
pathwalker

Release 1.1.1

David Scheliga

Jul 13, 2021

CONTENTS

1	Installation	3
1.1	API reference	3
1.1.1	path_is_relative_to	3
1.1.2	keep_root_paths	4
1.1.3	walk_folder_paths	6
1.1.4	walk_file_paths	6
2	Basic Usage	9
3	Indices and tables	11
	Index	13

pathwalker is a micro module containing 2 helper methods for walking either directories ([*pathwalker.walk_folder_paths\(\)*](#)) or file paths ([*pathwalker.walk_file_paths\(\)*](#)) providing an additional filtering by an unix filepath pattern.

INSTALLATION

Install the latest release from pip.

```
$ pip install pathwalker
```

1.1 API reference

<code>pathwalker.path_is_relative_to(...)</code>	Return whether or not this path is relative to the <i>other path</i> .
<code>pathwalker.keep_root_paths(paths)</code>	Keeps root paths within a list of paths.
<code>pathwalker.walk_folder_paths(root_path[, ...])</code>	Yields only paths of directories.
<code>pathwalker.walk_file_paths(root_path[, ...])</code>	Yields only file paths.

1.1.1 path_is_relative_to

`pathwalker.path_is_relative_to(path_to_check: pathlib.Path, other_path: pathlib.Path) → bool`

Return whether or not this path is relative to the *other path*.

Parameters

- **path_to_check** – The path to check for being a sub path of the *other path*.
- **other_path** – The other path, which may be a parent path of the *path to check*.

Returns bool

Examples

```
>>> from pathwalker import path_is_relative_to
>>> from pathlib import Path
>>> path_is_relative_to(path_to_check=Path("/a/b"), other_path=Path("/a"))
True
>>> path_is_relative_to(path_to_check=Path("/a/b"), other_path=Path("/c"))
False
>>> path_is_relative_to(path_to_check=Path("/ab"), other_path=Path("/a"))
False
```

1.1.2 keep_root_paths

`pathwalker.keep_root_paths(paths: List[Union[str, pathlib.Path]]) → List[pathlib.Path]`

Keeps root paths within a list of paths. Sub paths are dropped.

Notes

The purpose of this method is to get the minimum list of paths for a path recursion afterwards. This should avoid listing the items of sub paths or double entries within the list.

Parameters `paths` – Any paths; which will be resolved within this process.

Returns Resolved paths.

Return type List[Path]

Examples

```
>>> from doctestprinter import doctest_iter_print
>>> from pathlib import Path
```

The root should remain for later recursion.

```
>>> sample_paths = (
...     "./tests/resources/foo",
...     "./tests/resources/bar",
...     "./tests/resources/",
...     "./tests/resources/foo/bar",
...     "./tests/resources/another_bar",
... )
>>> cleared_sample_paths = keep_root_paths(paths=sample_paths)
>>> current_work_path = Path(".").resolve()
>>> doctest_iter_print(
...     cleared_sample_paths,
...     edits_item=lambda x: x.relative_to(current_work_path)
... )
tests/resources
```

```
>>> sample_paths = (
...     "./tests/resources/foo",
...     "./tests/resources/bar",
...     "./tests/resources/foo/bar",
...     "./tests/resources/another_bar",
... )
>>> cleared_sample_paths = keep_root_paths(
...     paths=sample_paths
... )
>>> current_work_path = Path(".").resolve()
>>> doctest_iter_print(
...     cleared_sample_paths,
...     edits_item=lambda x: x.relative_to(current_work_path)
... )
tests/resources/another_bar
```

(continues on next page)

(continued from previous page)

```
tests/resources/bar
tests/resources/foo
```

Double entries are removed from the list leaving single tree roots only.

```
>>> samples = (
...     "./tests/resources/foo",
...     "./tests/resources/foo",
...     "./tests/resources/bar",
...     "./tests/resources/foo/bar",
...     "./tests/resources/foo/bar",
...     "./tests/resources/another_bar",
...     "./tests/resources/another_bar",
... )
>>> cleared_sample_paths = keep_root_paths(
...     paths=samples
... )
>>> current_work_path = Path(".").resolve()
>>> doctest_iter_print(
...     cleared_sample_paths,
...     edits_item=lambda x: x.relative_to(current_work_path)
... )
tests/resources/another_bar
tests/resources/bar
tests/resources/foo
```

