StaticFrame

Release 0.4.3

unknown

Sep 24, 2019

QUICK START

Immutable data structures for one- and two-dimensional calculations with self-aligning, labelled axes.

- Code: https://github.com/InvestmentSystems/static-frame
- Packages: https://pypi.org/project/static-frame

CHAPTER

ONE

STATIC-FRAME

The StaticFrame library consists of the Series and Frame, immutable data structures for one- and two-dimensional calculations with self-aligning, labelled axes. StaticFrame meets the need for an immutable Pandas DataFrame with a consistent, functional interface. While many interfaces are similar to Pandas, StaticFrame deviates from Pandas in many ways: all data is immutable, and indices are always unique; the full range of NumPy data types is preserved, and date-time indices use discrete NumPy types; hierarchical indices are seamlessly integrated; and flexible approaches to element, row, and column iteration and function application are provided in a uniform interface.

Code: https://github.com/InvestmentSystems/static-frame

Docs: http://static-frame.readthedocs.io

Packages: https://pypi.org/project/static-frame

1.1 Installation

Install StaticFrame via PIP:

```
pip install static-frame
```

Or, install StaticFrame via conda:

```
conda install -c conda-forge static-frame
```

1.2 Dependencies

StaticFrame requires Python 3.6+ and NumPy 1.14.1+.

1.3 Quick-Start Guide

StaticFrame provides numerous methods for loading and creating data, either as a 1D `Series` or a 2D `Frame`. All creation routines are exposed as alternate constructors on the desired class, such as `Frame.from_records()`, `Frame.from_csv()` or `Frame.from_pandas()`.

For example, we can load JSON data from a URL using `Frame.from_json_url()`, and then use `Frame.head()` to reduce the displayed output to just the first five rows. (Passing explicit `dtypes` is only necessary on Windows.)

```
>>> import numpy as np
>>> import static_frame as sf
```

```
>>> frame = sf.Frame.from_json_url('https://jsonplaceholder.typicode.com/photos',␣
↪dtypes=dict(albumId=np.int64, id=np.int64))
```

```
>>> frame.head()
<Frame>
<Index> albumId id      title                url                  thumbnailUrl       ␣
↪ <<U12>
<Index>
0       1       1       accusamus beatae ... https://via.place... https://via.place...
1       1       2       reprehenderit est... https://via.place... https://via.place...
2       1       3       officia porro iur... https://via.place... https://via.place...
3       1       4       culpa odio esse r... https://via.place... https://via.place...
4       1       5       natus nisi omnis ... https://via.place... https://via.place...
<int64> <int64> <int64> <<U86>               <<U38>               <<U38>
```

Note: The Pandas CSV reader far out-performs the NumPy-based reader in StaticFrame: thus, for now, using `Frame.from_pandas(pd.read_csv(fp))` is recommended for loading CSV files.

For more information on Series and Frame constructors, see Container Import & Creation.

As with a NumPy array, the `Frame` exposes common attributes of shape and size.

```
>>> frame.shape
(5000, 5)
>>> frame.size
25000
>>> frame.nbytes
3320000
```

Unlike a NumPy array, a Frame stores heterogeneous types, where each column is a single type. StaticFrame preserves the full range of NumPy types, including fixed-size character strings. Character strings can be converted to Python objects or other types as needed with the `Frame.astype` interface, which exposes a `__getitem__` style interface for selecting columns to convert. As with all similar functions, a new `Frame` is returned.

```
>>> frame.dtypes
<Series>
<Index>
albumId      int64
id           int64
title        <U86
url          <U38
thumbnailUrl <U38
<<U12>       <object>
```

```
>>> frame.astype['title':](object).dtypes
<Series>
<Index>
albumId      int64
id           int64
title        object
url          object
```

(continues on next page)

(continued from previous page)

```
thumbnailUrl object
<<U12>       <object>
```

Utility functions common to Pandas users are available on `Frame` and `Series`, such as `Series.unique()`, `Series.isna()`, and `Series.any()`.

```
>>> frame['albumId'].unique().tolist()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,␣
↪24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,␣
↪45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,␣
↪66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,␣
↪87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
>>> frame['id'].isna().any()
False
```

Note: For more information on Series and Frame utility functions, see Transformations & Utilities.

StaticFrame interfaces for extracting data will be familiar to Pandas users, though with a number of interface refinements to remove redundancies and increase consistency. On a `Frame`, `__getitem__` is (exclusively) a column selector; `loc` and `iloc` are (with one argument) row selectors or (with two arguments) row and column selectors.

For example we can select a single column with `__getitem__`:

```
>>> frame['albumId'].tail()
<Series: albumId>
<Index>
4995              100
4996              100
4997              100
4998              100
4999              100
<int64>           <int64>
```

Consistent with other `__getitem__` style selectors, a slice or a list can be used to select columns:

```
>>> frame['id':'title'].head()
<Frame>
<Index> id      title                <<U12>
<Index>
0       1       accusamus beatae ...
1       2       reprehenderit est...
2       3       officia porro iur...
3       4       culpa odio esse r...
4       5       natus nisi omnis ...
<int64> <int64> <<U86>
```

The `loc` interface, with one argument, returns a `Series` for the row found at the given index label.

```
>>> frame.loc[4]
<Series: 4>
<Index>
albumId      1
id           5
title        natus nisi omnis ...
url          https://via.place...
```

(continues on next page)

(continued from previous page)

```
thumbnailUrl https://via.place...
<<U12>       <object>
```

With two arguments, `loc` can select both rows and columns at the same time:

```
>>> frame.loc[4:8, ['albumId', 'title']]
<Frame>
<Index> albumId title                <<U12>
<Index>
4       1       natus nisi omnis ...
5       1       accusamus ea aliq...
6       1       officia delectus ...
7       1       aut porro officii...
<int64> <int64> <<U86>
```

Where the `loc` interface uses index and column labels, the `iloc` interface uses integer offsets from zero, just as if the `Frame` were a NumPy array. For example, we can select the last row with `-1`:

```
>>> frame.iloc[-1]
<Series: 4999>
<Index>
albumId        100
id             5000
title          error quasi sunt ...
url            https://via.place...
thumbnailUrl   https://via.place...
<<U12>         <object>
```

Or, using two arguments, we can select the first two columns of the last two rows:

```
>>> frame.iloc[-2:, 0:2]
<Frame>
<Index> albumId id      <<U12>
<Index>
4998    100     4999
4999    100     5000
<int64> <int64> <int64>
```

Just as with Pandas, expressions can be used in `__getitem__`, `loc`, and `iloc` statements to create more narrow selections. For example, we can select all "albumId" greater than or equal to 98.

```
>>> frame.loc[frame['albumId'] >= 98, ['albumId', 'title']].head()
<Frame>
<Index> albumId title                <<U12>
<Index>
4850    98      aut aut nulla vol...
4851    98      ducimus neque del...
4852    98      fugit officiis su...
4853    98      pariatur temporib...
4854    98      qui inventore inc...
<int64> <int64> <<U86>
```

However, unlike Pandas, `__getitem__`, `loc`, and `iloc` cannot be used for assignment or in-place mutation on a `Frame` or `Series`. Throughout StaticFrame, all underlying NumPy arrays, and all container attributes, are immutable. Making data and objects immutable reduces opportunities for coding errors and offers, in some situations, greater efficiency by avoiding defensive copies.

```
>>> frame.loc[4854, 'albumId']
98
>>> frame.loc[4854, 'albumId'] = 200
Traceback (most recent call last):
TypeError: 'InterfaceGetItem' object does not support item assignment
>>> frame.values[4854, 0] = 200
Traceback (most recent call last):
ValueError: assignment destination is read-only
```

Note: For more information on Series and Frame selection interfaces, see Selection.

Instead of in-place assignment, an `assign` interface object (similar to the `Frame.astype` interface shown above) is provided to expose `__getitem__`, `loc`, and `iloc` interfaces that, when called with an argument, return a new object with the desired changes. These interfaces expose the full range of expressive assignment-like idioms found in Pandas and NumPy. Arguments can be single values, or `Series` and `Frame` objects, where assignment will align on the Index.

```
>>> frame_new = frame.assign.loc[4854, 'albumId'](200)
>>> frame_new.loc[4854, 'albumId']
200
```

This pattern of specialized interfaces is used throughout StaticFrame, such as with the `Frame.mask` and `Frame.drop` interfaces. For example, `Frame.mask` can be used to create a Boolean `Frame` that sets rows to True if their "id" is even:

```
>>> frame.mask.loc[frame['id'] % 2 == 0].head()
<Frame>
<Index> albumId id     title  url    thumbnailUrl <<U12>
<Index>
0       False   False  False  False  False
1       True    True   True   True   True
2       False   False  False  False  False
3       True    True   True   True   True
4       False   False  False  False  False
<int64> <bool>  <bool> <bool> <bool> <bool>
```

Or, using the `Frame.drop` interface, a new `Frame` can be created by dropping rows with even "id" values and dropping URL columns specified in a list:

```
>>> frame.drop.loc[frame['id'] % 2 == 0, ['thumbnailUrl', 'url']].head()
<Frame>
<Index> albumId id      title                <<U12>
<Index>
0       1       1       accusamus beatae ...
2       1       3       officia porro iur...
4       1       5       natus nisi omnis ...
6       1       7       officia delectus ...
8       1       9       qui eius qui aute...
<int64> <int64> <int64> <<U86>
```

Note: For more information on Series and Frame interfaces, see Assignment / Dropping / Masking.

Iteration of rows, columns, and elements, as well as function application on those values, is unified under a family of generator interfaces. These interfaces are distinguished by the form of the data iterated (`Series`, `namedtuple`,

or `array`) and whether key-value pairs (e.g., `Frame.iter_series_items()`) or just values (e.g., `Frame.iter_series()`) are yielded. For example, we can iterate over each row of a `Frame` and yield a corresponding `Series`:

```
>>> next(iter(frame.iter_series(axis=1)))
<Series>
<Index>
albumId      1
id           1
title        accusamus beatae ...
url          https://via.place...
thumbnailUrl https://via.place...
<<U12>       <object>
```

Or we can iterate over rows as named tuples, applying a function that matches a substring of the "title" or returns None, then drop those None records:

```
>>> frame.iter_tuple(axis=1).apply(lambda r: r.title if 'voluptatem' in r.title else␣
↪None).dropna().head()
<Series>
<Index>
19       assumenda volupta...
27       non neque eligend...
29       odio enim volupta...
31       ad enim dignissim...
40       in voluptatem dol...
<int64>  <object>
```

Element iteration and function application works the same way as for rows or columns (though without an `axis` argument). For example, here each URL is processed with the same string transformation function:

```
>>> frame[['thumbnailUrl', 'url']].iter_element().apply(lambda c: c.replace('https://
↪', '')).iloc[-4:]
<Frame>
<Index> thumbnailUrl         url                 <<U12>
<Index>
4996    via.placeholder.c... via.placeholder.c...
4997    via.placeholder.c... via.placeholder.c...
4998    via.placeholder.c... via.placeholder.c...
4999    via.placeholder.c... via.placeholder.c...
<int64> <object>             <object>
```

Group-by functionality is exposed in a similar manner with `Frame.iter_group_items()` and `Frame.iter_group()`.

```
>>> next(iter(frame.iter_group('albumId', axis=0))).shape
(50, 5)
```

Function application to a group `Frame` can be used to produce a `Series` indexed by the group label. For example, a `Series`, indexed by "albumId", can be produced to show the number of unique titles found per album.

```
>>> frame.iter_group('albumId', axis=0).apply(lambda g: len(g['title'].unique()),␣
↪dtype=np.int64).head()
<Series>
<Index>
1        50
2        50
```

(continues on next page)

(continued from previous page)

```
3         50
4         50
5         50
<int64>   <int64>
```

Note: For more information on Series and Frame iterators and tools for function application, see Iterators.

If performing calculations on a `Frame` that result in a `Series` with a compatible `Index`, a grow-only `FrameGO` can be used to add `Series` as new columns. This limited form of mutation, i.e., only the addition of columns, provides a convenient compromise between mutability and immutability. (Underlying NumPy array data always remains immutable.)

A `FrameGO` can be efficiently created from a `Frame`, as underling NumPy arrays do not have to be copied:

```
>>> frame_go = frame.to_frame_go()
```

We can obtain a track number within each album, assuming the records are sorted, by creating the following generator expression pipe-line. Using a `Frame` grouped by "albumId", `zip` together as pairs the `Frame.index` and a contiguous integer sequence via `range()`; `chain` all of those iterables, and then pass the resulting generator to `Series.from_items()`. (As much as possible, StaticFrame supports generators as arguments wherever an ordered sequence is expected.)

```
>>> from itertools import chain
>>> index_to_track = chain.from_iterable(zip(g.index, range(len(g))) for g in frame_
↪go.iter_group('albumId'))
>>> frame_go['track'] = sf.Series.from_items(index_to_track, dtype=np.int64) + 1
```

```
>>> frame_go.iloc[45:55]
<FrameGO>
<IndexGO> albumId id      title                url                  thumbnailUrl
↪   track   <<U12>
<Index>
45        1       46      quidem maiores in... https://via.place... https://via.place.
↪.. 46
46        1       47      et soluta est        https://via.place... https://via.place.
↪.. 47
47        1       48      ut esse id           https://via.place... https://via.place.
↪.. 48
48        1       49      quasi quae est mo... https://via.place... https://via.place.
↪.. 49
49        1       50      et inventore quae... https://via.place... https://via.place.
↪.. 50
50        2       51      non sunt voluptat... https://via.place... https://via.place.
↪.. 1
51        2       52      eveniet pariatur ... https://via.place... https://via.place.
↪.. 2
52        2       53      soluta et harum a... https://via.place... https://via.place.
↪.. 3
53        2       54      ut ex quibusdam d... https://via.place... https://via.place.
↪.. 4
54        2       55      voluptatem conseq... https://via.place... https://via.place.
↪.. 5
<int64>   <int64> <int64> <<U86>              <<U38>               <<U38>
↪   <int64>
```

Unlike with Pandas, StaticFrame `Index` objects always enforce uniqueness (there is no "verify_integrity" option: integrity is never optional). Thus, an index can never be set from non-unique data:

```
>>> frame_go.set_index('albumId')
Traceback (most recent call last):
static_frame.core.exception.ErrorInitIndex: labels (5000) have non-unique values (100)
```

For a data set such as the one used in this example, a hierarchical index, by "albumId" and "track", is practical. StaticFrame implements hierarchical indices as `IndexHierarchy` objects. The `Frame.set_index_hierarchy()` method, given columns in a `Frame`, can be used to create a hierarchical index:

```
>>> frame_h = frame_go.set_index_hierarchy(['albumId', 'track'], drop=True)
>>> frame_h.head()
<FrameGO>
<IndexGO>                id      title                url                 ␣
↪thumbnailUrl         <<U12>
<IndexHierarchy>
1                1       1       accusamus beatae ... https://via.place... https://
↪via.place...
1                2       2       reprehenderit est... https://via.place... https://
↪via.place...
1                3       3       officia porro iur... https://via.place... https://
↪via.place...
1                4       4       culpa odio esse r... https://via.place... https://
↪via.place...
1                5       5       natus nisi omnis ... https://via.place... https://
↪via.place...
<int64>          <int64> <int64> <<U86>               <<U38>               <<U38>
```

Hierarchical indices permit specifying selectors, per axis, at each hierarchical level. To distinguish hierarchical levels from axis arguments in a `loc` expression, the `HLoc` wrapper, exposing a `__getitem__` interface, can be used. For example, we can select, from all albums, the second and fifth track, and then only the "title" and "url" columns.

```
>>> frame_h.loc[sf.HLoc[:, [2,5]], ['title', 'url']].head()
<FrameGO>
<IndexGO>                title                url                  <<U12>
<IndexHierarchy>
1                2       reprehenderit est... https://via.place...
1                5       natus nisi omnis ... https://via.place...
2                2       eveniet pariatur ... https://via.place...
2                5       voluptatem conseq... https://via.place...
3                2       eaque iste corpor... https://via.place...
<int64>          <int64> <<U86>               <<U38>
```

Just as a hierarchical selection can reside in a `loc` expression with an `HLoc` wrapper, an integer index selection can reside in a `loc` expression with an `ILoc` wrapper. For example, the previous row selection is combined with the selection of the last column:

```
>>> frame_h.loc[sf.HLoc[:, [2,5]], sf.ILoc[-1]].head()
<Series: thumbnailUrl>
<IndexHierarchy>
1                    2       https://via.place...
1                    5       https://via.place...
2                    2       https://via.place...
2                    5       https://via.place...
3                    2       https://via.place...
<int64>              <int64> <<U38>
```

Note: For more information on Index and IndexHierarchy, see Index Manipulation.

While StaticFrame offers many of the features of Pandas and similar data structures, exporting directly to NumPy arrays (via the `.values` attribute) or to Pandas is supported for functionality not found in StaticFrame or compatibility with other libraries. For example, a `Frame` can export to a Pandas `DataFrame` with `Frame.to_pandas()`.

```
>>> df = frame_go.to_pandas()
```

CHAPTER

TWO

LICENSE

MIT License

Copyright (c) 2012-2019 Research Affiliates

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

IN NO EVENT SHALL RESEARCH AFFILATES, LLC ("RA") BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF RA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

RA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE AND ACCOMPANYING DOCUMENTATION OR TEXT, IF ANY, PROVIDED HEREUNDER IS PROVIDED "AS IS". RA HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

CHAPTER

THREE

ABOUT STATICFRAME

The StaticFrame library consists of the Series and Frame, immutable data structures for one- and two-dimensional calculations with self-aligning, labelled axes. StaticFrame offers an alternative to Pandas. While many interfaces for data extraction and manipulation are similar to Pandas, StaticFrame deviates from Pandas in many ways: all data is immutable, and all indices must be unique; all vector processing uses NumPy, and the full range of NumPy data types is preserved; the implementation is concise and lightweight; consistent naming and interfaces are used throughout; and flexible approaches to iteration and function application, with built-in options for parallelization, are provided.

Alpha Release

The current release of StaticFrame is an alpha release, meaning that interfaces may change, functionality may be incomplete, and there may be significant flaws. Please assist in development by reporting any bugs or request any missing features.

https://github.com/InvestmentSystems/static-frame/issues

StaticFrame is not a drop-in replacement for Pandas. While some conventions and API components are directly borrowed from Pandas, some are completely different, either by necessity (due to the immutable data model) or by choice (offering more uniform, less redundant, and more explicit interfaces). As StaticFrame does not support in-place mutation, architectures that made significant use of mutability in Pandas will require refactoring.

StaticFrame is lightweight. It has few dependencies (Pandas is not a dependency). The core library is less than 10,000 lines of code, less than 5% the size of the Pandas code base[1].

StaticFrame does not aspire to be an all-in-one framework for all aspects of data processing and visualization. StaticFrame focuses on providing efficient and powerful data structures with consistent, clear, and stable interfaces.

StaticFrame aspires to have comparable or better performance than Pandas. While this is already the case for some core operations (See *Performance*), some important functions are far more performant in Pandas (such as reading delimited text files via `pd.read_csv`). StaticFrame provides easy conversion to and from Pandas to bridge needed functionality or performance.

StaticFrame does not implement its own types or numeric computation routines, relying entirely on NumPy. NumPy offers desirable stability in performance and interface. For working with SciPy and related tools, StaticFrame exposes easy access to NumPy arrays.

[1] The Pandas 2.0 Design Docs state that the Pandas codebase has over 200,000 lines of code: https://pandas-dev.github.io/pandas2/goals.html

CHAPTER

FOUR

IMMUTABILITY

The *static_frame.Series* and *static_frame.Frame* store data in immutable NumPy arrays. Once created, array values cannot be changed. StaticFrame manages NumPy arrays, setting the `ndarray.flags.writeable` attribute to False on all managed and returned NumPy arrays.

```
>>> import static_frame as sf
>>> import numpy as np

>>> s = sf.Series((67, 62, 27, 14), index=('Jupiter', 'Saturn', 'Uranus', 'Neptune'),␣
↪dtype=np.int64)
>>> s #doctest: +NORMALIZE_WHITESPACE
<Series>
<Index>
Jupiter  67
Saturn   62
Uranus   27
Neptune  14
<<U7>    <int64>
>>> s['Jupiter'] = 68
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'Series' object does not support item assignment
>>> s.iloc[0] = 68
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'InterfaceGetItem' object does not support item assignment
>>> s.values[0] = 68
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: assignment destination is read-only
```

To mutate values in a `Series` or `Frame`, a copy must be made. Convenient functional interfaces to assign to a copy are provided, using conventions familiar to NumPy and Pandas users.

```
>>> s.assign['Jupiter'](69)
<Series>
<Index>
Jupiter  69
Saturn   62
Uranus   27
Neptune  14
<<U7>    <int64>
>>> s.assign['Uranus':](s['Uranus':] - 2)
```

(continues on next page)

(continued from previous page)

```
<Series>
<Index>
Jupiter  67
Saturn   62
Uranus   25
Neptune  12
<<U7>   <int64>
>>> s.assign.iloc[[0, 3]]((68, 11))
<Series>
<Index>
Jupiter  68
Saturn   62
Uranus   27
Neptune  11
<<U7>    <int64>
```

Immutable data has the overwhelming benefit of providing the confidence that a client of a `Series` or `Frame` cannot mutate its data. This removes the need for many unnecessary copies, and forces clients to only make copies when absolutely necessary.

There is no guarantee that using immutable data will produce correct code or more resilient and robust libraries. It is true, however, that using immutable data removes countless opportunities for introducing flaws in data processing routines and libraries.

CHAPTER

FIVE

HISTORY

The ideas behind StaticFrame developed out of years of work with Pandas and related tabular data structures by the Investment Systems team at Research Affiliates, LLC. In May of 2017 Christopher Ariza proposed the basic model to the Investment Systems team and began implementation. The first public release was in May 2018.

CHAPTER

SIX

PRESENTATIONS

The following presentations and interviews describe StaticFrame in greater depth.

