richreports

Release 0.2.0

Andrei Lapets

Mar 31, 2024

CONTENTS

1 Installation and Usage **3**
- 1.1 Examples . . . 3

2 Development **5**
- 2.1 Documentation . . . 5
- 2.2 Testing and Conventions . . . 5
- 2.3 Contributions . . . 6
- 2.4 Versioning . . . 6
- 2.5 Publishing . . . 6
  - 2.5.1 richreports module . . . 6

Python Module Index **11**

Index **13**

Library that supports the construction of human-readable, interactive static analysis reports that consist of decorated concrete syntax representations of programs.

CHAPTER

ONE

INSTALLATION AND USAGE

This library is available as a package on PyPI:

```
python -m pip install richreports
```

The library can be imported in the usual way:

```
from richreports import *
```

1.1 Examples

This library supports the enrichment of concrete syntax strings with delimiters. A `report` instance can be created from a concrete string and then enriched:

```
>>> r = report(
...     'def f(x, y):\n' +
...     '    return x + y'
... )
>>> r.enrich((2, 11), (2, 15), '(', ')')
>>> for line in r.render().split('\n'):
...     print(line)
def f(x, y):
    return (x + y)
```

This makes it possible to succinctly build up reports that correspond to structured representation formats such as HTML:

```
>>> r.enrich((1, 0), (2, 15), '<b>', '</b>', True)
>>> for line in r.render().split('\n'):
...     print(line)
<b>def f(x, y):</b>
<b>    return (x + y)</b>
>>> r.enrich((1, 0), (2, 15), '<div>\n', '\n</div>')
>>> for line in r.render().split('\n'):
...     print(line)
<div>
<b>def f(x, y):</b>
<b>    return (x + y)</b>
</div>
```

CHAPTER

TWO

DEVELOPMENT

All installation and development dependencies are fully specified in `pyproject.toml`. The `project.optional-dependencies` object is used to specify optional requirements for various development tasks. This makes it possible to specify additional options (such as `docs`, `lint`, and so on) when performing installation using pip:

```
python -m pip install .[docs,lint]
```

2.1 Documentation

The documentation can be generated automatically from the source files using Sphinx:

```
python -m pip install .[docs]
cd docs
sphinx-apidoc -f -E --templatedir=_templates -o _source .. && make html
```

2.2 Testing and Conventions

All unit tests are executed and their coverage is measured when using pytest (see the `pyproject.toml` file for configuration details):

```
python -m pip install .[test]
python -m pytest
```

Alternatively, all unit tests are included in the module itself and can be executed using doctest:

```
python src/richreports/richreports.py -v
```

Style conventions are enforced using Pylint:

```
python -m pip install .[lint]
python -m pylint src/richreports
```

2.3 Contributions

In order to contribute to the source code, open an issue or submit a pull request on the GitHub page for this library.

2.4 Versioning

Beginning with version 0.1.0, the version number format for this library and the changes to the library associated with version number increments conform with Semantic Versioning 2.0.0.

2.5 Publishing

This library can be published as a package on PyPI by a package maintainer. First, install the dependencies required for packaging and publishing:

```
python -m pip install .[publish]
```

Ensure that the correct version number appears in `pyproject.toml`, and that any links in this README document to the Read the Docs documentation of this package (or its dependencies) have appropriate version numbers. Also ensure that the Read the Docs project for this library has an automation rule that activates and sets as the default all tagged versions. Create and push a tag for this version (replacing `?.?.?` with the version number):

```
git tag ?.?.?
git push origin ?.?.?
```

Remove any old build/distribution files. Then, package the source into a distribution archive:

```
rm -rf build dist src/*.egg-info
python -m build --sdist --wheel .
```

Finally, upload the package distribution archive to PyPI:

```
python -m twine upload dist/*
```

2.5.1 richreports module

Library that supports the construction of human-readable, interactive static analysis reports that consist of decorated concrete syntax representations of programs.

class location(*iterable=()*, */*)

Bases: `Tuple[int, int]`

Data structure for representing a location within a report as a tuple of two integers: the line number and the column on that line.

Because this class is derived from the `tuple` type, relational operators can be used to determine whether one location appears before or after another.

```
>>> location((12, 24)) < location((13, 24))
True
>>> location((13, 23)) < location((13, 24))
```

(continues on next page)

(continued from previous page)

```
True
>>> location((13, 24)) < location((13, 24))
False
>>> location((14, 0)) < location((13, 0))
False
>>> location((14, 23)) < location((13, 41))
False
>>> location((12, 24)) <= location((13, 24))
True
>>> location((13, 23)) <= location((13, 24))
True
>>> location((13, 24)) <= location((13, 24))
True
>>> location((14, 0)) <= location((13, 0))
False
>>> location((14, 23)) <= location((13, 41))
False
```

class report(*string: str*, *line: int = 1*, *column: int = 0*)

Bases: `object`

Data structure that represents the raw concrete syntax string as a two-dimensional array of two-sided stacks. Each stack holds delimiters (left and right) that may appear before or after that character in the rendered version of the report.

```
>>> r = report(
...     'def f(x, y):\n' +
...     '    return x + y'
... )
```

