coincidence

Release 0.6.6

Helper functions for pytest.

Dominic Davis-Foster

Apr 24, 2024

Contents

Chapter

ONE

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
```

Chapter

TWO

coincidence

Helper functions for pytest.

Data:

PEP_563	`True` if the current Python version implements **PEP 563** – Postponed Evaluation of Annotations.

Functions:

pytest_report_header(config)	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`

Chapter

THREE

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:

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

Chapter

FOUR

coincidence.params

pytest.mark.parametrize decorators.

Functions:

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

Chapter

FIVE

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:

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

Functions:

advanced_data_regression(datadir, ...)	Pytest fixture for performing regression tests on lists, dictionaries and namedtuples.
advanced_file_regression(datadir, ...)	Pytest fixture for performing regression tests on strings, bytes and files.
check_file_regression(data, file_regression)	Check the given data against that in the reference file.
check_file_output(filename, file_regression)	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:

check(contents[, encoding, extension, ...])	Checks the contents against a previously recorded version, or generates a new file.
check_bytes(contents, **kwargs)	Checks the bytes contents against a previously recorded version, or generates a new file.
check_file(filename[, extension, newline])	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

Chapter

SIX

coincidence.selectors

Pytest decorators for selectively running tests.

Functions:

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

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`.

Chapter

SEVEN

coincidence.utils

Test helper utilities.

Functions:

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

Chapter

EIGHT

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 whey 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

Index

Symbols

A

C

G

I

M

N

O

P

S

T

W