- PyData LA 2018: https://pyvideo.org/pydata-la-2018/staticframe-an-immutable-alternative-to-pandas.html
- PyCon US 2019, lightning talk (starting at 53:00): https://pyvideo.org/pycon-us-2019/friday-lightning-talksbreak-pycon-2019.html
- Talk Python to Me, interview: https://talkpython.fm/episodes/show/204/staticframe-like-pandas-but-safer

CHAPTER

SEVEN

CONTRIBUTORS

These members of the Investment Systems team have contributed greatly to the design of StaticFrame:

- Brandt Bucher
- Guru Devanla
- John Hawk
- Adam Kay
- Mark LeMoine
- Myrl Marmarelis
- Tom Rutherford
- Yu Tomita
- Quang Vu

Thanks also for additional contributions from GitHub users.

https://github.com/InvestmentSystems/static-frame/graphs/contributors

CHAPTER

EIGHT

PERFORMANCE

StaticFrame benchmarks its performance with one-to-one comparisons to functionally equivalent Pandas operations. Such measures of performance are poor indicators of real-world use-cases, but provide some insight into general run-time characteristics.

8.1 StaticFrame v. Pandas

The following tables illustrate absolute run-times ("pd", "sf", where smaller is better) and run-time ratios ("pd_faster", "sf_faster", where larger is better) for functionally equivalent operations. Where a value is given for a run-time ratio, that package is faster; e.g., where "sf_faster" is 4.7, that operation is 4.7 times faster in StaticFrame than it is in Pandas. For details and code for each operation, see *Performance Comparison Test Code*.

8.1.1 StaticFrame 0.3.4 v. Pandas 0.23.4

test_class	pd	sf	pd_faster	sf_faster
FrameFloat_H1D_add_series_partial	2.8959	0.6154		4.7059
FrameFloat_H2D_add_series_partial	4.6065	3.1625		1.4566
FrameFloat_apply_axis0	1.8324	0.8498		2.1563
FrameFloat_apply_axis1	10.8848	3.0343		3.5872
FrameFloat_dropna_any_axis0	4.6445	3.6647		1.2674
FrameFloat_dropna_any_axis1	4.2284	0.2998		14.1058
FrameFloat_from_records	1.6269	1.0615		1.5326
FrameFloat_isna	0.1937	0.2523	1.3029	
FrameFloat_slice_loc_column	0.1004	0.0731		1.3738
FrameFloat_slice_loc_columns	0.0864	0.0744		1.1616
FrameFloat_slice_loc_index	0.2667	0.143		1.8659
FrameFloat_slice_loc_indices	0.2425	9.0009	37.1208	
FrameFloat_sum_skipna_axis0	1.3131	0.7694		1.7066
FrameFloat_sum_skipna_axis1	1.4416	0.6903		2.0885
FrameMixed_from_records	0.9984	1.0093	1.011	
FrameMixed_slice_loc_column	0.1009	0.0733		1.3763
FrameMixed_slice_loc_columns	18.3912	0.5906		31.1382
FrameMixed_slice_loc_index	0.5076	1.9941	3.9287	
FrameMixed_slice_loc_indices	0.3242	6.878	21.2172	
FrameStrFloat_init	0.2942	1.0577	3.5952	
IndexHierarchy2d_from_labels	0.4343	1.2591	2.8991	
IndexHierarchy2d_from_product	0.1985	0.0284		6.9958

Continued on next page

Table 1 – continued from previous page

test_class	pd	sf	pd_faster	sf_faster
IndexHierarchy3d_from_labels	0.5877	1.6323	2.7773	
IndexHierarchy3d_from_product	0.0862	0.0089		9.6377
IndexStr_init	0.1334	0.5931	4.446	
SeriesFloatH2DString_loc_slice	0.1185	0.6692	5.6458	
SeriesFloatH2DString_loc_target	0.0208	0.0033		6.2216
SeriesFloatH3DString_loc_slice_slice_target	0.4136	7.0483	17.0431	
SeriesFloatH3DString_loc_slice_target_slice	2.8101	0.3292		8.5355
SeriesFloatH3DString_loc_target	0.058	0.0046		12.7345
SeriesIntFloat_dropna	0.067	0.1664	2.483	
SeriesIntFloat_fillna	0.0375	0.0085		4.3834
SeriesIntFloat_init	0.0081	0.0091	1.1157	
SeriesIntFloat_isnull	0.0164	0.0026		6.385
SeriesIntObjStr_dropna	0.1786	0.3344	1.8725	
SeriesIntObjStr_fillna	0.1569	0.1247		1.2582
SeriesIntObjStr_isnull	0.0952	0.0988	1.0372	
SeriesIntObj_dropna	0.1716	0.3029	1.7657	
SeriesIntObj_fillna	0.3919	0.1236		3.1709
SeriesIntObj_isnull	0.0919	0.1014	1.1025	
SeriesStrFloat_dropna	0.0674	0.8448	12.5296	
SeriesStrFloat_fillna	0.0371	0.0085		4.3628
SeriesStrFloat_isnull	0.0164	0.0026		6.3598
SeriesStrObj_init	0.2408	0.4165	1.7295	

8.2 Performance Comparison Test Code

Performance tests are based on the following implementations. Follow the source links to view the relevant code.

class FrameFloat_H1D_add_series_partial
Adding series that only partially match the index

class FrameFloat_H2D_add_series_partial
Adding series that only partially match the index

class FrameFloat_apply_axis0

class FrameFloat_apply_axis1

class FrameFloat_dropna_any_axis0

class FrameFloat_dropna_any_axis1

class FrameFloat_from_records

class FrameFloat_isna

class FrameFloat_slice_loc_column

class FrameFloat_slice_loc_columns

class FrameFloat_slice_loc_index

class FrameFloat_slice_loc_indices

class FrameFloat_sum_skipna_axis0

class FrameFloat_sum_skipna_axis1

class FrameMixed_from_records

class FrameMixed_slice_loc_column

class FrameMixed_slice_loc_columns

class FrameMixed_slice_loc_index

class FrameMixed_slice_loc_indices

class FrameStrFloat_init

class IndexHierarchy2d_from_labels

class IndexHierarchy2d_from_product

class IndexHierarchy3d_from_labels

class IndexHierarchy3d_from_product

class IndexStr_init
Index construction for string labels.

class PerfTest

class SeriesFloatH2DString_loc_slice

class SeriesFloatH2DString_loc_target

class SeriesFloatH3DString_loc_slice_slice_target

class SeriesFloatH3DString_loc_slice_target_slice

class SeriesFloatH3DString_loc_target
Selecting single value from 3-level hierarchy.

class SeriesIntFloat_apply

class SeriesIntFloat_drop_duplicated

class SeriesIntFloat_dropna

class SeriesIntFloat_fillna

class SeriesIntFloat_fillna_forward

class SeriesIntFloat_init

class SeriesIntFloat_isnull

class SeriesIntObjStr_apply

class SeriesIntObjStr_dropna

class SeriesIntObjStr_fillna

class SeriesIntObjStr_fillna_forward

class SeriesIntObjStr_isnull

class SeriesIntObj_apply

class SeriesIntObj_drop_duplicated

class SeriesIntObj_dropna

class SeriesIntObj_fillna

class SeriesIntObj_fillna_forward

class SeriesIntObj_isnull

class SeriesStrFloat_apply

```
class SeriesStrFloat_dropna
class SeriesStrFloat_fillna
class SeriesStrFloat_fillna_forward
class SeriesStrFloat_isna
class SeriesStrObj_init
```

CHAPTER

NINE

WHAT IS NEW IN STATIC FRAME

9.1 0.5.0-dev

Added `interface` attribute to all containers, providing a hierarchical presentation of all interfaces.

Added `display_tall()` and `display_wide()` convenienve methods to all containers.

9.2 0.4.3

Fixed issues in `FrameGO` setitem and using binary operators between `Frame` and `FrameGO`.

9.3 0.4.2

Corrected flaw in axis 1 statistical operations with `Frame` constructed from mixed sized `TypeBlocks`.

Added `Series.loc_min`, `Series.loc_max`, `Series.iloc_min`, `Series.iloc_max`.

Added `Frame.loc_min`, `Frame.loc_max`, `Frame.iloc_min`, `Frame.iloc_max`,

9.4 0.4.1

`iter_element().apply` now properly preserves index and column types.

Using `Frame.from_records` with an empty iterable or iterator will deliver a `ErrorInitFrame`.

Matrix multiplication implemented for `Index`, `Series`, and `Frame`.

Added `Frame.from_records_items` constructor.

Improved dtype selection in `FrameGO` set item and related functions.

`IndexHierarchy.from_labels` now accepts an `index_constructors` argument.

`Frame.set_index_hierarchy` now accepts an `index_constructors` argument.

`IndexHierarhcy.from_product() now attempts to use ``name` of provided indicies for the `IndexHierarchy` name, when all names are non-None.

Added `IndexHierarchy.dtypes` and `IndexHierarchy.index_types`, returning `Series` indexed by `name` when possible.

9.5 0.4.0

Improved handling for special cases `Series` initialization, including initialization from iterables of lists.

The `Series` initializer no longer accepts dictionaries; `Series.from_dict` is added for explicit creation from mappings.

`IndexAutoFactory` suport removed from `Series.reindex` and `Frame.reindex` and added to `Series.relabel` and `Frame.relabel`.

The following `Series` and `Frame` methods are renamed: `reindex_flat`, `reindex_add_level`, and `reindex_drop_level` are now `relabel_flat`, `relabel_add_level`, and `relabel_drop_level`.

Implemented `Frame.from_sql` constructor.

9.6 0.3.9

`IndexAutoFactory` introduced to consolidate creation of auto-incremented integer indices, and provide a single token to force auto-incremented integer indices in other contexts where `index` arguments are taken.

`IndexAutoFactory` support implemented for the `index` argument in `Series.from_concat` and `Series.reindex`.

`IndexAutoFactory` support implemented for the `index` and `columns` argument in `Frame.from_concat` and `Frame.reindex`.

Added new `DisplayaConfig` parameters to format floating-point values: `value_format_float_positional`, `value_format_float_scientific`, `value_format_complex_positional`, `value_format_complex_scientific`,

Set default `value_format_float_scientific` and `value_format_complex_scientific` to avoid truncation of scientific notation in output displays.

9.7 0.3.8

All duplicate-handling functions now support heterogenously typed object arrays with unsortable (but hashable) types.

Operations on all indices now preserve order when indices are equal.

Functions with the `skipna` argument now properly skip `None` in `Frames` with built with object arrays.

`Frame.to_csv` now uses the argument name *delimiter* instead of *sep*, aligning with the usage in `Frame.from_csv`.

9.8 0.3.7

Completed implementation of `Frame.fillna_forward`, `Frame.fillna_backward`, `Frame.fillna_leading`, `Frame.fillna_trailing`.

Fixed issue exposed in FrameGO.sort_values() due to NumPy integers being used for selection.

`IndexHierarchy.sort()`, `IndexHierarchy.isin()`, `IndexHierarchy.roll()` now implemented.

`Series.sort_index()` now properly propagates `IndexBase` subclasses.

`Frame.sort_index()` and `Frame.sort_columns()` now properly propagate `IndexBase` subclasses.

All containers now derive from `ContainerBase`, simplyfying inheritance and `ContainerMeta` application.

`Index` objects based on `np.datetime64` now accept `np.datetime64` objects in `loc` expressions.

All construction from Python iterables now better handle array creation from diverse Python objects.

9.9 0.3.6

`Frame.to_frame_go` now properly handles `IndexHierarchy` columns.

Improved creation of `IndexHierarchy` from other `IndexHierarchy` or `IndexHierarchyGO`.

`Frame` initializer now exposes `index_constructor` and `columns_constructor` arguments.

`Frame.from_records` now efficiently uses `dict_view` objects containing row records.

`Frame` now supports shapes of all zero and non-zero combinations of index and column lengths; `Frame` construction will raise an exception if attempting to set a value in an unfillable Frame shape.

`Frame`, `Series`, `Index`, and `IndexHierarchy` all have improved implementations of `cumprod` and `cumsum` methods.

9.10 0.3.5

Improved type handling of `np.datetime64` typed columns in `Frame`.

Added `median` method to all `MetaOperatorDelegate` classes, inlcuding `Series`, `Index`, and `Frame`.

`Frame` and `Series` sort methods now propagate `name` attributes.

`Index.from_pandas()` now correctly collects `name` / `names` attributes from Pandas indexes.

Implemented `Series.fillna_forward`, `Series.fillna_backward`, `Series.fillna_leading`, `Series.fillna_trailing`.

Fixed flaw in dropping columns from a `Frame` (via `Frame.set_index` or the `Frame.drop` interface), whereby sometimes (depending on `TypeBlocks` structure) the drop would not be executed.

`Index` objects based on `np.datetime64` now limit `__init__` arguments only to those relevant for those derived classes.

`Index` objects based on `np.datetime64` now support transformations from both `datetime.timedelta` as well as `np.timedelta64`.

Index objects based on `np.datetime64` now support selection with slices with `np.datetime64` units different than those used in the `Index`.

9.11 0.3.4

Added `dtypes` argument to all relevant `Frame` constructors; `dtypes` can now be specified with a dictionary.

Deprecated instantiating a `Frame` from `dict`; added `Frame.from_dict` for explicit `Frame` creation from a `dict`.

9.12 0.3.3

Improvements to all `datetime64` based indicies: direct creation from labels now properly parses values into `datetime64`, and `loc`-style lookups now handle partial matches on lower-resolution datetimes. Added `IndexSecond` and `IndexMillisecond` Index classes.

Index can now be constructed directly from an `IndexHierarchy` (resulting in an Index of tuples)

Improvements to application of ellipsis when normalizing width in `Display` string representations.

`Frame.values` now always returns a 2D NumPy array.

`Series.iloc`, when a non-mulitple selection is given, now returns a single element, not a `Series`.

9.13 0.3.2

`IndexHierarchy.drop_level()` and related methods have been updated such that negative integers drop innermost levels, and postive integers drop outermost levels. This is an API breaking change.

Fixed missing handling for all-missing in `Series.dropna`.

Improved `loc` and `HLoc` usage on Series with `IndexHierarchy` to insure a Series is returned when a multiple selection is used.

`IndexHierarchy.from_labels()` now returns proper error message for invalid tree forms.

9.14 0.3.1

Implemented Series.iter_group_index(), Series.iter_group_index_items(), Frame.iter_group_index(), Frame.iter_group_index_items() for producing iterators (and targets of function application) based on groupings of the index; particularly useful for IndexHierarhcy.

Implemented Series.from_concat; improved Frame.from_concat in concatenating indices with diverse types. Frame.from_concat() now accepts Series.

Added `Index.iter_label()` and `IndexHierarchy.iter_label()`, for variable depth label iteration, particularly useful for IndexHierarchy.

Improved initializer behavior of IndexDate, IndexYearMonth, IndexYear to apply expected dtype when creating arrays from non-array initializers, allowing conversion of string date representations to proper date types.

Added `Index.to_pandas` and specialized methods on `IndexDate` and derived classes. Added `IndexHierarchy.to_pandas`.

Added support for `Series` as an argument to `FrameGO.extend()`.

Added `Series.to_frame()` and `Series.to_frame_go()`.

The `name` attribute is now implemented for all containers; all constructors now take a `name` argument, and a `rename` method is available. Extracting columns, rows, and setting indices on `Frame` all propagate name attributes appropriately.

The default `Series` display has been updated to show the "<Series>" label above the index, consistent with the presentation of `Frame`.

The `Frame.from_records()` method has been extended to support explicitly passing dtypes per column, which permits avoiding type discovery through observing the first record or relying on NumPy's type discovery in array creation.

The `Frame.from_concat()` constructor now handles hierarchical indices correctly.

9.15 0.3.0

The `Index.keys()` method now returns the underlying KeysView from the Index's dictionary.

All primary containers (i.e., Series, Frame, and Index) now display HTML tables in Jupyter Notebooks. This is implemented via the `_repr_html_()` methods.

All primary containers now feature a `to_html()` method.

All primary containers now feature a `to_html_datatables()` method, which authors a complete HTML file with DataTables/JavaScript-powered table viewing, sorting, and searching.

StaticFrame's display infrastructure now permits individually coloring types by category, as well as different display formats for supporting HTML output.

StaticFrame's display infrastructure now shows hierarchical indices, used for either indices or columns, in the same display grid used for other display components.

The `DisplayConfig` class has been expanded to permit definition of colors, specified in hexadecimal integers or string codes, for all type categories, as well as independent settings for type delimiters, and a new setting for `display_format`.

The following `DisplayFormats` have been created and implemented: `terminal`, `html_datatables`, `html_table`, and `html_pre`.

CHAPTER

TEN

BORING INDICES & WHERE TO FIND THEM: THE AUTO-INCREMENTED INTEGER INDEX IN STATICFRAME

This article is part of a series exploring the features and design of StaticFrame, a Python package that offers data structures similar to the Pandas DataFrame and Series, but with an immutable data model.

This article demonstrates how StaticFrame exposes functionality for creating the most boring index object: the auto-incremented integer index (AIII). An AIII makes an axis selectable with integers, just as a NumPy array; it makes `loc` selection equivalent to `iloc` selection; and it is closely related to "auto increment" integer columns found in databases, such as in MySQL (the `AUTO_INCREMENT` keyword), SQLite (the `AUTOINCREMENT` keyword), or PostgreSQL (the `SERIAL` pseudo-type).

While index objects that provide scrutable labels into data are a key feature of libraries like Pandas and StaticFrame, there are many situations where the simple, inscrutable AIII is needed, such as when data does not have a meaningful index, or in concatenation of data with redundant indices. Offering convenient and consistent approaches to creating these indices supports creating more maintainable code.

All examples use StaticFrame 0.4.0 or later (https://pypi.org/project/static-frame) and import with the following convention:

```
>>> import static_frame as sf
```

10.1 Reindexing & Relabeling

We will take a brief detour to consider how reindexing and relabeling work in Pandas and StaticFrame.

Changing an index on a `Series` or `Frame` could be done in at least two ways: (1) create a new container with a new index of any size, supplying labels with values from the old container if those labels are in the old index (i.e., alignment based on index labels) or (2) create a new container with a new index of the same size, reusing the same values in the same position (alignment based on position).

Following the precedent of Pandas, StaticFrame implements `Series.reindex()` and `Frame.reindex()` with the former interpretation: alignment based on index labels. As shown in the example below, the new index only matches and retains two of the four previous values:

```
>>> s1 = sf.Series((x * 100 for x in range(1, 5)), index=tuple('wxyz'))
>>> s1
<Series>
<Index>
w         100
x         200
y         300
z         400
<<U1>     <int64>
```

```
>>> s1.reindex(tuple('stwx'), fill_value=0)
<Series>
<Index>
s         0
t         0
w         100
x         200
<<U1>     <int64>
```

To handle the latter interpretation, alignment based on position, Pandas offers at least two approaches: the mutable `index` attribute can be directly assigned, or the `set_axis()` function can be used.

StaticFrame names all methods "relabel" that supply a new or transformed index of the same size, to be aligned by position. The `Series.relabel()` method can be used to create a new index by transforming old index labels (via a function or mapping), or by supplying an appropriately sized index initializer. As NumPy arrays in StaticFrame are immutable, relabeling is efficient: underlying data is never copied.

```
>>> s1.relabel(tuple('abcd'))
<Series>
<Index>
a         100
b         200
c         300
d         400
<<U1>     <int64>
```

10.2 Setting an Auto-Incremented Integer Index

A common use of index assignment based on position is "resetting" the index: replacing an existing index with an auto-incremented integer index (AIII). AIIIs are given to `Series` and `Frame` created without explicit index arguments; they are also useful when combining data that does not have a "natural" index along an axis.

While Pandas offers a discrete method for this operation, `reset_index()`, that function is made complex due to the `drop` and `inplace` parameters. For example, `reset_index()` will produce, from a `pd.Series`, a new `pd.Series` or a `pd.Frame` depending on if `drop` is `True` or `False`, and exposes a conflicting parameter configuration if `drop` is `False` and `inplace` is `True`, raising "TypeError: Cannot reset_index inplace on a Series to create a DataFrame."

A goal in StaticFrame's API design is to avoid, as much as possible, interfaces that permit conflicting, non-orthogonal arguments.

In addition to relabeling, another case where an AIII is frequently needed is in concatenating numerous `Series` or `Frame`. For example, when concatenating a `Frame`, one axis might be aligned while the other, extended axis requires an AIII. Deviating in naming from of the `reset_index()` method, Pandas supports this with a Boolean `ignore_index` parameter provided to the `pd.concat()` function.

Another goal of StaticFrame's API design is to support common interfaces wherever possible. Reusing, across diverse interfaces, the same mechanism for creating AIIIs is thus desirable.

10.3 The `IndexAutoFactory` Type

Rather than specialized functions or arguments, AIIIs in StaticFrame can be created on `Series` or `Frame` by passing a special value, an `IndexAutoFactory` object, to index initializer arguments. This is presently supported

for `Series.relabel()`, `Frame.relabel()`, `Series.from_concat()`, and `Frame.from_concat()`. `Series` and `Frame` initializers similarly can take an `IndexAutoFactory`.

By using a special type that can be supplied to existing `index` or `columns` arguments, StaticFrame avoids non-orthogonal arguments and offers a consistent interface for producing AIIIs.

10.4 Resetting an Index when Relabeling

By accepting an `IndexAutoFactory` argument, a `relabel()` method can be used to cover the functionality of the Pandas `reset_index()` method.

