
coincidence

Release 0.6.6

Helper functions for pytest.

Dominic Davis-Foster

Mar 04, 2024

Contents

| | | |
|----------|---|-----------|
| 1 | Installation | 1 |
| 1.1 | from PyPI | 1 |
| 1.2 | from Anaconda | 1 |
| 1.3 | from GitHub | 1 |
| 2 | coincidence | 3 |
| 2.1 | PEP_563 | 3 |
| 2.2 | pytest_report_header | 3 |
| 3 | coincidence.fixtures | 5 |
| 3.1 | fixed_datetime | 5 |
| 3.2 | path_separator | 5 |
| 3.3 | tmp_pathplus | 6 |
| 4 | coincidence.params | 7 |
| 4.1 | count | 7 |
| 4.2 | whitespace_perms | 7 |
| 4.3 | testing_boolean_values | 8 |
| 4.4 | param | 8 |
| 4.5 | parametrized_versions | 9 |
| 5 | coincidence.regressions | 11 |
| 5.1 | AdvancedDataRegressionFixture | 11 |
| 5.2 | AdvancedFileRegressionFixture | 12 |
| 5.3 | SupportsAsDict | 13 |
| 5.4 | advanced_data_regression | 13 |
| 5.5 | advanced_file_regression | 13 |
| 5.6 | check_file_regression | 13 |
| 5.7 | check_file_output | 14 |
| 6 | coincidence.selectors | 15 |
| 6.1 | min_version | 16 |
| 6.2 | max_version | 16 |
| 6.3 | only_version | 16 |
| 6.4 | not_windows | 16 |
| 6.5 | only_windows | 16 |
| 6.6 | not_pypy | 17 |
| 6.7 | only_pypy | 17 |
| 6.8 | not_macos | 17 |
| 6.9 | only_macos | 17 |
| 6.10 | not_docker | 17 |
| 6.11 | not_linux | 17 |

| | | |
|----------|----------------------------|-----------|
| 6.12 | only_linux | 18 |
| 6.13 | only_docker | 18 |
| 6.14 | platform_boolean_factory | 18 |
| 7 | coincidence_utils | 19 |
| 7.1 | generate_truthy_values | 19 |
| 7.2 | generate_falsy_values | 19 |
| 7.3 | is_docker | 19 |
| 7.4 | with_fixed_datetime | 20 |
| 8 | Changelog | 21 |
| 8.1 | 0.6.0 | 21 |
| 8.2 | 0.5.0 | 21 |
| 8.3 | 0.4.3 | 21 |
| 8.5 | 0.4.1 | 21 |
| 8.7 | 0.4.0 | 21 |
| 8.9 | 0.3.1 | 21 |
| 8.11 | 0.3.0 | 22 |
| 8.13 | 0.2.3 | 22 |
| 8.15 | 0.2.0 | 22 |
| 8.17 | 0.1.2 | 22 |
| 8.18 | 0.1.1 | 23 |
| 8.19 | 0.1.0 | 23 |
| | Python Module Index | 25 |
| | Index | 27 |

Installation

1.1 from PyPI

```
$ python3 -m pip install coincidence --user
```

1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge  
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install coincidence
```

1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/python-coincidence/coincidence@master --user
```


coincidence

Helper functions for pytest.

Data:

| | |
|----------------------|--|
| <code>PEP_563</code> | <code>True</code> if the current Python version implements PEP 563 – Postponed Evaluation of Annotations. |
|----------------------|--|

Functions:

| | |
|---|--|
| <code>pytest_report_header(config)</code> | Prints the start time of the pytest session. |
|---|--|

PEP_563 = False

Type: `bool`

`True` if the current Python version implements **PEP 563** – Postponed Evaluation of Annotations.

Note: This is currently set to `False` until the future of typing PEPs has been determined. No released versions of Python currently have **PEP 563** enabled by default.

Changed in version 0.6.0: Temporarily set to `False` regardless of version.

pytest_report_header (*config*)

Prints the start time of the pytest session.

Return type `str`

coincidence.fixtures

Pytest fixtures.

To enable the fixtures add the following to `conftest.py` in your test directory:

```
pytest_plugins = ("coincidence", )
```

See the [pytest documentation](#) for more information.

