
python_{*h*}*angmanDocumentation*

Release 2.2.2

Manu Phatak

December 19, 2015

1	Contents:	1
1.1	python_hangman	1
1.2	Installation	2
1.3	Design	2
1.4	Goals	3
1.5	Contributing	4
1.6	Credits	6
1.7	History	6
1.8	hangman package	7
2	Feedback	11
3	Indices and tables	13
	Python Module Index	15

Contents:

1.1 python_hangman

A well tested, cli, python version-agnostic, multi-platform hangman game. It's built following a TDD workflow and a MVC design pattern. Each component services a sensibly distinct logical purpose. Python Hangman is a version agnostic, tox tested, travis-backed program! Documented and distributed.

1.1.1 Features

- Hangman!
- Documentation: https://python_hangman.readthedocs.org
- Open Source: https://github.com/bionikspoon/python_hangman
- Idiomatic code.
- Thoroughly tested with very high coverage.
- Python version agnostic.
- Demonstrates MVC design out of the scope of web development.
- MIT license

1.1.2 Compatibility

- Python 2.6
- Python 2.7
- Python 3.3
- Python 3.4
- Python 3.5
- PyPy

1.1.3 Call Diagram

1.1.4 Credits

Tools used in rendering this package:

- Cookiecutter
- [bionikspoon/cookiecutter-pypackage](#) forked from [audreyr/cookiecutter-pypackage](#)

1.2 Installation

At the command line either via `easy_install` or `pip`:

```
$ mkvirtualenv hangman # optional for venv users
$ pip install python_hangman
$ hangman
```

Uninstall:

```
$ pip uninstall python_hangman
```

1.3 Design

This game roughly follows the **Model-View-Controller(MVC)** pattern. In the latest overhaul, these roles have been explicitly named: `hangman.model`, `hangman.view`, `hangman.controller`.

Traditionally in MVC the controller is the focal point. It tells the view what information to collect from the user and what to show. It uses that information to communicate with the model—also, the data persistence later—and determine the next step. This Hangman MVC adheres to these principals

1.3.1 Model

The model is very simply the hangman game instance—`hangman.model.Hangman`. It's a class. Every class should have “state” and the methods of that class should manage that state. In this case, the “state” is the current “state of the game”. The public API are for managing that state.

The entirety of the game logic is contained in `hangman.model.Hangman`. You could technically play the game in the python console by instantiating the class, submitting guesses with the method `hangman.model.Hangman.guess()` and printing the game state.

For example:

```
>>> from hangman.model import Hangman
>>> game = Hangman(answer='hangman')
>>> game.guess('a')
hangman(status='_A__A_', misses=[], remaining_turns=10)

>>> game.guess('n').guess('z').guess('e')
hangman(status='_AN__AN', misses=['E', 'Z'], remaining_turns=8)

>>> game.status
'_AN__AN'
```

```
>>> game.misses
['E', 'Z']

>>> game.remaining_turns
8
```

1.3.2 View

hangman.view is a collection of stateless functions that represent the presentation layer. When called these functions handles printing the art to the console, and collecting input from the user.

1.3.3 Controller

In this program, the controller is actually the “game_loop”—*hangman.controller.game_loop()*. I still think of it as a controller because the role it plays—communicating I/O from the view with the model-persistence layer.

The controller tells the view later what to print and what data to collect. It uses that information update the state of the game (model) and handle game events.

1.4 Goals

1.4.1 2.0.0

MVC pattern. The goal was to explicitly demonstrate an MVC pattern out of the scope of web development.

Idiomatic code. In this overhaul there’s a big emphasis on idiomatic code. The code should be describing its’ own intention with the clarity your grandmother could read.

1.4.2 1.0.0

Learning! This was a Test Driven Development(TDD) exercise.

