hamingja Documentation

Release 0.0.2

Florian Kromer

January 17, 2016

Contents

Contents:

CHAPTER 1

About the project

1.1 Introduction

Note: "... [T]he pre-Christian Germanic understanding of luck is very different from our own. ... [L]uck was a quality inherent in the man and his lineage, a part of his personality similar to his strength ... . Luck, the hamingja, is a personal entity in its own right, is part of the self, and can be split off from the other components of the self in certain circumstances. When a person dies, his or her hamingja is often reincarnated in one of his or her descendents, ..." from norse-mythology.org.

hamingja is a pre-processor for C code used in the embedded safety domain. **hamingja** is executed on source files prior to the compilation with a compiler.

The source code is scanned for enumerations. There are assigned explicite values for the enumeration constants. If enumeration constants do not have explicite values assigned to them they get a value assigned. If enumeration constants do already have values assigned to them they are replaced. The hamming distance between each value is satisfied over the whole sw project.

Have a look into the *Functional requirements* for a listing of all capabilities.

CHAPTER 2

Constraints

- Limited spare time of project contributers.
- Project owner is no full-time python developer.

CHAPTER 3

Requirements

3.1 Non-functional requirements

1. Unit tests (e.g. with unittest) shall be executed. (safe code)
2. Integration tests (e.g. with mock in addition to unittest) shall be executed. (safe code)
3. A code coverage analysis shall be executed (e.g. with coverage). (safe code)
4. The project shall be distributed as python package. (reusability)
5. Paths/files which shall be pre-processed shall be configurable. (flexibility)
6. Static code analysis (e.g. with pylint) shall be executed. Alternative tools are: frosted, pychecker (safe code)
7. Code check analysis (e.g. with pylint) shall be executed. Alternative tools are: pep8 (maintainability)
8. A CI environment (e.g. with Travis CI) for automation shall be used. (maintainability)
9. The official python coding style guide (e.g. PEP8) shall be used. (maintainability)

3.2 Functional requirements

1. Enumeration constants shall be identified.
2. All enumeration constants within the whole sw project shall be counted.
3. As many random numbers as the enumeration constant count shall be generated.
4. The random numbers shall satisfy the hamming distance to each other -> safe random number.
5. If an enumeration constant has a value assigned the value shall be replaced with a safe random number.
6. If an enumeration constant has no value assigned there shall be assigned a safe random number.

CHAPTER 4

Operation

4.1 Usage

4.1.1 First time setup

- Fork the project per github-URL. Visit the git documentation website for further information about how to use git. You may also use "Download ZIP" from the hamingja github project website.
- Integrate the virtual environment of hamingja into your virtualenv environment. Visit the virtualenv documentation website and virtualenvwrapper documentation website for further information about the setup and management of virtual environments. Copy the hamingja vritual environment [/venv/hamingja] into your directory containing the virtual environments. The packages may be installed from the requirement file [/venv/hamingja/requirements.txt].
- Run hamingja with this virtual enviroment.

4.1.2 Executing hamingja

4.2 Contribution

4.2.1 An exemplary development environment

- Install Python.
- Install an IDE (e.g. Ninja IDE for Linux from the Ninja IDE website).
- Get the code like already described.
- Happy coding...
- Create a free account on the GitHub website.
- Install a GUI for Git (e.g. gitg from the gitg website).
- Happy contributing over GitHub...

4.2.2 Run the static code analysis

Run `make lint` in the root directory of the project. The lint analysis depends on the configuration file **.pylintrc** in the root directory.

4.2.3 Run the "similar code" analysis

Run `make lint-sim` in the root directory of the project. The lint analysis depends on the configuration file **.pylintsimrc** in the root directory. The text output is generated in files **pylint_hamingja_[module].txt**.

4.2.4 Run the "class check" analysis

Run `make lint-class` in the root directory of the project. The lint analysis depends on the configuration file **.pylintclassrc** in the root directory. The text output is generated in files **pylint_hamingja_[module].txt**.

4.2.5 Run the production code coverage analysis

Run `make cover-prod` in the root directory of the project. The coverage analysis depends on the configuration file **.prodcoveragerc** in the root directory. The html output is generated in the **/cov/prod** directory with **index.html** as entry point. Select specific modules for detailed coverage information of the modules lines.

4.2.6 Run the test code coverage analysis

Run `make cover-test` in the root directory of the project. The coverage analysis depends on the configuration file **.testcoveragerc** in the root directory. The html output is generated in the **/cov/test** directory with **index.html** as entry point. Select specific modules for detailed coverage information of the modules lines.

4.2.7 Run the tests

Run `make tests` in the root directory of the project.

4.2.8 Run the docstring coverage analysation and code documentation

Run `make docu` in the root directory of the project.

CHAPTER 5

Verification

5.1 Production Code Coverage

Follow this link to the code coverage results for further details about the code coverage of the package.

CHAPTER 6

Code

6.1 Production Code

6.1.1 Package modules

lexer	

6.1.2 Modules

6.2 Test Code

6.2.1 Package modules

tests	This module contains the class "Test".

6.2.2 Modules

This module contains the class "Test".

class tests.**Test**(*methodName='runTest'*)

Wrapper for all unit test cases of the package "unittest". Inherits from the TestCase object.

testAbsoluteFalse()

FAILs always for testing if the test framework is set up correctly.

testAsoluteTruth()

PASSes always...

CHAPTER 7

Resources

7.1 Video tutorials

Walkthrough “working with python virtual environments”

Tutorial on lex by Jonathan Engelsma

Tutorial on yacc by Jonathan Engelsma

7.2 Guides

Example of a chemical equation parser implementation

7.3 Manuals

PLY documentation

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

t

T