Functions:

| | |
|--|---|
| <code>fixed_datetime(monkeypatch)</code> | Pytest fixture to pretend the current datetime is 2:20 AM on 13th October 2020. |
| <code>path_separator(request)</code> | Parametrized pytest fixture which returns the current filesystem path separator and skips the test for the other. |
| <code>tmp_pathplus(tmp_path)</code> | Pytest fixture which returns a temporary directory in the form of a <code>PathPlus</code> object. |

fixture `fixed_datetime`

Scope: function

Pytest fixture to pretend the current datetime is 2:20 AM on 13th October 2020.

See also: The `with_fixed_datetime()` contextmanager.

Attention: The monkeypatching only works when datetime is used and imported like:

```
import datetime
print(datetime.datetime.now())
```

Using `from datetime import datetime` won't work.

Return type `Iterator`

fixture `path_separator`

Scope: function

Parametrized pytest fixture which returns the current filesystem path separator and skips the test for the other.

This is useful when the test output differs on platforms with `\` as the path separator, such as windows.

New in version 0.4.0.

Return type `str`

fixture tmp_pathplus**Scope:** function

Pytest fixture which returns a temporary directory in the form of a `PathPlus` object.

The directory is unique to each test function invocation, created as a sub directory of the base temporary directory.

Use it as follows:

```
pytest_plugins = ("coincidence", )

def test_something(tmp_pathplus: PathPlus):
    assert True
```

Return type `PathPlus`

coincidence.params

pytest.mark.parametrize decorators.

Functions:

| | |
|--|--|
| <code>count(stop[, start, step])</code> | Returns a <code>pytest.mark.parametrize</code> decorator which provides a list of numbers between <code>start</code> and <code>stop</code> with an interval of <code>step</code> . |
| <code>whitespace_perms([ratio])</code> | Returns a <code>pytest.mark.parametrize</code> decorator which provides permutations of whitespace. |
| <code>testing_boolean_values([extra_truthy, ...])</code> | Returns a <code>pytest.mark.parametrize</code> decorator which provides a list of strings, integers and booleans, and the boolean representations of them. |
| <code>param(*values[, marks, id, idx, key])</code> | Specify a parameter in <code>pytest.mark.parametrize</code> calls or parametrized fixtures. |
| <code>parametrized_versions(*versions[, reasons])</code> | Return a list of parametrized version numbers. |

count (*stop*, *start*=0, *step*=1)

Returns a `pytest.mark.parametrize` decorator which provides a list of numbers between `start` and `stop` with an interval of `step`.

The single parametrized argument is `count`.

Parameters

- **stop** (*int*) – The stop value passed to `range`.
- **start** (*int*) – The start value passed to `range`. Default 0.
- **step** (*int*) – The step passed to `range`. Default 1.

Return type `MarkDecorator`

whitespace_perms (*ratio*=0.5)

Returns a `pytest.mark.parametrize` decorator which provides permutations of whitespace.

For this function whitespace is only `_\n\t\r`.

Not all permutations are returned, as there are a lot of them; instead a random selection of the permutations is returned. By default ½ of the permutations are returned, but this can be configured using the `ratio` argument.

The single parametrized argument is `char`.

Parameters **ratio** (*float*) – The ratio of the number of permutations to select to the total number of permutations. Default 0.5.

Return type `MarkDecorator`

testing_boolean_values (*extra_truthy=()*, *extra_falsy=()*, *ratio=1*)

Returns a `pytest.mark.parametrize` decorator which provides a list of strings, integers and booleans, and the boolean representations of them.

The parametrized arguments are `boolean_string` for the input value, and `expected_boolean` for the expected output.

Optionally, a random selection of the values can be returned using the `ratio` argument.

Parameters

- **extra_truthy** (*Sequence*) – Additional values to treat as `True`. Default `()`.
- **extra_falsy** (*Sequence*) – Additional values to treat as `False`. Default `()`.
- **ratio** (*float*) – The ratio of the number of values to select to the total number of values. Default `1`.

Return type `MarkDecorator`

param (**values*, *marks=()*, *id=None*, *idx=None*, *key=None*)

Specify a parameter in `pytest.mark.parametrize` calls or `parametrized fixtures`.

