AssertionLib Documentation

Release 3.2.1

B. F. van Beek

Oct 12, 2021

CONTENTS

Contents:

CHAPTER

ONE

ASSERTIONLIB 3.2.1

A package for performing assertions and providing informative exception messages.

1.1 Installation

- PyPi: `pip install AssertionLib`
- GitHub: `pip install git+https://github.com/nlesc-nano/AssertionLib`

1.2 Usage

A comprehensive overview of all available assertion methods is provided in the documentation. A few examples of some basic assertion:

```
>>> import numpy as np
>>> from assertionlib import assertion

# Assert the output of specific callables
>>> assertion.eq(5, 5)  # 5 == 5
>>> assertion.lt(5, 6)  # 5 < 6
>>> assertion.gt(6, 5)  # 5 > 6
>>> assertion.isinstance(5, int)
>>> assertion.hasattr(5, '__init__')
>>> assertion.any([False, False, True])
>>> assertion.isfinite(1.0)

# Simply assert a value
>>> assertion(5 == 5)
>>> assertion(isinstance(5, int))

# Apply post-processing before conducting the assertion
>>> ar_large = np.ones(10)
>>> ar_small = np.zeros(10)
>>> assertion.gt(ar_large, ar_small, post_process=np.all)  # all(ar_large > ar_small)

# Perform an assertion which will raise an AssertionError
>>> assertion.eq(5, 6, message='Fancy custom error message')  # 5 == 6
Traceback (most recent call last):
  ...
AssertionError: output = eq(a, b); assert output
```

(continues on next page)

(continued from previous page)

```
exception: AssertionError = AssertionError('Fancy custom error message')

output: bool = False
a: int = 5
b: int = 6
```

A few examples of AssertionErrors raised due to incorrect method signatures:

```
>>> from assertionlib import assertion

>>> assertion.len(5)
Traceback (most recent call last):
  ...
AssertionError: output = len(obj); assert output

exception: TypeError = TypeError("object of type 'int' has no len()")

output: NoneType = None
obj: int = 5
```

```
>>> from assertionlib import assertion

>>> assertion.eq(5, 5, 5, 5)
Traceback (most recent call last):
  ...
AssertionError: output = eq(a, b, _a, _b); assert output

exception: TypeError = TypeError('eq expected 2 arguments, got 4')

output: NoneType = None
a: int = 5
b: int = 5
_a: int = 5
_b: int = 5
```

A demonstration of the `exception` parameter. Providing an exception type will assert that the provided exception is raised during/before the assertion process:

```
>>> from assertionlib import assertion

>>> len(5)
Traceback (most recent call last):
  ...
TypeError: object of type 'int' has no len()
```

```
>>> from assertionlib import assertion

>>> assertion.len(5, exception=TypeError)  # i.e. len(5) should raise a TypeError
>>> assertion.len([5], exception=TypeError)
Traceback (most recent call last):
  ...
AssertionError: output = len(obj); assert output

exception: AssertionError = AssertionError("Failed to raise 'TypeError'")

output: int = 1
```

(continues on next page)

(continued from previous page)

```
obj: list = [5]
```

Lastly, the output of custom callables can be asserted in one of the following two ways, supplying the callable to `AssertionManager.assert()` or creating a custom assertion method and adding it to an instance with `AssertionManager.add_to_instance()`:

```
>>> from assertionlib import assertion

>>> def my_fancy_func(a: object) -> bool:
...     return False

# Approach #1, supply to-be asserted callable to assertion.assert_()
>>> assertion.assert_(my_fancy_func, 5)
Traceback (most recent call last):
  ...
AssertionError: output = my_fancy_func(a); assert output

exception: AssertionError = AssertionError(None)

output: bool = False
a: int = 5
```

```
>>> from assertionlib import assertion

# Approach #2, permanantly add a new bound method using assertion.add_to_instance()
>>> assertion.add_to_instance(my_fancy_func)
>>> assertion.my_fancy_func(5)
Traceback (most recent call last):
  ...
AssertionError: output = my_fancy_func(a); assert output

exception: AssertionError = AssertionError(None)

output: bool = False
a: int = 5
```

CHAPTER

TWO

API

2.1 assertionlib

A package for performing assertion operations.

2.1.1 assertionlib.dataclass

A class with a number of generic pre-defined (magic) methods inspired by the builtin `dataclasses` module introduced in Python 3.7.

2.1.2 assertionlib.functions

Various functions related to the *assertionlib.AssertionManager* class.

2.1.3 assertionlib.manager

A module containing the actual *assertionlib.AssertionManager* class.

2.1.4 assertionlib.ndrepr

A module for holding the *assertionlib.NDRepr* class, a subclass of the builtin `reprlib.Repr` class.

2.2 Index

2.2.1 assertionlib.manager

A module containing the actual *AssertionManager* class.

Index

assertion	An instance of *AssertionManager*.
AssertionManager([repr_instance])	A class for performing assertions and providing informative exception messages.
*AssertionManager.assert_*(func, *args[, ...])	Perform the following assertion: `assert func(*args, **kwargs)`.
AssertionManager.__call__(value, *[, ...])	Equivalent to `assert value`.
AssertionManager.add_to_instance(func[, ...])	Add a new custom assertion method to this instance.

Assertions based on the builtin `operator` module.

AssertionManager.abs	Perform the following assertion: `assert abs(a)`.
AssertionManager.add	Perform the following assertion: `assert add(a, b)`.
*AssertionManager.and_*	Perform the following assertion: `assert and_(a, b)`.
AssertionManager.concat	Perform the following assertion: `assert concat(a, b)`.
AssertionManager.contains	Perform the following assertion: `assert contains(a, b)`.
AssertionManager.countOf	Perform the following assertion: `assert countOf(a, b)`.
AssertionManager.eq	Perform the following assertion: `assert eq(a, b)`.
AssertionManager.floordiv	Perform the following assertion: `assert floordiv(a, b)`.
AssertionManager.ge	Perform the following assertion: `assert ge(a, b)`.
AssertionManager.getitem	Perform the following assertion: `assert getitem(a, b)`.
AssertionManager.gt	Perform the following assertion: `assert gt(a, b)`.
AssertionManager.index	Perform the following assertion: `assert index(a)`.
AssertionManager.indexOf	Perform the following assertion: `assert indexOf(a, b)`.
AssertionManager.inv	Perform the following assertion: `assert inv(a)`.
AssertionManager.invert	Perform the following assertion: `assert invert(a)`.
*AssertionManager.is_*	Perform the following assertion: `assert is_(a, b)`.
AssertionManager.is_not	Perform the following assertion: `assert is_not(a, b)`.
AssertionManager.le	Perform the following assertion: `assert le(a, b)`.
AssertionManager.lshift	Perform the following assertion: `assert lshift(a, b)`.
AssertionManager.lt	Perform the following assertion: `assert lt(a, b)`.

continues on next page

Table 2 – continued from previous page

AssertionManager.matmul	Perform the following assertion: `assert matmul(a, b)`.
AssertionManager.mod	Perform the following assertion: `assert mod(a, b)`.
AssertionManager.mul	Perform the following assertion: `assert mul(a, b)`.
AssertionManager.ne	Perform the following assertion: `assert ne(a, b)`.
AssertionManager.neg	Perform the following assertion: `assert neg(a)`.
*AssertionManager.not_*	Perform the following assertion: `assert not_(a)`.
*AssertionManager.or_*	Perform the following assertion: `assert or_(a, b)`.
AssertionManager.pos	Perform the following assertion: `assert pos(a)`.
AssertionManager.pow	Perform the following assertion: `assert pow(a, b)`.
AssertionManager.rshift	Perform the following assertion: `assert rshift(a, b)`.
AssertionManager.sub	Perform the following assertion: `assert sub(a, b)`.
AssertionManager.truediv	Perform the following assertion: `assert truediv(a, b)`.
AssertionManager.truth	Perform the following assertion: `assert truth(a)`.
AssertionManager.length_hint	Perform the following assertion: `assert length_hint(obj, default=default)`.

Assertions based on the builtin `os.path` module.

AssertionManager.isabs	Perform the following assertion: `assert isabs(s)`.
AssertionManager.isdir	Perform the following assertion: `assert isdir(s)`.
AssertionManager.isfile	Perform the following assertion: `assert isfile(path)`.
AssertionManager.islink	Perform the following assertion: `assert islink(path)`.
AssertionManager.ismount	Perform the following assertion: `assert ismount(path)`.

Assertions based on the builtin `math` module.

AssertionManager.allclose	Perform the following assertion: `assert isclose(a, b, rel_tol=rel_tol, abs_tol=abs_tol)`.
AssertionManager.isclose	Perform the following assertion: `assert isclose(a, b, rel_tol=rel_tol, abs_tol=abs_tol)`.
AssertionManager.isfinite	Perform the following assertion: `assert isfinite(x)`.

continues on next page

Table 4 – continued from previous page

AssertionManager.isinf	Perform the following assertion: `assert isinf(x).`
AssertionManager.isnan	Perform the following assertion: `assert isnan(x).`

Assertions based on the builtin `builtins` module.

AssertionManager.callable	Perform the following assertion: `assert callable(obj).`
AssertionManager.hasattr	Perform the following assertion: `assert hasattr(obj, name).`
AssertionManager.isinstance	Perform the following assertion: `assert isinstance(obj, class_or_tuple).`
AssertionManager.issubclass	Perform the following assertion: `assert issubclass(cls, class_or_tuple).`
AssertionManager.len	Perform the following assertion: `assert len(obj).`
AssertionManager.any	Perform the following assertion: `assert any(iterable).`
AssertionManager.all	Perform the following assertion: `assert all(iterable).`
AssertionManager.isdisjoint	Perform the following assertion: `assert isdisjoint(a, b).`
AssertionManager.issuperset	Perform the following assertion: `assert issuperset(a, b).`
AssertionManager.issubset	Perform the following assertion: `assert issubset(a, b).`
AssertionManager.round	Perform the following assertion: `assert round(number, ndigits=ndigits).`

Miscellaneous assertions.

AssertionManager.len_eq	Perform the following assertion: `assert len_eq(a, b).`
AssertionManager.str_eq	Perform the following assertion: `assert str_eq(a, b, str_converter=str_converter).`
AssertionManager.shape_eq	Perform the following assertion: `assert shape_eq(a, b).`
AssertionManager.function_eq	Perform the following assertion: `assert function_eq(func1, func2).`

API

assertionlib.manager.**assertion** : **AssertionManager**
An instance of *AssertionManager*.

class assertionlib.manager.**AssertionManager**(*repr_instance: Optional[reprlib.Repr] = <assertionlib.ndrepr.NDRepr object>*)
A class for performing assertions and providing informative exception messages.

A number of usage examples are provided in the the documentation.

Parameters **repr_instance** (reprlib.Repr, optional) – An instance of reprlib.Repr for formatting Exception messages. The passed instance should have access to a bound callable by the name of repr, which in turn should produce a string representation of any passed objects. If None, default the builtin repr() function. See also *AssertionManager.repr_instance*.

repr_instance
An instance of reprlib.Repr for formatting Exception messages. The passed instance should have access to a bound callable by the name of repr, which in turn should produce a string representation of passed objects. If None, default the builtin repr() function.

Type reprlib.Repr, optional

repr_fallback
A fallback value in case *AssertionManager.repr_instance* is None.

Type Callable[[Any], str]

maxstring_fallback
A fallback value in case *AssertionManager.repr_instance* is None.

Type int

AssertionManager.**assert_**(*func: Callable[[...], T], *args: Any, invert: bool = False, exception: Optional[Type[Exception]] = None, post_process: Optional[Callable[[T], Any]] = None, message: Optional[str] = None, **kwargs: Any*) → None
Perform the following assertion: assert func(*args, **kwargs).

Examples

For example assert 5 == 5 is equivalent to AssertionManager().assert_(operator.eq, 5, 5).

Parameters

- **func** (Callable[..., T]) – The callable whose output will be evaluated.
- ***args** (Any) – Positional arguments for **func**.

Keyword Arguments

- **invert** (bool, optional) – If True, invert the output of the assertion: assert not func(*args, **kwargs).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation. The only dissalowed value is AssertionError.
- **post_process** (Callable[[T], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example functions would be the likes of any() and all().

- **message** (str, optional) – A custom error message to-be passed to the `assert` statement.
- ****kwargs** (Any, optional) – Keyword arguments for **func**.

Return type None

See also:

AssertionManager.__call__() Equivalent to `assert value`.

AssertionManager.**__call__**(*value: T*, ***, *invert: bool = False*, *post_process: Optional[Callable[[T], Any]] = None*, *message: Optional[str] = None*) → None

Equivalent to `assert value`.

Examples

```
>>> from assertionlib import assertion

>>> assertion(5 == 5)
>>> assertion(5 == 6)
Traceback (most recent call last):
    ...
AssertionError: output = (value); assert output

exception: AssertionError = 'None'

output: bool = False
value: bool = False
```

Parameters **value** (T) – The to-be asserted value.

Keyword Arguments

- **invert** (bool) – If `True`, invert the output of the assertion: `assert not value`.
- **post_process** (Callable[[T], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example functions would be the likes of `any()` and `all()`.
- **message** (str, optional) – A custom error message to-be passed to the `assert` statement.

