Chicken Turtle Util Documentation

Release 4.1.2

Tim Diels

Jun 12, 2017

Contents

Chicken Turtle Util (CTU) is a Python utility library. It was renamed to pytil, for the latest version, see pytil.

The API reference starts with an overview of all the features and then gets down to the nitty gritty details of each of them. Most of the reference provides examples. For a full overview of features see the module contents overview of the API reference and the table of contents of the user guide (in the sidebar) as they are complementary.

The API reference makes heavy use of a type language; for example, to describe exactly what arguments can be passed to a function.

Dependencies are grouped by module. For example, when using `chicken_turtle_util.data_frame`, you should `pip install 'chicken_turtle_util[data_frame]'`. To install dependencies of all modules, use `pip install 'chicken_turtle_util[all]'`. If you are not familiar with pip, see pip's quickstart guide.

While all features are documented and tested, the API is changed frequently. When doing so, the major version is bumped and a changelog is kept to help upgrade. Fixes will not be backported. It is recommended to pin the major version in your setup.py, e.g. for 2.x.y:

```
install_requires = ['chicken_turtle_util==2.*', ...]
```

Contents:

CHAPTER 1

API reference

See modules for a short description of each modules. For a full listing of the contents of all modules, see the module contents overview.

The API reference makes heavy use of a type language; for example, to describe exactly what arguments can be passed to a function.

Modules

algorithms	
asyncio	Extensions to asyncio.
click	*click* utilities
configuration	
data_frame	
debug	
dict	
exceptions	Exception classes: *UserException* and *InvalidOperationError*.
function	Function manipulation, like functools.
hashlib	hashlib additions
http	HTTP utilities.
inspect	Similar to *inspect* module.
iterable	Utility functions for working with iterables.
logging	Logging utilities.
multi_dict	multi-dict utilities. Multi-dicts can map keys to multiple values.
observable	Observable collections.
path	Extensions to pathlib.
pymysql	
series	

Continued on next page

Table 1.1 – continued from previous page

set	Set utilities.
sqlalchemy	
test	Test utilities.

chicken_turtle_util.asyncio

Extensions to asyncio.

Requires Python >=3.5

stubborn_gather	Stubbornly wait for awaitables, despite some of them raising

chicken_turtle_util.asyncio.**stubborn_gather**(**awaitables*)

Stubbornly wait for awaitables, despite some of them raising

Like a more stubborn version of asyncio.gather.

Continue until all awaitables have finished or have raised. If one or more awaitables raise, a new *Exception* is raised with the traceback and message of each exception as message. However, if all exceptions raised are *asyncio.CancelledError*, *asyncio.CancelledError* is raised instead.

If cancelled, cancels the (futures associated with the) awaitables.

Parameters **awaitables** : iterable(awaitable)

Awaitables to await

Returns results :: (any, ...)

Return of each awaitable. The return of `awaitables[i]` is `results[i]`.

chicken_turtle_util.click

click utilities

argument	Like *click.argument*, but by default `required=True`
assert_runs	Invoke click command and assert it completes successfully
option	Like *click.option*, but by default `show_default=True, required=True`
password_option	Like click.option, but by default `prompt=True, hide_input=True, show_default=False, required=True`.

chicken_turtle_util.click.**assert_runs**(**args*, ***kwargs*)

Invoke click command and assert it completes successfully

Parameters ***args, **kwargs**

CliRunner.invoke arguments, excluding catch_exceptions

Returns **result** : click.testing.Result

chicken_turtle_util.exceptions

Exception classes: *UserException* and *InvalidOperationError*.

If you miss the ability to pass args to any of these exceptions, note that you actually can. For example:

```
>>> ex = Exception(1, 2, 3)
>>> ex.args
(1, 2, 3)
```

You can only use positional arguments though.

`exc_info`	Get exc_info tuple from exception
`InvalidOperationError`	When an operation is illegal/invalid (in the current state), regardless of what arguments you throw at it.
`UserException`	Exception with message to show the user.

exception `chicken_turtle_util.exceptions.`**`InvalidOperationError`**

When an operation is illegal/invalid (in the current state), regardless of what arguments you throw at it.

An operation is a method/function call, the getting or setting of an attribute.

When the issue is with an argument, use ValueError, not this.

exception `chicken_turtle_util.exceptions.`**`UserException`**(*message*, **args*)

Exception with message to show the user.