Examples:

```
@pytest.mark.parametrize("test_input, expected", [
    ("3+5", 8),
    param("6*9", 42, marks=pytest.mark.xfail),
    param("2**2", 4, idx=0),
    param("3**2", 9, id="3^2"),
    param("sqrt(9)", 3, key=itemgetter(0)),
])
def test_eval(test_input, expected):
    assert eval(test_input) == expected
```

New in version 0.4.0.

Parameters

- ***values** (*~T*) – Variable args of the values of the parameter set, in order.
- **marks** (*Union[MarkDecorator, Collection[Union[MarkDecorator, Mark]]]*) – A single mark or a list of marks to be applied to this parameter set. Default `()`.
- **id** (*Optional[str]*) – The id to attribute to this parameter set. Default `None`.
- **idx** (*Optional[int]*) – The index of the value in `*values` to use as the id. Default `None`.
- **key** (*Optional[Callable[[Tuple[~T, ...]], str]]*) – A callable which is given values (as a `tuple`) and returns the value to use as the id. Default `None`.

Return type `ParameterSet`

Overloads

- `param(values: object, marks = (), id: Optional[str] = ...)`
- `param(values: object, marks = (), idx: Optional[int])`
- `param(values, marks = (), key: Optional[Callable[[Tuple[~T, ...]], str]])`

parametrized_versions (*versions, reasons=())

Return a list of parametrized version numbers.

Examples:

```
@pytest.mark.parametrize(
    "version",
    parametrized_versions(
        3.6,
        3.7,
        3.8,
        reason="Output differs on each version.",
    ),
)
def test_something(version: str):
    pass
```

```
@pytest.fixture(
    params=parametrized_versions(
        3.6,
        3.7,
        3.8,
        reason="Output differs on each version.",
    ),
)
def version(request):
    return request.param

def test_something(version: str):
    pass
```

New in version 0.4.0.

Parameters

- ***versions** (Union[str, float, Tuple[int, ...]]) – The Python versions to parametrize.
- **reasons** (Union[str, Iterable[Optional[str]]]) – The reasons to use when skipping versions. Either a string value to use for all versions, or a list of values which correspond to *versions. Default ().

Return type List[ParameterSet]

coincidence.regressions

Regression test helpers.

To enable the fixtures in this module add the following to `conftest.py` in your test directory:

```
pytest_plugins = ("coincidence", )
```

Classes:

| | |
|--|--|
| <code>AdvancedDataRegressionFixture(datadir, ...)</code> | Subclass of <code>DataRegressionFixture</code> with support for additional types. |
| <code>AdvancedFileRegressionFixture(datadir, ...)</code> | Subclass of <code>FileRegressionFixture</code> with UTF-8 by default and some extra methods. |
| <code>SupportsAsDict</code> | <code>typing.Protocol</code> for classes like <code>collections.namedtuple()</code> and <code>typing.NamedTuple</code> which implement an <code>_asdict()</code> method. |

Functions:

| | |
|---|--|
| <code>advanced_data_regression(datadir, ...)</code> | Pytest fixture for performing regression tests on lists, dictionaries and namedtuples. |
| <code>advanced_file_regression(datadir, ...)</code> | Pytest fixture for performing regression tests on strings, bytes and files. |
| <code>check_file_regression(data, file_regression)</code> | Check the given data against that in the reference file. |
| <code>check_file_output(filename, file_regression)</code> | Check the content of the given text file against the reference file. |

class `AdvancedDataRegressionFixture` (*datadir, original_datadir, request*)

Bases: `DataRegressionFixture`

Subclass of `DataRegressionFixture` with support for additional types.

The following types and their subclasses are supported:

- `collections.abc.Mapping`, `typing.Mapping` (including `dict` and `typing.Dict`)
- `collections.abc.Sequence`, `typing.Sequence` (including `list`, `typing.Tuple` etc.)
- `collections.OrderedDict`, `typing.OrderedDict`
- `collections.Counter`, `typing.Counter`
- `types.MappingProxyType` (cannot be subclassed)
- `_pytest.capture.CaptureResult` (the type of `capsys.readouterr()`)
- Any type which implements the `SupportsAsDict` protocol (including `collections.namedtuple()` and `typing.NamedTuple`)

check (*data_dict*, *basename=None*, *fullpath=None*)

Checks data against a previously recorded version, or generates a new file.

