#)
- [Jaakko Pallari](#)
- [Ryan Olf](#)

### Closed issues and pulled patches:

[#13](#) [#15](#) [#17](#) [#19](#) [#26](#) [#27](#) [#29](#) [#30](#) [#33](#) [#38](#) [#44](#) [#52](#) [#53](#) [#54](#) [#59](#) [#62](#)

### v0.1.1 (2012-12-01)

- Add experimental “full-name” support [3].
- [PR-11](#) fixes Makefile for BSD make (thanks, [@goro1080](#)!).
- Fix [issue-9](#): line number sent to the server was shifted when the cursor is

at the beginning of line.

- Fix [issue-10](#): `get_in_function_call` was called in non-python-mode buffer.
- Fix [issue-7](#): server process was killed unexpectedly.
- Add `jedi:setup-keys`. You don't need to manually add Jedi commands to `python-mode-map` now.

Contributions from:

- [Kiyono Goro](#)

Closed issues and pulled patches:

[3] | *jedi:get-full-name-\** functions require developmental version of [Jedi](#). See also: [Request: Definition.fullname · Issue #61 · davidhalter/jedi](#)

## vo.1.0 (2012-11-09)

- [PR-8](#) adds ELDoc like argument highlighting (thanks, [@syohex](#)!).
- [PR-2](#) adds meta-data in header comment for ELPA (thanks, [@syohex](#)!).
- [PR-1](#) fixes Makefile for newer pip version (thanks, [@L42y](#)!).
- First version.

Contributions from:

- [L42y](#)
- [Syohei YOSHIDA](#)

Closed issues and pulled patches: