doctestprinter

Release 1.3.0

D. Scheliga

Mar 19, 2021

CONTENTS

doctestprinter contains convenience functions to print outputs more adequate for doctests.

Example features:

- removes trailing whitespaces: pandas.DataFrame generates trailing whitespaces, which interferes with auto text 'trailing whitespace' removal features, leading to failed tests.
- maximum line width: break long sequences at whitespaces to a paragraph.

CHAPTER

ONE

INDICES AND TABLES

- genindex

CHAPTER

TWO

INSTALLATION

Either install the current release from pip ...

```
pip install doctestprinter
```

... or the latest development state of the gitlab repository.

```
$ pip install git+https://gitlab.com/david.scheliga/doctestprinter.git@dev --upgrade
```

doctestprinter.doctest_print(anything_to_print)	The general printing method for doctests.
doctestprinter.doctest_iter_print(..., [, ...])	Prints the first level of the iterable or mapping.
doctestprinter.EditingItem(editor, ...)	*EditingItem* is a wrapper class for the *edit_item* argument of `doctest_iter_print()`.
doctestprinter.prepare_print(anything_to_print)	Prepares anything for printing.
doctestprinter.prepare_pandas([...])	Builds explicitly string representations for *pandas.DataFrame* and *pandas.Series* objects.
doctestprinter.print_pandas([...])	Prints explicitly *pandas.DataFrame* and *pandas.Series* objects.
doctestprinter.round_collections(item_to_round)	Rounds items within collections (dict, list, tuple).
doctestprinter.repr_posix_path(any_path)	Represents the path on a Windows machine as a PosixPath representation turning back slashes to forward slashes.
doctestprinter.set_in_quotes(item)	Set the string represenation of anything in quotes.
doctestprinter.strip_base_path(...)	Strips the given *base path* from the *path to show* and performing `repr_posix_path()` on the result.
doctestprinter.strip_trailing_tabs(text)	Strips trailing tabs from the text.
doctestprinter.strip_trailing_whitespaces(text)	Strips trailing whitespaces from the text.
doctestprinter.strip_trailing_whitespaces_and_tabs(text)	Strips both trailing whitespaces and tabs from the text.

2.1 doctestprinter.doctest_print

doctestprinter.**doctest_print**(*anything_to_print: Any*, *max_line_width: Optional[int] = 0*, *indent: Optional[str] = None*)

The general printing method for doctests.

Notes

The argument *max_line_width* will break lines a whitespaces. If the single text exceeds the maximum linewidth it will not be broken within this implementation.

Parameters

- **anything_to_print** (*Any*) – Anything which will be converted into a string and post-processed, with default methods.
- **max_line_width** (*Optional[int]*) – Sets the maximum linewidth of the print.
- **indent** (*str*) – Additional indentation added to the docstring.

Examples

```
>>> test_text = (
...     "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed"
...     " do eiusmod tempor      incididunt ut labore et dolore magna"
...     " aliqua. Ut enim ad  minim veniam, quis nostrud exercitation"
...     " ullamco laboris  nisi ut  aliquip  ex ea commodo consequat."
... )
>>> doctest_print(test_text[:84])
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
>>> doctest_print(test_text, max_line_width=60)
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor      incididunt ut labore et dolore magna aliqua.
Ut enim ad  minim veniam, quis nostrud exercitation ullamco laboris
nisi ut  aliquip  ex ea commodo consequat.
>>> doctest_print(test_text, max_line_width=60, indent="    ")
    Lorem ipsum dolor sit amet, consectetur adipiscing elit,
    sed do eiusmod tempor      incididunt ut labore et dolore
    magna aliqua. Ut enim ad  minim veniam, quis nostrud exercitation
    ullamco laboris  nisi ut  aliquip  ex ea commodo consequat.
>>> small_list = list(iter("abcdefghij"))
>>> doctest_print(small_list)
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
>>> doctest_print(small_list, max_line_width=60)
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

2.2 doctestprinter.doctest_iter_print

doctestprinter.**doctest_iter_print**(*iterable_to_print: Union[Mapping, Iterable]*, *max_line_width: Optional[int] = 0*, *indent: Optional[str] = None*, *edits_item: Optional[Callable[[Any], Any]] = None*)

Prints the first level of the iterable or mapping.

Parameters

- **iterable_to_print** (*Union[Mapping, Iterable]*) – A Mapping or Iterable which first level will be iterated and printed.
- **max_line_width** (*Optional[int]*) – Sets the maximum linewidth of the print.
- **indent** (*Optional[str]*) – Additional indentation. The items of mappings will be indented additionally.
- **edits_item** (*Callable[[Any], Any]*) – A callable which takes the 1st level item and returing the state, which should be printed.

Examples

```
>>> sample_mapping = {"a": "mapping  ", "with": 3, "i": "tems  "}
>>> doctest_iter_print(sample_mapping)
a:
  mapping
with:
  3
i:
  tems
>>> doctest_iter_print(sample_mapping, indent="..")
..a:
....mapping
..with:
....3
..i:
....tems
>>> doctest_iter_print([1, 2, {"a": "mapping  ", "with": 3, "i": "tems  "}])
1
2
{'a': 'mapping  ', 'with': 3, 'i': 'tems  '}
>>> doctest_iter_print(["abc", "abcd", "abcde"], edits_item=lambda x: x[:3])
abc
abc
abc
>>> doctest_iter_print({"edit": 1, "item": 2}, edits_item=lambda x: x**2)
edit:
  1
item:
  4
>>> sample_dict = {"first_level": {"second": "level", "a number": 1.234567}}
>>> doctest_iter_print(sample_dict, edits_item=round_collections)
first_level:
  {'second': 'level', 'a number': 1.235}
```

2.3 doctestprinter.EditingItem

class doctestprinter.**EditingItem**(*editor: Callable*, **editor_args*, ***editor_kwargs*)

EditingItem is a wrapper class for the *edit_item* argument of *doctest_iter_print()*.

Parameters

- **editor** (*Callable[[Any], Any]*) – A callable which first argument is the item to be edited, before printing.
- **editor_args** – Arguments for the *editor*.
- **editor_kwargs** – Keyword arguments for the *editor*.

Examples

prepare_pandas accepts 4 arguments. *edits_item* of *doctest_iter_print()* takes a Callable and passes each item within the iteration as the first argument to the *editor*. Therefor *edits_pandas* allows to predefine the arguments for any Callable, which provides additional functionality.

```
>>> from doctestprinter import (
...     EditingItem, doctest_iter_print, prepare_pandas
... )
>>> from pandas import DataFrame
>>> frame_1 = DataFrame.from_records({"Alpha": 1, "Beta": 2}, index=[1.234])
>>> frame_2 = DataFrame({"Gamma":3, "Delta": 4}, index=[2.345])
>>> edits_pandas = EditingItem(prepare_pandas, formats="{:>.1f}#{:>8}")
>>> doctest_iter_print([frame_1, frame_2], edits_item=edits_pandas)
      Alpha      Beta
1.2       1         2
      Gamma     Delta
2.3       3         4
```

__init__(*editor: Callable*, **editor_args*, ***editor_kwargs*)

EditingItem is a wrapper class for the *edit_item* argument of *doctest_iter_print()*.

Parameters

- **editor** (*Callable[[Any], Any]*) – A callable which first argument is the item to be edited, before printing.
- **editor_args** – Arguments for the *editor*.
- **editor_kwargs** – Keyword arguments for the *editor*.

Examples

prepare_pandas accepts 4 arguments. *edits_item* of *doctest_iter_print()* takes a Callable and passes each item within the iteration as the first argument to the *editor*. Therefor *edits_pandas* allows to predefine the arguments for any Callable, which provides additional functionality.

```
>>> from doctestprinter import (
...     EditingItem, doctest_iter_print, prepare_pandas
... )
>>> from pandas import DataFrame
>>> frame_1 = DataFrame.from_records({"Alpha": 1, "Beta": 2}, index=[1.234])
>>> frame_2 = DataFrame({"Gamma":3, "Delta": 4}, index=[2.345])
```

(continues on next page)

(continued from previous page)

```
>>> edits_pandas = EditingItem(prepare_pandas, formats="{:>.1f}#{:>8}")
>>> doctest_iter_print([frame_1, frame_2], edits_item=edits_pandas)
        Alpha      Beta
1.2         1         2
        Gamma     Delta
2.3         3         4
```

Methods

__init__(editor, *editor_args, **editor_kwargs)	*EditingItem* is a wrapper class for the *edit_item* argument of *doctest_iter_print()*.

2.4 doctestprinter.prepare_print

doctestprinter.**prepare_print**(*anything_to_print: Any*, *max_line_width: Optional[int] = 0*, *indent: Optional[str] = None*) → str

Prepares anything for printing.

Notes

The argument *max_line_width* will break lines a whitespaces. If the single text exceeds the maximum linewidth it will not be broken within this implementation.

Parameters

- **anything_to_print** (*Any*) – Anything which will be converted into a string and post-processed, with default methods.
- **max_line_width** (*Optional[int]*) – Sets the maximum linewidth of the print.
- **indent** (*str*) – Additional indentation added to the docstring.

Returns str

Examples

```
>>> test_text = (
...     "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed"
...     " do eiusmod tempor      incididunt ut labore et dolore magna"
...     " aliqua. Ut enim ad  minim veniam, quis nostrud exercitation"
...     " ullamco laboris  nisi ut  aliquip  ex ea commodo consequat."
... )
>>> print(prepare_print(test_text[:84]))
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
>>> print(prepare_print(test_text, max_line_width=60))
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor      incididunt ut labore et dolore magna aliqua.
Ut enim ad  minim veniam, quis nostrud exercitation ullamco laboris
nisi ut  aliquip  ex ea commodo consequat.
>>> print(prepare_print(test_text, max_line_width=60, indent="    "))
    Lorem ipsum dolor sit amet, consectetur adipiscing elit,
```

(continues on next page)

(continued from previous page)

```
    sed do eiusmod tempor      incididunt ut labore et dolore
    magna aliqua. Ut enim ad  minim veniam, quis nostrud exercitation
    ullamco laboris  nisi ut  aliquip  ex ea commodo consequat.
>>> small_list = list(iter("abcdefghij"))
>>> print(prepare_print(small_list))
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
>>> print(prepare_print(small_list, max_line_width=60))
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

2.5 doctestprinter.prepare_pandas

doctestprinter.**prepare_pandas**(*frame_or_series: Optional = None*, *formats: Optional[str] = None*, *max_line_count: Optional[int] = None*, *max_title_width: Optional[int] = None*) → str

Builds explicitly string representations for *pandas.DataFrame* and *pandas.Series* objects.

Parameters

- **frame_or_series** – The DataFrame or Series to be represented.
- **formats** (*str; optional*) – Concatenated format specifiers for the index column(s) and value column(s). Default *{:>f}#{:>f}*
- **max_line_count** (*int; optional*) – Defines the max lines which should be print. Its half wide is used to show either head and tail. Default is 60.
- **max_title_width** (*int; optional*) – Maximum printed width of the column title. Default is 16.

Returns str

Notes

For further details see *print_pandas()*.

2.6 doctestprinter.print_pandas

doctestprinter.**print_pandas**(*frame_or_series: Optional = None*, *formats: Optional[str] = None*, *max_line_count: Optional[int] = None*, *max_title_width: Optional[int] = None*)

Prints explicitly *pandas.DataFrame* and *pandas.Series* objects.

Parameters

- **frame_or_series** – The DataFrame or Series to be print.
- **formats** (*str; optional*) – Concatenated format specifiers for the index column(s) and value column(s). Default *{:>f}#{:>f}*
- **max_line_count** (*int; optional*) – Defines the max lines which should be print. Its half wide is used to show either head and tail. Default is 60.
- **max_title_width** (*int; optional*) – Maximum printed width of the column title. Default is 16.

Warning: This method doesn't reproduce the exact representation as pandas does. This is intentional as this function was written to takle issues regarding usage of doctests in combination with pytest.

Notes

This function was written in response to failed doctests in pytest due to changed formatting behaviour in between running the doctest outside and inside pytest.