Parameters

- **data_dict** (`Union[Sequence, SupportsAsDict, Mapping, MappingProxyType, CaptureResult]`)
- **basename** (`Optional[str]`) – The basename of the file to test/record. If not given the name of the test is used. Default `None`.
- **fullpath** (`Optional[str]`) – The complete path to use as a reference file. This option will ignore `datadir` fixture when reading *expected* files, but will still use it to write *obtained* files. Useful if a reference file is located in the session data dir, for example. Default `None`.

Note: `basename` and `fullpath` are exclusive.

class AdvancedFileRegressionFixture (*datadir*, *original_datadir*, *request*)

Bases: `FileRegressionFixture`

Subclass of `FileRegressionFixture` with UTF-8 by default and some extra methods.

New in version 0.2.0.

Methods:

| | |
|--|---|
| <code>check</code> (<i>contents</i> [, <i>encoding</i> , <i>extension</i> , ...]) | Checks the contents against a previously recorded version, or generates a new file. |
| <code>check_bytes</code> (<i>contents</i> , <i>**kwargs</i>) | Checks the bytes contents against a previously recorded version, or generates a new file. |
| <code>check_file</code> (<i>filename</i> [, <i>extension</i> , <i>newline</i>]) | Check the content of the given text file against the reference file. |

check (*contents*, *encoding='UTF-8'*, *extension='.txt'*, *newline=None*, *basename=None*, *fullpath=None*, *binary=False*, *obtained_filename=None*, *check_fn=None*)

Checks the contents against a previously recorded version, or generates a new file.

Parameters

- **contents** (`Union[str, StringList]`)
- **extension** (`str`) – The extension of the reference file. Default `'.txt'`.
- ****kwargs** – Additional keyword arguments passed to `pytest_regressions.file_regression.FileRegressionFixture.check()`.

See also: `check_file_regression()`

check_bytes (*contents*, ***kwargs*)

Checks the bytes contents against a previously recorded version, or generates a new file.

Parameters

- **contents** (`bytes`)
- ****kwargs** – Additional keyword arguments passed to `pytest_regressions.file_regression.FileRegressionFixture.check()`.

check_file (*filename*, *extension=None*, *newline='\n'*, ***kwargs*)
 Check the content of the given text file against the reference file.

Parameters

- **filename** (`Union[str, Path, PathLike]`)
- **extension** (`Optional[str]`) – The extension of the reference file. If `None` the extension is determined from `filename`. Default `None`.
- **newline** (`Optional[str]`) – Controls how universal newlines mode works. See `open()`. Default `'\n'`.
- ****kwargs** – Additional keyword arguments passed to `pytest_regressions.file_regression.FileRegressionFixture.check()`.

See also: `check_file_output()`

protocol SupportsAsDict

Bases: `Protocol`

`typing.Protocol` for classes like `collections.namedtuple()` and `typing.NamedTuple` which implement an `_asdict()` method.

This protocol is `runtime checkable`.

Classes that implement this protocol must have the following methods / attributes:

`_asdict()`

Return a new dict which maps field names to their corresponding values.

Return type `Dict[str, Any]`

`__non_callable_proto_members__ = {}`

Type: `set`

fixture advanced_data_regression

Scope: function

Pytest fixture for performing regression tests on lists, dictionaries and namedtuples.

Return type `AdvancedDataRegressionFixture`

fixture advanced_file_regression

Scope: function

Pytest fixture for performing regression tests on strings, bytes and files.

New in version 0.2.0.

Return type `AdvancedFileRegressionFixture`

check_file_regression (*data*, *file_regression*, *extension='.txt'*, ***kwargs*)

Check the given data against that in the reference file.

Parameters

- **data** (`Union[str, StringList]`)
- **file_regression** (`FileRegressionFixture`) – The file regression fixture for the test.

- **extension** (`str`) – The extension of the reference file. Default `'.txt'`.
- ****kwargs** – Additional keyword arguments passed to `pytest_regressions.file_regression.FileRegressionFixture.check()`.

See also: `AdvancedFileRegressionFixture.check()`

Return type `bool`

check_file_output (`filename`, `file_regression`, `extension=None`, `newline='\n'`, `**kwargs`)

Check the content of the given text file against the reference file.