Parameters **message** : str

User-friendly message

chicken_turtle_util.function

Function manipulation, like functools. Contains only *compose*, compose functions.

`compose`	Compose functions

`chicken_turtle_util.function.`**`compose`**(**functions*)

Compose functions

Like the `o` operator in math.

Parameters **functions** : collection(any -> any)

Collection of one or more functions to compose.

Returns any -> any

Function composed of *functions*

Raises **ValueError**

When `len(functions) < 1`

Examples

`compose(f1, f2)` is equivalent to `f1 o f2`, or to `lambda x: f1(f2(x))`

chicken_turtle_util.hashlib

hashlib additions

base85_digest	Get base 85 encoded digest of hash

chicken_turtle_util.hashlib.**base85_digest**(*hash_*)

Get base 85 encoded digest of hash

Parameters **hash_** : hash

hashlib hash object. E.g. the return of hashlib.sha512()

Returns str

base 85 encoded digest

chicken_turtle_util.http

HTTP utilities. Contains only *download*, download a http resource.

download	Download an HTTP resource to a file

chicken_turtle_util.http.**download**(*url*, *destination*)

Download an HTTP resource to a file

Parameters **url** : str

HTTP resource to download

destination : pathlib.Path

Location at which to store downloaded resource. If *destination* does not exist, it's assumed to be a file path. If *destination* exists and is a file, it is overwritten. If *destination* exists and is a directory, the file will be saved inside the directory with as name the file name suggested by a server, if any, or the last part of the URL otherwise (excluding query and fragment parts).

Returns **path** : pathlib.Path

Path to the downloaded file.

name : str or None

File name suggested by the server or None if none was suggested.

chicken_turtle_util.inspect

Similar to *inspect* module. Contains only function call inspect utilities

call_args	Get function call arguments as a single dict

chicken_turtle_util.inspect.**call_args**(*f*, *args=()*, *kwargs={}*)

Get function call arguments as a single dict

Parameters **f** : function

The function of the function call

args : iterable(any)

Arguments of the function call

kwargs : {str => any}

Keyword arguments of the function call

Returns {arg_name :: str => arg_value :: any}

Dict of arguments including *args*, *kwargs* and any missing optional arguments of *f*.

Examples

```
>>> def f(a=1, *my_args, k=None, **kwargs):
...     pass
...
>>> call_args(f)
{'a': 1, 'k': None, '*args': ()}
>>> call_args(f, [3])
{'a': 3, 'k': None, '*args': ()}
>>> call_args(f, [3], dict(k='some'))
{'a': 3, 'k': 'some', '*args': ()}
>>> call_args(f, [3, 4])
{'a': 3, 'k': None, '*args': (4,)}
>>> call_args(f, dict(other='some'))
{'a': 1, 'k': None, 'other': 'some', '*args': ()}
>>> def g():
...     pass
...
>>> call_args(g)
{}
```

chicken_turtle_util.iterable

Utility functions for working with iterables. *sliding_window*, ...

See also

itertools more_itertools

flatten	Flatten shallowly zero or more times
is_sorted	Get whether iterable is sorted ascendingly
partition	Split iterable into partitions
sliding_window	Iterate using a sliding window

chicken_turtle_util.iterable.**flatten**(*iterable*, *times=1*)

Flatten shallowly zero or more times

Does not flatten *str* and *bytes*. Order is stably maintained (i.e. no 2 items swap places, even if they're equal).

Parameters **iterable** : iterable(any) except str or bytes

Iterable to flatten. May be any iterable other than *str* or *bytes*. May have irregular depth.

times : int, optional

The number of times to flatten shallowly or, equivalently, the number of levels of depth to remove. Should be 0 or more.

Yields any

Items of iterable flattened to depth `depth(iterable) - times`

Raises **ValueError**

If input is invalid.