Other reasons are the different *NaN* value representation, representation of floats on different operating systems and lack of defining a string formatting behavior for each column.

In between different python versions, which changes between *nan* and *NaN*. In cases of running the tests on Linux and Windows on different machines the float representation in the footer of a Series print changes leading to failed doctests. First problem is solved by lowercase *NaN* to *nan* and second case is solved by a changed Series representation.

Examples

The default format specification is {:>g}. But the intention of this function is to define a specific format specification for each test to fix the doctest result for any python version within a tox test.

```
>>> single_index_test_frame = DataFrame(
...     np.linspace(1/3, 1000/3, num=4).reshape(2, 2),
...     columns=["x", "y"],
...     index=Index([0.1, 0.211], name="t")
... )
>>> print_pandas(single_index_test_frame)
                x        y
t
0.100    0.333333  111.333
0.211  222.333333  333.333
>>> print_pandas(single_index_test_frame, "{:>.1f}#{:>4.1f}{:>e}")
         x             y
t
0.1    0.3  1.113333e+02
0.2  222.3  3.333333e+02
```

The main task of *print_pandas()* is the possibility to fix the format of each column within a DataFrame.

```
>>> index_items = zip("aabb", np.linspace(0.0, 1/3, num=4))
>>> sample_index = MultiIndex.from_tuples(index_items, names=["x1", "x2"])
>>> sample_frame = DataFrame(
...     np.linspace(1/3, 1000/3, num=12).reshape(4, 3),
...     index=sample_index,
...     columns=["Alpha", "Beta", "G"]
... )
>>> print_pandas(sample_frame, "{:>4}{:>.2f}#{:.0f}{:.1e}{:.5g}")
           Alpha     Beta        G
x1    x2
   a  0.00      0  3.1e+01   60.879
      0.11     91  1.2e+02  151.697
   b  0.22    182  2.1e+02  242.515
      0.33    273  3.0e+02  333.333
```

2.7 doctestprinter.round_collections

doctestprinter.**round_collections**(*item_to_round: Any*, *digits: int = 3*) → Any

Rounds items within collections (dict, list, tuple). This method also supports object, which implements a *round(digits)* method.

Parameters

- **item_to_round** (*Any*) – An item with the potential to be rounded. If the item cannot be rounded it will be returned instead.
- **digits** (*int*) – Remaining digits of the rounded number. Default is 3.

Returns Any

Examples

```
>>> sample_dict = {"a_number": 1.234567, "not": "a number"}
>>> round_collections(item_to_round=sample_dict, digits=4)
{'a_number': 1.2346, 'not': 'a number'}
>>> sample_list = [1.234567, "not a number"]
>>> round_collections(item_to_round=sample_list, digits=4)
[1.2346, 'not a number']
>>> sample_tuple = (1.234567, "not a number")
>>> round_collections(item_to_round=sample_tuple, digits=4)
(1.2346, 'not a number')
```

Invoking the *round(digit)* method of objects.

```
>>> import numpy
>>> sample_array = numpy.array([1.234567])
>>> round_collections(item_to_round=sample_array, digits=4)
array([1.2346])
>>> from pandas import Series
>>> sample_series = Series([1.234567])
>>> round_collections(item_to_round=sample_series, digits=4)
0    1.2346
dtype: float64
>>> from pandas import DataFrame
>>> sample_frame = DataFrame([[1.234567, "not a number"]])
>>> round_collections(item_to_round=sample_frame, digits=4)
        0             1
0  1.2346  not a number
```

Or just do nothing.

```
>>> round_collections(item_to_round="nothing at all", digits=4)
'nothing at all'
```

2.8 doctestprinter.repr_posix_path

doctestprinter.**repr_posix_path**(*any_path: Union[str, pathlib.Path]*) → str

Represents the path on a Windows machine as a Posix-Path representation turning back slashes to forward slashes.

Examples

```
>>> repr_posix_path("c:\\a\\path")
'/c/a/path'
>>> repr_posix_path(".\\a\\path")
'./a/path'
>>> repr_posix_path(".\\a\\path")
'./a/path'
```

Parameters **any_path** (*str, Path*) – Any type of path representation.