Parameters

- **filename** (`Union[str, Path, PathLike]`)
- **file_regression** (`FileRegressionFixture`) – The file regression fixture for the test.
- **extension** (`Optional[str]`) – The extension of the reference file. If `None` the extension is determined from `filename`. Default `None`.
- **newline** (`Optional[str]`) – Controls how universal newlines mode works. See `open()`. Default `'\n'`.
- ****kwargs** – Additional keyword arguments passed to `pytest_regressions.file_regression.FileRegressionFixture.check()`.

See also: `AdvancedFileRegressionFixture.check_file()`

Return type `bool`

coincidence.selectors

Pytest decorators for selectively running tests.

Functions:

| | |
|--|--|
| <code>min_version(version[, reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test if the current Python version is less than the required one. |
| <code>max_version(version[, reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test if the current Python version is greater than the required one. |
| <code>only_version(version[, reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test if the current Python version not the required one. |
| <code>not_windows([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test if the current platform is Windows. |
| <code>only_windows([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test unless the current platform is Windows. |
| <code>not_pypy([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test if the current Python implementation is PyPy. |
| <code>only_pypy([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test unless the current Python implementation is PyPy. |
| <code>not_macos([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test if the current platform is macOS. |
| <code>only_macos([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test unless the current platform is macOS. |
| <code>not_docker([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test if running on Docker. |
| <code>not_linux([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test if the current platform is Linux. |
| <code>only_linux([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test unless the current platform is Linux. |
| <code>only_docker([reason])</code> | Factory function to return a <code>@pytest.mark.skipif</code> decorator which will skip a test unless running on Docker. |
| <code>platform_boolean_factory(condition, platform)</code> | Factory function to return decorators such as <code>not_pypy()</code> and <code>only_windows()</code> . |

min_version (*version*, *reason=None*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test if the current Python version is less than the required one.

Parameters

- **version** (`Union[str, float, Tuple[int, ...]]`) – The version number to compare to `sys.version_info`.
- **reason** (`Optional[str]`) – The reason to display when skipping. Default 'Requires Python <version> or greater.'.

Return type `MarkDecorator`

max_version (*version*, *reason=None*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test if the current Python version is greater than the required one.

Parameters

- **version** (`Union[str, float, Tuple[int, ...]]`) – The version number to compare to `sys.version_info`.
- **reason** (`Optional[str]`) – The reason to display when skipping. Default 'Not needed after Python <version>.'.

Return type `MarkDecorator`

only_version (*version*, *reason=None*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test if the current Python version not the required one.

Parameters

- **version** (`Union[str, float, Tuple[int, ...]]`) – The version number to compare to `sys.version_info`.
- **reason** (`Optional[str]`) – The reason to display when skipping. Default 'Not needed on Python <version>.'.

Return type `MarkDecorator`

not_windows (*reason='Not required on Windows'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test if the current platform is Windows.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Not required on Windows'.

Return type `MarkDecorator`

only_windows (*reason='Only required on Windows'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test unless the current platform is Windows.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Only required on Windows'.

Return type `MarkDecorator`

not_pypy (*reason='Not required on PyPy'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test if the current Python implementation is PyPy.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Not required on PyPy'.

Return type `MarkDecorator`

only_pypy (*reason='Only required on PyPy'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test unless the current Python implementation is PyPy.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Only required on PyPy'.

Return type `MarkDecorator`

not_macos (*reason='Not required on macOS'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test if the current platform is macOS.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Not required on macOS'.

Return type `MarkDecorator`

only_macos (*reason='Only required on macOS'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test unless the current platform is macOS.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Only required on macOS'.

Return type `MarkDecorator`

not_docker (*reason='Not required on Docker'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test if running on Docker.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Not required on Docker'.

Return type `MarkDecorator`

not_linux (*reason='Not required on Linux'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test if the current platform is Linux.

New in version 0.2.0.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Not required on Linux'.

Return type `MarkDecorator`

only_linux (*reason='Only required on Linux'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test unless the current platform is Linux.

New in version 0.2.0.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Only required on Linux'.

Return type `MarkDecorator`

only_docker (*reason='Only required on Docker'*)

Factory function to return a `@pytest.mark.skipif` decorator which will skip a test unless running on Docker.