Warnings: This method does resolve the paths. Non existing paths will not be dropped. Also this function will not raise a `FileNotExist-Error` for non existing paths.

```
>>> samples = (
...     "./tests/resources/foo",
...     "./not/existing",
...     "./not/existing/either",
... )
>>> cleared_sample_paths = keep_root_paths(
...     paths=samples
... )
>>> current_work_path = Path(".").resolve()
>>> doctest_iter_print(
...     cleared_sample_paths,
...     edits_item=lambda x: x.relative_to(current_work_path)
... )
not/existing
tests/resources/foo
```

1.1.3 walk_folder_paths

`pathwalker.walk_folder_paths(root_path: Union[str, pathlib.Path], filter_pattern: Optional[str] = None, recursive: bool = False) → Iterator[pathlib.Path]`

Yields only paths of directories.

Parameters

- **root_path** (*Path*) – Root path to walk through.
- **filter_pattern** (*str*) – Unix path pattern for filtering retrieved paths.
- **recursive** (*bool*) – Returns also paths of all sub folders.

Yields *Path*

Examples

```
>>> from doctestprinter import doctest_iter_print
>>> from pathwalker import walk_folder_paths
>>> found_folders = sorted(
...     walk_folder_paths("./tests", filter_pattern = "[!..]*"),
...     key=lambda x: str(x)
... )
>>> doctest_iter_print(found_folders)
tests/resources
```

```
>>> found_folders = sorted(
...     walk_folder_paths("./tests", filter_pattern = "[!..]*", recursive=True),
...     key=lambda x: str(x)
... )
>>> doctest_iter_print(found_folders)
tests/resources
tests/resources/another_bar
tests/resources/bar
tests/resources/foo
tests/resources/foo/bar
```

1.1.4 walk_file_paths

`pathwalker.walk_file_paths(root_path: Union[str, pathlib.Path], filter_pattern: Optional[str] = None, recursive: bool = False) → Generator[pathlib.Path, None, None]`

Yields only file paths.

Parameters

- **root_path** (*Path*) – Root path to walk through.
- **filter_pattern** (*str*) – Unix path pattern for filtering retrieved paths.
- **recursive** (*bool*) – Returns also paths of all sub folders.

Yields *Path*

Examples

```
>>> from doctestprinter import doctest_iter_print
>>> from pathwalker import walk_file_paths
>>> found_files = sorted(
...     walk_file_paths("tests/.", filter_pattern = "[!._]*.py", recursive=True),
...     key=lambda x: str(x)
... )
>>> doctest_iter_print(found_files)
tests/path_test.py
tests/resources/foo.py
tests/test_common_paths.py
```


BASIC USAGE

Walk through folders only.

```
>>> from pathwalker import walk_folder_paths
>>> for found_folder in walk_folder_paths(".", filter_pattern = "[!._]*"):
...     print(found_folder)
docs
tests
```

Walk through files only.

```
>>> from doctestprinter import doctest_iter_print
>>> from pathwalker import walk_file_paths
>>> found_files = sorted(
...     walk_file_paths(".", filter_pattern = "[!._]*.py", recursive=True),
...     key=lambda x: str(x)
... )
>>> doctest_iter_print(found_files)
docs/conf.py
pathwalker.py
setup.py
tests/path_test.py
```


INDICES AND TABLES

- `genindex`

INDEX

K

`keep_root_paths()` (*in module pathwalker*), [4](#)

P

`path_is_relative_to()` (*in module pathwalker*), [3](#)

W

`walk_file_paths()` (*in module pathwalker*), [6](#)

`walk_folder_paths()` (*in module pathwalker*), [6](#)