For example, the `IndexAutoFactory` class can be given as the `index` argument to `Series.relabel()` to produce a new `Series` with an AIII. As mentioned above, as underlying NumPy arrays are immutable in StaticFrame, this is a no-copy operation.

```
>>> s1.relabel(sf.IndexAutoFactory)
<Series>
<Index>
0         100
1         200
2         300
3         400
<int64>   <int64>
```

The benefit of having a specific type, rather than using `None`, to signify application of an AIII is made more clear in the context of `Frame.relabel()`, where both a `columns` and `index` argument can be set independently. The example bellow demonstrates creating a `Frame`, setting an AIII on both axis, and setting an AIII on `columns` while doing relabeling on the `index`.

```
>>> f1 = sf.Frame.from_dict(dict(a=(1,2), b=(True, False)), index=tuple('xy'))
>>> f1
<Frame>
<Index> a       b      <<U1>
<Index>
x       1       True
y       2       False
<<U1>   <int64> <bool>
```

```
>>> f1.relabel(index=sf.IndexAutoFactory, columns=sf.IndexAutoFactory)
<Frame>
<Index> 0       1      <int64>
<Index>
0       1       True
1       2       False
<int64> <int64> <bool>
```

```
>>> f1.relabel(index=tuple('ab'), columns=sf.IndexAutoFactory)
<Frame>
<Index> 0       1      <int64>
<Index>
a       1       True
b       2       False
<<U1>   <int64> <bool>
```

10.5 Resetting an Index when Concatenating

Concatinating `Series` and `Frame` is a context where supplying a new index is often desirable along the extended axis. The `IndexAutoFactory` type can be used here to supply that index.

For example, when concatenating (vertically stacking) with `Series.from_concat()`, we must supply a new index if the resulting index is not unique. Unlike Pandas, StaticFrame requires all indices to have unique values.

```
>>> s1
<Series>
<Index>
w         100
x         200
y         300
z         400
<<U1>     <int64>
```

```
>>> sf.Series.from_concat((s1, s1), index=tuple('abcdefgh'))
<Series>
<Index>
a         100
b         200
c         300
d         400
e         100
f         200
g         300
h         400
<<U1>     <int64>
```

However, if an AIII is needed, the `IndexAutoFactory` type can be used with the same interface:

```
>>> sf.Series.from_concat((s1, s1), index=sf.IndexAutoFactory)
<Series>
<Index>
0         100
1         200
2         300
3         400
4         100
5         200
6         300
7         400
<int64>   <int64>
```

The same approach is used with `Frame.from_concat()`, where both `columns` and `index` arguments are exposed. For example, two `Series` can be horizontally "stacked" along axis 1 to produce a new `Frame`. If the `Series.name` attributes are unique, they can be used to create the columns; otherwise, new columns can be supplied or an `IndexAutoFactory` value can be provided.

```
>>> s2 = s1 * .5
>>> sf.Frame.from_concat((s1, s2), axis=1, columns=sf.IndexAutoFactory)
<Frame>
<Index> 0         1         <int64>
<Index>
w         100       50.0
x         200       100.0
```

(continues on next page)

(continued from previous page)

```
y       300     150.0
z       400     200.0
<<U1>   <int64> <float64>
```

Similarly, concatenating along axis 1 (horizontally stacking) the same `Frame` multiple times results in non-unique columns, which raises an `Exception` in StaticFrame. To avoid this, the `IndexAutoFactory` can be supplied.

```
>>> sf.Frame.from_concat((f1, f1), axis=1, columns=sf.IndexAutoFactory)
<Frame>
<Index> 0       1      2       3      <int64>
<Index>
x       1       True   1       True
y       2       False  2       False
<<U1>   <int64> <bool> <int64> <bool>
```

10.6 Consistent Interfaces for More Maintainable Code

Resetting an index is not a complex operation. However, how to provide the option to create an AIII within diverse interfaces is not obvious. The approach taken with StaticFrame offers a consistent interface, leading to more maintainable code.

For more information about StaticFrame, see the documentation (http://static-frame.readthedocs.io) or project (https://github.com/InvestmentSystems/static-frame) sites. Feedback is encouraged.

CHAPTER

ELEVEN

API OVERVIEW

For each container, the complete public API is presented. Note that interface endpoints, such as `iter_element`, are expanded to show all interface sub components.

This is designed as an overview; for detailed documentation, start with *Structures*.

11.1 Series

11.1.1 Series: Attribute

static_frame.Series.T	
static_frame.Series.dtype	Return the dtype of the underlying NumPy array. Re…
static_frame.Series.index	The IndexBase instance assigned for labels.
static_frame.Series.mloc	The memory location, represented as an integer, of…
static_frame.Series.name	
static_frame.Series.nbytes	Return the total bytes of the underlying NumPy arr…
static_frame.Series.ndim	Return the number of dimensions, which for a Serie…
static_frame.Series.shape	Return a tuple describing the shape of the underly…
static_frame.Series.size	Return the size of the underlying NumPy array. Ret…

11.1.2 Series: Constructor

static_frame.Series.__init__()	
static_frame.Series.from_concat()	Concatenate multiple Series into a new Series. Arg…
static_frame.Series.from_dict()	Series construction from a dictionary, where the f…
static_frame.Series.from_items()	Series construction from an iterator or generator…
static_frame.Series.from_pandas()	Given a Pandas Series, return a Series. Args: valu…

11.1.3 Series: Dictionary-Like

static_frame.Series.__contains__()	Inclusion of value in index labels.
static_frame.Series.__iter__()	Iterator of index labels, same as Series.keys.
static_frame.Series.__reversed__()	Returns a reverse iterator on the series' index.
static_frame.Series.get()	Return the value found at the index key, else the...
static_frame.Series.items()	Iterator of pairs of index label and value.
static_frame.Series.keys()	Iterator of index labels.
static_frame.Series.values	

11.1.4 Series: Display

static_frame.Series.__repr__()	
static_frame.Series.__str__()	Return str(self).
static_frame.Series.display()	Return a Display of the Series.
static_frame.Series.display_tall()	
static_frame.Series.display_wide()	
static_frame.Series.interface	A Frame documenting the interface of this class.

11.1.5 Series: Exporter

static_frame.Series.to_frame()	Return a :py:class:static_frame.Frame view of this...
static_frame.Series.to_frame_go()	Return :py:class:static_frame.FrameGO view of this...
static_frame.Series.to_html()	Return an HTML table representation of this Series...
static_frame.Series.to_html_datatables()	Return a complete HTML representation of this Seri...
static_frame.Series.to_pairs()	Return a tuple of tuples, where each inner tuple i...
static_frame.Series.to_pandas()	Return a Pandas Series.

11.1.6 Series: Iterator

static_frame.Series.iter_element(axis)	
static_frame.Series.iter_element(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_element(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.Series.iter_element(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.Series.iter_element(axis).apply_pool()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_element_items(axis)	
static_frame.Series.iter_element_items(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_element_items(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.Series.iter_element_items(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.Series.iter_element_items(axis).apply_pool()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_group(axis)	
static_frame.Series.iter_group(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_group(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.Series.iter_group(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.Series.iter_group(axis).apply_pool()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_group_index(axis)	
static_frame.Series.iter_group_index(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_group_index(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.Series.iter_group_index(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.Series.iter_group_index(axis).apply_pool()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_group_index_items(axis)	
static_frame.Series.iter_group_index_items(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_group_index_items(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.Series.iter_group_index_items(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.Series.iter_group_index_items(axis).apply_pool()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_group_items(axis)	
static_frame.Series.iter_group_items(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.Series.iter_group_items(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.Series.iter_group_items(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.Series.iter_group_items(axis).apply_pool()	Apply passed function to each object iterated, whe…

11.1.7 Series: Method

static_frame.Series.__len__()	Length of values.
static_frame.Series.all()	Logical and over values along the specified axis....
static_frame.Series.any()	Logical or over values along the specified axis. A...
static_frame.Series.astype	Return a Series with type determined by dtype argu...
static_frame.Series.clip()	Apply a clip opertion to this Series. Note that cl...
static_frame.Series.cumprod()	Return the cumulative product over the specified a...
static_frame.Series.cumsum()	Return the cumulative sum over the specified axis....
static_frame.Series.drop_duplicated()	Return a Series with duplicated values removed.
static_frame.Series.dropna()	Return a new Series after removing values of NaN o...
static_frame.Series.duplicated()	Return a same-sized Boolean Series that shows True...
static_frame.Series.fillna()	Return a new Series after replacing null (NaN or N...
static_frame.Series.fillna_backward()	Return a new Series after feeding backward the las...
static_frame.Series.fillna_forward()	Return a new Series after feeding forward the last...
static_frame.Series.fillna_leading()	Return a new Series after filling leading (and onl...
static_frame.Series.fillna_trailing()	Return a new Series after filling trailing (and on...
static_frame.Series.head()	Return a Series consisting only of the top element...
static_frame.Series.iloc_max()	Return the integer index corresponding to the maxi...
static_frame.Series.iloc_min()	Return the integer index corresponding to the mini...
static_frame.Series.isin()	Return a same-sized Boolean Series that shows if t...
static_frame.Series.isna()	Return a same-indexed, Boolean Series indicating w...
static_frame.Series.loc_max()	Return the label corresponding to the maximum valu...
static_frame.Series.loc_min()	Return the label corresponding to the minimum valu...
static_frame.Series.max()	Return the maximum along the specified axis. Args:...
static_frame.Series.mean()	Return the mean along the specified axis. Args: ax...
static_frame.Series.median()	Return the median along the specified axis. Args:...
static_frame.Series.min()	Return the minimum along the specified axis. Args:...
static_frame.Series.notna()	Return a same-indexed, Boolean Series indicating w...
static_frame.Series.prod()	Return the product along the specified axis. Args:...
static_frame.Series.reindex()	Return a new Series with labels defined by the pro...
static_frame.Series.relabel()	Return a new Series with transformed labels on the...
static_frame.Series.relabel_add_level()	Return a new Series, adding a new root level to an...
static_frame.Series.relabel_drop_level()	Return a new Series, dropping one or more levels f...
static_frame.Series.relabel_flat()	Return a new Series, where an IndexHierarchy (if d...
static_frame.Series.rename()	Return a new Series with an updated name attribute...
static_frame.Series.roll()	Return a Series with values rotated forward and wr...
static_frame.Series.shift()	Return a Series with values shifted forward on the...
static_frame.Series.sort_index()	Return a new Series ordered by the sorted Index.
static_frame.Series.sort_values()	Return a new Series ordered by the sorted values.
static_frame.Series.std()	Return the standard deviaton along the specified a...
static_frame.Series.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.Series.tail()	Return a Series consisting only of the bottom elem...
static_frame.Series.transpose()	The transpositon of a Series is itself.
static_frame.Series.unique()	Return a NumPy array of unqiue values.
static_frame.Series.var()	Return the variance along the specified axis. Args...

11.1.8 Series: Operator Binary

static_frame.Series.__add__()	Same as a + b.
static_frame.Series.__and__()	Same as a & b.
static_frame.Series.__eq__()	Same as a == b.
static_frame.Series.__floordiv__()	Same as a // b.
static_frame.Series.__ge__()	Same as a >= b.
static_frame.Series.__gt__()	Same as a > b.
static_frame.Series.__le__()	Same as a <= b.
static_frame.Series.__lt__()	Same as a < b.
static_frame.Series.__matmul__()	Same as a @ b.
static_frame.Series.__mod__()	Same as a % b.
static_frame.Series.__mul__()	Same as a * b.
static_frame.Series.__ne__()	Same as a != b.
static_frame.Series.__or__()	Same as a | b.
static_frame.Series.__pow__()	Same as a ** b.
static_frame.Series.__radd__()	Same as a + b.
static_frame.Series.__rfloordiv__()	Same as a // b.
static_frame.Series.__rmatmul__()	Same as a @ b.
static_frame.Series.__rmul__()	Same as a * b.
static_frame.Series.__rshift__()	Same as a >> b.
static_frame.Series.__rsub__()	Same as a - b.
static_frame.Series.__rtruediv__()	Same as a / b.
static_frame.Series.__sub__()	Same as a - b.
static_frame.Series.__truediv__()	Same as a / b.
static_frame.Series.__xor__()	Same as a ^ b.

11.1.9 Series: Operator Unary

static_frame.Series.__abs__()	Same as abs(a).
static_frame.Series.__invert__()	Same as ~a.
static_frame.Series.__neg__()	Same as -a.
static_frame.Series.__pos__()	Same as +a.

11.1.10 Series: Selector

static_frame.Series[]	Selector of values by label. Args: key: A loc sele…
static_frame.Series.assign.iloc[]	Integer-position based selection.
static_frame.Series.assign.loc[]	Label-based selection.
static_frame.Series.assign[]	Label-based selection.
static_frame.Series.drop.iloc[]	Integer-position based selection.
static_frame.Series.drop.loc[]	Label-based selection.
static_frame.Series.drop[]	Label-based selection.
static_frame.Series.iloc[]	
static_frame.Series.loc[]	
static_frame.Series.mask.iloc[]	Integer-position based selection.
static_frame.Series.mask.loc[]	Label-based selection.
static_frame.Series.mask[]	Label-based selection.
static_frame.Series.masked_array.iloc[]	Integer-position based selection.
static_frame.Series.masked_array.loc[]	Label-based selection.
static_frame.Series.masked_array[]	Label-based selection.

11.2 Frame

11.2.1 Frame: Attribute

static_frame.Frame.T	Return a transposed version of the Frame.
static_frame.Frame.columns	The IndexBase instance assigned for column labels.
static_frame.Frame.dtypes	Return a Series of dytpes for each realizable colu…
static_frame.Frame.index	The IndexBase instance assigned for row labels.
static_frame.Frame.mloc	The memory locations, represented as an array of i…
static_frame.Frame.name	
static_frame.Frame.nbytes	Return the total bytes of the underlying NumPy arr…
static_frame.Frame.ndim	Return the number of dimensions, which for a Frame…
static_frame.Frame.shape	Return a tuple describing the shape of the underly…
static_frame.Frame.size	Return the size of the underlying NumPy array. Ret…

11.2.2 Frame: Constructor

static_frame.Frame.__init__()	
static_frame.Frame.from_concat()	Concatenate multiple Frames into a new Frame. If i...
static_frame.Frame.from_csv()	Create a Frame from a file path or a file-like obj...
static_frame.Frame.from_dict()	Create a Frame from a dictionary, or any object th...
static_frame.Frame.from_element_iloc_items()	Given an iterable of pairs of iloc coordinates and...
static_frame.Frame.from_element_loc_items()	This function is partialed (setting the index and...
static_frame.Frame.from_items()	Frame constructor from an iterator or generator of...
static_frame.Frame.from_json()	Frame constructor from an in-memory JSON document....
static_frame.Frame.from_json_url()	Frame constructor from a JSON documenst provided v...
static_frame.Frame.from_pandas()	Given a Pandas DataFrame, return a Frame. Args: va...
static_frame.Frame.from_records()	Frame constructor from an iterable of rows, where...
static_frame.Frame.from_records_items()	Frame constructor from iterable of pairs of index...
static_frame.Frame.from_sql()	Frame constructor from an SQL query and a database...
static_frame.Frame.from_structured_array()	Convert a NumPy structed array into a Frame. Args:...
static_frame.Frame.from_tsv()	Specialized version of Frame.from_csv for TSV file...

11.2.3 Frame: Dictionary-Like

static_frame.Frame.__contains__()	Inclusion of value in column labels.
static_frame.Frame.__iter__()	Iterator of column labels, same as Frame.keys.
static_frame.Frame.__reversed__()	Returns a reverse iterator on the frame's columns.
static_frame.Frame.get()	Return the value found at the columns key, else th...
static_frame.Frame.items()	Iterator of pairs of column label and correspondin...
static_frame.Frame.keys()	Iterator of column labels.
static_frame.Frame.values	A 2D array of values. Note: type coercion might be...

11.2.4 Frame: Display

static_frame.Frame.__repr__()	
static_frame.Frame.__str__()	Return str(self).
static_frame.Frame.display()	
static_frame.Frame.display_tall()	
static_frame.Frame.display_wide()	
static_frame.Frame.interface	A Frame documenting the interface of this class.

11.2.5 Frame: Exporter

static_frame.Frame.to_csv()	Given a file path or file-like object, write the F...
static_frame.Frame.to_frame_go()	Return a FrameGO view of this Frame. As underlying...
static_frame.Frame.to_html()	Return an HTML table representation of this Frame...
static_frame.Frame.to_html_datatables()	Return a complete HTML representation of this Fram...
static_frame.Frame.to_pairs()	Return a tuple of major axis key, minor axis key v...
static_frame.Frame.to_pandas()	Return a Pandas DataFrame.
static_frame.Frame.to_tsv()	Given a file path or file-like object, write the F...

11.2.6 Frame: Iterator

static_frame.Frame.iter_array(axis)	
static_frame.Frame.iter_array(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_array(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_array(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_array(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_array_items(axis)	
static_frame.Frame.iter_array_items(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_array_items(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_array_items(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_array_items(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_element(axis)	
static_frame.Frame.iter_element(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_element(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_element(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_element(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_element_items(axis)	
static_frame.Frame.iter_element_items(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_element_items(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_element_items(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_element_items(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_group(axis)	
static_frame.Frame.iter_group(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_group(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_group(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_group(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_group_index(axis)	
static_frame.Frame.iter_group_index(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_group_index(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_group_index(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_group_index(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_group_index_items(axis)	
static_frame.Frame.iter_group_index_items(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_group_index_items(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_group_index_items(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_group_index_items(axis).apply_pool()	Apply passed function to each c

Con

Table 2 – continued from previous page

static_frame.Frame.iter_group_items(axis)	
static_frame.Frame.iter_group_items(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_group_items(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_group_items(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_group_items(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_series(axis)	
static_frame.Frame.iter_series(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_series(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_series(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_series(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_series_items(axis)	
static_frame.Frame.iter_series_items(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_series_items(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_series_items(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_series_items(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_tuple(axis)	
static_frame.Frame.iter_tuple(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_tuple(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_tuple(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_tuple(axis).apply_pool()	Apply passed function to each c
static_frame.Frame.iter_tuple_items(axis)	
static_frame.Frame.iter_tuple_items(axis).apply()	Apply passed function to each c
static_frame.Frame.iter_tuple_items(axis).apply_iter()	Generator that applies the passe
static_frame.Frame.iter_tuple_items(axis).apply_iter_items()	Generator that applies function
static_frame.Frame.iter_tuple_items(axis).apply_pool()	Apply passed function to each c

11.2.7 Frame: Method

static_frame.Frame.__len__()	Length of rows in values.
static_frame.Frame.all()	Logical and over values along the specified axis....
static_frame.Frame.any()	Logical or over values along the specified axis. A...
static_frame.Frame.astype	Retype one or more columns. Can be used as as func...
static_frame.Frame.astype[]	Selector of columns by label. Args: key: A loc sel...
static_frame.Frame.clip()	Apply a clip opertion to this Frame. Note that cli...
static_frame.Frame.cumprod()	Return the cumulative product over the specified a...
static_frame.Frame.cumsum()	Return the cumulative sum over the specified axis....
static_frame.Frame.drop_duplicated()	Return a Frame with duplicated values removed.
static_frame.Frame.dropna()	Return a new Frame after removing rows (axis 0) or...
static_frame.Frame.duplicated()	Return an axis-sized Boolean Series that shows Tru...
static_frame.Frame.fillna()	Return a new Frame after replacing null (NaN or No...
static_frame.Frame.fillna_backward()	Return a new Frame after filling backward null (Na...
static_frame.Frame.fillna_forward()	Return a new Frame after filling forward null (NaN...
static_frame.Frame.fillna_leading()	Return a new Frame after filling leading (and only...
static_frame.Frame.fillna_trailing()	Return a new Frame after filling trailing (and onl...
static_frame.Frame.head()	Return a Frame consisting only of the top rows as...
static_frame.Frame.iloc_max()	Return the integer indices corresponding to the ma...
static_frame.Frame.iloc_min()	Return the integer indices corresponding to the mi...
static_frame.Frame.isin()	Return a same-sized Boolean Frame that shows if th...
static_frame.Frame.isna()	Return a same-indexed, Boolean Frame indicating Tr...

Continued on next page

Table 3 – continued from previous page

static_frame.Frame.loc_max()	Return the labels corresponding to the maximum val…
static_frame.Frame.loc_min()	Return the labels corresponding to the minimum val…
static_frame.Frame.max()	Return the maximum along the specified axis. Args:…
static_frame.Frame.mean()	Return the mean along the specified axis. Args: ax…
static_frame.Frame.median()	Return the median along the specified axis. Args:…
static_frame.Frame.min()	Return the minimum along the specified axis. Args:…
static_frame.Frame.notna()	Return a same-indexed, Boolean Frame indicating Tr…
static_frame.Frame.prod()	Return the product along the specified axis. Args:…
static_frame.Frame.reindex()	Return a new Frame with labels defined by the prov…
static_frame.Frame.relabel()	Return a new Frame with transformed labels on the…
static_frame.Frame.relabel_add_level()	Return a new Frame, adding a new root level to an…
static_frame.Frame.relabel_drop_level()	Return a new Frame, dropping one or more levels fr…
static_frame.Frame.relabel_flat()	Return a new Frame, where an IndexHierarchy (if de…
static_frame.Frame.rename()	Return a new Frame with an updated name attribute.
static_frame.Frame.roll()	Args: include_index: Determine if index is include…
static_frame.Frame.set_index()	Return a new frame produced by setting the given c…
static_frame.Frame.set_index_hierarchy()	Given an iterable of column labels, return a new F…
static_frame.Frame.shift()	
static_frame.Frame.sort_columns()	Return a new Frame ordered by the sorted Columns.
static_frame.Frame.sort_index()	Return a new Frame ordered by the sorted Index.
static_frame.Frame.sort_values()	Return a new Frame ordered by the sorted values, w…
static_frame.Frame.std()	Return the standard deviaton along the specified a…
static_frame.Frame.sum()	Sum values along the specified axis. Args: axis: A…
static_frame.Frame.tail()	Return a Frame consisting only of the bottom rows…
static_frame.Frame.transpose()	Return a tansposed version of the Frame.
static_frame.Frame.unique()	Return a NumPy array of unqiue values. If the axis…
static_frame.Frame.var()	Return the variance along the specified axis. Args…

11.2.8 Frame: Operator Binary

static_frame.Frame.__add__()	Same as a + b.
static_frame.Frame.__and__()	Same as a & b.
static_frame.Frame.__eq__()	Same as a == b.
static_frame.Frame.__floordiv__()	Same as a // b.
static_frame.Frame.__ge__()	Same as a >= b.
static_frame.Frame.__gt__()	Same as a > b.
static_frame.Frame.__le__()	Same as a <= b.
static_frame.Frame.__lt__()	Same as a < b.
static_frame.Frame.__matmul__()	Same as a @ b.
static_frame.Frame.__mod__()	Same as a % b.
static_frame.Frame.__mul__()	Same as a * b.
static_frame.Frame.__ne__()	Same as a != b.
static_frame.Frame.__or__()	Same as a | b.
static_frame.Frame.__pow__()	Same as a ** b.
static_frame.Frame.__radd__()	Same as a + b.
static_frame.Frame.__rfloordiv__()	Same as a // b.
static_frame.Frame.__rmatmul__()	Same as a @ b.
static_frame.Frame.__rmul__()	Same as a * b.
static_frame.Frame.__rshift__()	Same as a >> b.
static_frame.Frame.__rsub__()	Same as a - b.
static_frame.Frame.__rtruediv__()	Same as a / b.
static_frame.Frame.__sub__()	Same as a - b.
static_frame.Frame.__truediv__()	Same as a / b.
static_frame.Frame.__xor__()	Same as a ^ b.