Examples

```
>>> list(flatten([[2, 3], 1, [5, [7, 8]]]))
[2, 3, 1, 5, [7, 8]]
```

```
>>> list(flatten([[2, 3], 1, [5, [7, 8]]], times=2))
[2, 3, 1, 5, 7, 8]
```

```
>>> list(flatten([[2, 3], 1, [5, [7, 8]]], times=3))
[2, 3, 1, 5, 7, 8]
```

```
>>> flatten([iter([2, 3]), 1, [5, iter([7, 8])]])
iter([2, 3, 1, 5, iter([7, 8])])
```

```
>>> list(flatten([[2, 3], 1, [5, [7, 8]]], times=0))
[[2, 3], 1, [5, [7, 8]]]
```

chicken_turtle_util.iterable.**is_sorted**(*iterable*)

Get whether iterable is sorted ascendingly

Parameters **iterable** : iterable(comparable)

Iterable whose ordering to check

Returns bool

Whether iterable is sorted

chicken_turtle_util.iterable.**partition**(*iterable*, *key*)

Split iterable into partitions

Parameters **iterable** : iterable(item :: any)

Iterable to split into partitions

key : (item :: any) -> (partition_id :: any)

Function that assigns an item of the iterable to a partition

Returns **partitioning** : {(partition_id :: any)

Partitioning. Ordering of items is maintained within each *partition*. I.e. each *partition* is a subsequence of *iterable*.

chicken_turtle_util.iterable.**sliding_window**(*iterable*, *size=2*)

Iterate using a sliding window

Parameters **iterable** : iterable(any)

Iterable to slide a window across

size : int, optional

Window size

Yields (any, ...)

Iterator slices of size *size*, taken from start to end through the iterator.

Raises **ValueError**

When `ilen(iterable) < size or size < 1`

See also:

`more_itertools.chunked` Divide iterable into (non-overlapping) chunks of given size

Examples

```
>>> list(sliding_window(range(4)))
[(0,1), (1,2), (2,3)]
```

```
>>> list(sliding_window(range(4), size=3))
[(0,1,2), (1,2,3)]
```

```
>>> list(sliding_window(range(1)))
[]
```

chicken_turtle_util.logging

Logging utilities.

configure	Configure root logger to log INFO to stderr and DEBUG to log file.
set_level	Temporarily change log level of logger

chicken_turtle_util.logging.**configure**(*log_file*)

Configure root logger to log INFO to stderr and DEBUG to log file.

The log file is appended to. Stderr uses a terse format, while the log file uses a verbose unambiguous format.

Root level is set to INFO.

Parameters **log_file** : Path

File to log to

Returns **stderr_handler** : logging.StreamHandler

Handler that logs to stderr

file_handler : logging.FileHandler

Handler that logs to log_file

chicken_turtle_util.logging.**set_level**(*logger*, *level*)

Temporarily change log level of logger

Parameters **logger** : str or Logger

Logger name

level

Log level to set

Examples

```
>>> with set_level('sqlalchemy.engine', logging.INFO):
...     pass # sqlalchemy log level is set to INFO in this block
```

chicken_turtle_util.multi_dict

multi-dict utilities. Multi-dicts can map keys to multiple values.

A multi-dict (or multi map) is a dict that maps each key to one or more values.

MultiDict	A multi-dict view of a `{hashable => {hashable}}` dict.

class chicken_turtle_util.multi_dict.**MultiDict**(*dict_*)

A multi-dict view of a `{hashable => {hashable}}` dict.

A light wrapper offering a few methods for working with multi-dicts.

Parameters **dict_** : {hashable => {hashable}}

Dict to access as a multi-dict

Notes

A multi-dict (or multi map) is a dict that maps each key to one or more values.

`MultiDict`s provided by other libraries tend to be more feature rich, while this interface is far more conservative. Instead of wrapping, they provide an interface that mixes regular and multi-dict access. Additionally, other `MultiDict`'s map keys to lists of values, allowing a key to map to the same value multiple times.

chicken_turtle_util.observable

Observable collections. Only contains *Set* currently.

Set	Observable set

class chicken_turtle_util.observable.**Set**(**args*, ***kwargs*)

Observable set

chicken_turtle_util.path

Extensions to pathlib.

assert_equals	Assert 2 files are equal
assert_mode	Assert last 3 octal mode digits match given mode exactly
chmod	Change file mode bits
hash	Hash file or directory
read	Get file contents
remove	Remove file or directory (recursively), unless it's missing
write	Create or overwrite file with contents

chicken_turtle_util.path.**assert_mode**(*path*, *mode*)

Assert last 3 octal mode digits match given mode exactly

Parameters **path** : pathlib.Path

Path whose mode to assert

mode : int

Expected mode

chicken_turtle_util.path.**chmod**(*path*, *mode*, *operator='='*, *recursive=False*)

Change file mode bits

When recursively chmodding a directory, executable bits in *mode* are ignored when applying to a regular file. E.g. `chmod(path, mode=0o777, recursive=True)` would apply `mode=0o666` to regular files.