The individual lines in the supplied string can be retrieved via the `lines` attribute.

```
>>> list(r.lines)
['def f(x, y):', '    return x + y']
```

Delimiters can be added around a range within the report by specifying the locations corresponding to the endpoints (inclusive) of the range.

```
>>> r.enrich((2, 11), (2, 15), '(', ')')
>>> for line in r.render().split('\n'):
...     print(line)
def f(x, y):
    return (x + y)
```

The optional `enrich_intermediate_lines` parameter can be used to delimit all complete lines that appear between the supplied endpoints.

```
>>> r.enrich((1, 0), (2, 15), '<b>', '</b>', enrich_intermediate_lines=True)
>>> for line in r.render().split('\n'):
...     print(line)
<b>def f(x, y):</b>
<b>    return (x + y)</b>
```

By default, the `enrich_intermediate_lines` parameter is set to `False`.

```
>>> r.enrich((1, 0), (2, 15), '<div>\n', '\n</div>')
>>> for line in r.render().split('\n'):
...     print(line)
<div>
<b>def f(x, y):</b>
<b>    return (x + y)</b>
</div>
```

The optional `skip_whitespace` parameter (which is set to `False` by default) can be used to ensure that left-hand delimiters skip over whitespace (moving to the right and down) and, likewise, that right-hand delimiters skip over whitespace (moving to the left and up).

```
>>> r = report(
...     '   \n' +
...     '\n' +
...     '   \n' +
...     '    def f(x, y):\n' +
...     '        return x + y        \n' +
...     '                            \n' +
...     '                            \n' +
...     '                            '
... )
>>> r.enrich((2, 0), (5, 20), '<b>', '</b>', skip_whitespace=True)
>>> for line in r.render().split('\n'):
...     print(line)



    <b>def f(x, y):
        return x + y</b>


```

If the delimited text consists of whitespace and `skip_whitespace` is `True`, no delimiters are added.

```
>>> r.enrich((6, 0), (6, 20), '<i>', '</i>', skip_whitespace=True)
>>> r.enrich((1, 0), (1, 3), '<i>', '</i>', skip_whitespace=True)
>>> r.enrich((2, 0), (3, 3), '<i>', '</i>', skip_whitespace=True)
>>> for line in r.render().split('\n'):
...     print(line)



    <b>def f(x, y):
        return x + y</b>


```

If `enrich_intermediate_lines` and `skip_whitespace` are both `True`, then individual lines between the first occurrence of a left-hand delimiter and the last occurrence of a right-hand delimiter are delimited as if each line was being enriched individually with `skip_whitespace` set to `True`.

```
>>> r = report(
```

(continues on next page)

(continued from previous page)

```
...     '    \n' +
...     '\n' +
...     '    def f(x, y):\n' +
...     '                               \n' +
...     '\n' +
...     '                               \n' +
...     '        return x + y           \n' +
...     '\n' +
...     '                               \n' +
...     '                               '
... )
>>> r.enrich(
...     (1, 3), (10, 20),
...     '<b>', '</b>',
...     enrich_intermediate_lines=True, skip_whitespace=True
... )
>>> for line in r.render().split('\n'):
...     print(line)


    <b>def f(x, y):</b>



        <b>return x + y</b>
```

It is possible to specify at what value the line and column numbering schemes begin by supplying the optional `line` and `column` arguments to the instance constructor.

```
>>> r = report('    def f(x, y):\n        return x + y', line=1, column=0)
>>> r.enrich((1, 0), (2, 20), '<b>', '</b>', skip_whitespace=True)
>>> list(r.render().split('\n'))
['    <b>def f(x, y):', '        return x + y</b>']
>>> r = report('    def f(x, y):\n        return x + y', line=0, column=0)
>>> r.enrich((0, 0), (1, 20), '<b>', '</b>', skip_whitespace=True)
>>> list(r.render().split('\n'))
['    <b>def f(x, y):', '        return x + y</b>']
```

enrich(*start: Union[Tuple[int, int],* location*]*, *end: Union[Tuple[int, int],* location*]*, *left: str*, *right: str*, *enrich_intermediate_lines=False*, *skip_whitespace=False*)

Add a pair of left and right delimiters around a given range within this report instance.

```
>>> r = report(
...     'def f(x, y):\n' +
...     '    return x + y'
... )
>>> r.enrich((1, 0), (2, 15), '<b>', '</b>', True)
>>> for line in r.render().split('\n'):
...     print(line)
<b>def f(x, y):</b>
```

(continues on next page)

(continued from previous page)

```
<b>    return x + y</b>
```

render() → str

Return the report (incorporating all delimiters) as a string.

```
>>> r = report(
...     'def f(x, y):\n' +
...     '    return x + y'
... )
>>> r.enrich((1, 0), (2, 16), '<b>', '</b>')
>>> for line in r.render().split('\n'):
...     print(line)
<b>def f(x, y):
    return x + y</b>
```

PYTHON MODULE INDEX

r

richreports.richreports, 6

INDEX

E

enrich() (*report method*), 9

L

location (*class in richreports.richreports*), 6

M

module
    richreports.richreports, 6

R

render() (*report method*), 10
report (*class in richreports.richreports*), 7
richreports.richreports
    module, 6