Returns str

2.9 doctestprinter.set_in_quotes

doctestprinter.**set_in_quotes**(*item: Any*) → str

Set the string represenation of anything in quotes.

Parameters **item** (*Any*) – The item which will be set in quotes.

Returns str

Examples

```
>>> from doctestprinter import set_in_quotes
>>> print(set_in_quotes("   a  "))
'   a  '
>>> print(set_in_quotes(""))
''
>>> print(set_in_quotes("None"))
'None'
```

```
>>> from doctestprinter import doctest_iter_print
>>> sample = ["o   ", " o  ", "  o ", "   o"]
>>> doctest_iter_print(sample, edits_item=set_in_quotes)
'o   '
' o  '
'  o '
'   o'
```

2.10 doctestprinter.strip_base_path

doctestprinter.**strip_base_path**(*base_path_to_strip: Union[str, pathlib.Path]*, *path_to_show: Union[str, pathlib.Path]*) → str

Strips the given *base path* from the *path to show* and performing *repr_posix_path()* on the result.

Examples

```
>>> strip_base_path("/a/root/path", "/a/root/path/some/place")
'... /some/place'
>>> strip_base_path("\\a\\root\\path", "/a/root/path/some/place")
'... /some/place'
>>> strip_base_path("/a/root/path", "\\a\\root\\path\\some\\place")
'... /some/place'
```

Parameters

- **base_path_to_strip** – The base path, which should be removed from the view.
- **path_to_show** – The path which is going to be viewed.

Returns str

2.11 doctestprinter.strip_trailing_tabs

doctestprinter.**strip_trailing_tabs**(*text: str*) → str

Strips trailing tabs from the text. Introduced in 1.1.0.

Parameters **text** (*str*) – Text from which trailing tabs should be stripd.

Returns str

Examples

```
>>> sample_text = "A sample text with\t\n trailing tabs.\t\n\t"
>>> strip_trailing_tabs(sample_text)
'A sample text with\n trailing tabs.\n'
>>> sample_text = "A sample text with\t\n\ttrailing tabs.\t\nEnd.\t"
>>> strip_trailing_tabs(sample_text)
'A sample text with\n\ttrailing tabs.\nEnd.'
```

2.12 doctestprinter.strip_trailing_whitespaces

doctestprinter.**strip_trailing_whitespaces**(*text: str*) → str

Strips trailing whitespaces from the text. Introduced in 1.1.0.

Parameters **text** (*str*) – Text from which trailing whitespaces should be stripd.

Returns str

Examples

```
>>> sample_text = "A sample text with    \n trailing whitespaces.   \n   "
>>> strip_trailing_whitespaces(sample_text)
'A sample text with\n trailing whitespaces.'
>>> sample_text = "A sample text with    \n trailing whitespaces.   \nEnd. "
>>> strip_trailing_whitespaces(sample_text)
'A sample text with\n trailing whitespaces.\nEnd.'
```

2.13 doctestprinter.strip_trailing_whitespaces_and_tabs

doctestprinter.**strip_trailing_whitespaces_and_tabs**(*text: str*) → str

Strips both trailing whitespaces and tabs from the text. Introduced in 1.1.0.

Parameters **text** (*str*) – Text from which trailing tabs should be removed.

Returns str

Examples

```
>>> sample_text = "A sample text with    \n trailing whitespaces.   \n\t"
>>> strip_trailing_whitespaces_and_tabs(sample_text)
'A sample text with\n trailing whitespaces.'
>>> sample_text = "A sample text with    \n trailing whitespaces.   \nEnd.\t"
>>> strip_trailing_whitespaces_and_tabs(sample_text)
'A sample text with\n trailing whitespaces.\nEnd.'
>>> sample_text = "A sample text with    \n trailing whitespaces.   \n   "
>>> strip_trailing_whitespaces(sample_text)
'A sample text with\n trailing whitespaces.'
>>> sample_text = "A sample text with    \n trailing whitespaces.   \nEnd. "
>>> strip_trailing_whitespaces_and_tabs(sample_text)
'A sample text with\n trailing whitespaces.\nEnd.'
```

INDEX

Symbols

D

E

P

R

S