Symlinks are ignored.

Parameters **path** : Path

Path to chmod

mode : int

Mode bits to apply, e.g. `0o777`.

operator : '+' or '-' or '='

How to apply the mode bits to the file. If '=', assign mode, if '+', add to current mode, if '-', subtract from current mode.

recursive : bool

Whether to chmod recursively. If recursive, applies modes in a top-down fashion, like the chmod command.

chicken_turtle_util.path.**hash**(*path*, *hash_function=<built-in function openssl_sha512>*)

Hash file or directory

Parameters **path** : pathlib.Path

File or directory to hash

hash_function : () -> hash

Function which returns hashlib hash objects

Returns hash

hashlib hash object of file/directory contents. File/directory stat data is ignored. The directory digest covers file/directory contents and their location relative to the directory being digested. The directory name itself is ignored.

chicken_turtle_util.path.**read**(*path*)

Get file contents

Parameters **path** : pathlib.Path

Path of file to read

Returns str

File contents

chicken_turtle_util.path.**remove**(*path*, *force=False*)

Remove file or directory (recursively), unless it's missing

On NFS file systems, if a directory contains .nfs* temporary files (sometimes created when deleting a file), it waits for them to go away.

Parameters **path** : Path

Path to remove

force : bool

If True, will remove files and directories even if they are read-only (as if first doing chmod -R +w)

chicken_turtle_util.path.**write**(*path*, *contents*, *mode=None*)

Create or overwrite file with contents

Missing parent directories of *path* will be created.

Parameters **path** : pathlib.Path

Path to file to write to

contents : str

Contents to write to file

mode : int or None

If set, also chmod file

chicken_turtle_util.set

Set utilities. Contains only *merge_by_overlap*, merges overlapping sets in place.

merge_by_overlap	Of a list of sets, merge those that overlap, in place.

chicken_turtle_util.set.**merge_by_overlap**(*sets*)

Of a list of sets, merge those that overlap, in place.

The result isn't necessarily a subsequence of the original *sets*.

Parameters **sets** : [{any}]

Sets of which to merge those that overlap. Empty sets are ignored.

Notes

Implementation is based on this StackOverflow answer. It outperforms all other algorithms in the thread (visited at dec 2015) on python3.4 using a wide range of inputs.

Examples

```
>>> merge_by_overlap([{1,2}, set(), {2,3}, {4,5,6}, {6,7}])
[{1,2,3}, {4,5,6,7}]
```

chicken_turtle_util.test

Test utilities.

assert_text_contains	Assert long string contains given string
assert_text_equals	Assert long strings are equal
assert_matches	
assert_search_matches	
temp_dir_cwd	pytest fixture that sets current working directory to a temporary directory

chicken_turtle_util.test.**temp_dir_cwd**(*tmpdir*)
pytest fixture that sets current working directory to a temporary directory

chicken_turtle_util.test.**assert_text_contains**(*whole*, *part*)
Assert long string contains given string

chicken_turtle_util.test.**assert_text_equals**(*actual*, *expected*)
Assert long strings are equal

chicken_turtle_util.test.**assert_matches**(*actual*, *pattern*, *flags=0*)

chicken_turtle_util.test.**assert_search_matches**(*actual*, *pattern*, *flags=0*)

Module contents overview

algorithms

multi_way_partitioning	
spread_points_in_hypercube	
toset_from_tosets	

asyncio

stubborn_gather	Stubbornly wait for awaitables, despite some of them raising

click

argument	Like *click.argument*, but by default `required=True`
assert_runs	Invoke click command and assert it completes successfully

Continued on next page

Table 1.18 – continued from previous page

option	Like *click.option*, but by default show_default=True, required=True
password_option	Like click.option, but by default prompt=True, hide_input=True, show_default=False, required=True.

configuration

ConfigurationLoader

data_frame

assert_equals
equals
replace_na_with_none
split_array_like

debug

pretty_memory_info

dict

pretty_print_head
DefaultDict
invert
assign

exceptions

exc_info	Get exc_info tuple from exception
UserException	Exception with message to show the user.
InvalidOperationError	When an operation is illegal/invalid (in the current state), regardless of what arguments you throw at it.