Parameters **reason** (`str`) – The reason to display when skipping. Default 'Only required on Docker'.

Return type `MarkDecorator`

platform_boolean_factory (*condition, platform, versionadded=None, *, module=None*)

Factory function to return decorators such as `not_pypy()` and `only_windows()`.

Parameters

- **condition** (`bool`) – Should evaluate to `True` if the test should be skipped.
- **platform** (`str`)
- **versionadded** (`Optional[str]`) – Default `None`.
- **module** (`Optional[str]`) – The module to set the function as belonging to in `__module__`. If `None` `__module__` is set to `'coincidence.selectors'`. Default `None`.

Return type `Tuple[Callable[...MarkDecorator], Callable[...MarkDecorator]]`

Returns 2-element tuple of `not_function`, `only_function`.

`coincidence.utils`

Test helper utilities.

Functions:

| | |
|--|---|
| <code>generate_truthy_values([extra_truthy, ratio])</code> | Returns an iterator of strings, integers and booleans which should be considered <code>True</code> . |
| <code>generate_falsy_values([extra_falsy, ratio])</code> | Returns an iterator of strings, integers and booleans which should be considered <code>False</code> . |
| <code>is_docker()</code> | Returns whether the current Python instance is running in Docker. |
| <code>with_fixed_datetime(fixed_datetime)</code> | Context manager to set a fixed datetime for the duration of the <code>with</code> block. |

generate_truthy_values (*extra_truthy=()*, *ratio=1*)

Returns an iterator of strings, integers and booleans which should be considered `True`.

Optionally, a random selection of the values can be returned using the `ratio` argument.

Parameters

- **extra_truthy** (`Iterable[Union[str, int, ~_T]]`) – Additional values which should be considered `True`. Default `()`.
- **ratio** (`float`) – The ratio of the number of values to select to the total number of values. Default `1`.

Return type `Iterator[Union[str, int, ~_T]]`

generate_falsy_values (*extra_falsy=()*, *ratio=1*)

Returns an iterator of strings, integers and booleans which should be considered `False`.

Optionally, a random selection of the values can be returned using the `ratio` argument.

Parameters

- **extra_falsy** (`Iterable[Union[str, int, ~_T]]`) – Additional values which should be considered `True`. Default `()`.
- **ratio** (`float`) – The ratio of the number of values to select to the total number of values. Default `1`.

Return type `Iterator[Union[str, int, ~_T]]`

is_docker ()

Returns whether the current Python instance is running in Docker.

Return type `bool`

with_fixed_datetime (*fixed_datetime*)

Context manager to set a fixed datetime for the duration of the with block.

Parameters `fixed_datetime` (`datetime`)

See also: The *fixed_datetime* fixture.

Attention: The monkeypatching only works when datetime is used and imported like:

```
import datetime
print(datetime.datetime.now())
```

Using `from datetime import datetime` won't work.

Return type `Iterator`

Changelog

0.6.0

`coincidence.PEP_563()` is temporarily set to `False` for all versions until the future of typing PEPs has been determined. No released versions of Python currently have **PEP 563** enabled by default.

0.5.0

- `coincidence.regressions.AdvancedDataRegressionFixture()` – Add support for `pathlib` and `domdf_python_tools.paths.PathPlus`.

0.4.3

Bugs Fixed

- `coincidence.utils.with_fixed_datetime()` – Correctly handle monkeypatching of datetime in PyPy.

0.4.1

Bugs Fixed

- `coincidence.PEP_563()` – Is now `True` on Python ≥ 3.11 , per the deferral of **PEP 563**.

0.4.0

Additions

Fixtures

- `coincidence.fixtures.path_separator`

Functions

- `coincidence.params.param()`
- `coincidence.params.parametrized_versions()`

0.3.1

Bugs Fixed

coincidence.regressions – Ensure the custom YAML representers are only configured if PyYAML can be imported.

0.3.0

coincidence.regressions.AdvancedDataRegressionFixture

- Handle `toml.decoder.InlineTableDict` the `toml` module is available.
- Improve handling of custom subclasses, especially for nested types.

0.2.3

Bugs Fixed

Disabled the entry point as it was resulting in a confused plugin loading order and did not work.

