zentropi

Release 0.1.3

May 20, 2017

Contents

CHAPTER 1

Overview

docs	
tests	
package	

Zentropi: Script Your World.

Zentropi is for you if:

- You like the idea of making your own bots.
- You want to build your personal internet-of-things.
- You are curious about automation at work and/or home.
- You care about your time; automation lets you do more of things you enjoy.
- You consider building your own tools to tackle the increasing complexity of your responsibilities.
- You imagine a world where your real and virtual worlds can interact with each other seamlessly.
- You dream of raising your own army of minions... doing your work tirelessly while you take a vaca...

..where were we? Ah, yes, if any of these ^^ sound familiar, you are not alone!

Installation

Note: Requires Python 3.5+ and compiler toolchain to build c-extensions.

Stable:

```
$ python3 -m venv zen
$ source zen/bin/activate
$ pip install zentropi
```

Current:

```
$ python3 -m venv zen
$ source zen/bin/activate
$ git clone https://github.com/zentropi/python-zentropi.git
$ cd python-zentropi
$ pip install -e .
```

Install steps for Ubuntu and MacOS: https://zentropi.readthedocs.io/en/latest/installation.html

Documentation

https://zentropi.readthedocs.io/

Example

We will make a toy agent that responds to the message "hello" with a "hello, world".

The above illustration shows how the concepts and objects are logically arranged and connected within Zentropi. We will go deeper into these in the READMEs along with examples, for now let us jump straight to the code:

```
from zentropi import Agent, on_message, run_agents, ZentropiShell


class HelloBot(Agent):
    @on_message('hello')
    def say_hello(self, message):
        return 'hello, world'


if __name__ == '__main__':
    hello_bot = HelloBot(name='hello_bot')
    shell = ZentropiShell(name='shell')
    run_agents(hello_bot, shell)
```

Save this as `hello.py` and run with `$ python hello.py`

You should see this on your screen:

```
$ python hello.py
  @shell: '*** started'
  @shell: 'shell-starting'
  @shell: 'shell-ready'
```

We can type any message at the prompt and the shell agent will broadcast it for us. Go ahead and type `hello`, followed by ENTER.

```
hello
  @shell: 'hello'
  @hello_bot: 'hello, world' {'text': 'hello, world'}
  @shell: 'shell-ready'
exit
```

Type `exit` or press Ctrl-D to leave the shell.

What next?

Zentropi is still being developed and is not production-ready, however it is already useful to experiment and build toys.

Check out what is already possible in the examples directory:

https://github.com/zentropi/python-zentropi/tree/master/examples

CHAPTER 2

Installation

If you already have virtual-environment and python-dev set up:

Stable:

:: $ python3 -m venv zen $ source zen/bin/activate $ pip install zentropi

Current:

```
$ python3 -m venv zen
$ source zen/bin/activate
$ git clone https://github.com/zentropi/python-zentropi.git
$ cd python-zentropi
$ pip install -e .
```

Linux

Tested on Ubuntu 16.04

We will need to set up Ubuntu for Python development before we can install and use Zentropi, follow the steps below to update your system with necessary packages:

1. Confirm that Python 3.5 or above is available

```
$ sudo apt update
$ sudo apt install python3
$ python3 -V
```

Above command should output text similar to `Python 3.5.X`; at the time of writing, it was `Python 3.5.2`.

If the command fails, run `sudo apt install python3`.

2. Install external requirements

We will first set up the tools to compile and install Python libraries that are needed by Zentropi.

```
$ sudo apt install gcc libssl-dev python3-dev python3-setuptools python3-venv
```

3. Install and create venv

```
$ python3 -m venv zen
```

This creates a python virtual-environment, which keeps your installed libraries separate from rest of the system.

4. Activate the venv

```
$ source zen/bin/activate
```

You will need to *activate* the zen virtual-environment before working with zentropi. A shortcut would be to add an alias to your `~/.profile` or `~/.bash_profile`:

```
alias zen="source /path/to/zen/bin/activate"
```

You can run `source ~/.profile` or open a new terminal window and type `zen` to activate the virtual-environment.

5. Install zentropi

Stable:

```
$ pip install zentropi
```

Current:

```
$ git clone https://github.com/zentropi/python-zentropi.git
$ cd python-zentropi
$ pip install -e .
```

MacOS

Tested on MacOS Sierra.

If you are starting from scratch for python, we will also set up brew package manager. Visit https://brew.sh/ for instructions. Once you have brew set up (and it may take a long while, depending on your network - if you have never developed software on your Mac before, getting all the tools is a considerable download, 2-5 GB of data.

You will not need to download as much again until the tools are updated, which is not too often for the big packages.

More hints: http://docs.python-guide.org/en/latest/starting/install3/osx/#install3-osx

1. Update package manager

```
$ brew update
```

2. Install python

```
$ brew install python3
```

3. Install and create venv

```
$ python3 -m venv zen
```

This creates a python virtual-environment, which keeps your installed libraries separate from rest of the system.

4. Activate the venv

```
$ source zen/bin/activate
```

You will need to *activate* the zen virtual-environment before working with zentropi. A shortcut would be to add an alias to your `~/.profile` or `~/.bash_profile`:

```
alias zen="source /path/to/zen/bin/activate"
```

You can run `source ~/.profile` or open a new terminal window and type `zen` to activate the virtual-environment.

5. Install zentropi

Stable:

```
$ pip install zentropi
```

Current:

```
$ git clone https://github.com/zentropi/python-zentropi.git
$ cd python-zentropi
$ pip install -e .
```

CHAPTER 3

Usage

To use zentropi in a project:

```
import zentropi
```

CHAPTER 4

Reference

zentropi

class zentropi.**Events**(*callback=None*)

class Zentropian: self.events = Events(callback=self._trigger_frame_handler)

zen.emit('event-name', data={}, internal=False)

@zen.on_event('event-name', parse=True) def handle_event_name(event):

pass

class zentropi.**Field**(*default: typing.Any = None*, ***, *name: typing.Union[str, NoneType] = None*) → None

A Field allows us to store data with extra information associated with it.

Example

```
>>> from zentropi import Field
>>> test_field = Field(42, name='the_answer')
>>> test_field.default
42
>>> test_field.value
42
```

```
>>> test_field.name
'the_answer'
>>> test_field.kind
'Field'
>>> test_field.describe()
{'kind': 'Field', 'name': 'the_answer', 'value': 42}
```

clean(*value: typing.Any*) → typing.Any
Override: Convert types or clean up text; return cleaned value.

validate(*value: typing.Any*) → bool
Override: Return True if given value is valid, this should only accept cleaned values and return False if the value is not valid. Alternatively, raise an exception.

class zentropi.**KINDS**
An enumeration.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Documentation improvements

zentropi could always use more documentation, whether as part of the official zentropi docs, in docstrings, or even on the web in blog posts, articles, and such.

Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/zentropi/python-zentropi/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

Development

To set up *python-zentropi* for local development:

1. Fork python-zentropi (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/python-zentropi.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Harshad Sharma - https://github.com/hiway/
- Ashwini Chaudhary - https://github.com/ashwch

CHAPTER 7

Changelog

0.1.0 (soon)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

z

Index