
boolrule Documentation

Release 0.3.0

Steve Webster

Feb 09, 2023

Contents

1	boolrule	3
2	Installation	5
3	Usage	7
4	boolrule API	9
5	Writing boolean expressions	11
6	Contributing	13
7	Credits	17
8	History	19
9	Indices and tables	21
	Index	23

boolrule is a simple boolean expression evaluation engine.

boolrule was built by the team at tails.com to evaluate conditional edges between nodes in a graph-like structure, though we've found numerous uses for it since.

Contents:

Simple boolean expression evaluation engine.

- Free software: MIT license
- Documentation: <https://boolrule.readthedocs.io>.

1.1 Features

Compare simple boolean statements:

```
>>> rule = BoolRule('5 > 3')
>>> rule.test()
True
>>> rule = BoolRule('5 < 3')
>>> rule.test()
False
```

Evaluate boolean statements against a context dict:

```
>>> can_buy_beer = BoolRule('user.age_years >= 18')
>>> can_buy_beer.test({'user':{'age_years': 12}})
False
>>> can_buy_beer.test({'user':{'age_years': 20}})
True
```

Combine conditions with and and or operators to produce complex expressions:

```
>>> is_hipster = BoolRule('address.postcode.outcode in ("E1","E2") or user.has_beard_
↩= true')
>>> address = {
>>>     'postcode': {
>>>         'outcode': 'E1'
>>>     }
>>> }
>>> is_hipster.test({'has_beard': False, 'address': address})
True
```

1.2 Credits

Made possible by the excellent [pyparsing](#) library.

This package was created with [Cookiecutter](#) and the [audreyr/cookiecutter-pypackage](#) project template.

2.1 Stable release

To install boolrule, run this command in your terminal:

```
$ pip install boolrule
```

This is the preferred method to install boolrule, as it will always install the most recent stable release.

If you don't have [pip](#) installed, this [Python installation guide](#) can guide you through the process.

2.2 From sources

The sources for boolrule can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/tailsdotcom/boolrule
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/tailsdotcom/boolrule/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```


The entirety of boolrule's functionality is encapsulated in the BoolRule class.

3.1 Getting started

The simplest use case is evaluating simple, self-contained expressions:

```
from boolrule import BoolRule

expression = '5 > 10'
rule = BoolRule(expression)
rule.test() # False
```

However, the real power of boolrule comes when the expression makes use of values from the context dict passed to the *test()* method:

```
from boolrule import BoolRule

expression = 'content.is_published = true and user.level in content.allowed_levels'
rule = BoolRule(expression)

context = {
    'user': {
        'level': 'super',
    },
    'content': {
        "is_published": True,
        'allowed_levels': [
            'admin',
            'super'
        ]
    },
}
```

(continues on next page)

(continued from previous page)

```
if rule.test(context):  
    # Let the user see the content  
    pass
```

3.2 Lazy compilation

By default the expression is compiled when you create a new `BoolRule` object. If you're instantiating a lot of `BoolRule` instances but are only likely to call `test` on a few of them (because you're looking for just the first match, for example) then you can use the optional `lazy`` argument in the call to `BoolRule`` to defer compilation until the first call to `test()`:

```
rules = [  
    BoolRule(expression, lazy=True)  
    for expression in expressions  
]  
  
if any(r in rules.test(context)):  
    # Do a thing  
    pass
```

4.1 BoolRule

class boolrule.**BoolRule** (*query*, *lazy=False*)

Represents a boolean expression and provides a *test* method to evaluate the expression and determine its truthiness.

Parameters

- **query** – A string containing the query to be evaluated
- **lazy** – If `True`, parse the query the first time it's tested rather than immediately. This can help with performance if you instantiate a lot of rules and only end up evaluating a small handful.

test (*context=None*)

Test the expression against the given context and return the result.

Parameters **context** – A dict context to evaluate the expression against.

Returns `True` if the expression successfully evaluated against the context, or `False` otherwise.

4.2 Exceptions

class boolrule.**MissingVariableException**

Raised when an expression contains a property path that's not supplied in the context.

class boolrule.**UnknownOperatorException**

Raised when an expression uses an unknown operator.

This should never be thrown since the operator won't be correctly parsed as a token by `pyarsing`, but it's useful to have this hanging around for when additional operators are being added.

Writing boolean expressions

The grammar supported by boolrule is fairly simple but powerful.

5.1 Whitespace

Except within string literals, all whitespace is ignored.