Return type None

AssertionManager.**add_to_instance**(*func: Callable*, *name: Optional[str] = None*, *override_attr: bool = False*) → None

Add a new custom assertion method to this instance.

The new method name is added to `AssertionManager._PRIVATE_ATTR`.

Parameters **func** (Callable) – The callable whose output will be asserted in the to-be created method.

Keyword Arguments

- **name** (str, optional) – The name of the new method. If `None`, use the name of **func**.
- **override_attr** (bool) – If `False`, raise an `AttributeError` if a method with the same name already exists in this instance.

Return type None

Raises **AttributeError** – Raised if `override_attr=False` and a method with the same name already exists in this instance.

Assertions based on the builtin `operator` module

AssertionManager.**abs**(*a*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert abs(a)`.

Parameters **a** – The positional-only argument `a` of `abs()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not abs(a)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type None

See also:

abs() Same as abs(a).

AssertionManager.**add**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert add(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `add()`.
- **b** – The positional-only argument `b` of `add()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not add(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type None

See also:

add() Same as a + b.

AssertionManager.**and_**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert and_(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `and_()`.
- **b** – The positional-only argument `b` of `and_()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not and_(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

and_() Same as a & b.

AssertionManager.**concat**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert concat(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `concat()`.
- **b** – The positional-only argument `b` of `concat()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not concat(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

concat() Same as a + b, for a and b sequences.

AssertionManager.**contains**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert contains(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `contains()`.
- **b** – The positional-only argument `b` of `contains()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not contains(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

contains() Same as b in a (note reversed operands).

AssertionManager.**countOf**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert countOf(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `countOf()`.
- **b** – The positional-only argument `b` of `countOf()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not countOf(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

countOf() Return the number of times b occurs in a.

AssertionManager.**eq**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert eq(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `eq()`.
- **b** – The positional-only argument `b` of `eq()`.

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not eq(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

eq() Same as a == b.

AssertionManager.**floordiv**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert floordiv(a, b).

Parameters

- **a** – The positional-only argument a of floordiv().
- **b** – The positional-only argument b of floordiv().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not floordiv(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

floordiv() Same as a // b.

AssertionManager.**ge**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert ge(a, b).

Parameters

- **a** – The positional-only argument a of ge().
- **b** – The positional-only argument b of ge().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not ge(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.

- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

ge() Same as a >= b.

AssertionManager.**getitem**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: assert getitem(a, b).

Parameters

- **a** – The positional-only argument a of getitem().
- **b** – The positional-only argument b of getitem().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not getitem(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

getitem() Same as a[b].

AssertionManager.**gt**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: assert gt(a, b).

Parameters

- **a** – The positional-only argument a of gt().
- **b** – The positional-only argument b of gt().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not gt(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().

- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

gt() Same as a > b.

AssertionManager.**index**(*a*, */*, ***kwargs: Any*) → None
Perform the following assertion: assert index(a).

Parameters **a** – The positional-only argument a of index().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not index(a).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

index() Same as a.__index__()

AssertionManager.**indexOf**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: assert indexOf(a, b).

Parameters

- **a** – The positional-only argument a of indexOf().
- **b** – The positional-only argument b of indexOf().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not indexOf(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

indexOf() Return the first index of b in a.

AssertionManager.**inv**(*a*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert inv(a)`.

Parameters **a** – The positional-only argument `a` of `inv()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not inv(a)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

inv() Same as ~a.

AssertionManager.**invert**(*a*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert invert(a)`.

Parameters **a** – The positional-only argument `a` of `invert()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not invert(a)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

invert() Same as ~a.

AssertionManager.**is_**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert is_(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `is_()`.
- **b** – The positional-only argument `b` of `is_()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not is_(a, b)`.

- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

is_() Same as a is b.

AssertionManager.**is_not**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert is_not(a, b).

Parameters

- **a** – The positional-only argument a of is_not().
- **b** – The positional-only argument b of is_not().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not is_not(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

is_not() Same as a is not b.

AssertionManager.**le**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert le(a, b).

Parameters

- **a** – The positional-only argument a of le().
- **b** – The positional-only argument b of le().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not le(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.

- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

le() Same as a <= b.

AssertionManager.**lshift**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert lshift(a, b).

Parameters

- **a** – The positional-only argument a of lshift().
- **b** – The positional-only argument b of lshift().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not lshift(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

lshift() Same as a << b.

AssertionManager.**lt**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert lt(a, b).

Parameters

- **a** – The positional-only argument a of lt().
- **b** – The positional-only argument b of lt().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not lt(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().

- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

lt() Same as a < b.

AssertionManager.**matmul**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert matmul(a, b)`.

Parameters

- **a** – The positional-only argument a of matmul().
- **b** – The positional-only argument b of matmul().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: `assert not matmul(a, b)`.
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

matmul() Same as a @ b.

AssertionManager.**mod**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert mod(a, b)`.

Parameters

- **a** – The positional-only argument a of mod().
- **b** – The positional-only argument b of mod().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: `assert not mod(a, b)`.
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

mod() Same as a % b.

AssertionManager.**mul**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert mul(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `mul()`.
- **b** – The positional-only argument `b` of `mul()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not mul(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

mul() Same as a * b.

AssertionManager.**ne**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert ne(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `ne()`.
- **b** – The positional-only argument `b` of `ne()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not ne(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

ne() Same as a != b.

AssertionManager.**neg**(*a*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert neg(a)`.

Parameters **a** – The positional-only argument `a` of `neg()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not neg(a)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

neg() Same as -a.

AssertionManager.**not_**(*a*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert not_(a)`.

Parameters **a** – The positional-only argument `a` of `not_()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not not_(a)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

not_() Same as not a.

AssertionManager.**or_**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert or_(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `or_()`.
- **b** – The positional-only argument `b` of `or_()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not or_(a, b)`.

- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

or_() Same as a | b.

AssertionManager.**pos**(*a*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert pos(a).

Parameters **a** – The positional-only argument a of pos().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not pos(a).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

pos() Same as +a.

AssertionManager.**pow**(*a*, *b*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert pow(a, b).

Parameters

- **a** – The positional-only argument a of pow().
- **b** – The positional-only argument b of pow().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not pow(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

pow() Same as a ** b.

AssertionManager.**rshift**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert rshift(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `rshift()`.
- **b** – The positional-only argument `b` of `rshift()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not rshift(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type None

See also:

rshift() Same as a >> b.

AssertionManager.**sub**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert sub(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `sub()`.
- **b** – The positional-only argument `b` of `sub()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not sub(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type None

See also:

sub() Same as a - b.

AssertionManager.**truediv**(*a*, *b*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert truediv(a, b)`.

Parameters

- **a** – The positional-only argument `a` of `truediv()`.
- **b** – The positional-only argument `b` of `truediv()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not truediv(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

truediv() Same as a / b.

AssertionManager.**truth**(*a*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert truth(a)`.

Parameters **a** – The positional-only argument `a` of `truth()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not truth(a)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

truth() Return True if a is true, False otherwise.

AssertionManager.**length_hint**(*obj*, *default=0*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert length_hint(obj, default=default)`.

Parameters

- **obj** – The positional-only argument `obj` of `length_hint()`.

- **default** – The positional-only argument `default` of `length_hint()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not length_hint(obj, default=default)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

length_hint() Return an estimate of the number of items in obj. This is useful for presizing containers when building from an iterable. If the object supports len(), the result will be exact. Otherwise, it may over- or under-estimate by an arbitrary amount. The result will be an integer >= 0.

Assertions based on the builtin `os.path` module

`AssertionManager.`**isabs**(*s*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert isabs(s)`.

Parameters **s** – The positional-only argument `s` of `isabs()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not isabs(s)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

isabs() Test whether a path is absolute

`AssertionManager.`**isdir**(*s*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert isdir(s)`.

Parameters **s** – The positional-only argument `s` of `isdir()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not isdir(s)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.

- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

isdir() Return true if the pathname refers to an existing directory.

AssertionManager.**isfile**(*path*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert isfile(path).

Parameters **path** – The positional-only argument path of isfile().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not isfile(path).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

isfile() Test whether a path is a regular file

AssertionManager.**islink**(*path*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert islink(path).

Parameters **path** – The positional-only argument path of islink().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not islink(path).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

islink() Test whether a path is a symbolic link

AssertionManager.**ismount**(*path*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert ismount(path)`.

Parameters **path** – The positional-only argument `path` of `ismount()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not ismount(path)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

ismount() Test whether a path is a mount point

Assertions based on the builtin `math` module

AssertionManager.**allclose**(*a*, *b*, */*, ***, *rel_tol=1e-09*, *abs_tol=0.0*, ***kwargs: Any*) → None

Perform the following assertion: `assert isclose(a, b, rel_tol=rel_tol, abs_tol=abs_tol)`.

Parameters

- **a** – The positional-only argument `a` of `isclose()`.
- **b** – The positional-only argument `b` of `isclose()`.
- **rel_tol** – The keyword-only argument `rel_tol` of `isclose()`.
- **abs_tol** – The keyword-only argument `abs_tol` of `isclose()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not isclose(a, b, rel_tol=rel_tol, abs_tol=abs_tol)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

isclose() Determine whether two floating point numbers are close in value. rel_tol maximum difference for being considered "close", relative to the magnitude of the input values abs_tol maximum difference for being considered "close", regardless of the magnitude of the input values Return True if a is close in value to b, and False otherwise. For the values to be considered close, the difference between them must be smaller than at least one of the tolerances. -inf, inf and NaN behave similarly to the IEEE 754 Standard. That is, NaN is not close to anything, even itself. inf and -inf are only close to themselves.

AssertionManager.**isclose**(*a*, *b*, */*, *, *rel_tol=1e-09*, *abs_tol=0.0*, ***kwargs: Any*) → None

Perform the following assertion: `assert isclose(a, b, rel_tol=rel_tol, abs_tol=abs_tol)`.

Parameters

- **a** – The positional-only argument `a` of `isclose()`.
- **b** – The positional-only argument `b` of `isclose()`.
- **rel_tol** – The keyword-only argument `rel_tol` of `isclose()`.
- **abs_tol** – The keyword-only argument `abs_tol` of `isclose()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not isclose(a, b, rel_tol=rel_tol, abs_tol=abs_tol)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

isclose() Determine whether two floating point numbers are close in value. rel_tol maximum difference for being considered "close", relative to the magnitude of the input values abs_tol maximum difference for being considered "close", regardless of the magnitude of the input values Return True if a is close in value to b, and False otherwise. For the values to be considered close, the difference between them must be smaller than at least one of the tolerances. -inf, inf and NaN behave similarly to the IEEE 754 Standard. That is, NaN is not close to anything, even itself. inf and -inf are only close to themselves.

AssertionManager.**isfinite**(*x*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert isfinite(x)`.

Parameters **x** – The positional-only argument `x` of `isfinite()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not isfinite(x)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.

- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

isfinite() Return True if x is neither an infinity nor a NaN, and False otherwise.

AssertionManager.**isinf**(*x*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert isinf(x).

Parameters **x** – The positional-only argument x of isinf().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not isinf(x).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

isinf() Return True if x is a positive or negative infinity, and False otherwise.

AssertionManager.**isnan**(*x*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert isnan(x).

Parameters **x** – The positional-only argument x of isnan().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not isnan(x).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

isnan() Return True if x is a NaN (not a number), and False otherwise.

Assertions based on the builtin `builtins` module

AssertionManager.**callable**(*obj*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert callable(obj)`.

Parameters **obj** – The positional-only argument `obj` of `callable()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not callable(obj)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

callable()

Return whether the object is callable (i.e., some kind of function).

Note that classes are callable, as are instances of classes with a __call__() method.

AssertionManager.**hasattr**(*obj*, *name*, */*, ***kwargs: Any*) → None
Perform the following assertion: `assert hasattr(obj, name)`.

Parameters

- **obj** – The positional-only argument `obj` of `hasattr()`.
- **name** – The positional-only argument `name` of `hasattr()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not hasattr(obj, name)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

hasattr()

Return whether the object has an attribute with the given name.

This is done by calling getattr(obj, name) and catching AttributeError.

AssertionManager.**isinstance**(*obj*, *class_or_tuple*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert isinstance(obj, class_or_tuple)`.

Parameters

- **obj** – The positional-only argument `obj` of `isinstance()`.
- **class_or_tuple** – The positional-only argument `class_or_tuple` of `isinstance()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not isinstance(obj, class_or_tuple)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

isinstance()

Return whether an object is an instance of a class or of a subclass thereof.

A tuple, as in `isinstance(x, (A, B, ...))`, may be given as the target to check against. This is equivalent to `isinstance(x, A) or isinstance(x, B) or ...` etc.

AssertionManager.**issubclass**(*cls*, *class_or_tuple*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert issubclass(cls, class_or_tuple)`.

Parameters

- **cls** – The positional-only argument `cls` of `issubclass()`.
- **class_or_tuple** – The positional-only argument `class_or_tuple` of `issubclass()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not issubclass(cls, class_or_tuple)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

issubclass()

Return whether 'cls' is a derived from another class or is the same class.

A tuple, as in `issubclass(x, (A, B, ...))`, may be given as the target to check against. This is equivalent to `issubclass(x, A) or issubclass(x, B) or ...` etc.

`AssertionManager.`**`len`**(*obj*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert len(obj)`.

Parameters **obj** – The positional-only argument `obj` of `len()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not len(obj)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

len() Return the number of items in a container.

`AssertionManager.`**`any`**(*iterable*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert any(iterable)`.

Parameters **iterable** – The positional-only argument `iterable` of `any()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not any(iterable)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

any()

Return True if bool(x) is True for any x in the iterable.

If the iterable is empty, return False.

`AssertionManager.`**`all`**(*iterable*, */*, ***kwargs: Any*) → None

Perform the following assertion: `assert all(iterable)`.

Parameters **iterable** – The positional-only argument `iterable` of `all()`.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not all(iterable)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

all()

Return True if bool(x) is True for all values x in the iterable.

If the iterable is empty, return True.

AssertionManager.**isdisjoint**(*a: Iterable[Hashable], b: Iterable[Hashable], /, **kwargs: Any*) → None

Perform the following assertion: `assert isdisjoint(a, b)`.

Parameters

- **a** – The positional-only argument `a` of *isdisjoint()*.
- **b** – The positional-only argument `b` of *isdisjoint()*.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not isdisjoint(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

isdisjoint() Check if **a** has no elements in **b**.

AssertionManager.**issuperset**(*a: Iterable[Hashable], b: Iterable[Hashable], /, **kwargs: Any*) → None

Perform the following assertion: `assert issuperset(a, b)`.

Parameters

- **a** – The positional-only argument `a` of *issuperset()*.

- **b** – The positional-only argument b of *issuperset()*.

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not issuperset(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

issuperset() Check if **a** contains all elements from **b**.

AssertionManager.**issubset**(*a: Iterable[Hashable], b: Iterable[Hashable], /, **kwargs: Any*) → None

Perform the following assertion: assert issubset(a, b).

Parameters

- **a** – The positional-only argument a of *issubset()*.
- **b** – The positional-only argument b of *issubset()*.

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not issubset(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

issubset() Check if **b** contains all elements in **a**.

AssertionManager.**round**(*number*, */*, *, *ndigits=None*, ***kwargs: Any*) → None

Perform the following assertion: assert round(number, ndigits=ndigits).

Parameters

- **number** – The positional-only argument number of round().
- **ndigits** – The keyword-only argument ndigits of round().

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not round(number, ndigits=ndigits).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

round()

Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None. Otherwise the return value has the same type as the number. ndigits may be negative.

Miscellaneous assertions

AssertionManager.**len_eq**(*a: Sized*, *b: int*, */*, ***kwargs: Any*) → None

Perform the following assertion: assert len_eq(a, b).

Parameters

- **a** – The positional-only argument a of *len_eq()*.
- **b** – The positional-only argument b of *len_eq()*.

Keyword Arguments

- **invert** (bool) – If True, invert the output of the assertion: assert not len_eq(a, b).
- **exception** (type [Exception], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (Callable[[Any], bool], optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of any() and all().
- **message** (str, optional) – A custom error message to-be passed to the assert statement.

Return type None

See also:

len_eq() Check if the length of **a** is equivalent to **b**: len(a) == b.

AssertionManager.**str_eq**(*a: T*, *b: str*, */*, ***, *str_converter: Callable[[T], str] = <built-in function repr>*, ***kwargs: Any*) → None

Perform the following assertion: assert str_eq(a, b, str_converter=str_converter).

Parameters

- **a** – The positional-only argument a of *str_eq()*.

- **b** – The positional-only argument `b` of *str_eq()*.
- **str_converter** – The keyword-only argument `str_converter` of *str_eq()*.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not str_eq(a, b, str_converter=str_converter)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

str_eq() Check if the string-representation of **a** is equivalent to **b**: `repr(a) == b`.

AssertionManager.**shape_eq**(*a: numpy.ndarray, b: Union[numpy.ndarray, Tuple[int, ...]], /, **kwargs: Any*) → None

Perform the following assertion: `assert shape_eq(a, b)`.

Parameters

- **a** – The positional-only argument `a` of *shape_eq()*.
- **b** – The positional-only argument `b` of *shape_eq()*.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not shape_eq(a, b)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

shape_eq() Check if the shapes of **a** and **b** are equivalent: `a.shape == getattr(b, 'shape', b)`. **b** should be either an object with the `shape` attribute (*e.g.* a NumPy array) or a `tuple` representing a valid array shape.

AssertionManager.**function_eq**(*func1: function, func2: function, /, **kwargs: Any*) → None

Perform the following assertion: `assert function_eq(func1, func2)`.

Parameters

- **func1** – The positional-only argument `func1` of *function_eq()*.

- **func2** – The positional-only argument `func2` of *function_eq()*.

Keyword Arguments

- **invert** (`bool`) – If `True`, invert the output of the assertion: `assert not function_eq(func1, func2)`.
- **exception** (`type` [`Exception`], optional) – Assert that **exception** is raised during/before the assertion operation.
- **post_process** (`Callable[[Any], bool]`, optional) – Apply post-processing to the to-be asserted data before asserting aforementioned data. Example values would be the likes of `any()` and `all()`.
- **message** (`str`, optional) – A custom error message to-be passed to the `assert` statement.

Return type `None`

See also:

function_eq() Check if two functions are equivalent by checking if their `__code__` is identical. **func1** and **func2** should be instances of `FunctionType` or any other object with access to the `__code__` attribute.

2.2.2 assertionlib.ndrepr

A module for holding the *NDRepr* class, a subclass of the builtin `reprlib.Repr` class.

Index

NDRepr(**kwargs)	A subclass of `reprlib.Repr` with methods for handling additional object types.

Type-specific repr methods:

NDRepr.repr_float	Create a `str` representation of a `float` instance.
NDRepr.repr_Exception	Create a `str` representation of an :exc`Exception` instance.
NDRepr.repr_Signature	Create a `str` representation of a `Signature` instance.
NDRepr.repr_method	Create a `str` representation of a bound method.
NDRepr.repr_method_descriptor	Create a `str` representation of an unbound method.
NDRepr.repr_function	Create a `str` representation of a function.
NDRepr.repr_builtin_function_or_method	Create a `str` representation of a builtin function or method.
NDRepr.repr_type	Create a `str` representation of a `type` object.
NDRepr.repr_module	Create a `str` representation of a module.
NDRepr.repr_dict_keys	Create a `str` representation of a `KeysView`.
NDRepr.repr_dict_values	Create a `str` representation of a `ValuesView`.
NDRepr.repr_dict_items	Create a `str` representation of a `ItemsView`.
NDRepr.repr_Molecule	Create a `str` representation of a `plams.Molecule` instance.

continues on next page

Table 8 – continued from previous page

NDRepr.repr_Settings	Create a `str` representation of a `plams.Settings` instance.
NDRepr.repr_Atom	Create a `str` representation of a `plams.Atom` instance.
NDRepr.repr_Bond	Create a `str` representation of a `plams.Bond` instance.
NDRepr.repr_ndarray	Create a `str` representation of a `numpy.ndarray` instance.
NDRepr.repr_DataFrame	Create a `str` representation of a `pandas.DataFrame` instance.
NDRepr.repr_Series	Create a `str` representation of a `pandas.Series` instance.
NDRepr.repr_Dataset	Create a `str` representation of a `h5py.Dataset` instance.