function

compose	Compose functions

hashlib

base85_digest	Get base 85 encoded digest of hash

http

download	Download an HTTP resource to a file

inspect

call_args	Get function call arguments as a single dict

iterable

sliding_window	Iterate using a sliding window
partition	Split iterable into partitions
is_sorted	Get whether iterable is sorted ascendingly
flatten	Flatten shallowly zero or more times

logging

configure	Configure root logger to log INFO to stderr and DEBUG to log file.
set_level	Temporarily change log level of logger

multi_dict

MultiDict	A multi-dict view of a `{hashable => {hashable}}` dict.

observable

Set	Observable set

path

assert_equals	Assert 2 files are equal
assert_mode	Assert last 3 octal mode digits match given mode exactly
chmod	Change file mode bits
hash	Hash file or directory
read	Get file contents
remove	Remove file or directory (recursively), unless it's missing

Continued on next page

Table 1.32 – continued from previous page

write	Create or overwrite file with contents

pymysql

patch

series

assert_equals
equals
invert
split

set

merge_by_overlap	Of a list of sets, merge those that overlap, in place.

sqlalchemy

log_sql
pretty_sql

test

assert_text_contains	Assert long string contains given string
assert_text_equals	Assert long strings are equal
assert_matches	
assert_search_matches	
temp_dir_cwd	pytest fixture that sets current working directory to a temporary directory

CHAPTER 2

Python type language

When documenting code, it is often necessary to refer to the type of an argument or a return. Here, I introduce a language for doing so in a semi-formal manner.

First off, I define these pseudo-types:

- iterable: something you can iterate over once (or more) using *iter*
- iterator: something you can call *next* on
- collection: something you can iterate over multiple times

I define the rest of the type language through examples:

```
pathlib.Path
```

Expects a *pathlib.Path*-like, i.e. anything that looks like a *pathlib.Path* (duck typing) is allowed. None is not allowed.

```
exact(pathlib.Path)
```

Expects a *Path* or derived class instance, so no duck typing (and no None).

```
pathlib.Path or None
```

Expect a *pathlib.Path*-like or None. When None is allowed it must be explicitly specified like this.

```
bool or int
```

Expect a boolean or an int.

```
{bool}
```

A set of booleans.

```
{any}
```

A set of anything.

```
{'apples' => bool, 'name' => str}
```

A dictionary with keys 'apples' and 'name' which respectively have a boolean and a string as value. (Note that the `:` token is already used by Sphinx, and -> is usually used for lambdas, so we use => instead).

```
dict(apples=bool, name=str)
```

Equivalent to the previous example.

```
Parameters
----------
field : str
dict_ : {field => bool}
```

A dictionary with one key, specified by the value of *field*, another parameter (but can be any expression, e.g. a global).

```
{apples => bool, name => str}
```

Not equivalent to the apples dict earlier. *apples* and *name* are references to the value used as a key.

```
(bool,)
```

Tuple of a single bool.

```
[bool]
```

List of 0 or more booleans.

```
[(bool, bool)]
```

List of tuples of boolean pairs.

```
[(first :: bool, second :: bool)]
```

Equivalent type compared to the previous example, but you can more easily refer to the first and second bool in your parameter description this way.

```
{item :: int}
```

Set of int. We can refer to the set elements as *item*.

```
iterable(bool)
```

Iterable of bool. Something you can call *iter* on.

```
iterator(bool)
```

Iterator of bool. Something you can call *next* on.

```
type_of(expression)
```

Type of expression, avoid when possible in order to be as specific as possible.

```
Parameters
----------
a : SomeType
b : type_of(a.nodes[0].key_function)
```

b has the type of the retrieved function.

```
(int, str, k=int) -> bool
```

Function that takes an int and a str as positional args, an int as keyword arg named 'k' and returns a bool.

```
func :: int -> bool
```

Function that takes an int and returns a bool. We can refer to it as *func*.

CHAPTER 3

Developer documentation

Documentation for developers/contributors of Chicken Turtle Util.

The project follows a simple project structure and associated workflow. Please read its documentation.

Project decisions

API design

If it's a path, expect a *pathlib.Path*, not a *str*.

If extending a module from another project, e.g. *pandas*, use the same name as the module. While a `from pandas import *` would allow the user to access functions of the real pandas module through the extended module, we have no control over additions to the real pandas, which could lead to name clashes later on, so don't.