Also, explored:

- Tox, test automation
- Travis CI
- Python version agnostic programming
- Setuptools
- Publishing on pip
- Coverage via coveralls
- Documentation with sphinx and ReadTheDocs
- Cookiecutter development

1.5 Contributing

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

You can contribute in many ways:

1.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/bionikspoon/python_hangman/issues.

If you are 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.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/bionikspoon/python_hangman/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 contributions are welcome :)

1.5.2 Get Started!

Ready to contribute? Here’s how to set up *python_hangman* for local development.

1. Fork the *python_hangman* repo on GitHub.
2. Clone your fork locally:


```
$ git clone git@github.com:your_name_here/python_hangman.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv python_hangman
$ cd python_hangman/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b feature/name-of-your-feature
$ git checkout -b hotfix/name-of-your-bugfix
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 hangman tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. 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
```

7. Submit a pull request through the GitHub website.

1.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4, 3.5, and PyPy. Check https://travis-ci.org/bionikspoon/python_hangman/pull_requests and make sure that the tests pass for all supported Python versions.

1.5.4 Tips

To run a subset of tests:

```
$ py.test tests/test_hangman.py
```

1.6 Credits

1.6.1 Development Lead

- Manu Phatak <bionikspoon@gmail.com>

1.6.2 Contributors

None yet. Why not be the first?

1.7 History

1.7.1 Next Release

- Stay Posted

1.7.2 2.2.0 (2015-18-05)

- Fixed max recursion issue with game loop.
- Updated requirements.
- Removed gratuitous docs – less is more.
- 2.2.1 Handle ctrl+d EOF to exit.
- 2.2.2 Fix broken coverage report.

1.7.3 2.1.0 (2015-18-05)

- Updated docs, divided and automated in a more reasonable way.
- renamed the github repo to mirror pypi name.
- 2.1.1 Fix pypi's rst render

1.7.4 2.0.0 (2015-12-05)

- Establishing a changelog.
- Massive refactoring, explicit MVC structure.
- Code is even more idiomatic!
- Created a *FlashMessage* utility.
- Removed poorly implemented classes in favor of stateless functions.
- Add, Remove support for py35, py32.
- 100% code coverage. (2 untestable, inconsequential lines ignored)

1.8 hangman package

1.8.1 python_nhangman

A well tested, cli, python version-agnostic, multi-platform hangman game. It's built following a TDD workflow and a MVC design pattern. Each component services a sensibly distinct logical purpose. Python Hangman is a version agnostic, tox tested, travis-backed program! Documented and distributed.

1.8.2 Submodules

hangman.__main__

Entry point for hangman command.

hangman.controller

hangman.controller.**game_loop**(game=hangman(status='_____', misses=[], remaining_turns=10),
flash=<hangman.utils.FlashMessage object>)

Run a single game.

Parameters

- **game** (hangman.model.Hangman) – Hangman game instance.
- **flash** (hangman.utils.FlashMessage) – FlashMessage utility

hangman.controller.**run**(game=hangman(status='_____', misses=[], remaining_turns=10),
flash=<hangman.utils.FlashMessage object>)

Run game_loop and handle exiting.

Logic is separated from game_loop to cleanly avoid python recursion limits.

Parameters

- **game** (hangman.model.Hangman) – Hangman game instance.
- **flash** (hangman.utils.FlashMessage) – FlashMessage utility

hangman.model

class hangman.model.**Hangman**(answer=None)

Bases: object

The the logic for managing the status of the game and raising key game related events.

```
>>> from hangman.model import Hangman
>>> game = Hangman(answer='hangman')
>>> game.guess('a')
hangman(status='_A__A_', misses=[], remaining_turns=10)
```

```
>>> game.guess('n').guess('z').guess('e')
hangman(status='_AN__AN', misses=['E', 'Z'], remaining_turns=8)
```

```
>>> game.status
'_AN__AN'
```

```
>>> game.misses
['E', 'Z']
```

```
>>> game.remaining_turns
8
```

MAX_TURNS = 10

guess (*letter*)

Add letter to hits or misses.

hits

List of hits.