API

class assertionlib.ndrepr.**NDRepr**(***kwargs: Union[int, Mapping[str, Any]]*)

A subclass of `reprlib.Repr` with methods for handling additional object types.

Has additional methods for handling:

- PLAMS Molecules, Atoms, Bonds and Settings
- NumPy arrays
- Pandas Series and DataFrames
- Callables

Parameters ****kwargs** (*object*) – User-specified values for one or more *NDRepr* instance attributes. An `AttributeError` is raised upon encountering unrecognized keys.

maxSignature

The maximum length of callables' signatures before further parameters are truncated. See also *NDRepr.repr_Signature()*.

Type `int`

maxfloat

The number of to-be displayed `float` decimals. See also *NDRepr.repr_float()*.

Type `int`

maxMolecule

The maximum number of to-be displayed atoms and bonds in PLAMS molecules. See also *NDRepr.repr_Molecule()*.

Type `int`

maxndarray

The maximum number of items in a `numpy.ndarray` row. Passed as argument to the `numpy.printoptions()` function:

- `threshold = self.maxndarray`
- `edgeitems = self.maxndarray // 2`

See also *NDRepr.repr_ndarray()*.

Type int

maxSeries
The maximum number of rows per pandas.Series instance. Passed as value to pandas.options.display.

- pandas.options.display.max_rows = self.series

See also *NDRepr.repr_Series()*.

Type int

maxDataFrame
The maximum number of rows per pandas.DataFrame instance. Passed as values to pandas.options.display:

- pandas.options.display.max_rows = self.maxdataframe
- pandas.options.display.max_columns = self.maxdataframe // 2

See also *NDRepr.repr_DataFrame()*.

Type int

np_printoptions
Additional keyword arguments for numpy.printoptions().

Note: Arguments provided herein will take priority over those specified internally in *NDRepr.repr_ndarray()*.

Type dict

pd_printoptions
Additional "keyword arguments" for pandas.options.

Note: Arguments provided herein will take priority over those specified internally in *NDRepr.repr_DataFrame()* and *NDRepr.repr_Series()*.

Type dict

NDRepr.**repr_float**(*obj: float*, *level: int*) → str
Create a str representation of a float instance.

NDRepr.**repr_Exception**(*obj: Exception*, *level: int*) → str
Create a str representation of an :exc`Exception` instance.

NDRepr.**repr_Signature**(*obj: inspect.Signature*, *level: int*) → str
Create a str representation of a Signature instance.

NDRepr.**repr_method**(*obj: builtins.method*, *level: int*) → str
Create a str representation of a bound method.

NDRepr.**repr_method_descriptor**(*obj: builtins.method_descriptor*, *level: int*) → str
Create a str representation of an unbound method.

NDRepr.**repr_function**(*obj: builtins.function*, *level: int*) → str
Create a str representation of a function.

NDRepr.**repr_builtin_function_or_method**(*obj: builtins.builtin_function_or_method*, *level: int*) → str
Create a str representation of a builtin function or method.

NDRepr.**repr_type**(*obj: type*, *level: int*) → str
Create a str representation of a type object.

NDRepr.**repr_module**(*obj: builtins.module*, *level: int*) → str
Create a str representation of a module.

NDRepr.**repr_dict_keys**(*obj: KeysView[Any]*, *level: int*) → str
Create a str representation of a KeysView.

NDRepr.**repr_dict_values**(*obj: ValuesView[Any]*, *level: int*) → str
Create a str representation of a ValuesView.

NDRepr.**repr_dict_items**(*obj: ItemsView[Any, Any]*, *level: int*) → str
Create a str representation of a ItemsView.

NDRepr.**repr_Molecule**(*obj: scm.plams.mol.molecule.Molecule*, *level: int*) → str
Create a str representation of a plams.Molecule instance.

NDRepr.**repr_Settings**(*obj: scm.plams.core.settings.Settings*, *level: int*) → str
Create a str representation of a plams.Settings instance.

NDRepr.**repr_Atom**(*obj: scm.plams.mol.molecule.Atom*, *level: int*) → str
Create a str representation of a plams.Atom instance.

NDRepr.**repr_Bond**(*obj: scm.plams.mol.molecule.Bond*, *level: int*) → str
Create a str representation of a plams.Bond instance.

NDRepr.**repr_ndarray**(*obj: numpy.ndarray*, *level: int*) → str
Create a str representation of a numpy.ndarray instance.

NDRepr.**repr_DataFrame**(*obj: pandas.core.frame.DataFrame*, *level: int*) → str
Create a str representation of a pandas.DataFrame instance.

NDRepr.**repr_Series**(*obj: pandas.core.series.Series*, *level: int*) → str
Create a str representation of a pandas.Series instance.

NDRepr.**repr_Dataset**(*obj: h5py._h1.dataset.Dataset*, *level: int*) → str
Create a str representation of a h5py.Dataset instance.

2.2.3 assertionlib.dataclass

A class with a number of generic pre-defined (magic) methods inspired by dataclass of Python 3.7.

Index

AbstractDataClass()	A dataclass with a number of generic pre-defined (magic) methods.
AbstractDataClass.__repr__()	Return a (machine readable) string representation of this instance.
AbstractDataClass.__eq__(value)	Check if this instance is equivalent to **value**.
AbstractDataClass.__hash__()	Return the hash of this instance.
AbstractDataClass.__copy__()	Return a shallow copy of this instance; see *AbstractDataClass.copy()*.

continues on next page

Table 9 – continued from previous page

AbstractDataClass.__deepcopy__([memo])	Return a deep copy of this instance; see *AbstractDataClass.copy()*.".
AbstractDataClass.copy([deep])	Return a shallow or deep copy of this instance.
AbstractDataClass.as_dict([return_private])	Construct a dictionary from this instance with all non-private instance variables.
AbstractDataClass.from_dict(dct)	Construct a instance of this objects' class from a dictionary with keyword arguments.
AbstractDataClass.inherit_annotations()	A decorator for inheriting annotations and docstrings.