11.2.9 Frame: Operator Unary

static_frame.Frame.__abs__()	Same as abs(a).
static_frame.Frame.__invert__()	Same as ~a.
static_frame.Frame.__neg__()	Same as -a.
static_frame.Frame.__pos__()	Same as +a.

11.2.10 Frame: Selector

static_frame.Frame[]	Selector of columns by label. Args: key: A loc sel...
static_frame.Frame.assign.iloc[]	Integer-position based selection.
static_frame.Frame.assign.loc[]	Label-based selection.
static_frame.Frame.assign[]	Label-based selection.
static_frame.Frame.drop.iloc[]	Integer-position based selection.
static_frame.Frame.drop.loc[]	Label-based selection.
static_frame.Frame.drop[]	Label-based selection.
static_frame.Frame.iloc[]	
static_frame.Frame.loc[]	
static_frame.Frame.mask.iloc[]	Integer-position based selection.
static_frame.Frame.mask.loc[]	Label-based selection.
static_frame.Frame.mask[]	Label-based selection.
static_frame.Frame.masked_array.iloc[]	Integer-position based selection.
static_frame.Frame.masked_array.loc[]	Label-based selection.
static_frame.Frame.masked_array[]	Label-based selection.

11.3 FrameGO

11.3.1 FrameGO: Attribute

static_frame.FrameGO.T	Return a transposed version of the Frame.
static_frame.FrameGO.columns	The IndexBase instance assigned for column labels.
static_frame.FrameGO.dtypes	Return a Series of dytpes for each realizable colu...
static_frame.FrameGO.index	The IndexBase instance assigned for row labels.
static_frame.FrameGO.mloc	The memory locations, represented as an array of i...
static_frame.FrameGO.name	
static_frame.FrameGO.nbytes	Return the total bytes of the underlying NumPy arr...
static_frame.FrameGO.ndim	Return the number of dimensions, which for a Frame...
static_frame.FrameGO.shape	Return a tuple describing the shape of the underly...
static_frame.FrameGO.size	Return the size of the underlying NumPy array. Ret...

11.3.2 FrameGO: Constructor

static_frame.FrameGO.__init__()	
static_frame.FrameGO.from_concat()	Concatenate multiple Frames into a new Frame. If i...
static_frame.FrameGO.from_csv()	Create a Frame from a file path or a file-like obj...
static_frame.FrameGO.from_dict()	Create a Frame from a dictionary, or any object th...
static_frame.FrameGO.from_element_iloc_items()	Given an iterable of pairs of iloc coordinates and...
static_frame.FrameGO.from_element_loc_items()	This function is partialed (setting the index and...
static_frame.FrameGO.from_items()	Frame constructor from an iterator or generator of...
static_frame.FrameGO.from_json()	Frame constructor from an in-memory JSON document....
static_frame.FrameGO.from_json_url()	Frame constructor from a JSON documenst provided v...
static_frame.FrameGO.from_pandas()	Given a Pandas DataFrame, return a Frame. Args: va...
static_frame.FrameGO.from_records()	Frame constructor from an iterable of rows, where...
static_frame.FrameGO.from_records_items()	Frame constructor from iterable of pairs of index...
static_frame.FrameGO.from_sql()	Frame constructor from an SQL query and a database...
static_frame.FrameGO.from_structured_array()	Convert a NumPy structed array into a Frame. Args:...
static_frame.FrameGO.from_tsv()	Specialized version of Frame.from_csv for TSV file...

11.3.3 FrameGO: Dictionary-Like

static_frame.FrameGO.__contains__()	Inclusion of value in column labels.
static_frame.FrameGO.__iter__()	Iterator of column labels, same as Frame.keys.
static_frame.FrameGO.__reversed__()	Returns a reverse iterator on the frame's columns.
static_frame.FrameGO.get()	Return the value found at the columns key, else th...
static_frame.FrameGO.items()	Iterator of pairs of column label and correspondin...
static_frame.FrameGO.keys()	Iterator of column labels.
static_frame.FrameGO.values	A 2D array of values. Note: type coercion might be...

11.3.4 FrameGO: Display

static_frame.FrameGO.__repr__()	
static_frame.FrameGO.__str__()	Return str(self).
static_frame.FrameGO.display()	
static_frame.FrameGO.display_tall()	
static_frame.FrameGO.display_wide()	
static_frame.FrameGO.interface	A Frame documenting the interface of this class.

11.3.5 FrameGO: Exporter

static_frame.FrameGO.to_csv()	Given a file path or file-like object, write the F...
static_frame.FrameGO.to_frame()	Return Frame version of this Frame.
static_frame.FrameGO.to_frame_go()	Return a FrameGO version of this Frame.
static_frame.FrameGO.to_html()	Return an HTML table representation of this Frame...
static_frame.FrameGO.to_html_datatables()	Return a complete HTML representation of this Fram...
static_frame.FrameGO.to_pairs()	Return a tuple of major axis key, minor axis key v...
static_frame.FrameGO.to_pandas()	Return a Pandas DataFrame.
static_frame.FrameGO.to_tsv()	Given a file path or file-like object, write the F...

11.3.6 FrameGO: Iterator

static_frame.FrameGO.iter_array(axis)	
static_frame.FrameGO.iter_array(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_array(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_array(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_array(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_array_items(axis)	
static_frame.FrameGO.iter_array_items(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_array_items(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_array_items(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_array_items(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_element(axis)	
static_frame.FrameGO.iter_element(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_element(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_element(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_element(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_element_items(axis)	
static_frame.FrameGO.iter_element_items(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_element_items(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_element_items(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_element_items(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_group(axis)	
static_frame.FrameGO.iter_group(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_group(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_group(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_group(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_group_index(axis)	
static_frame.FrameGO.iter_group_index(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_group_index(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_group_index(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_group_index(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_group_index_items(axis)	
static_frame.FrameGO.iter_group_index_items(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_group_index_items(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_group_index_items(axis).apply_iter_items()	Generator that applies functi

C

Table 4 – continued from previous page

static_frame.FrameGO.iter_group_index_items(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_group_items(axis)	
static_frame.FrameGO.iter_group_items(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_group_items(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_group_items(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_group_items(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_series(axis)	
static_frame.FrameGO.iter_series(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_series(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_series(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_series(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_series_items(axis)	
static_frame.FrameGO.iter_series_items(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_series_items(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_series_items(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_series_items(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_tuple(axis)	
static_frame.FrameGO.iter_tuple(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_tuple(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_tuple(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_tuple(axis).apply_pool()	Apply passed function to eac
static_frame.FrameGO.iter_tuple_items(axis)	
static_frame.FrameGO.iter_tuple_items(axis).apply()	Apply passed function to eac
static_frame.FrameGO.iter_tuple_items(axis).apply_iter()	Generator that applies the pa
static_frame.FrameGO.iter_tuple_items(axis).apply_iter_items()	Generator that applies functi
static_frame.FrameGO.iter_tuple_items(axis).apply_pool()	Apply passed function to eac

11.3.7 FrameGO: Method

static_frame.FrameGO.__len__()	Length of rows in values.
static_frame.FrameGO.all()	Logical and over values along the specified axis....
static_frame.FrameGO.any()	Logical or over values along the specified axis. A...
static_frame.FrameGO.astype	Retype one or more columns. Can be used as as func...
static_frame.FrameGO.astype[]	Selector of columns by label. Args: key: A loc sel...
static_frame.FrameGO.clip()	Apply a clip opertion to this Frame. Note that cli...
static_frame.FrameGO.cumprod()	Return the cumulative product over the specified a...
static_frame.FrameGO.cumsum()	Return the cumulative sum over the specified axis....
static_frame.FrameGO.drop_duplicated()	Return a Frame with duplicated values removed.
static_frame.FrameGO.dropna()	Return a new Frame after removing rows (axis 0) or...
static_frame.FrameGO.duplicated()	Return an axis-sized Boolean Series that shows Tru...
static_frame.FrameGO.extend()	Extend this FrameGO (in-place) with another Frame'...
static_frame.FrameGO.extend_items()	Given an iterable of pairs of column name, column...
static_frame.FrameGO.fillna()	Return a new Frame after replacing null (NaN or No...
static_frame.FrameGO.fillna_backward()	Return a new Frame after filling backward null (Na...
static_frame.FrameGO.fillna_forward()	Return a new Frame after filling forward null (NaN...
static_frame.FrameGO.fillna_leading()	Return a new Frame after filling leading (and only...
static_frame.FrameGO.fillna_trailing()	Return a new Frame after filling trailing (and onl...
static_frame.FrameGO.head()	Return a Frame consisting only of the top rows as...
static_frame.FrameGO.iloc_max()	Return the integer indices corresponding to the ma...

Continued on next page

Table 5 – continued from previous page

static_frame.FrameGO.iloc_min()	Return the integer indices corresponding to the mi…
static_frame.FrameGO.isin()	Return a same-sized Boolean Frame that shows if th…
static_frame.FrameGO.isna()	Return a same-indexed, Boolean Frame indicating Tr…
static_frame.FrameGO.loc_max()	Return the labels corresponding to the maximum val…
static_frame.FrameGO.loc_min()	Return the labels corresponding to the minimum val…
static_frame.FrameGO.max()	Return the maximum along the specified axis. Args:…
static_frame.FrameGO.mean()	Return the mean along the specified axis. Args: ax…
static_frame.FrameGO.median()	Return the median along the specified axis. Args:…
static_frame.FrameGO.min()	Return the minimum along the specified axis. Args:…
static_frame.FrameGO.notna()	Return a same-indexed, Boolean Frame indicating Tr…
static_frame.FrameGO.prod()	Return the product along the specified axis. Args:…
static_frame.FrameGO.reindex()	Return a new Frame with labels defined by the prov…
static_frame.FrameGO.relabel()	Return a new Frame with transformed labels on the…
static_frame.FrameGO.relabel_add_level()	Return a new Frame, adding a new root level to an…
static_frame.FrameGO.relabel_drop_level()	Return a new Frame, dropping one or more levels fr…
static_frame.FrameGO.relabel_flat()	Return a new Frame, where an IndexHierarchy (if de…
static_frame.FrameGO.rename()	Return a new Frame with an updated name attribute.
static_frame.FrameGO.roll()	Args: include_index: Determine if index is include…
static_frame.FrameGO.set_index()	Return a new frame produced by setting the given c…
static_frame.FrameGO.set_index_hierarchy()	Given an iterable of column labels, return a new F…
static_frame.FrameGO.shift()	
static_frame.FrameGO.sort_columns()	Return a new Frame ordered by the sorted Columns.
static_frame.FrameGO.sort_index()	Return a new Frame ordered by the sorted Index.
static_frame.FrameGO.sort_values()	Return a new Frame ordered by the sorted values, w…
static_frame.FrameGO.std()	Return the standard deviaton along the specified a…
static_frame.FrameGO.sum()	Sum values along the specified axis. Args: axis: A…
static_frame.FrameGO.tail()	Return a Frame consisting only of the bottom rows…
static_frame.FrameGO.transpose()	Return a tansposed version of the Frame.
static_frame.FrameGO.unique()	Return a NumPy array of unqiue values. If the axis…
static_frame.FrameGO.var()	Return the variance along the specified axis. Args…

11.3.8 FrameGO: Operator Binary

static_frame.FrameGO.__add__()	Same as a + b.
static_frame.FrameGO.__and__()	Same as a & b.
static_frame.FrameGO.__eq__()	Same as a == b.
static_frame.FrameGO.__floordiv__()	Same as a // b.
static_frame.FrameGO.__ge__()	Same as a >= b.
static_frame.FrameGO.__gt__()	Same as a > b.
static_frame.FrameGO.__le__()	Same as a <= b.
static_frame.FrameGO.__lt__()	Same as a < b.
static_frame.FrameGO.__matmul__()	Same as a @ b.
static_frame.FrameGO.__mod__()	Same as a % b.
static_frame.FrameGO.__mul__()	Same as a * b.
static_frame.FrameGO.__ne__()	Same as a != b.
static_frame.FrameGO.__or__()	Same as a | b.
static_frame.FrameGO.__pow__()	Same as a ** b.
static_frame.FrameGO.__radd__()	Same as a + b.
static_frame.FrameGO.__rfloordiv__()	Same as a // b.
static_frame.FrameGO.__rmatmul__()	Same as a @ b.
static_frame.FrameGO.__rmul__()	Same as a * b.
static_frame.FrameGO.__rshift__()	Same as a >> b.
static_frame.FrameGO.__rsub__()	Same as a - b.
static_frame.FrameGO.__rtruediv__()	Same as a / b.
static_frame.FrameGO.__sub__()	Same as a - b.
static_frame.FrameGO.__truediv__()	Same as a / b.
static_frame.FrameGO.__xor__()	Same as a ^ b.

11.3.9 FrameGO: Operator Unary

static_frame.FrameGO.__abs__()	Same as abs(a).
static_frame.FrameGO.__invert__()	Same as ~a.
static_frame.FrameGO.__neg__()	Same as -a.
static_frame.FrameGO.__pos__()	Same as +a.

11.3.10 FrameGO: Selector

static_frame.FrameGO[]	Selector of columns by label. Args: key: A loc sel...
static_frame.FrameGO.assign.iloc[]	Integer-position based selection.
static_frame.FrameGO.assign.loc[]	Label-based selection.
static_frame.FrameGO.assign[]	Label-based selection.
static_frame.FrameGO.drop.iloc[]	Integer-position based selection.
static_frame.FrameGO.drop.loc[]	Label-based selection.
static_frame.FrameGO.drop[]	Label-based selection.
static_frame.FrameGO.iloc[]	
static_frame.FrameGO.loc[]	
static_frame.FrameGO.mask.iloc[]	Integer-position based selection.
static_frame.FrameGO.mask.loc[]	Label-based selection.
static_frame.FrameGO.mask[]	Label-based selection.
static_frame.FrameGO.masked_array.iloc[]	Integer-position based selection.
static_frame.FrameGO.masked_array.loc[]	Label-based selection.
static_frame.FrameGO.masked_array[]	Label-based selection.

11.4 Index

11.4.1 Index: Attribute

static_frame.Index.STATIC	bool(x) -> bool Returns True when the argument x i...
static_frame.Index.depth	int([x]) -> integer int(x, base=10) -> integer Con...
static_frame.Index.drop	
static_frame.Index.dtype	Return the dtype of the underlying NumPy array. Re...
static_frame.Index.mloc	The memory location, represented as an integer, of...
static_frame.Index.name	
static_frame.Index.nbytes	Return the total bytes of the underlying NumPy arr...
static_frame.Index.ndim	Return the number of dimensions. Returns: :py:clas...
static_frame.Index.positions	Return the immutable positions array. This is need...
static_frame.Index.shape	Return a tuple describing the shape of the underly...
static_frame.Index.size	Return the size of the underlying NumPy array. Ret...

11.4.2 Index: Constructor

static_frame.Index.__init__()	
static_frame.Index.from_labels()	Construct an Index from an iterable of labels, whe...
static_frame.Index.from_pandas()	Given a Pandas index, return the appropriate Index...

11.4.3 Index: Dictionary-Like

static_frame.Index.__contains__()	Return True if value in the labels.
static_frame.Index.__iter__()	Iterate over labels.
static_frame.Index.__reversed__()	Returns a reverse iterator on the index labels.
static_frame.Index.get()	Return the value found at the index key, else the...
static_frame.Index.items()	Iterator of pairs of index label and value.
static_frame.Index.keys()	Iterator of index labels.
static_frame.Index.values	A 1D array of labels.

11.4.4 Index: Display

static_frame.Index.__repr__()	
static_frame.Index.__str__()	Return str(self).
static_frame.Index.display()	
static_frame.Index.display_tall()	
static_frame.Index.display_wide()	
static_frame.Index.interface	A Frame documenting the interface of this class.

11.4.5 Index: Exporter

static_frame.Index.to_html()	Return an HTML table representation of this Index...
static_frame.Index.to_html_datatables()	Return a complete HTML representation of this Inde...
static_frame.Index.to_pandas()	Return a Pandas Index.
static_frame.Index.to_series()	Return a Series with values from this Index's labe...

11.4.6 Index: Iterator

static_frame.Index.iter_label(axis)	
static_frame.Index.iter_label(axis).apply()	Apply passed function to each object iterated, whe...
static_frame.Index.iter_label(axis).apply_iter()	Generator that applies the passed function to each...
static_frame.Index.iter_label(axis).apply_iter_items()	Generator that applies function to each element it...
static_frame.Index.iter_label(axis).apply_pool()	Apply passed function to each object iterated, whe...

11.4.7 Index: Method

static_frame.Index.__len__()	
static_frame.Index.add_level()	Return an IndexHierarhcy with an added root level.
static_frame.Index.all()	Logical and over values along the specified axis....
static_frame.Index.any()	Logical or over values along the specified axis. A...
static_frame.Index.copy()	Return a new Index.
static_frame.Index.cumprod()	Return the cumulative product over the specified a...
static_frame.Index.cumsum()	Return the cumulative sum over the specified axis....
static_frame.Index.intersection()	Perform intersection with another Index, container...
static_frame.Index.isin()	Return a Boolean array showing True where a label...
static_frame.Index.loc_to_iloc()	Note: Boolean Series are reindexed to this index,...
static_frame.Index.max()	Return the maximum along the specified axis. Args:...
static_frame.Index.mean()	Return the mean along the specified axis. Args: ax...
static_frame.Index.median()	Return the median along the specified axis. Args:...
static_frame.Index.min()	Return the minimum along the specified axis. Args:...
static_frame.Index.prod()	Return the product along the specified axis. Args:...
static_frame.Index.relabel()	Return a new Index with labels replaced by the cal...
static_frame.Index.rename()	Return a new Frame with an updated name attribute.
static_frame.Index.roll()	Return an Index with values rotated forward and wr...
static_frame.Index.sort()	Return a new Index with the labels sorted. Args: k...
static_frame.Index.std()	Return the standard deviaton along the specified a...
static_frame.Index.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.Index.union()	Perform union with another Index, container, or Nu...
static_frame.Index.values_at_depth()	Return an NP array for the depth_level specified.
static_frame.Index.var()	Return the variance along the specified axis. Args...

11.4.8 Index: Operator Binary

static_frame.Index.__add__()	Same as a + b.
static_frame.Index.__and__()	Same as a & b.
static_frame.Index.__eq__()	Same as a == b.
static_frame.Index.__floordiv__()	Same as a // b.
static_frame.Index.__ge__()	Same as a >= b.
static_frame.Index.__gt__()	Same as a > b.
static_frame.Index.__le__()	Same as a <= b.
static_frame.Index.__lt__()	Same as a < b.
static_frame.Index.__matmul__()	Same as a @ b.
static_frame.Index.__mod__()	Same as a % b.
static_frame.Index.__mul__()	Same as a * b.
static_frame.Index.__ne__()	Same as a != b.
static_frame.Index.__or__()	Same as a | b.
static_frame.Index.__pow__()	Same as a ** b.
static_frame.Index.__radd__()	Same as a + b.
static_frame.Index.__rfloordiv__()	Same as a // b.
static_frame.Index.__rmatmul__()	Same as a @ b.
static_frame.Index.__rmul__()	Same as a * b.
static_frame.Index.__rshift__()	Same as a >> b.
static_frame.Index.__rsub__()	Same as a - b.
static_frame.Index.__rtruediv__()	Same as a / b.
static_frame.Index.__sub__()	Same as a - b.
static_frame.Index.__truediv__()	Same as a / b.
static_frame.Index.__xor__()	Same as a ^ b.

11.4.9 Index: Operator Unary

static_frame.Index.__abs__()	Same as abs(a).
static_frame.Index.__invert__()	Same as ~a.
static_frame.Index.__neg__()	Same as -a.
static_frame.Index.__pos__()	Same as +a.

11.4.10 Index: Selector

static_frame.Index[]	Extract a new index given an iloc key.
static_frame.Index.iloc[]	
static_frame.Index.loc[]	

11.5 IndexGO

11.5.1 IndexGO: Attribute

static_frame.IndexGO.STATIC	bool(x) -> bool Returns True when the argument x i...
static_frame.IndexGO.depth	int([x]) -> integer int(x, base=10) -> integer Con...
static_frame.IndexGO.drop	
static_frame.IndexGO.dtype	Return the dtype of the underlying NumPy array. Re...
static_frame.IndexGO.mloc	The memory location, represented as an integer, of...
static_frame.IndexGO.name	
static_frame.IndexGO.nbytes	Return the total bytes of the underlying NumPy arr...
static_frame.IndexGO.ndim	Return the number of dimensions. Returns: :py:clas...
static_frame.IndexGO.positions	Return the immutable positions array. This is need...
static_frame.IndexGO.shape	Return a tuple describing the shape of the underly...
static_frame.IndexGO.size	Return the size of the underlying NumPy array. Ret...

11.5.2 IndexGO: Constructor

static_frame.IndexGO.__init__()	
static_frame.IndexGO.from_labels()	Construct an Index from an iterable of labels, whe...
static_frame.IndexGO.from_pandas()	Given a Pandas index, return the appropriate Index...