5.2 Literals

Numeric literals are written as bare numbers. Floating point and exponent-based numbers are supported:

```
10      # int
-10     # int with optional sign
10.5    # float (without optional sign)
10.5E-3 # equivalent to 0.0105
```

String literals can be single or double quoted:

```
"Hello, world"
'boolrule rulez'
```

Boolean literals are the bare values `true` and `false`

None type is the bare value `none`

5.3 Property paths

In order to reference values from the context passed into the `test()` method you specify the path to the property as a dot-separated identifier:

```
foo
foo.bar
foo.bar.baz
```

At evaluation time, these will map to either object attributes or dict keys in that order.

5.4 Basic comparison operators

Operator	Description	Example
<code>=</code> , <code>==</code> , <code>eq</code>	Equality	<code>foo == 5</code>
<code>!=</code> , <code>!=</code> , <code>ne</code> ,	Inequality	<code>bar != 5</code>
<code>></code> , <code>gt</code>	Greater than	<code>foo > 5</code>
<code>>=</code> , <code>ge</code> ,	Greater than or equal to	<code>foo >= 5</code>
<code><</code> , <code>lt</code>	Less than	<code>foo < 5</code>
<code><=</code> , <code>le</code> ,	Less than or equal to	<code>foo <= 5</code>
<code>is</code>	Identity	<code>foo is True</code>
<code>isnot</code>	Inverse identity	<code>foobar isnot True</code>

5.5 Logical operators

Operator	Description	Example
<code>and</code>	Logical and	<code>foo == 5 and bar < 10</code>
<code>or</code>	Logical or	<code>bar == 5 or bar < 10</code>

5.6 Set operators

Operator	Description	Example
<code>in</code> ,	Is a member of	<code>foo in ("a", "b", "c")</code>
<code>notin</code> ,	Is not a member of	<code>foo notin ("a", "b")</code>
	Is a subset of	<code>("a", "b") ("a", "b", "c")</code>
	Is a superset of	<code>("a", "b", "c") ("a", "b")</code>
	Intersects	<code>("a", "b") ("a", "b", "c")</code>
<code>not</code>	Does not intersect	<code>("a", "b") not ("c", "d")</code>

5.7 Nested expressions

You can use parentheses to nest expressions:

```
foo > 5 and (10 < bar or bar > 20)
```


Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at <https://github.com/tailsdotcom/boolrule/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.

6.1.2 Fix Bugs

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

6.1.3 Implement Features

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

6.1.4 Write Documentation

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

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/tailsdotcom/boolrule/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 :)

6.2 Get Started!

Ready to contribute? Here's how to set up *boolrule* for local development.

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

```
$ git clone git@github.com:your_name_here/boolrule.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 boolrule
$ cd boolrule/
$ python setup.py develop
```

4. Create a branch for local development:

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

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 boolrule tests
$ python setup.py test or 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.

6.3 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. Check <https://github.com/tailsdotcom/boolrule/actions/workflows/ci.yml> and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_boolrule
```


7.1 Development Lead

- Steve Webster <spjwebster@gmail.com>

7.2 Contributors

- Barnaby Shearer <b@zi.is>
- Mauro Canetti <mauro.canetti@gmail.com>

8.1 0.3.5 (2024-02-09)

- Introduce not operator

8.2 0.3.4 (2021-09-24)

- Fixed error caused by early return when evaluating nested expressions.

8.3 0.3.3 (2021-07-15)

- Upgrade dependencies.

8.4 0.3.2 (2020-09-23)

- Add Type hinting.

8.5 0.3.1 (2020-09-09)

- Raise an exception when the whole expression cannot be parsed. Previous behaviour would discard the segment that didn't match the expression grammar.

8.6 0.3.0 (2018-01-15)

- Add None type and is/isnot operators (contributed by ocurero)

8.7 0.2.0 (2016-10-27)

- Fixed error caused by refactor from internal codebase that was preventing deep context level values from being referenced in a substitution value

8.8 0.1.2 (2016-09-30)

- Improved documentation

8.9 0.1.1 (2016-09-30)

- Made `context` optional
- Improved documentation

8.10 0.1.0 (2016-09-30)

- First release on PyPI.

CHAPTER 9

Indices and tables

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

B

`BoolRule` (*class in boolrule*), 9

M

`MissingVariableException` (*class in boolrule*), 9

T

`test()` (*boolrule.BoolRule method*), 9

U

`UnknownOperatorException` (*class in boolrule*), 9