API

class assertionlib.dataclass.**AbstractDataClass**

A dataclass with a number of generic pre-defined (magic) methods.

Provides methods for:

- String conversion: *AbstractDataClass.__repr__()*.
- Object comparisons: *AbstractDataClass.__eq__()*.
- Hash construction: *AbstractDataClass.__hash__()*.
- Copying: *AbstractDataClass.copy()*, *AbstractDataClass.__copy__()* and *AbstractDataClass.__deepcopy__()*.
- Dictionary interconversion: *AbstractDataClass.as_dict()* and *AbstractDataClass.from_dict()*.
- Inherting method docstrings and annotations: *AbstractDataClass.inherit_annotations()*.

_PRIVATE_ATTR

A class variable with the names of private instance variable. These attributes will be excluded whenever calling *AbstractDataClass.as_dict()*, printing or comparing objects. The set is unfrozen (and added as instance variables) the moment a class instance is initiated.

Type frozenset [str] or set [str]

_HASHABLE

A class variable denoting whether or not class instances are hashable. The *AbstractDataClass.__hash__* method will be unavailable if False.

Type bool

_hash

An attribute for caching the hash() of this instance. Only available if *AbstractDataClass._HASHABLE* is True.

Type int

AbstractDataClass.**__repr__**() → str

Return a (machine readable) string representation of this instance.

The string representation consists of this instances' class name in addition to all (non-private) instance variables.

Returns A string representation of this instance.

Return type str

See also:

AbstractDataClass._PRIVATE_ATTR A set with the names of private instance variables.

AbstractDataClass._repr_fallback Fallback function for *AbstractDataClass.__repr__()* incase of recursive calls.

AbstractDataClass._str_iterator() Return an iterable for the iterating over this instances' attributes.

AbstractDataClass._str() Returns a string representation of a single **key/value** pair.

AbstractDataClass.**__eq__**(*value: Any*) → bool
Check if this instance is equivalent to **value**.

The comparison checks if the class type of this instance and **value** are identical and if all (non-private) instance variables are equivalent.

Returns Whether or not this instance and **value** are equivalent.

Return type bool

See also:

AbstractDataClass._PRIVATE_ATTR A set with the names of private instance variables.

AbstractDataClass._eq Return if **v1** and **v2** are equivalent.

AbstractDataClass._eq_fallback Fallback function for *AbstractDataClass.__eq__()* incase of recursive calls.

AbstractDataClass.**__hash__**() → int
Return the hash of this instance.

The returned hash is constructed from two components: * The hash of this instances' class type. * The hashes of all key/value pairs in this instances' (non-private) attributes.

If an unhashable instance variable is encountered, *e.g.* a list, then its id() is used for hashing.

This method will raise a TypeError if the class attribute *AbstractDataClass._HASHABLE* is False.

See also:

AbstractDataClass._PRIVATE_ATTR A set with the names of private instance variables.

AbstractDataClass._HASHABLE Whether or not this class is hashable.

AbstractDataClass._hash_fallback Fallback function for *AbstractDataClass.__hash__()* incase of recursive calls.

AbstractDataClass._hash An instance variable for caching the hash() of this instance.

AbstractDataClass.**__copy__**() → AT
Return a shallow copy of this instance; see *AbstractDataClass.copy()*.

AbstractDataClass.**__deepcopy__**(*memo: Optional[Dict[int, Any]] = None*) → AT
Return a deep copy of this instance; see *AbstractDataClass.copy()*.".

AbstractDataClass.**copy**(*deep: bool = False*) → AT
Return a shallow or deep copy of this instance.

Parameters **deep** (bool) – Whether or not to return a deep or shallow copy.

Returns A new instance constructed from this instance.

Return type *AbstractDataClass*

AbstractDataClass.**as_dict**(*return_private: bool = False*) → Dict[str, Any]
Construct a dictionary from this instance with all non-private instance variables.

The returned dictionary values are shallow copies.

Parameters **return_private** (bool) – If True, return both public and private instance variables. Private instance variables are defined in *AbstractDataClass._PRIVATE_ATTR*.

Returns A dictionary with keyword arguments for initializing a new instance of this class.

Return type dict [str, Any]

See also:

AbstractDataClass.from_dict() Construct a instance of this objects' class from a dictionary with keyword arguments.

AbstractDataClass._PRIVATE_ATTR A set with the names of private instance variables.

classmethod AbstractDataClass.**from_dict**(*dct: Mapping[str, Any]*) → AT
Construct a instance of this objects' class from a dictionary with keyword arguments.

Parameters **dct** (Mapping [str, Any]) – A dictionary with keyword arguments for constructing a new *AbstractDataClass* instance.

Returns A new instance of this object's class constructed from **dct**.

Return type *AbstractDataClass*

See also:

AbstractDataClass.as_dict() Construct a dictionary from this instance with all non-private instance variables.

classmethod AbstractDataClass.**inherit_annotations**() → Callable[[FT], FT]
A decorator for inheriting annotations and docstrings.

Can be applied to methods of *AbstractDataClass* subclasses to automatically inherit the docstring and annotations of identical-named functions of its superclass.

References to *AbstractDataClass* are replaced with ones pointing to the respective subclass.

Returns A decorator for updating the annotations and docstring of a callable.

Return type type

Examples

```
>>> class SubClass(AbstractDataClass):
...
...     @AbstractDataClass.inherit_annotations()
...     def __copy__(self): pass

>>> print(SubClass.__copy__.__doc__)
Return a shallow copy of this instance; see :meth:`SubClass.copy`.

>>> print(SubClass.__copy__.__annotations__)
{'self': ~AT, 'return': ~AT}
```

2.2.4 assertionlib.functions

Various functions related to the *AssertionManager* class.

Index

get_sphinx_domain(func[, module_mapping])	Create a Sphinx domain for **func**.
create_assertion_doc(func)	Create a new NumPy style assertion docstring from the docstring of **func**.
bind_callable(class_type, func[, name, warn])	Take a callable and use it to create a new assertion method for **class_type**.
to_positional(func)	Decorate a function's `__signature__` such that all positional-or-keyword arguments are changed to either positional- or keyword-only.

API

assertionlib.functions.**get_sphinx_domain**(*func: Callable*, *module_mapping: Mapping[str, str] = mappingproxy({'genericpath': 'os.path', 'posixpath': 'os.path', '_operator': 'operator'})*) → str

Create a Sphinx domain for **func**.