11.5.3 IndexGO: Dictionary-Like

static_frame.IndexGO.__contains__()	Return True if value in the labels.
static_frame.IndexGO.__iter__()	Iterate over labels.
static_frame.IndexGO.__reversed__()	Returns a reverse iterator on the index labels.
static_frame.IndexGO.get()	Return the value found at the index key, else the...
static_frame.IndexGO.items()	Iterator of pairs of index label and value.
static_frame.IndexGO.keys()	Iterator of index labels.
static_frame.IndexGO.values	A 1D array of labels.

11.5.4 IndexGO: Display

static_frame.IndexGO.__repr__()	
static_frame.IndexGO.__str__()	Return str(self).
static_frame.IndexGO.display()	
static_frame.IndexGO.display_tall()	
static_frame.IndexGO.display_wide()	
static_frame.IndexGO.interface	A Frame documenting the interface of this class.

11.5.5 IndexGO: Exporter

static_frame.IndexGO.to_html()	Return an HTML table representation of this In-dex…
static_frame.IndexGO. to_html_datatables()	Return a complete HTML representation of this Inde…
static_frame.IndexGO.to_pandas()	Return a Pandas Index.
static_frame.IndexGO.to_series()	Return a Series with values from this Index's labe…

11.5.6 IndexGO: Iterator

static_frame.IndexGO.iter_label(axis)	
static_frame.IndexGO.iter_label(axis).apply()	Apply passed function to each object iter-ated, whe…
static_frame.IndexGO.iter_label(axis). apply_iter()	Generator that applies the passed function to each…
static_frame.IndexGO.iter_label(axis). apply_iter_items()	Generator that applies function to each ele-ment it…
static_frame.IndexGO.iter_label(axis). apply_pool()	Apply passed function to each object iter-ated, whe…

11.5.7 IndexGO: Method

static_frame.IndexGO.__len__()	
static_frame.IndexGO.add_level()	Return an IndexHierarhcy with an added root level.
static_frame.IndexGO.all()	Logical and over values along the specified axis....
static_frame.IndexGO.any()	Logical or over values along the specified axis. A...
static_frame.IndexGO.append()	append a value
static_frame.IndexGO.copy()	Return a new Index.
static_frame.IndexGO.cumprod()	Return the cumulative product over the specified a...
static_frame.IndexGO.cumsum()	Return the cumulative sum over the specified axis....
static_frame.IndexGO.extend()	Append multiple values Args: values: can be a gene...
static_frame.IndexGO.intersection()	Perform intersection with another Index, container...
static_frame.IndexGO.isin()	Return a Boolean array showing True where a label...
static_frame.IndexGO.loc_to_iloc()	Note: Boolean Series are reindexed to this index,...
static_frame.IndexGO.max()	Return the maximum along the specified axis. Args:...
static_frame.IndexGO.mean()	Return the mean along the specified axis. Args: ax...
static_frame.IndexGO.median()	Return the median along the specified axis. Args:...
static_frame.IndexGO.min()	Return the minimum along the specified axis. Args:...
static_frame.IndexGO.prod()	Return the product along the specified axis. Args:...
static_frame.IndexGO.relabel()	Return a new Index with labels replaced by the cal...
static_frame.IndexGO.rename()	Return a new Frame with an updated name attribute.
static_frame.IndexGO.roll()	Return an Index with values rotated forward and wr...
static_frame.IndexGO.sort()	Return a new Index with the labels sorted. Args: k...
static_frame.IndexGO.std()	Return the standard deviaton along the specified a...
static_frame.IndexGO.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.IndexGO.union()	Perform union with another Index, container, or Nu...
static_frame.IndexGO.values_at_depth()	Return an NP array for the depth_level specified.
static_frame.IndexGO.var()	Return the variance along the specified axis. Args...

11.5.8 IndexGO: Operator Binary

static_frame.IndexGO.__add__()	Same as a + b.
static_frame.IndexGO.__and__()	Same as a & b.
static_frame.IndexGO.__eq__()	Same as a == b.
static_frame.IndexGO.__floordiv__()	Same as a // b.
static_frame.IndexGO.__ge__()	Same as a >= b.
static_frame.IndexGO.__gt__()	Same as a > b.
static_frame.IndexGO.__le__()	Same as a <= b.
static_frame.IndexGO.__lt__()	Same as a < b.
static_frame.IndexGO.__matmul__()	Same as a @ b.
static_frame.IndexGO.__mod__()	Same as a % b.
static_frame.IndexGO.__mul__()	Same as a * b.
static_frame.IndexGO.__ne__()	Same as a != b.
static_frame.IndexGO.__or__()	Same as a | b.
static_frame.IndexGO.__pow__()	Same as a ** b.
static_frame.IndexGO.__radd__()	Same as a + b.
static_frame.IndexGO.__rfloordiv__()	Same as a // b.
static_frame.IndexGO.__rmatmul__()	Same as a @ b.
static_frame.IndexGO.__rmul__()	Same as a * b.
static_frame.IndexGO.__rshift__()	Same as a >> b.
static_frame.IndexGO.__rsub__()	Same as a - b.
static_frame.IndexGO.__rtruediv__()	Same as a / b.
static_frame.IndexGO.__sub__()	Same as a - b.
static_frame.IndexGO.__truediv__()	Same as a / b.
static_frame.IndexGO.__xor__()	Same as a ^ b.

11.5.9 IndexGO: Operator Unary

static_frame.IndexGO.__abs__()	Same as abs(a).
static_frame.IndexGO.__invert__()	Same as ~a.
static_frame.IndexGO.__neg__()	Same as -a.
static_frame.IndexGO.__pos__()	Same as +a.

11.5.10 IndexGO: Selector

static_frame.IndexGO[]	Extract a new index given an iloc key.
static_frame.IndexGO.iloc[]	
static_frame.IndexGO.loc[]	

11.6 IndexHierarchy

11.6.1 IndexHierarchy: Attribute

static_frame.IndexHierarchy.STATIC	bool(x) -> bool Returns True when the argument x i...
static_frame.IndexHierarchy.depth	
static_frame.IndexHierarchy.dtype	Return the dtype of the underlying NumPy array. Re...
static_frame.IndexHierarchy.dtypes	Return a Series of dytpes for each index depth. Re...
static_frame.IndexHierarchy.index_types	Return a Series of Index classes for each index de...
static_frame.IndexHierarchy.mloc	The memory location, represented as an integer, of...
static_frame.IndexHierarchy.name	
static_frame.IndexHierarchy.nbytes	Return the total bytes of the underlying NumPy arr...
static_frame.IndexHierarchy.ndim	Return the number of dimensions. Returns: :py:clas...
static_frame.IndexHierarchy.shape	Return a tuple describing the shape of the underly...
static_frame.IndexHierarchy.size	Return the size of the underlying NumPy array. Ret...

11.6.2 IndexHierarchy: Constructor

static_frame.IndexHierarchy.__init__()	Args: levels: IndexLevels instance, or, optionally...
static_frame.IndexHierarchy.from_labels()	Construct an IndexHierarhcy from an iterable of la...
static_frame.IndexHierarchy.from_pandas()	Given a Pandas index, return the appropriate Index...
static_frame.IndexHierarchy.from_product()	Given groups of iterables, return an IndexHierarch...
static_frame.IndexHierarchy.from_tree()	Convert into a IndexHierarchy a dictionary definin...

11.6.3 IndexHierarchy: Dictionary-Like

static_frame.IndexHierarchy.__contains__()	Determine if a leaf loc is contained in this Index...
static_frame.IndexHierarchy.__iter__()	Iterate over labels.
static_frame.IndexHierarchy.__reversed__()	Returns a reverse iterator on the index labels.
static_frame.IndexHierarchy.get()	Return the value found at the index key, else the...
static_frame.IndexHierarchy.keys()	Iterator of index labels.
static_frame.IndexHierarchy.values	An 2D array of labels. Note: type coercion might b...

11.6.4 IndexHierarchy: Display

static_frame.IndexHierarchy.__repr__()	
static_frame.IndexHierarchy.__str__()	Return str(self).
static_frame.IndexHierarchy.display()	
static_frame.IndexHierarchy.display_tall()	
static_frame.IndexHierarchy.display_wide()	
static_frame.IndexHierarchy.interface	A Frame documenting the interface of this class.

11.6.5 IndexHierarchy: Exporter

static_frame.IndexHierarchy.to_frame()	Return the index as a Frame.
static_frame.IndexHierarchy.to_html()	Return an HTML table representation of this In-dex...
static_frame.IndexHierarchy.to_html_datatables()	Return a complete HTML representation of this Inde...
static_frame.IndexHierarchy.to_pandas()	Return a Pandas MultiIndex.

11.6.6 IndexHierarchy: Iterator

static_frame.IndexHierarchy.iter_label(axis)	
static_frame.IndexHierarchy.iter_label(axis).apply()	Apply passed function to each object it-erated, whe...
static_frame.IndexHierarchy.iter_label(axis).apply_iter()	Generator that applies the passed func-tion to each...
static_frame.IndexHierarchy.iter_label(axis).apply_iter_items()	Generator that applies function to each element it...
static_frame.IndexHierarchy.iter_label(axis).apply_pool()	Apply passed function to each object it-erated, whe...

11.6.7 IndexHierarchy: Method

static_frame.IndexHierarchy.__len__()	
static_frame.IndexHierarchy.add_level()	Return an IndexHierarchy with a new root level add...
static_frame.IndexHierarchy.all()	Logical and over values along the specified axis....
static_frame.IndexHierarchy.any()	Logical or over values along the specified axis. A...
static_frame.IndexHierarchy.copy()	Return a new IndexHierarchy. This is not a deep co...
static_frame.IndexHierarchy.cumprod()	Return the cumulative product over the specified a...
static_frame.IndexHierarchy.cumsum()	Return the cumulative sum over the specified axis....
static_frame.IndexHierarchy.drop_level()	Return an IndexHierarchy with one or more leaf lev...
static_frame.IndexHierarchy.flat()	Return a flat, one-dimensional index of tuples for...
static_frame.IndexHierarchy.intersection()	Perform intersection with another Index, container...
static_frame.IndexHierarchy.isin()	Return a Boolean array showing True where one or m...
static_frame.IndexHierarchy.loc_to_iloc()	Given iterable of GetItemKeyTypes, apply to each l...
static_frame.IndexHierarchy.max()	Return the maximum along the specified axis. Args:...
static_frame.IndexHierarchy.mean()	Return the mean along the specified axis. Args: ax...
static_frame.IndexHierarchy.median()	Return the median along the specified axis. Args:...
static_frame.IndexHierarchy.min()	Return the minimum along the specified axis. Args:...
static_frame.IndexHierarchy.prod()	Return the product along the specified axis. Args:...
static_frame.IndexHierarchy.relabel()	Return a new IndexHierarchy with labels replaced b...
static_frame.IndexHierarchy.rename()	Return a new Frame with an updated name attribute.
static_frame.IndexHierarchy.roll()	Return an Index with values rotated forward and wr...
static_frame.IndexHierarchy.sort()	Return a new Index with the labels sorted. Args: k...
static_frame.IndexHierarchy.std()	Return the standard deviaton along the specified a...
static_frame.IndexHierarchy.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.IndexHierarchy.union()	Perform union with another Index, container, or Nu...
static_frame.IndexHierarchy.values_at_depth()	Return an NP array for the depth_level specified.
static_frame.IndexHierarchy.var()	Return the variance along the specified axis. Args...

11.6.8 IndexHierarchy: Operator Binary

static_frame.IndexHierarchy.__add__()	Same as a + b.
static_frame.IndexHierarchy.__and__()	Same as a & b.
static_frame.IndexHierarchy.__eq__()	Same as a == b.
static_frame.IndexHierarchy.__floordiv__()	Same as a // b.
static_frame.IndexHierarchy.__ge__()	Same as a >= b.
static_frame.IndexHierarchy.__gt__()	Same as a > b.
static_frame.IndexHierarchy.__le__()	Same as a <= b.
static_frame.IndexHierarchy.__lt__()	Same as a < b.
static_frame.IndexHierarchy.__matmul__()	Same as a @ b.
static_frame.IndexHierarchy.__mod__()	Same as a % b.
static_frame.IndexHierarchy.__mul__()	Same as a * b.
static_frame.IndexHierarchy.__ne__()	Same as a != b.
static_frame.IndexHierarchy.__or__()	Same as a | b.
static_frame.IndexHierarchy.__pow__()	Same as a ** b.
static_frame.IndexHierarchy.__radd__()	Same as a + b.
static_frame.IndexHierarchy.__rfloordiv__()	Same as a // b.
static_frame.IndexHierarchy.__rmatmul__()	Same as a @ b.
static_frame.IndexHierarchy.__rmul__()	Same as a * b.
static_frame.IndexHierarchy.__rshift__()	Same as a >> b.
static_frame.IndexHierarchy.__rsub__()	Same as a - b.
static_frame.IndexHierarchy.__rtruediv__()	Same as a / b.
static_frame.IndexHierarchy.__sub__()	Same as a - b.
static_frame.IndexHierarchy.__truediv__()	Same as a / b.
static_frame.IndexHierarchy.__xor__()	Same as a ^ b.

11.6.9 IndexHierarchy: Operator Unary

static_frame.IndexHierarchy.__abs__()	Same as abs(a).
static_frame.IndexHierarchy.__invert__()	Same as ~a.
static_frame.IndexHierarchy.__neg__()	Same as -a.
static_frame.IndexHierarchy.__pos__()	Same as +a.

11.6.10 IndexHierarchy: Selector

static_frame.IndexHierarchy[]	Extract a new index given an iloc key.
static_frame.IndexHierarchy.iloc[]	
static_frame.IndexHierarchy.loc[]	

11.7 IndexHierarchyGO

11.7.1 IndexHierarchyGO: Attribute

static_frame.IndexHierarchyGO.STATIC	bool(x) -> bool Returns True when the argument x i...
static_frame.IndexHierarchyGO.depth	
static_frame.IndexHierarchyGO.dtype	Return the dtype of the underlying NumPy array. Re...
static_frame.IndexHierarchyGO.dtypes	Return a Series of dytpes for each index depth. Re...
static_frame.IndexHierarchyGO.index_types	Return a Series of Index classes for each index de...
static_frame.IndexHierarchyGO.mloc	The memory location, represented as an integer, of...
static_frame.IndexHierarchyGO.name	
static_frame.IndexHierarchyGO.nbytes	Return the total bytes of the underlying NumPy arr...
static_frame.IndexHierarchyGO.ndim	Return the number of dimensions. Returns: :py:clas...
static_frame.IndexHierarchyGO.shape	Return a tuple describing the shape of the underly...
static_frame.IndexHierarchyGO.size	Return the size of the underlying NumPy array. Ret...

11.7.2 IndexHierarchyGO: Constructor

static_frame.IndexHierarchyGO.__init__()	Args: levels: IndexLevels instance, or, optionally...
static_frame.IndexHierarchyGO.from_labels()	Construct an IndexHierarhcy from an iterable of la...
static_frame.IndexHierarchyGO.from_pandas()	Given a Pandas index, return the appropriate Index...
static_frame.IndexHierarchyGO.from_product()	Given groups of iterables, return an IndexHierarch...
static_frame.IndexHierarchyGO.from_tree()	Convert into a IndexHierarchy a dictionary definin...

11.7.3 IndexHierarchyGO: Dictionary-Like

static_frame.IndexHierarchyGO.__contains__()	Determine if a leaf loc is contained in this Index...
static_frame.IndexHierarchyGO.__iter__()	Iterate over labels.
static_frame.IndexHierarchyGO.__reversed__()	Returns a reverse iterator on the index labels.
static_frame.IndexHierarchyGO.get()	Return the value found at the index key, else the...
static_frame.IndexHierarchyGO.keys()	Iterator of index labels.
static_frame.IndexHierarchyGO.values	An 2D array of labels. Note: type coercion might b...

11.7.4 IndexHierarchyGO: Display

static_frame.IndexHierarchyGO.__repr__()	
static_frame.IndexHierarchyGO.__str__()	Return str(self).
static_frame.IndexHierarchyGO.display()	
static_frame.IndexHierarchyGO.display_tall()	
static_frame.IndexHierarchyGO.display_wide()	
static_frame.IndexHierarchyGO.interface	A Frame documenting the interface of this class.

11.7.5 IndexHierarchyGO: Exporter

static_frame.IndexHierarchyGO.to_frame()	Return the index as a Frame.
static_frame.IndexHierarchyGO.to_html()	Return an HTML table representation of this Index...
static_frame.IndexHierarchyGO.to_html_datatables()	Return a complete HTML representation of this Inde...
static_frame.IndexHierarchyGO.to_pandas()	Return a Pandas MultiIndex.

11.7.6 IndexHierarchyGO: Iterator

static_frame.IndexHierarchyGO.iter_label(axis)	
static_frame.IndexHierarchyGO.iter_label(axis).apply()	Apply passed function to each object iterated, whe...
static_frame.IndexHierarchyGO.iter_label(axis).apply_iter()	Generator that applies the passed function to each...
static_frame.IndexHierarchyGO.iter_label(axis).apply_iter_items()	Generator that applies function to each element it...
static_frame.IndexHierarchyGO.iter_label(axis).apply_pool()	Apply passed function to each object iterated, whe...

11.7.7 IndexHierarchyGO: Method

static_frame.IndexHierarchyGO.__len__()	
static_frame.IndexHierarchyGO.add_level()	Return an IndexHierarchy with a new root level add...
static_frame.IndexHierarchyGO.all()	Logical and over values along the specified axis....
static_frame.IndexHierarchyGO.any()	Logical or over values along the specified axis. A...
static_frame.IndexHierarchyGO.append()	Append a single label to this index.
static_frame.IndexHierarchyGO.copy()	Return a new IndexHierarchy. This is not a deep co...
static_frame.IndexHierarchyGO.cumprod()	Return the cumulative product over the specified a...
static_frame.IndexHierarchyGO.cumsum()	Return the cumulative sum over the specified axis....
static_frame.IndexHierarchyGO.drop_level()	Return an IndexHierarchy with one or more leaf lev...
static_frame.IndexHierarchyGO.extend()	Extend this IndexHiearchy in-place
static_frame.IndexHierarchyGO.flat()	Return a flat, one-dimensional index of tuples for...
static_frame.IndexHierarchyGO.intersection()	Perform intersection with another Index, container...
static_frame.IndexHierarchyGO.isin()	Return a Boolean array showing True where one or m...
static_frame.IndexHierarchyGO.loc_to_iloc()	Given iterable of GetItemKeyTypes, apply to each l...
static_frame.IndexHierarchyGO.max()	Return the maximum along the specified axis. Args:...
static_frame.IndexHierarchyGO.mean()	Return the mean along the specified axis. Args: ax...
static_frame.IndexHierarchyGO.median()	Return the median along the specified axis. Args:...
static_frame.IndexHierarchyGO.min()	Return the minimum along the specified axis. Args:...
static_frame.IndexHierarchyGO.prod()	Return the product along the specified axis. Args:...
static_frame.IndexHierarchyGO.relabel()	Return a new IndexHierarchy with labels replaced b...
static_frame.IndexHierarchyGO.rename()	Return a new Frame with an updated name attribute.
static_frame.IndexHierarchyGO.roll()	Return an Index with values rotated forward and wr...
static_frame.IndexHierarchyGO.sort()	Return a new Index with the labels sorted. Args: k...
static_frame.IndexHierarchyGO.std()	Return the standard deviaton along the specified a...
static_frame.IndexHierarchyGO.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.IndexHierarchyGO.union()	Perform union with another Index, container, or Nu...
static_frame.IndexHierarchyGO.values_at_depth()	Return an NP array for the depth_level specified.
static_frame.IndexHierarchyGO.var()	Return the variance along the specified axis. Args...

11.7.8 IndexHierarchyGO: Operator Binary

static_frame.IndexHierarchyGO.__add__()	Same as a + b.
static_frame.IndexHierarchyGO.__and__()	Same as a & b.
static_frame.IndexHierarchyGO.__eq__()	Same as a == b.
static_frame.IndexHierarchyGO.__floordiv__()	Same as a // b.
static_frame.IndexHierarchyGO.__ge__()	Same as a >= b.
static_frame.IndexHierarchyGO.__gt__()	Same as a > b.
static_frame.IndexHierarchyGO.__le__()	Same as a <= b.
static_frame.IndexHierarchyGO.__lt__()	Same as a < b.
static_frame.IndexHierarchyGO.__matmul__()	Same as a @ b.
static_frame.IndexHierarchyGO.__mod__()	Same as a % b.
static_frame.IndexHierarchyGO.__mul__()	Same as a * b.
static_frame.IndexHierarchyGO.__ne__()	Same as a != b.
static_frame.IndexHierarchyGO.__or__()	Same as a | b.
static_frame.IndexHierarchyGO.__pow__()	Same as a ** b.
static_frame.IndexHierarchyGO.__radd__()	Same as a + b.
static_frame.IndexHierarchyGO.__rfloordiv__()	Same as a // b.
static_frame.IndexHierarchyGO.__rmatmul__()	Same as a @ b.
static_frame.IndexHierarchyGO.__rmul__()	Same as a * b.
static_frame.IndexHierarchyGO.__rshift__()	Same as a >> b.
static_frame.IndexHierarchyGO.__rsub__()	Same as a - b.
static_frame.IndexHierarchyGO.__rtruediv__()	Same as a / b.
static_frame.IndexHierarchyGO.__sub__()	Same as a - b.
static_frame.IndexHierarchyGO.__truediv__()	Same as a / b.
static_frame.IndexHierarchyGO.__xor__()	Same as a ^ b.

11.7.9 IndexHierarchyGO: Operator Unary

static_frame.IndexHierarchyGO.__abs__()	Same as abs(a).
static_frame.IndexHierarchyGO.__invert__()	Same as ~a.
static_frame.IndexHierarchyGO.__neg__()	Same as -a.
static_frame.IndexHierarchyGO.__pos__()	Same as +a.