Decorators and context managers should not be provided directly but should be returned by a function. This avoids confusion over whether or not parentheses should be used `@f` vs `@f()`, and parameters can easily be added in the future.

If a module is a collection of instances of something, give it a plural name, else make it singular. E.g. *exceptions* for a collection of *Exception* classes, but *function* for a set of related functions operating on functions.

API implementation

Do not prefix imports with underscore. When importing things, they also are exported, but *help* or Sphinx documentation will not include them and thus a user should realise they should not be used. E.g. `import numpy as np` in *module.py* can be accessed with *module.np*, but it isn't mentioned in *help(module)* or Sphinx documentation.

CHAPTER 4

Changelog

Semantic versioning is used (starting with v3.0.0).

4.1.2

Announce rename to pytil.

4.1.1

- Fixes:
  - add missing keys to `extras_require`: `hashlib`, `multi_dict`, `test`

4.1.0

- Backwards incompatible changes: None
- Enhancements/additions:
  - `click.assert_runs`: pass on extra args to click's `invoke()`
  - `path.chmod`, `path.remove`: ignore disappearing children instead of raising
  - Add `exceptions.exc_info`: exc_info tuple as seen in function parameters in the `traceback` standard module
  - Add `extras_require['all']` to `setup.py`: union of all extra dependencies
- Fixes:
  - `path.chmod`: do not follow symlinks
  - `iterable.flatten`: removed debug prints: `+`, `-`

- Internal / implementation details:
  - use simple project structure instead of Chicken Turtle Project
  - `pytest-catchlog` instead of `pytest-capturelog`
  - `extras_require['dev']`: test dependencies were missing
  - `test_http` created `existing_file` in working dir instead of in test dir

v4.0.1

- Fixed: README formatting error

v4.0.0

- Major:
  - `path.digest` renamed to `path.hash` (and added `hash_function` parameter)
  - renamed `cli` to `click`
  - require Python 3.5 or newer
  - Changed: `asyncio.stubborn_gather`:
    - raise `CancelledError` if all its awaitables raised `CancelledError`.
    - raise summary exception if any awaitable raises exception other than `CancelledError`
    - log exceptions, as soon as they are raised
- Minor:
  - Added:
    - `click.assert_runs`
    - `hashlib.base85_digest`
    - `logging.configure`
    - `path.assert_equals`
    - `path.assert_mode`
    - `test.assert_matches`
    - `test.assert_search_matches`
    - `test.assert_text_contains`
    - `test.assert_text_equals`
- Fixes:
  - `path.remove`: raised when `path.is_symlink()` or contains a symlink
  - `path.digest/hash`: directory hash collisions were more likely than necessary
  - `pymysql.patch`: change was not picked up in recent pymysql versions

v3.0.1

- Fixed: README formatting error

v3.0.0

- Removed:
  - `cli.Context`, `cli.BasicsMixin`, `cli.DatabaseMixin`, `cli.OutputDirectoryMixin`
  - `pyqt` module
  - `URL_MAX_LENGTH`
  - `various` module: `Object`, `PATH_MAX_LENGTH`
- Enhanced:
  - `data_frame.split_array_like`: `columns` defaults to `df.columns`
  - `sqlalchemy.pretty_sql`: much better formatting
- Added:
  - `algorithms.toset_from_tosets`: Create totally ordered set (toset) from tosets
  - `configuration.ConfigurationLoader`: loads a single configuration from one or more files directory according to XDG standards
  - `data_frame.assert_equals`: Assert 2 data frames are equal
  - `data_frame.equals`: Get whether 2 data frames are equal
  - `dict.assign`: assign one dict to the other through mutations
  - `exceptions.InvalidOperationError`: raise when an operation is illegal/invalid, regardless of the arguments you throw at it (in the current state).
  - `inspect.call_args`: Get function call arguments as a single dict
  - `observable.Set`: set which can be observed for changes
  - `path.chmod`: change file or directory mode bits (optionally recursively)
  - `path.digest`: Get SHA512 checksum of file or directory
  - `path.read`: get file contents
  - `path.remove`: remove file or directory (recursively), unless it's missing
  - `path.write`: create or overwrite file with contents
  - `series.assert_equals`: Assert 2 series are equal
  - `series.equals`: Get whether 2 series are equal
  - `series.split`: Split values
  - `test.temp_dir_cwd`: pytest fixture that sets current working directory to a temporary directory

v2.0.4

No changelog

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index