Examples

```
>>> from collections import OrderedDict
>>> from assertionlib.functions import get_sphinx_domain

>>> value1: str = get_sphinx_domain(int)
>>> print(value1)
:class:`int<python:int>`

>>> value2: str = get_sphinx_domain(list.count)
>>> print(value2)
:meth:`list.count()<python:list.count>`

>>> value3: str = get_sphinx_domain(OrderedDict)
>>> print(value3)
:class:`~collections.OrderedDict`

>>> value4: str = get_sphinx_domain(OrderedDict.keys)
>>> print(value4)
:meth:`~collections.OrderedDict.keys`
```

Parameters

- **func** (Callable) – A class or (builtin) method or function.
- **module_mapping** (dict [str, str]) – A dictionary for mapping `__module__` values to actual module names. Useful for whenever there is a discrepancy between the two, *e.g.* the *genericpath* module of `os.path.join()`.

Returns A string with a valid Sphinx refering to **func**.

Return type str

Raises **TypeError** – Raised if **func** is neither a class or a (builtin) function or method.

assertionlib.functions.**create_assertion_doc**(*func: Callable*) → str

Create a new NumPy style assertion docstring from the docstring of **func**.

The summary of **funcs'** docstring, if available, is added to the "See also" section, in addition with an intersphinx-compatible link to **func**.

Examples

```
>>> from assertionlib.functions import create_assertion_doc

>>> docstring: str = create_assertion_doc(isinstance)
>>> print(docstring)
Perform the following assertion: :code:`assert isinstance(obj, class_or_tuple)`.

Parameters
----------
obj
    The positional-only argument ``obj`` of :func:`isinstance()<python:isinstance>
↪`.

class_or_tuple
    The positional-only argument ``class_or_tuple`` of :func:`isinstance()
↪<python:isinstance>`.


Keyword Arguments
-----------------
invert : :class:`bool`
    If :data:`True`, invert the output of the assertion: :code:`assert not␣
↪isinstance(obj, class_or_tuple)`.

exception : :class:`type` [:exc:`Exception`], optional
    Assert that **exception** is raised during/before the assertion operation.

post_process : :data:`Callable[[Any], bool]<typing.Callable>`, optional
    Apply post-processing to the to-be asserted data before asserting␣
↪aforementioned data.
    Example values would be the likes of :func:`any()<python:any>` and␣
↪:func:`all()<python:all>`.

message : :class:`str`, optional
    A custom error message to-be passed to the ``assert`` statement.


:rtype: :data:`None`

See also
--------
:func:`isinstance()<python:isinstance>`
        Return whether an object is an instance of a class or of a subclass␣
↪thereof.

    A tuple, as in ``isinstance(x, (A, B, ...))``, may be given as the target to
```

(continues on next page)

(continued from previous page)

```
    check against. This is equivalent to ``isinstance(x, A) or isinstance(x, B)
    or ...`` etc.
```

Parameters **func** (Callable) – A callable whose output is to-be asserted.

Returns A new docstring constructed from **funcs'** docstring.

Return type str

assertionlib.functions.**bind_callable**(*class_type: Union[type, Any]*, *func: Callable*, *name: Optional[str] = None*, *warn: bool = True*) → None

Take a callable and use it to create a new assertion method for **class_type**.

The created callable will have the same signature as **func** except for one additional keyword argument by the name of `func` (default value: `False`). Setting this keyword argument to `True` will invert the output of the assertion, *i.e.* it changes `assert func(...)` into `assert not func(...)`.

Examples

Supplying the builtin `len()` function will create (and bind) a callable which performs the `assert len(obj)` assertion.

Parameters

- **class_type** (type or Any) – A class (*i.e.* a type instance) or class instance.
- **func** (Callable) – A callable object whose output will be asserted by the created method.
- **name** (str, optional) – The name of the name of the new method. If None, use the name of **func**.

Return type None

assertionlib.functions.**to_positional**(*func: FT*) → FT

Decorate a function's `__signature__` such that all positional-or-keyword arguments are changed to either positional- or keyword-only.

Example

```
>>> from inspect import signature
>>> from assertionlib.functions import to_positional

>>> def func1(a: int, b: int = 0) -> int:
...     pass

>>> @to_positional
... def func2(a: int, b: int = 0) -> int:
...     pass

>>> print(signature(func1), signature(func2), sep='\n')
(a: int, b: int = 0) -> int
(a: int, /, *, b: int = 0) -> int
```

2.2.5 assertionlib.assertion_functions

A module with various new assertion functions.

Index

len_eq(a, b, /)	Check if the length of **a** is equivalent to **b**: `len(a) == b`.
str_eq(a, b, /, *[, str_converter])	Check if the string-representation of **a** is equivalent to **b**: `repr(a) == b`.
shape_eq(a, b, /)	Check if the shapes of **a** and **b** are equivalent: `a.shape == getattr(b, 'shape', b)`.
isdisjoint(a, b, /)	Check if **a** has no elements in **b**.
issuperset(a, b, /)	Check if **a** contains all elements from **b**.
issubset(a, b, /)	Check if **b** contains all elements in **a**.
function_eq(func1, func2, /)	Check if two functions are equivalent by checking if their `__code__` is identical.

API

`assertionlib.assertion_functions.`**`len_eq`**(*a: Sized*, *b: int*, */*) → bool
Check if the length of **a** is equivalent to **b**: `len(a) == b`.

`assertionlib.assertion_functions.`**`str_eq`**(*a: T*, *b: str*, */*, ***, *str_converter: Callable[[T], str] = <built-in function repr>*) → bool
Check if the string-representation of **a** is equivalent to **b**: `repr(a) == b`.

`assertionlib.assertion_functions.`**`shape_eq`**(*a: numpy.ndarray*, *b: Union[numpy.ndarray, Tuple[int, ...]]*, */*) → bool
Check if the shapes of **a** and **b** are equivalent: `a.shape == getattr(b, 'shape', b)`.

b should be either an object with the `shape` attribute (*e.g.* a NumPy array) or a `tuple` representing a valid array shape.

`assertionlib.assertion_functions.`**`isdisjoint`**(*a: Iterable[Hashable]*, *b: Iterable[Hashable]*, */*) → bool
Check if **a** has no elements in **b**.

`assertionlib.assertion_functions.`**`issuperset`**(*a: Iterable[Hashable]*, *b: Iterable[Hashable]*, */*) → bool
Check if **a** contains all elements from **b**.

`assertionlib.assertion_functions.`**`issubset`**(*a: Iterable[Hashable]*, *b: Iterable[Hashable]*, */*) → bool
Check if **b** contains all elements in **a**.

`assertionlib.assertion_functions.`**`function_eq`**(*func1: function*, *func2: function*, */*) → bool
Check if two functions are equivalent by checking if their `__code__` is identical.

func1 and **func2** should be instances of `FunctionType` or any other object with access to the `__code__` attribute.

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

E

F

G

H

I

L

M

N

O

P

R

S

T