The way of enabling the plugin reverts to:

```
# conftest.py
pytest_plugins = ("coincidence", )
```

0.2.0

- Switched to *whely* as the build backend.
- Added support for PyPy 3.7
- ~~Added an entry point for *pytest* to avoid the need to enable the plugin in *conftest*.~~ (reverted in 0.2.3)

Additions

Classs

- *coincidence.regressions.AdvancedFileRegressionFixture*

Fixtures

- *coincidence.regressions.advanced_file_regression*

Functions

- *coincidence.selectors.not_linux()*
- *coincidence.selectors.only_linux()*

0.1.2

- `coincidence.regressions.AdvancedDataRegressionFixture.check()` – Add support for `_pytest.capture.CaptureResult`.

0.1.1

- `coincidence.regressions.AdvancedDataRegressionFixture` – Add a fake version when PyYAML cannot be imported.

0.1.0

Initial release.

Python Module Index

C

- `coincidence`, [3](#)
- `coincidence.fixtures`, [5](#)
- `coincidence.params`, [7](#)
- `coincidence.regressions`, [11](#)
- `coincidence.selectors`, [15](#)
- `coincidence.utils`, [19](#)

Symbols

`__non_callable_proto_members__`
(*SupportsAsDict* attribute), 13
`_asdict()` (*SupportsAsDict* method), 13

A

`AdvancedDataRegressionFixture` (*class in*
coincidence.regressions), 11
`AdvancedFileRegressionFixture` (*class in*
coincidence.regressions), 12

C

`check()` (*AdvancedDataRegressionFixture* method), 12
`check()` (*AdvancedFileRegressionFixture* method), 12
`check_bytes()` (*AdvancedFileRegressionFixture*
method), 12
`check_file()` (*AdvancedFileRegressionFixture*
method), 12
`check_file_output()` (*in module*
coincidence.regressions), 14
`check_file_regression()` (*in module*
coincidence.regressions), 13
`coincidence`
 module, 3
`coincidence.fixtures`
 module, 5
`coincidence.params`
 module, 7
`coincidence.regressions`
 module, 11
`coincidence.selectors`
 module, 15
`coincidence.utils`
 module, 19
`count()` (*in module* *coincidence.params*), 7

G

`generate_falsy_values()` (*in module*
coincidence.utils), 19
`generate_truthy_values()` (*in module*
coincidence.utils), 19

I

`is_docker()` (*in module* *coincidence.utils*), 19

M

`max_version()` (*in module* *coincidence.selectors*), 16
`min_version()` (*in module* *coincidence.selectors*), 15
`module`
 coincidence, 3
 coincidence.fixtures, 5
 coincidence.params, 7
 coincidence.regressions, 11
 coincidence.selectors, 15
 coincidence.utils, 19

N

`not_docker()` (*in module* *coincidence.selectors*), 17
`not_linux()` (*in module* *coincidence.selectors*), 17
`not_macos()` (*in module* *coincidence.selectors*), 17
`not_pypy()` (*in module* *coincidence.selectors*), 17
`not_windows()` (*in module* *coincidence.selectors*), 16

O

`only_docker()` (*in module* *coincidence.selectors*), 18
`only_linux()` (*in module* *coincidence.selectors*), 18
`only_macos()` (*in module* *coincidence.selectors*), 17
`only_pypy()` (*in module* *coincidence.selectors*), 17
`only_version()` (*in module* *coincidence.selectors*),
 16
`only_windows()` (*in module* *coincidence.selectors*),
 16

P

`param()` (*in module* *coincidence.params*), 8
`parametrized_versions()` (*in module*
coincidence.params), 9
`PEP_563` (*in module* *coincidence*), 3
`platform_boolean_factory()` (*in module*
coincidence.selectors), 18
`pytest_report_header()` (*in module*
coincidence), 3
Python Enhancement Proposals
 PEP 563, 3, 21

S

`SupportsAsDict` (*protocol in*
coincidence.regressions), 13

T

`testing_boolean_values()` (*in module*
coincidence.params), [7](#)

W

`whitespace_perms()` (*in module*
coincidence.params), [7](#)

`with_fixed_datetime()` (*in module*
coincidence.utils), [19](#)