11.7.10 IndexHierarchyGO: Selector

static_frame.IndexHierarchyGO[]	Extract a new index given an iloc key.
static_frame.IndexHierarchyGO.iloc[]	
static_frame.IndexHierarchyGO.loc[]	

11.8 IndexDate

11.8.1 IndexDate: Attribute

static_frame.IndexDate.STATIC	bool(x) -> bool Returns True when the argument x i…
static_frame.IndexDate.depth	int([x]) -> integer int(x, base=10) -> integer Con…
static_frame.IndexDate.drop	
static_frame.IndexDate.dtype	Return the dtype of the underlying NumPy array. Re…
static_frame.IndexDate.mloc	The memory location, represented as an integer, of…
static_frame.IndexDate.name	
static_frame.IndexDate.nbytes	Return the total bytes of the underlying NumPy arr…
static_frame.IndexDate.ndim	Return the number of dimensions. Returns: :py:clas…
static_frame.IndexDate.positions	Return the immutable positions array. This is need…
static_frame.IndexDate.shape	Return a tuple describing the shape of the underly…
static_frame.IndexDate.size	Return the size of the underlying NumPy array. Ret…

11.8.2 IndexDate: Constructor

static_frame.IndexDate.__init__()	
static_frame.IndexDate.from_date_range()	Get an IndexDate instance over a range of dates, w…
static_frame.IndexDate.from_labels()	Construct an Index from an iterable of labels, whe…
static_frame.IndexDate.from_pandas()	Given a Pandas index, return the appropriate In-dex…
static_frame.IndexDate. from_year_month_range()	Get an IndexDate instance over a range of months,…
static_frame.IndexDate.from_year_range()	Get an IndexDate instance over a range of years, w…

11.8.3 IndexDate: Dictionary-Like

static_frame.IndexDate.__contains__()	Return True if value in the labels. Will only retu…
static_frame.IndexDate.__iter__()	Iterate over labels.
static_frame.IndexDate.__reversed__()	Returns a reverse iterator on the index labels.
static_frame.IndexDate.get()	Return the value found at the index key, else the…
static_frame.IndexDate.items()	Iterator of pairs of index label and value.
static_frame.IndexDate.keys()	Iterator of index labels.
static_frame.IndexDate.values	A 1D array of labels.

11.8.4 IndexDate: Display

static_frame.IndexDate.__repr__()	
static_frame.IndexDate.__str__()	Return str(self).
static_frame.IndexDate.display()	
static_frame.IndexDate.display_tall()	
static_frame.IndexDate.display_wide()	
static_frame.IndexDate.interface	A Frame documenting the interface of this class.

11.8.5 IndexDate: Exporter

static_frame.IndexDate.to_html()	Return an HTML table representation of this Index…
static_frame.IndexDate.to_html_datatables()	Return a complete HTML representation of this Inde…
static_frame.IndexDate.to_pandas()	Return a Pandas Index.
static_frame.IndexDate.to_series()	Return a Series with values from this Index's labe…

11.8.6 IndexDate: Iterator

static_frame.IndexDate.iter_label(axis)	
static_frame.IndexDate.iter_label(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.IndexDate.iter_label(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.IndexDate.iter_label(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.IndexDate.iter_label(axis).apply_pool()	Apply passed function to each object iterated, whe…

11.8.7 IndexDate: Method

static_frame.IndexDate.__len__()	
static_frame.IndexDate.add_level()	Return an IndexHierarhcy with an added root level.
static_frame.IndexDate.all()	Logical and over values along the specified axis....
static_frame.IndexDate.any()	Logical or over values along the specified axis. A...
static_frame.IndexDate.copy()	Return a new Index.
static_frame.IndexDate.cumprod()	Return the cumulative product over the specified a...
static_frame.IndexDate.cumsum()	Return the cumulative sum over the specified axis....
static_frame.IndexDate.intersection()	Perform intersection with another Index, container...
static_frame.IndexDate.isin()	Return a Boolean array showing True where a label...
static_frame.IndexDate.loc_to_iloc()	Specialized for IndexData indices to convert strin...
static_frame.IndexDate.max()	Return the maximum along the specified axis. Args:...
static_frame.IndexDate.mean()	Return the mean along the specified axis. Args: ax...
static_frame.IndexDate.median()	Return the median along the specified axis. Args:...
static_frame.IndexDate.min()	Return the minimum along the specified axis. Args:...
static_frame.IndexDate.prod()	Return the product along the specified axis. Args:...
static_frame.IndexDate.relabel()	Return a new Index with labels replaced by the cal...
static_frame.IndexDate.rename()	Return a new Frame with an updated name attribute.
static_frame.IndexDate.roll()	Return an Index with values rotated forward and wr...
static_frame.IndexDate.sort()	Return a new Index with the labels sorted. Args: k...
static_frame.IndexDate.std()	Return the standard deviaton along the specified a...
static_frame.IndexDate.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.IndexDate.union()	Perform union with another Index, container, or Nu...
static_frame.IndexDate.values_at_depth()	Return an NP array for the depth_level specified.
static_frame.IndexDate.var()	Return the variance along the specified axis. Args...

11.8.8 IndexDate: Operator Binary

static_frame.IndexDate.__add__()	Same as a + b.
static_frame.IndexDate.__and__()	Same as a & b.
static_frame.IndexDate.__eq__()	Same as a == b.
static_frame.IndexDate.__floordiv__()	Same as a // b.
static_frame.IndexDate.__ge__()	Same as a >= b.
static_frame.IndexDate.__gt__()	Same as a > b.
static_frame.IndexDate.__le__()	Same as a <= b.
static_frame.IndexDate.__lt__()	Same as a < b.
static_frame.IndexDate.__matmul__()	Same as a @ b.
static_frame.IndexDate.__mod__()	Same as a % b.
static_frame.IndexDate.__mul__()	Same as a * b.
static_frame.IndexDate.__ne__()	Same as a != b.
static_frame.IndexDate.__or__()	Same as a | b.
static_frame.IndexDate.__pow__()	Same as a ** b.
static_frame.IndexDate.__radd__()	Same as a + b.
static_frame.IndexDate.__rfloordiv__()	Same as a // b.
static_frame.IndexDate.__rmatmul__()	Same as a @ b.
static_frame.IndexDate.__rmul__()	Same as a * b.
static_frame.IndexDate.__rshift__()	Same as a >> b.
static_frame.IndexDate.__rsub__()	Same as a - b.
static_frame.IndexDate.__rtruediv__()	Same as a / b.
static_frame.IndexDate.__sub__()	Same as a - b.
static_frame.IndexDate.__truediv__()	Same as a / b.
static_frame.IndexDate.__xor__()	Same as a ^ b.

11.8.9 IndexDate: Operator Unary

static_frame.IndexDate.__abs__()	Same as abs(a).
static_frame.IndexDate.__invert__()	Same as ~a.
static_frame.IndexDate.__neg__()	Same as -a.
static_frame.IndexDate.__pos__()	Same as +a.

11.8.10 IndexDate: Selector

static_frame.IndexDate[]	Extract a new index given an iloc key.
static_frame.IndexDate.iloc[]	
static_frame.IndexDate.loc[]	

11.9 IndexYearMonth

11.9.1 IndexYearMonth: Attribute

static_frame.IndexYearMonth.STATIC	bool(x) -> bool Returns True when the argument x i...
static_frame.IndexYearMonth.depth	int([x]) -> integer int(x, base=10) -> integer Con...
static_frame.IndexYearMonth.drop	
static_frame.IndexYearMonth.dtype	Return the dtype of the underlying NumPy array. Re...
static_frame.IndexYearMonth.mloc	The memory location, represented as an integer, of...
static_frame.IndexYearMonth.name	
static_frame.IndexYearMonth.nbytes	Return the total bytes of the underlying NumPy arr...
static_frame.IndexYearMonth.ndim	Return the number of dimensions. Returns: :py:clas...
static_frame.IndexYearMonth.positions	Return the immutable positions array. This is need...
static_frame.IndexYearMonth.shape	Return a tuple describing the shape of the underly...
static_frame.IndexYearMonth.size	Return the size of the underlying NumPy array. Ret...

11.9.2 IndexYearMonth: Constructor

static_frame.IndexYearMonth.__init__()	
static_frame.IndexYearMonth.from_date_range()	Get an IndexYearMonth instance over a range of dat...
static_frame.IndexYearMonth.from_labels()	Construct an Index from an iterable of labels, whe...
static_frame.IndexYearMonth.from_pandas()	Given a Pandas index, return the appropriate Index...
static_frame.IndexYearMonth.from_year_month_range()	Get an IndexYearMonth instance over a range of mon...
static_frame.IndexYearMonth.from_year_range()	Get an IndexYearMonth instance over a range of yea...

11.9.3 IndexYearMonth: Dictionary-Like

static_frame.IndexYearMonth.__contains__()	Return True if value in the labels. Will only retu...
static_frame.IndexYearMonth.__iter__()	Iterate over labels.
static_frame.IndexYearMonth.__reversed__()	Returns a reverse iterator on the index labels.
static_frame.IndexYearMonth.get()	Return the value found at the index key, else the...
static_frame.IndexYearMonth.items()	Iterator of pairs of index label and value.
static_frame.IndexYearMonth.keys()	Iterator of index labels.
static_frame.IndexYearMonth.values	A 1D array of labels.

11.9.4 IndexYearMonth: Display

static_frame.IndexYearMonth.__repr__()	
static_frame.IndexYearMonth.__str__()	Return str(self).
static_frame.IndexYearMonth.display()	
static_frame.IndexYearMonth.display_tall()	
static_frame.IndexYearMonth.display_wide()	
static_frame.IndexYearMonth.interface	A Frame documenting the interface of this class.

11.9.5 IndexYearMonth: Exporter

static_frame.IndexYearMonth.to_html()	Return an HTML table representation of this Index…
static_frame.IndexYearMonth.to_html_datatables()	Return a complete HTML representation of this Inde…
static_frame.IndexYearMonth.to_pandas()	Return a Pandas Index.
static_frame.IndexYearMonth.to_series()	Return a Series with values from this Index's labe…

11.9.6 IndexYearMonth: Iterator

static_frame.IndexYearMonth.iter_label(axis)	
static_frame.IndexYearMonth.iter_label(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.IndexYearMonth.iter_label(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.IndexYearMonth.iter_label(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.IndexYearMonth.iter_label(axis).apply_pool()	Apply passed function to each object iterated, whe…

11.9.7 IndexYearMonth: Method

static_frame.IndexYearMonth.__len__()	
static_frame.IndexYearMonth.add_level()	Return an IndexHierarhcy with an added root level.
static_frame.IndexYearMonth.all()	Logical and over values along the specified axis....
static_frame.IndexYearMonth.any()	Logical or over values along the specified axis. A...
static_frame.IndexYearMonth.copy()	Return a new Index.
static_frame.IndexYearMonth.cumprod()	Return the cumulative product over the specified a...
static_frame.IndexYearMonth.cumsum()	Return the cumulative sum over the specified axis....
static_frame.IndexYearMonth.intersection()	Perform intersection with another Index, container...
static_frame.IndexYearMonth.isin()	Return a Boolean array showing True where a label...
static_frame.IndexYearMonth.loc_to_iloc()	Specialized for IndexData indices to convert strin...
static_frame.IndexYearMonth.max()	Return the maximum along the specified axis. Args:...
static_frame.IndexYearMonth.mean()	Return the mean along the specified axis. Args: ax...
static_frame.IndexYearMonth.median()	Return the median along the specified axis. Args:...
static_frame.IndexYearMonth.min()	Return the minimum along the specified axis. Args:...
static_frame.IndexYearMonth.prod()	Return the product along the specified axis. Args:...
static_frame.IndexYearMonth.relabel()	Return a new Index with labels replaced by the cal...
static_frame.IndexYearMonth.rename()	Return a new Frame with an updated name attribute.
static_frame.IndexYearMonth.roll()	Return an Index with values rotated forward and wr...
static_frame.IndexYearMonth.sort()	Return a new Index with the labels sorted. Args: k...
static_frame.IndexYearMonth.std()	Return the standard deviaton along the specified a...
static_frame.IndexYearMonth.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.IndexYearMonth.union()	Perform union with another Index, container, or Nu...
static_frame.IndexYearMonth.values_at_depth()	Return an NP array for the depth_level specified.
static_frame.IndexYearMonth.var()	Return the variance along the specified axis. Args...

11.9.8 IndexYearMonth: Operator Binary

static_frame.IndexYearMonth.__add__()	Same as a + b.
static_frame.IndexYearMonth.__and__()	Same as a & b.
static_frame.IndexYearMonth.__eq__()	Same as a == b.
static_frame.IndexYearMonth.__floordiv__()	Same as a // b.
static_frame.IndexYearMonth.__ge__()	Same as a >= b.
static_frame.IndexYearMonth.__gt__()	Same as a > b.
static_frame.IndexYearMonth.__le__()	Same as a <= b.
static_frame.IndexYearMonth.__lt__()	Same as a < b.
static_frame.IndexYearMonth.__matmul__()	Same as a @ b.
static_frame.IndexYearMonth.__mod__()	Same as a % b.
static_frame.IndexYearMonth.__mul__()	Same as a * b.
static_frame.IndexYearMonth.__ne__()	Same as a != b.
static_frame.IndexYearMonth.__or__()	Same as a | b.
static_frame.IndexYearMonth.__pow__()	Same as a ** b.
static_frame.IndexYearMonth.__radd__()	Same as a + b.
static_frame.IndexYearMonth.__rfloordiv__()	Same as a // b.
static_frame.IndexYearMonth.__rmatmul__()	Same as a @ b.
static_frame.IndexYearMonth.__rmul__()	Same as a * b.
static_frame.IndexYearMonth.__rshift__()	Same as a >> b.
static_frame.IndexYearMonth.__rsub__()	Same as a - b.
static_frame.IndexYearMonth.__rtruediv__()	Same as a / b.
static_frame.IndexYearMonth.__sub__()	Same as a - b.
static_frame.IndexYearMonth.__truediv__()	Same as a / b.
static_frame.IndexYearMonth.__xor__()	Same as a ^ b.

11.9.9 IndexYearMonth: Operator Unary

static_frame.IndexYearMonth.__abs__()	Same as abs(a).
static_frame.IndexYearMonth.__invert__()	Same as ~a.
static_frame.IndexYearMonth.__neg__()	Same as -a.
static_frame.IndexYearMonth.__pos__()	Same as +a.

11.9.10 IndexYearMonth: Selector

static_frame.IndexYearMonth[]	Extract a new index given an iloc key.
static_frame.IndexYearMonth.iloc[]	
static_frame.IndexYearMonth.loc[]	

11.10 IndexYear

11.10.1 IndexYear: Attribute

static_frame.IndexYear.STATIC	bool(x) -> bool Returns True when the argument x i…
static_frame.IndexYear.depth	int([x]) -> integer int(x, base=10) -> integer Con…
static_frame.IndexYear.drop	
static_frame.IndexYear.dtype	Return the dtype of the underlying NumPy array. Re…
static_frame.IndexYear.mloc	The memory location, represented as an integer, of…
static_frame.IndexYear.name	
static_frame.IndexYear.nbytes	Return the total bytes of the underlying NumPy arr…
static_frame.IndexYear.ndim	Return the number of dimensions. Returns: :py:clas…
static_frame.IndexYear.positions	Return the immutable positions array. This is need…
static_frame.IndexYear.shape	Return a tuple describing the shape of the underly…
static_frame.IndexYear.size	Return the size of the underlying NumPy array. Ret…

11.10.2 IndexYear: Constructor

static_frame.IndexYear.__init__()	
static_frame.IndexYear.from_date_range()	Get an IndexYearMonth instance over a range of dat…
static_frame.IndexYear.from_labels()	Construct an Index from an iterable of labels, whe…
static_frame.IndexYear.from_pandas()	Given a Pandas index, return the appropriate Index…
static_frame.IndexYear.from_year_month_range()	Get an IndexYearMonth instance over a range of mon…
static_frame.IndexYear.from_year_range()	Get an IndexDate instance over a range of years, w…

11.10.3 IndexYear: Dictionary-Like

static_frame.IndexYear.__contains__()	Return True if value in the labels. Will only retu…
static_frame.IndexYear.__iter__()	Iterate over labels.
static_frame.IndexYear.__reversed__()	Returns a reverse iterator on the index labels.
static_frame.IndexYear.get()	Return the value found at the index key, else the…
static_frame.IndexYear.items()	Iterator of pairs of index label and value.
static_frame.IndexYear.keys()	Iterator of index labels.
static_frame.IndexYear.values	A 1D array of labels.

11.10.4 IndexYear: Display

static_frame.IndexYear.__repr__()	
static_frame.IndexYear.__str__()	Return str(self).
static_frame.IndexYear.display()	
static_frame.IndexYear.display_tall()	
static_frame.IndexYear.display_wide()	
static_frame.IndexYear.interface	A Frame documenting the interface of this class.

11.10.5 IndexYear: Exporter

static_frame.IndexYear.to_html()	Return an HTML table representation of this Index…
static_frame.IndexYear.to_html_datatables()	Return a complete HTML representation of this Inde…
static_frame.IndexYear.to_pandas()	Return a Pandas Index.
static_frame.IndexYear.to_series()	Return a Series with values from this Index's labe…

11.10.6 IndexYear: Iterator

static_frame.IndexYear.iter_label(axis)	
static_frame.IndexYear.iter_label(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.IndexYear.iter_label(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.IndexYear.iter_label(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.IndexYear.iter_label(axis).apply_pool()	Apply passed function to each object iterated, whe…

11.10.7 IndexYear: Method

static_frame.IndexYear.__len__()	
static_frame.IndexYear.add_level()	Return an IndexHierarhcy with an added root level.
static_frame.IndexYear.all()	Logical and over values along the specified axis....
static_frame.IndexYear.any()	Logical or over values along the specified axis. A...
static_frame.IndexYear.copy()	Return a new Index.
static_frame.IndexYear.cumprod()	Return the cumulative product over the specified a...
static_frame.IndexYear.cumsum()	Return the cumulative sum over the specified axis....
static_frame.IndexYear.intersection()	Perform intersection with another Index, container...
static_frame.IndexYear.isin()	Return a Boolean array showing True where a label...
static_frame.IndexYear.loc_to_iloc()	Specialized for IndexData indices to convert strin...
static_frame.IndexYear.max()	Return the maximum along the specified axis. Args:...
static_frame.IndexYear.mean()	Return the mean along the specified axis. Args: ax...
static_frame.IndexYear.median()	Return the median along the specified axis. Args:...
static_frame.IndexYear.min()	Return the minimum along the specified axis. Args:...
static_frame.IndexYear.prod()	Return the product along the specified axis. Args:...
static_frame.IndexYear.relabel()	Return a new Index with labels replaced by the cal...
static_frame.IndexYear.rename()	Return a new Frame with an updated name attribute.
static_frame.IndexYear.roll()	Return an Index with values rotated forward and wr...
static_frame.IndexYear.sort()	Return a new Index with the labels sorted. Args: k...
static_frame.IndexYear.std()	Return the standard deviaton along the specified a...
static_frame.IndexYear.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.IndexYear.union()	Perform union with another Index, container, or Nu...
static_frame.IndexYear.values_at_depth()	Return an NP array for the depth_level specified.
static_frame.IndexYear.var()	Return the variance along the specified axis. Args...

11.10.8 IndexYear: Operator Binary

static_frame.IndexYear.__add__()	Same as a + b.
static_frame.IndexYear.__and__()	Same as a & b.
static_frame.IndexYear.__eq__()	Same as a == b.
static_frame.IndexYear.__floordiv__()	Same as a // b.
static_frame.IndexYear.__ge__()	Same as a >= b.
static_frame.IndexYear.__gt__()	Same as a > b.
static_frame.IndexYear.__le__()	Same as a <= b.
static_frame.IndexYear.__lt__()	Same as a < b.
static_frame.IndexYear.__matmul__()	Same as a @ b.
static_frame.IndexYear.__mod__()	Same as a % b.
static_frame.IndexYear.__mul__()	Same as a * b.
static_frame.IndexYear.__ne__()	Same as a != b.
static_frame.IndexYear.__or__()	Same as a | b.
static_frame.IndexYear.__pow__()	Same as a ** b.
static_frame.IndexYear.__radd__()	Same as a + b.
static_frame.IndexYear.__rfloordiv__()	Same as a // b.
static_frame.IndexYear.__rmatmul__()	Same as a @ b.
static_frame.IndexYear.__rmul__()	Same as a * b.
static_frame.IndexYear.__rshift__()	Same as a >> b.
static_frame.IndexYear.__rsub__()	Same as a - b.
static_frame.IndexYear.__rtruediv__()	Same as a / b.
static_frame.IndexYear.__sub__()	Same as a - b.
static_frame.IndexYear.__truediv__()	Same as a / b.
static_frame.IndexYear.__xor__()	Same as a ^ b.

11.10.9 IndexYear: Operator Unary

static_frame.IndexYear.__abs__()	Same as abs(a).
static_frame.IndexYear.__invert__()	Same as ~a.
static_frame.IndexYear.__neg__()	Same as -a.
static_frame.IndexYear.__pos__()	Same as +a.

11.10.10 IndexYear: Selector

static_frame.IndexYear[]	Extract a new index given an iloc key.
static_frame.IndexYear.iloc[]	
static_frame.IndexYear.loc[]	

11.11 IndexMillisecond

11.11.1 IndexMillisecond: Attribute

static_frame.IndexMillisecond.STATIC	bool(x) -> bool Returns True when the argument x i...
static_frame.IndexMillisecond.depth	int([x]) -> integer int(x, base=10) -> integer Con...
static_frame.IndexMillisecond.drop	
static_frame.IndexMillisecond.dtype	Return the dtype of the underlying NumPy array. Re...
static_frame.IndexMillisecond.mloc	The memory location, represented as an integer, of...
static_frame.IndexMillisecond.name	
static_frame.IndexMillisecond.nbytes	Return the total bytes of the underlying NumPy arr...
static_frame.IndexMillisecond.ndim	Return the number of dimensions. Returns: :py:clas...
static_frame.IndexMillisecond.positions	Return the immutable positions array. This is need...
static_frame.IndexMillisecond.shape	Return a tuple describing the shape of the underly...
static_frame.IndexMillisecond.size	Return the size of the underlying NumPy array. Ret...