is_valid_answer (*word*)

Validate answer. Letters only. Max:16

is_valid_guess (*letter*)

Validate guess. Letters only. Max:1

misses

List of misses.

remaining_turns

Calculate number of turns remaining.

status

Build a string representation of status.

hangman.utils

App utilities.

class hangman.utils.**WordBank**

Bases: object

Default collection of words to choose from

WORDS = ['ATTEMPT', 'DOLL', 'ELLEN', 'FLOATING', 'PRIDE', 'HEADING', 'FILM', 'KIDS', 'MONKEY', 'LUNG

classmethod **get** ()

Get a random word from word list.

classmethod **set** (**values*)

Set word list.

class hangman.utils.**FlashMessage**

Bases: object

Basic “flash message” implementation.

game_lost = False

game_won = False

message = ''

exception hangman.utils.**GameLost**

Bases: exceptions.Exception

Raised when out of turns.


```
Dare to pick a letter:  
_
```

Parameters

- **game** (*hangman.Hangman*) – game instance
- **message** (*hangman.utils.FlashMessage*) – flash message

Raises *hangman.utils.GameOverNotificationComplete*

```
hangman.view.print_partial_body (picture, status)  
hangman.view.print_partial_hits (game_status)  
hangman.view.print_partial_message (flash, answer)  
hangman.view.print_partial_title ()  
hangman.view.print_spacer ()  
    Print empty line  
hangman.view.prompt_guess ()  
    Get a single letter.  
hangman.view.prompt_play_again ()  
    Prompt user to play again.  
hangman.view.say_goodbye ()  
    Write a goodbye message.
```

Feedback

If you have any suggestions or questions about **python_hangman** feel free to email me at bionikspoon@gmail.com.

If you encounter any errors or problems with **python_hangman**, please let me know! Open an Issue at the GitHub https://github.com/bionikspoon/python_hangman main repository.

Indices and tables

- `genindex`
- `modindex`
- `search`

h

- hangman, 7
- hangman.__main__, 7
- hangman.controller, 7
- hangman.model, 7
- hangman.utils, 8
- hangman.view, 9

B

build_partial_misses() (in module hangman.view), 9
build_partial_picture() (in module hangman.view), 9

D

draw_board() (in module hangman.view), 9

F

FlashMessage (class in hangman.utils), 8

G

game_loop() (in module hangman.controller), 7
game_lost (hangman.utils.FlashMessage attribute), 8
game_won (hangman.utils.FlashMessage attribute), 8
GameLost, 8
GameOverNotificationComplete, 9
GameWon, 8
get() (hangman.utils.WordBank class method), 8
guess() (hangman.model.Hangman method), 8

H

Hangman (class in hangman.model), 7
hangman (module), 7
hangman.__main__ (module), 7
hangman.controller (module), 7
hangman.model (module), 7
hangman.utils (module), 8
hangman.view (module), 9
hits (hangman.model.Hangman attribute), 8

I

is_valid_answer() (hangman.model.Hangman method), 8
is_valid_guess() (hangman.model.Hangman method), 8

M

MAX_TURNS (hangman.model.Hangman attribute), 8
message (hangman.utils.FlashMessage attribute), 8
misses (hangman.model.Hangman attribute), 8

P

print_partial_body() (in module hangman.view), 10
print_partial_hits() (in module hangman.view), 10
print_partial_message() (in module hangman.view), 10
print_partial_title() (in module hangman.view), 10
print_spacer() (in module hangman.view), 10
prompt_guess() (in module hangman.view), 10
prompt_play_again() (in module hangman.view), 10

R

remaining_turns (hangman.model.Hangman attribute), 8
run() (in module hangman.controller), 7

S

say_goodbye() (in module hangman.view), 10
set() (hangman.utils.WordBank class method), 8
status (hangman.model.Hangman attribute), 8

W

WordBank (class in hangman.utils), 8
WORDS (hangman.utils.WordBank attribute), 8