11.11.2 IndexMillisecond: Constructor

static_frame.IndexMillisecond.__init__()	
static_frame.IndexMillisecond.from_labels()	Construct an Index from an iterable of labels, whe...
static_frame.IndexMillisecond.from_pandas()	Given a Pandas index, return the appropriate Index...

11.11.3 IndexMillisecond: Dictionary-Like

static_frame.IndexMillisecond.__contains__()	Return True if value in the labels. Will only retu...
static_frame.IndexMillisecond.__iter__()	Iterate over labels.
static_frame.IndexMillisecond.__reversed__()	Returns a reverse iterator on the index labels.
static_frame.IndexMillisecond.get()	Return the value found at the index key, else the...
static_frame.IndexMillisecond.items()	Iterator of pairs of index label and value.
static_frame.IndexMillisecond.keys()	Iterator of index labels.
static_frame.IndexMillisecond.values	A 1D array of labels.

11.11.4 IndexMillisecond: Display

static_frame.IndexMillisecond.__repr__()	
static_frame.IndexMillisecond.__str__()	Return str(self).
static_frame.IndexMillisecond.display()	
static_frame.IndexMillisecond.display_tall()	
static_frame.IndexMillisecond.display_wide()	
static_frame.IndexMillisecond.interface	A Frame documenting the interface of this class.

11.11.5 IndexMillisecond: Exporter

static_frame.IndexMillisecond.to_html()	Return an HTML table representation of this Index…
static_frame.IndexMillisecond.to_html_datatables()	Return a complete HTML representation of this Inde…
static_frame.IndexMillisecond.to_pandas()	Return a Pandas Index.
static_frame.IndexMillisecond.to_series()	Return a Series with values from this Index's labe…

11.11.6 IndexMillisecond: Iterator

static_frame.IndexMillisecond.iter_label(axis)	
static_frame.IndexMillisecond.iter_label(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.IndexMillisecond.iter_label(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.IndexMillisecond.iter_label(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.IndexMillisecond.iter_label(axis).apply_pool()	Apply passed function to each object iterated, whe…

11.11.7 IndexMillisecond: Method

static_frame.IndexMillisecond.__len__()	
static_frame.IndexMillisecond.add_level()	Return an IndexHierarhcy with an added root level.
static_frame.IndexMillisecond.all()	Logical and over values along the specified axis....
static_frame.IndexMillisecond.any()	Logical or over values along the specified axis. A...
static_frame.IndexMillisecond.copy()	Return a new Index.
static_frame.IndexMillisecond.cumprod()	Return the cumulative product over the specified a...
static_frame.IndexMillisecond.cumsum()	Return the cumulative sum over the specified axis....
static_frame.IndexMillisecond.intersection()	Perform intersection with another Index, container...
static_frame.IndexMillisecond.isin()	Return a Boolean array showing True where a label...
static_frame.IndexMillisecond.loc_to_iloc()	Specialized for IndexData indices to convert strin...
static_frame.IndexMillisecond.max()	Return the maximum along the specified axis. Args:...
static_frame.IndexMillisecond.mean()	Return the mean along the specified axis. Args: ax...
static_frame.IndexMillisecond.median()	Return the median along the specified axis. Args:...
static_frame.IndexMillisecond.min()	Return the minimum along the specified axis. Args:...
static_frame.IndexMillisecond.prod()	Return the product along the specified axis. Args:...
static_frame.IndexMillisecond.relabel()	Return a new Index with labels replaced by the cal...
static_frame.IndexMillisecond.rename()	Return a new Frame with an updated name attribute.
static_frame.IndexMillisecond.roll()	Return an Index with values rotated forward and wr...
static_frame.IndexMillisecond.sort()	Return a new Index with the labels sorted. Args: k...
static_frame.IndexMillisecond.std()	Return the standard deviaton along the specified a...
static_frame.IndexMillisecond.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.IndexMillisecond.union()	Perform union with another Index, container, or Nu...
static_frame.IndexMillisecond.values_at_depth()	Return an NP array for the depth_level specified.
static_frame.IndexMillisecond.var()	Return the variance along the specified axis. Args...

11.11.8 IndexMillisecond: Operator Binary

static_frame.IndexMillisecond.__add__()	Same as a + b.
static_frame.IndexMillisecond.__and__()	Same as a & b.
static_frame.IndexMillisecond.__eq__()	Same as a == b.
static_frame.IndexMillisecond.__floordiv__()	Same as a // b.
static_frame.IndexMillisecond.__ge__()	Same as a >= b.
static_frame.IndexMillisecond.__gt__()	Same as a > b.
static_frame.IndexMillisecond.__le__()	Same as a <= b.
static_frame.IndexMillisecond.__lt__()	Same as a < b.
static_frame.IndexMillisecond.__matmul__()	Same as a @ b.
static_frame.IndexMillisecond.__mod__()	Same as a % b.
static_frame.IndexMillisecond.__mul__()	Same as a * b.
static_frame.IndexMillisecond.__ne__()	Same as a != b.
static_frame.IndexMillisecond.__or__()	Same as a | b.
static_frame.IndexMillisecond.__pow__()	Same as a ** b.
static_frame.IndexMillisecond.__radd__()	Same as a + b.
static_frame.IndexMillisecond.__rfloordiv__()	Same as a // b.
static_frame.IndexMillisecond.__rmatmul__()	Same as a @ b.
static_frame.IndexMillisecond.__rmul__()	Same as a * b.
static_frame.IndexMillisecond.__rshift__()	Same as a >> b.
static_frame.IndexMillisecond.__rsub__()	Same as a - b.
static_frame.IndexMillisecond.__rtruediv__()	Same as a / b.
static_frame.IndexMillisecond.__sub__()	Same as a - b.
static_frame.IndexMillisecond.__truediv__()	Same as a / b.
static_frame.IndexMillisecond.__xor__()	Same as a ^ b.

11.11.9 IndexMillisecond: Operator Unary

static_frame.IndexMillisecond.__abs__()	Same as abs(a).
static_frame.IndexMillisecond.__invert__()	Same as ~a.
static_frame.IndexMillisecond.__neg__()	Same as -a.
static_frame.IndexMillisecond.__pos__()	Same as +a.

11.11.10 IndexMillisecond: Selector

static_frame.IndexMillisecond[]	Extract a new index given an iloc key.
static_frame.IndexMillisecond.iloc[]	
static_frame.IndexMillisecond.loc[]	

11.12 IndexSecond

11.12.1 IndexSecond: Attribute

static_frame.IndexSecond.STATIC	bool(x) -> bool Returns True when the argument x i...
static_frame.IndexSecond.depth	int([x]) -> integer int(x, base=10) -> integer Con...
static_frame.IndexSecond.drop	
static_frame.IndexSecond.dtype	Return the dtype of the underlying NumPy array. Re...
static_frame.IndexSecond.mloc	The memory location, represented as an integer, of...
static_frame.IndexSecond.name	
static_frame.IndexSecond.nbytes	Return the total bytes of the underlying NumPy arr...
static_frame.IndexSecond.ndim	Return the number of dimensions. Returns: :py:clas...
static_frame.IndexSecond.positions	Return the immutable positions array. This is need...
static_frame.IndexSecond.shape	Return a tuple describing the shape of the underly...
static_frame.IndexSecond.size	Return the size of the underlying NumPy array. Ret...

11.12.2 IndexSecond: Constructor

static_frame.IndexSecond.__init__()	
static_frame.IndexSecond.from_labels()	Construct an Index from an iterable of labels, whe...
static_frame.IndexSecond.from_pandas()	Given a Pandas index, return the appropriate Index...

11.12.3 IndexSecond: Dictionary-Like

static_frame.IndexSecond.__contains__()	Return True if value in the labels. Will only retu...
static_frame.IndexSecond.__iter__()	Iterate over labels.
static_frame.IndexSecond.__reversed__()	Returns a reverse iterator on the index labels.
static_frame.IndexSecond.get()	Return the value found at the index key, else the...
static_frame.IndexSecond.items()	Iterator of pairs of index label and value.
static_frame.IndexSecond.keys()	Iterator of index labels.
static_frame.IndexSecond.values	A 1D array of labels.

11.12.4 IndexSecond: Display

static_frame.IndexSecond.__repr__()	
static_frame.IndexSecond.__str__()	Return str(self).
static_frame.IndexSecond.display()	
static_frame.IndexSecond.display_tall()	
static_frame.IndexSecond.display_wide()	
static_frame.IndexSecond.interface	A Frame documenting the interface of this class.

11.12.5 IndexSecond: Exporter

static_frame.IndexSecond.to_html()	Return an HTML table representation of this Index…
static_frame.IndexSecond.to_html_datatables()	Return a complete HTML representation of this Inde…
static_frame.IndexSecond.to_pandas()	Return a Pandas Index.
static_frame.IndexSecond.to_series()	Return a Series with values from this Index's labe…

11.12.6 IndexSecond: Iterator

static_frame.IndexSecond.iter_label(axis)	
static_frame.IndexSecond.iter_label(axis).apply()	Apply passed function to each object iterated, whe…
static_frame.IndexSecond.iter_label(axis).apply_iter()	Generator that applies the passed function to each…
static_frame.IndexSecond.iter_label(axis).apply_iter_items()	Generator that applies function to each element it…
static_frame.IndexSecond.iter_label(axis).apply_pool()	Apply passed function to each object iterated, whe…

11.12.7 IndexSecond: Method

static_frame.IndexSecond.__len__()	
static_frame.IndexSecond.add_level()	Return an IndexHierarhcy with an added root level.
static_frame.IndexSecond.all()	Logical and over values along the specified axis....
static_frame.IndexSecond.any()	Logical or over values along the specified axis. A...
static_frame.IndexSecond.copy()	Return a new Index.
static_frame.IndexSecond.cumprod()	Return the cumulative product over the specified a...
static_frame.IndexSecond.cumsum()	Return the cumulative sum over the specified axis....
static_frame.IndexSecond.intersection()	Perform intersection with another Index, container...
static_frame.IndexSecond.isin()	Return a Boolean array showing True where a label...
static_frame.IndexSecond.loc_to_iloc()	Specialized for IndexData indices to convert strin...
static_frame.IndexSecond.max()	Return the maximum along the specified axis. Args:...
static_frame.IndexSecond.mean()	Return the mean along the specified axis. Args: ax...
static_frame.IndexSecond.median()	Return the median along the specified axis. Args:...
static_frame.IndexSecond.min()	Return the minimum along the specified axis. Args:...
static_frame.IndexSecond.prod()	Return the product along the specified axis. Args:...
static_frame.IndexSecond.relabel()	Return a new Index with labels replaced by the cal...
static_frame.IndexSecond.rename()	Return a new Frame with an updated name attribute.
static_frame.IndexSecond.roll()	Return an Index with values rotated forward and wr...
static_frame.IndexSecond.sort()	Return a new Index with the labels sorted. Args: k...
static_frame.IndexSecond.std()	Return the standard deviaton along the specified a...
static_frame.IndexSecond.sum()	Sum values along the specified axis. Args: axis: A...
static_frame.IndexSecond.union()	Perform union with another Index, container, or Nu...
static_frame.IndexSecond.values_at_depth()	Return an NP array for the depth_level specified.
static_frame.IndexSecond.var()	Return the variance along the specified axis. Args...

11.12.8 IndexSecond: Operator Binary

static_frame.IndexSecond.__add__()	Same as a + b.
static_frame.IndexSecond.__and__()	Same as a & b.
static_frame.IndexSecond.__eq__()	Same as a == b.
static_frame.IndexSecond.__floordiv__()	Same as a // b.
static_frame.IndexSecond.__ge__()	Same as a >= b.
static_frame.IndexSecond.__gt__()	Same as a > b.
static_frame.IndexSecond.__le__()	Same as a <= b.
static_frame.IndexSecond.__lt__()	Same as a < b.
static_frame.IndexSecond.__matmul__()	Same as a @ b.
static_frame.IndexSecond.__mod__()	Same as a % b.
static_frame.IndexSecond.__mul__()	Same as a * b.
static_frame.IndexSecond.__ne__()	Same as a != b.
static_frame.IndexSecond.__or__()	Same as a | b.
static_frame.IndexSecond.__pow__()	Same as a ** b.
static_frame.IndexSecond.__radd__()	Same as a + b.
static_frame.IndexSecond.__rfloordiv__()	Same as a // b.
static_frame.IndexSecond.__rmatmul__()	Same as a @ b.
static_frame.IndexSecond.__rmul__()	Same as a * b.
static_frame.IndexSecond.__rshift__()	Same as a >> b.
static_frame.IndexSecond.__rsub__()	Same as a - b.
static_frame.IndexSecond.__rtruediv__()	Same as a / b.
static_frame.IndexSecond.__sub__()	Same as a - b.
static_frame.IndexSecond.__truediv__()	Same as a / b.
static_frame.IndexSecond.__xor__()	Same as a ^ b.

11.12.9 IndexSecond: Operator Unary

static_frame.IndexSecond.__abs__()	Same as abs(a).
static_frame.IndexSecond.__invert__()	Same as ~a.
static_frame.IndexSecond.__neg__()	Same as -a.
static_frame.IndexSecond.__pos__()	Same as +a.

11.12.10 IndexSecond: Selector

static_frame.IndexSecond[]	Extract a new index given an iloc key.
static_frame.IndexSecond.iloc[]	
static_frame.IndexSecond.loc[]	

CHAPTER

TWELVE

STRUCTURES

12.1 Primary Containers

The primary components of the StaticFrame library are the one-dimensional *static_frame.Series* and the two-dimensional *static_frame.Frame* and *static_frame.FrameGO*.

While *Frame* and *Series* are immutable, the *FrameGO* permits a type of grow-only mutation, the addition of columns.

class Series(*values: Union[Iterable[Any], numpy.ndarray, Mapping[Hashable, Any], int, float, str, bool], *, index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], Type[static_frame.core.index_auto.IndexAutoFactory]] = None, name: Hashable = None, dtype: Union[str, numpy.dtype, type, None] = None, index_constructor: Callable[[...], IndexBase] = None, own_index: bool = False*)

A one-dimensional ordered, labelled collection, immutable and of fixed size.

Parameters

- **values** – An iterable of values, or a single object, to be aligned with the supplied (or automatically generated) index. Alternatively, a dictionary of index / value pairs can be provided.
- **index** – Optional index initializer. If provided in addition to data values, lengths must be compatible.
- **own_index** – Flag the passed index as ownable by this `Series`. Primarily used by internal clients.

```
>>> import numpy as np
>>> import static_frame as sf

>>> sf.Series.from_dict(dict(Mercury=167, Neptune=-200), dtype=np.int64)
<Series>
<Index>
Mercury  167
Neptune  -200
<<U7>    <int64>

>>> sf.Series((167, -200), index=('Mercury', 'Neptune'), dtype=np.int64)
<Series>
<Index>
Mercury  167
Neptune  -200
<<U7>    <int64>
```

class Frame(*data: Union[Iterable[Iterable[Any]], numpy.ndarray] = <object object>, *, index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], Type[static_frame.core.index_auto.IndexAutoFactory]] = None, columns: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], Type[static_frame.core.index_auto.IndexAutoFactory]] = None, name: Hashable = None, index_constructor: Callable[[...], IndexBase] = None, columns_constructor: Callable[[...], IndexBase] = None, own_data: bool = False, own_index: bool = False, own_columns: bool = False*)

A two-dimensional ordered, labelled collection, immutable and of fixed size.

Parameters

- **data** – A Frame initializer, given as either a NumPy array, a single value (to be used to fill a shape defined by `index` and `columns`), or an iterable suitable to given to the NumPy array constructor.
- **index** – Optional index initializer. If provided in addition to data values, lengths must be compatible.
- **columns** – Optional column initializer. If provided in addition to data values, lengths must be compatible.
- **own_data** – Flag the data values as ownable by this `Frame`. Primarily used by internal clients.
- **own_index** – Flag the passed index as ownable by this `Frame`. Primarily used by internal clients.
- **own_columns** – Flag the passed columns as ownable by this `Frame`. Primarily used by internal clients.

```
>>> sf.Frame(((-65, 227.9), (-200, 4495.1)), columns=('temperature', 'distance'),␣
↪index=('Mars', 'Neptune'))
<Frame>
<Index> temperature distance  <<U11>
<Index>
Mars    -65.0       227.9
Neptune -200.0      4495.1
<<U7>   <float64>   <float64>

>>> sf.Frame.from_dict(dict(diameter=(12756, 142984, 120536), mass=(5.97, 1898, 568)),
↪ index=('Earth', 'Jupiter', 'Saturn'), dtypes=dict(diameter=np.int64))
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Jupiter 142984   1898.0
Saturn  120536   568.0
<<U7>   <int64>  <float64>
```

class FrameGO(*data: Union[Iterable[Iterable[Any]], numpy.ndarray] = <object object>, *, index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], Type[static_frame.core.index_auto.IndexAutoFactory]] = None, columns: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], Type[static_frame.core.index_auto.IndexAutoFactory]] = None, name: Hashable = None, index_constructor: Callable[[...], IndexBase] = None, columns_constructor: Callable[[...], IndexBase] = None, own_data: bool = False, own_index: bool = False, own_columns: bool = False*)

A two-dimensional, ordered, labelled collection, immutable with grow-only columns. Initialization arguments are the same as for *Frame*.

```
>>> f = sf.FrameGO.from_dict(dict(diameter=(12756, 142984, 120536), mass=(5.97, 1898,␣
↪568)), index=('Earth', 'Jupiter', 'Saturn'), dtypes=dict(diameter=np.int64))
>>> f['radius'] = f['diameter'] * 0.5
>>> f
<FrameGO>
<IndexGO> diameter mass      radius    <<U8>
<Index>
Earth     12756    5.97      6378.0
Jupiter   142984   1898.0    71492.0
Saturn    120536   568.0     60268.0
<<U7>     <int64>  <float64> <float64>
```

12.2 Index Mappings

Index mapping classes are used to map labels to ordinal positions on the `Series`, `Frame`, and `FrameGO`.

class Index(*labels: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]], *, loc_is_iloc: bool = False, name: Hashable = None, dtype: Union[str, numpy.dtype, type, None] = None*)

A mapping of labels to positions, immutable and of fixed size. Used by default in *Series* and as index and columns in *Frame*.

Parameters

- **labels** – An iterable of unique, hashable values, or another `Index` or `IndexHierarchy`, to be used as the labels of the index.
- **name** – A hashable object to name the Index.
- **loc_is_iloc** – Optimization when a contiguous integer index is provided as labels. Generally only set by internal clients.
- **dtype** – Optional dtype to be used for labels.

```
>>> sf.Index(('Mercury', 'Mars'), dtype=object)
<Index>
Mercury
Mars
<object>

>>> sf.Index(name[:2].upper() for name in ('Mercury', 'Mars'))
<Index>
ME
MA
<object>
```

class IndexGO(*labels: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]], *, loc_is_iloc: bool = False, name: Hashable = None, dtype: Union[str, numpy.dtype, type, None] = None*)

A mapping of labels to positions, immutable with grow-only size. Used as columns in *FrameGO*.

Parameters

- **labels** – An iterable of unique, hashable values, or another `Index` or `IndexHierarchy`, to be used as the labels of the index.
- **name** – A hashable object to name the Index.
- **loc_is_iloc** – Optimization when a contiguous integer index is provided as labels. Generally only set by internal clients.
- **dtype** – Optional dtype to be used for labels.

```
>>> a = sf.IndexGO(('Uranus', 'Neptune'))
>>> a.append('Pluto')
>>> a
<IndexGO>
Uranus
Neptune
Pluto
<<U7>
```

class IndexYear(*labels: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]], *, name: Hashable = None*)

A mapping of years (via NumPy datetime64[Y]) to positions, immutable and of fixed size.

Parameters

- **labels** – Iterable of hashable values to be used as the index labels. If strings, NumPy datetime conversions will be applied.
- **name** – A hashable object to name the Index.

class IndexYearMonth(*labels: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]], *, name: Hashable = None*)

A mapping of year months (via NumPy datetime64[M]) to positions, immutable and of fixed size.

Parameters

- **labels** – Iterable of hashable values to be used as the index labels. If strings, NumPy datetime conversions will be applied.
- **name** – A hashable object to name the Index.

class IndexDate(*labels: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]], *, name: Hashable = None*)

A mapping of dates (via NumPy datetime64[D]) to positions, immutable and of fixed size.

Parameters

- **labels** – Iterable of hashable values to be used as the index labels. If strings, NumPy datetime conversions will be applied.
- **name** – A hashable object to name the Index.

class IndexSecond(*labels: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]], *, name: Hashable = None*)

A mapping of time stamps at the resolution of seconds (via NumPy datetime64[s]) to positions, immutable and of fixed size.

Parameters

- **labels** – Iterable of hashable values to be used as the index labels. If strings, NumPy datetime conversions will be applied.
- **name** – A hashable object to name the Index.

class IndexMillisecond(*labels: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]], *, name: Hashable = None*)

A mapping of time stamps at the resolutoin of milliseconds (via NumPy datetime64[ms]) to positions, immutable and of fixed size.

Parameters

- **labels** – Iterable of hashable values to be used as the index labels. If strings, NumPy datetime conversions will be applied.
- **name** – A hashable object to name the Index.

Deviations from Pandas

`Index` and `IndexGO` require that all labels are unique. Duplicated labels will raise an error in all cases. This deviates form Pandas, where Index objects permit duplicate labels. This also makes options like the `verify_integrity` argument to `pd.Series.set_index` and `pd.DataFrame.set_index` unnecessary.

---

class IndexHierarchy(*levels: Union[static_frame.core.index_level.IndexLevel, IndexHierarchy], *, name: Hashable = None*)

A hierarchy of *static_frame.Index* objects, defined as strict tree of uniform depth across all branches.

class IndexHierarchyGO(*levels: Union[static_frame.core.index_level.IndexLevel, IndexHierarchy], *, name: Hashable = None*)

A hierarchy of *static_frame.Index* objects that permits mutation only in the addition of new hierarchies or labels.

12.3 Utility Objects

The following objects are generally only created by internal clients, and thus are not fully documented here.

class TypeBlocks(**, blocks: List[numpy.ndarray], dtypes: List[numpy.dtype], index: List[Tuple[int, int]], shape: Tuple[int, int]*)

An ordered collection of type-heterogenous, immutable NumPy arrays, providing an external array-like interface of a single, 2D array. Used by *Frame* for core, unindexed array management.

A TypeBlocks instance can have a zero size shape (where the length of one axis is zero). Internally, when axis 0 (rows) is of size 0, we store similarly sized arrays. When axis 1 (columns) is of size 0, we do not store arrays, as such arrays do not define a type (as tyupes are defined by columns).

CHAPTER

THIRTEEN

CONTAINER IMPORT & CREATION

Rather than offering a complex default initializer with arguments that are sometimes irrelevant, `Series` and `Frame` containers offer numerous specialized constructors.

Both `Series` and `Frame` have `from_items` constructors that consume key/value pairs, such as those returned by `dict.items()` and similar functions.

13.1 Series

classmethod Series.**from_items**(*pairs: Iterable[Tuple[Hashable, Any]], *, dtype: Union[str, numpy.dtype, type, None] = None, name: Hashable = None, index_constructor: Optional[Callable[[...], static_frame.core.index_base.IndexBase]] = None*) → static_frame.core.series.Series

Series construction from an iterator or generator of pairs, where the first pair value is the index and the second is the value.

Parameters

- **pairs** – Iterable of pairs of index, value.
- **dtype** – dtype or valid dtype specifier.

Returns *static_frame.Series*

```
>>> sf.Series.from_items(zip(('Mercury', 'Jupiter'), (4879, 12756)), dtype=np.int64)
<Series>
<Index>
Mercury  4879
Jupiter  12756
<<U7>    <int64>
```

classmethod Series.**from_dict**(*mapping: Dict[Hashable, Any], *, dtype: Union[str, numpy.dtype, type, None] = None, name: Hashable = None, index_constructor: Optional[Callable[[...], static_frame.core.index_base.IndexBase]] = None*) → static_frame.core.series.Series

Series construction from a dictionary, where the first pair value is the index and the second is the value.

Parameters

- **mapping** – a dictionary or similar mapping interface.
- **dtype** – dtype or valid dtype specifier.

Returns *static_frame.Series*

classmethod Series.**from_concat**(*containers: Iterable[Series], *, index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], Type[static_frame.core.index_auto.IndexAutoFactory]] = None, name: Hashable = None*)

Concatenate multiple Series into a new Series.

Parameters

- **containers** – Iterable of `Series` from which values in the new `Series` are drawn.
- **index** – If None, the resultant index will be the concatenation of all indices (assuming they are unique in combination). If `IndexAutoFactory`, the resultant index is a auto-incremented integer index. Otherwise, the value is used as a index initializer.

classmethod Series.**from_pandas**(*value*, *, *own_data: bool = False*) → static_frame.core.series.Series

Given a Pandas Series, return a Series.

Parameters

- **value** – Pandas Series.
- **own_data** – If True, the underlying NumPy data array will be made immutable and used without a copy.
- **own_index** – If True, the underlying NumPy index label array will be made immutable and used without a copy.

Returns *static_frame.Series*

13.2 Frame

classmethod Frame.**from_items**(*pairs: Iterable[Tuple[Hashable, Iterable[Any]]], *, index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]] = None, fill_value: object = nan, dtypes: Union[Iterable[Union[str, numpy.dtype, type, None]], Dict[Hashable, Union[str, numpy.dtype, type, None]], None] = None, name: Hashable = None, consolidate_blocks: bool = False*)

Frame constructor from an iterator or generator of pairs, where the first value is the column name and the second value is an iterable of a single column's values.

Parameters

- **pairs** – Iterable of pairs of column name, column values.
- **index** – Iterable of values to create an Index.
- **fill_value** – If pairs include Series, they will be reindexed with the provided index; reindexing will use this fill value.
- **dtypes** – Optionally provide an iterable of dtypes, equal in length to the length of each row, or a mapping by column name. If a dtype is given as None, NumPy's default type determination will be used.
- **name** – A hashable object to name the Frame.
- **consolidate_blocks** – Optionally consolidate adjacent same-typed columns into contiguous arrays.

Returns *static_frame.Frame*

```
>>> sf.Frame.from_items((('diameter', (12756, 142984, 120536)), ('mass', (5.97, 1898,␣
↪568))), index=('Earth', 'Jupiter', 'Saturn'), dtypes=dict(diameter=np.int64))
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Jupiter 142984   1898.0
Saturn  120536   568.0
<<U7>   <int64>  <float64>
```

classmethod Frame.**from_records**(*records: Iterable[Any], *, index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], None] = None, columns: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], None] = None, dtypes: Union[Iterable[Union[str, numpy.dtype, type, None]], Dict[Hashable, Union[str, numpy.dtype, type, None]], None] = None, name: Hashable = None, consolidate_blocks: bool = False*) → static_frame.core.frame.Frame

Frame constructor from an iterable of rows, where rows are defined as iterables, including tuples, lists, and dictionaries. If each row is a NamedTuple or dictionary, and `columns` is not provided, column names will be derived from the dictionary keys or NamedTuple fields.

Note that rows defined as `Series` is not supported; use `Frame.from_concat`; for creating a `Frame` from a single dictionary, where keys are column labels and values are columns, use `Frame.from_dict`.

Parameters

- **records** – Iterable of row values, where row values are arrays, tuples, lists, dictionaries, or namedtuples.
- **index** – Optionally provide an iterable of index labels, equal in length to the number of records.
- **columns** – Optionally provide an iterable of column labels, equal in length to the length of each row.
- **dtypes** – Optionally provide an iterable of dtypes, equal in length to the length of each row, or a mapping by column name. If a dtype is given as None, NumPy's default type determination will be used.
- **name** – A hashable object to name the Frame.
- **consolidate_blocks** – Optionally consolidate adjacent same-typed columns into contiguous arrays.

Returns *static_frame.Frame*

```
>>> index = ('Mercury', 'Venus', 'Earth', 'Mars')
>>> columns = ('diameter', 'gravity', 'temperature')
>>> records = ((4879, 3.7, 167), (12104, 8.9, 464), (12756, 9.8, 15), (6792, 3.7, -
↪65))
>>> sf.Frame.from_records(records, index=index, columns=columns,␣
↪dtypes=dict(diameter=np.int64, temperature=np.int64))
<Frame>
<Index> diameter gravity   temperature <<U11>
<Index>
```

(continues on next page)

(continued from previous page)

```
Mercury 4879     3.7       167
Venus   12104    8.9       464
Earth   12756    9.8       15
Mars    6792     3.7       -65
<<U7>   <int64>  <float64> <int64>
```

classmethod Frame.**from_records_items**(*items: Iterator[Tuple[Hashable, Iterable[Any]]], *, columns: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], None] = None, dtypes: Union[Iterable[Union[str, numpy.dtype, type, None]], Dict[Hashable, Union[str, numpy.dtype, type, None]], None] = None, name: Hashable = None, consolidate_blocks: bool = False*) → static_frame.core.frame.Frame

Frame constructor from iterable of pairs of index value, row (where row is an iterable).

Parameters

- **items** – Iterable of pairs of index label, row values, where row values are arrays, tuples, lists, dictionaries, or namedtuples.
- **columns** – Optionally provide an iterable of column labels, equal in length to the length of each row.
- **dtypes** – Optionally provide an iterable of dtypes, equal in length to the length of each row, or a mapping by column name. If a dtype is given as None, NumPy's default type determination will be used.
- **name** – A hashable object to name the Frame.
- **consolidate_blocks** – Optionally consolidate adjacent same-typed columns into contiguous arrays.

Returns *static_frame.Frame*

classmethod Frame.**from_dict**(*mapping: Dict[Hashable, Iterable[Any]], *, index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]] = None, fill_value: object = nan, dtypes: Union[Iterable[Union[str, numpy.dtype, type, None]], Dict[Hashable, Union[str, numpy.dtype, type, None]], None] = None, name: Hashable = None, consolidate_blocks: bool = False*) → static_frame.core.frame.Frame

Create a Frame from a dictionary, or any object that has an items() method.

Parameters

- **mapping** – a dictionary or similar mapping interface.
- **dtypes** – Optionally provide an iterable of dtypes, equal in length to the length of each row, or a mapping by column name. If a dtype is given as None, NumPy's default type determination will be used.
- **name** – A hashable object to name the Frame.
- **consolidate_blocks** – Optionally consolidate adjacent same-typed columns into contiguous arrays.

classmethod Frame.**from_concat**(*frames: Iterable[Union[Frame, static_frame.core.series.Series]], *, axis: int = 0, union: bool = True, index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], Type[static_frame.core.index_auto.IndexAutoFactory]] = None, columns: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], Type[static_frame.core.index_auto.IndexAutoFactory]] = None, name: Hashable = None, consolidate_blocks: bool = False*) → static_frame.core.frame.Frame

Concatenate multiple Frames into a new Frame. If index or columns are provided and appropriately sized, the resulting Frame will use those indices. If the axis along concatenation (index for axis 0, columns for axis 1) is unique after concatenation, it will be preserved.

Parameters

- **frames** – Iterable of Frames.
- **axis** – Integer specifying 0 to concatenate supplied Frames vertically (aligning on columns), 1 to concatenate horizontally (aligning on rows).
- **union** – If True, the union of the aligned indices is used; if False, the intersection is used.
- **index** – Optionally specify a new index.
- **columns** – Optionally specify new columns.
- **name** – A hashable object to name the Frame.
- **consolidate_blocks** – Optionally consolidate adjacent same-typed columns into contiguous arrays.

Returns *static_frame.Frame*

```
>>> f1 = sf.Frame.from_dict(dict(diameter=(12756, 142984), mass=(5.97, 1898)), index=(
↪'Earth', 'Jupiter'))
>>> f2 = sf.Frame.from_dict(dict(mass=(0.642, 102), moons=(2, 14)), index=('Mars',
↪'Neptune'))
>>> sf.Frame.from_concat((f1, f2))
<Frame>
<Index> diameter  mass      moons     <<U8>
<Index>
Earth   12756.0   5.97      nan
Jupiter 142984.0  1898.0    nan
Mars    nan       0.642     2.0
Neptune nan       102.0     14.0
<<U7>   <float64> <float64> <float64>
>>> sf.Frame.from_concat((f1, f2), union=False)
<Frame>
<Index> mass      <<U8>
<Index>
Earth   5.97
Jupiter 1898.0
Mars    0.642
Neptune 102.0
<<U7>   <float64>
```

classmethod Frame.**from_csv**(*fp: Union[str, TextIO], *, delimiter: str = ',', index_column: Union[int, str, None] = None, skip_header: int = 0, skip_footer: int = 0, header_is_columns: bool = True, quote_char: str = '"', encoding: Optional[str] = None, dtypes: Union[Iterable[Union[str, numpy.dtype, type, None]], Dict[Hashable, Union[str, numpy.dtype, type, None]], None] = None, name: Hashable = None, consolidate_blocks: bool = False*) → static_frame.core.frame.Frame

Create a Frame from a file path or a file-like object defining a delimited (CSV, TSV) data file.

Parameters

- **fp** – A file path or a file-like object.
- **delimiter** – The character used to seperate row elements.
- **index_column** – Optionally specify a column, by position or name, to become the index.
- **skip_header** – Number of leading lines to skip.
- **skip_footer** – Numver of trailing lines to skip.
- **header_is_columns** – If True (by default), the header, the first line after the skip_header count of lines, is used to create the column labels.
- **dtypes** – Optionally provide an iterable of dtypes, equal in length to the length of each row, or a mapping by column name. If a dtype is given as None, NumPy's default type determination will be used.
- **name** – A hashable object to name the Frame.
- **consolidate_blocks** – Optionally consolidate adjacent same-typed columns into contiguous arrays.

Returns *static_frame.Frame*

```
>>> from io import StringIO
>>> filelike = StringIO('name,mass,temperature\\nVenus,4.87,464\\nNeptune,102,-200')
>>> sf.Frame.from_csv(filelike, index_column='name', dtypes=dict(temperature=np.
↪int64))
<Frame>
<Index> mass      temperature <<U11>
<Index>
Venus   4.87      464
Neptune 102.0     -200
<<U7>   <float64> <int64>
```

classmethod Frame.**from_tsv**(*fp*, ***kwargs*) → static_frame.core.frame.Frame

Specialized version of *Frame.from_csv()* for TSV files.

Returns *static_frame.Frame*

classmethod Frame.**from_json**(*json_data: str, *, dtypes: Union[Iterable[Union[str, numpy.dtype, type, None]], Dict[Hashable, Union[str, numpy.dtype, type, None]], None] = None, name: Hashable = None, consolidate_blocks: bool = False*) → static_frame.core.frame.Frame

Frame constructor from an in-memory JSON document.

Parameters

- **json_data** – a string of JSON, encoding a table as an array of JSON objects.

- **dtypes** – Optionally provide an iterable of dtypes, equal in length to the length of each row, or a mapping by column name. If a dtype is given as None, NumPy's default type determination will be used.
- **name** – A hashable object to name the Frame.
- **consolidate_blocks** – Optionally consolidate adjacent same-typed columns into contiguous arrays.

Returns *static_frame.Frame*

classmethod Frame.**from_json_url**(*url: str, *, dtypes: Union[Iterable[Union[str, numpy.dtype, type, None]], Dict[Hashable, Union[str, numpy.dtype, type, None]], None] = None, name: Hashable = None, consolidate_blocks: bool = False*) → static_frame.core.frame.Frame

Frame constructor from a JSON documenst provided via a URL.

Parameters

- **url** – URL to the JSON resource.
- **dtypes** – Optionally provide an iterable of dtypes, equal in length to the length of each row, or a mapping by column name. If a dtype is given as None, NumPy's default type determination will be used.
- **name** – A hashable object to name the Frame.
- **consolidate_blocks** – Optionally consolidate adjacent same-typed columns into contiguous arrays.

Returns *static_frame.Frame*

classmethod Frame.**from_sql**(*query: str, connection: sqlite3.Connection*) → static_frame.core.frame.Frame

Frame constructor from an SQL query and a database connection object.

Parameters

- **query** – A query string.
- **connection** – A DBAPI2 (PEP 249) Connection object, such as those returned from SQLite (via the sqlite3 module) or PyODBC.

classmethod Frame.**from_structured_array**(*array: numpy.ndarray, *, index_column: Union[int, str, None] = None, dtypes: Union[Iterable[Union[str, numpy.dtype, type, None]], Dict[Hashable, Union[str, numpy.dtype, type, None]], None] = None, name: Hashable = None, consolidate_blocks: bool = False*) → static_frame.core.frame.Frame

Convert a NumPy structed array into a Frame.

Parameters

- **array** – Structured NumPy array.
- **index_column** – Optionally provide the name or position offset of the column to use as the index.
- **dtypes** – Optionally provide an iterable of dtypes, equal in length to the length of each row, or a mapping by column name. If a dtype is given as None, NumPy's default type determination will be used.
- **name** – A hashable object to name the Frame.

- **consolidate_blocks** – Optionally consolidate adjacent same-typed columns into contiguous arrays.

Returns *static_frame.Frame*

```
>>> a = np.array([('Venus', 4.87, 464), ('Neptune', 102, -200)], dtype=[('name',␣
↪object), ('mass', 'f4'), ('temperature', 'i4')])
>>> sf.Frame.from_structured_array(a, index_column='name')
<Frame>
<Index>  mass              temperature <<U11>
<Index>
Venus    4.869999885559082 464
Neptune  102.0             -200
<object> <float32>         <int32>
```

classmethod `Frame.`**from_pandas**(*value*, *, *own_data: bool = False*) → static_frame.core.frame.Frame

Given a Pandas DataFrame, return a Frame.

Parameters

- **value** – Pandas DataFrame.
- **own_data** – If True, the underlying NumPy data array will be made immutable and used without a copy.

Returns *static_frame.Frame*

13.3 Index

While indices are often specified with their data at container creation, in some cases explicitly creating indices in advance of the data is practical.

classmethod `Index.`**from_labels**(*labels: Iterable[Sequence[Hashable]]*) → static_frame.core.index.Index

Construct an `Index` from an iterable of labels, where each label is a hashable. Provided for a compatible interface to `IndexHierarchy`.

classmethod `IndexHierarchy.`**from_product**(**levels*, *name: Hashable = None*) → IH

Given groups of iterables, return an `IndexHierarchy` made of the product of a values in those groups, where the first group is the top-most hierarchy.

Returns *static_frame.IndexHierarchy*

classmethod `IndexHierarchy.`**from_tree**(*tree*, *, *name: Hashable = None*) → IH

Convert into a `IndexHierarchy` a dictionary defining keys to either iterables or nested dictionaries of the same.

Returns *static_frame.IndexHierarchy*

classmethod `IndexHierarchy.`**from_labels**(*labels: Iterable[Sequence[Hashable]]*, *, *name: Hashable = None*, *index_constructors: Optional[Sequence[Callable[[...], IndexBase]]] = None*) → IH

Construct an `IndexHierarhcy` from an iterable of labels, where each label is tuple defining the component labels for all hierarchies.

Parameters **labels** – an iterator or generator of tuples.

Returns *static_frame.IndexHierarchy*

CHAPTER

FOURTEEN

SIZE, SHAPE & TYPE

static_frame.Series and *static_frame.Frame* store underlying data in one or two-dimensional NumPy arrays. Attributes similar to those found on NumPy arrays are available to describe the characteristics of the container.

14.1 Series

`Series.`**`shape`**
Return a tuple describing the shape of the underlying NumPy array.

Returns `tp.Tuple[int]`

`Series.`**`ndim`**
Return the number of dimensions, which for a *Series* is always 1.

Returns `int`

`Series.`**`size`**
Return the size of the underlying NumPy array.

Returns `int`

`Series.`**`nbytes`**
Return the total bytes of the underlying NumPy array.

Returns `int`

`Series.`**`dtype`**
Return the dtype of the underlying NumPy array.

Returns `numpy.dtype`

14.1.1 Examples

```
>>> s = sf.Series((1, 2, 67, 62, 27, 14), index=('Earth', 'Mars', 'Jupiter', 'Saturn',
↪ 'Uranus', 'Neptune'), dtype=np.int64)
>>> s
<Series>
<Index>
Earth    1
Mars     2
Jupiter  67
Saturn   62
Uranus   27
Neptune  14
```

(continues on next page)

(continued from previous page)

```
<<U7>    <int64>
>>> s.shape
(6,)
>>> s.ndim
1
>>> s.size
6
>>> s.nbytes
48
>>> s.dtype
dtype('int64')
```

14.2 Frame

`Frame.`**`shape`**
Return a tuple describing the shape of the underlying NumPy array.

Returns `tp.Tuple[int]`

`Frame.`**`ndim`**
Return the number of dimensions, which for a *Frame* is always 2.

Returns `int`

`Frame.`**`size`**
Return the size of the underlying NumPy array.

Returns `int`

`Frame.`**`nbytes`**
Return the total bytes of the underlying NumPy array.

Returns `int`

`Frame.`**`dtypes`**
Return a Series of dytpes for each realizable column.

Returns *static_frame.Series*

14.2.1 Examples

```
>>> f = sf.Frame.from_items((('diameter', (12756, 142984, 120536)), ('mass', (5.97,␣
↪1898, 568))), index=('Earth', 'Jupiter', 'Saturn'), dtypes=dict(diameter=np.int64))
>>> f
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Jupiter 142984   1898.0
Saturn  120536   568.0
<<U7>   <int64>  <float64>
>>> f.shape
(3, 2)
>>> f.ndim
```

(continues on next page)

(continued from previous page)

```
2
>>> f.size
6
>>> f.nbytes
48
>>> f.dtypes
<Series>
<Index>
diameter int64
mass     float64
<<U8>    <object>
```

CHAPTER

FIFTEEN

SELECTION

Data selection permits returning views of data contained within a container.

The two-dimensional `Frame` exposes three primary means of data selection: a root `__getitem__` interface, as well as `__getitem__` interfaces on `loc` and `iloc` attributes. While the one-dimensional `Series` provides the same interface, the root and `loc` `__getitem__` are identical./

As much as possible, slices or views of underlying data will be returned from selection operations. As underlying data is immutable, there is no risk of undesirable side-effects from returning views of underlying data.

15.1 Series

Series[key]

Series.loc[key]

Series.iloc[key]

Return the values specified by `key`.

Parameters **key** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root `__getitem__` takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices.

Returns *static_frame.Series*

```
>>> s = sf.Series((1, 2, 67, 62, 27, 14), index=('Earth', 'Mars', 'Jupiter', 'Saturn',
↪ 'Uranus', 'Neptune'), dtype=np.int64)
>>> s
<Series>
<Index>
Earth    1
Mars     2
Jupiter  67
Saturn   62
Uranus   27
Neptune  14
<<U7>    <int64>

>>> s['Mars']
2
>>> s['Mars':]
<Series>
<Index>
Mars     2
Jupiter  67
```

(continues on next page)

(continued from previous page)

```
Saturn   62
Uranus   27
Neptune  14
<<U7>    <int64>
>>> s[['Mars', 'Saturn']]
<Series>
<Index>
Mars     2
Saturn   62
<<U7>    <int64>
>>> s[s > 60]
<Series>
<Index>
Jupiter  67
Saturn   62
<<U7>    <int64>

>>> s.iloc[-2:]
<Series>
<Index>
Uranus   27
Neptune  14
<<U7>    <int64>
```

15.2 Frame

Frame[key]

Frame.loc[key]

Frame.iloc[key]

Return the values specified by `key`.

Parameters **key** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root __getitem__ takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices. The root __getitem__ interface is a column selector; `loc` and `iloc` interfaces accept one or two arguments, for either row selection or row and column selection (respectively).

Returns *static_frame.Frame*

```
>>> index = ('Mercury', 'Venus', 'Earth', 'Mars')
>>> columns = ('diameter', 'gravity', 'temperature')
>>> records = ((4879, 3.7, 167), (12104, 8.9, 464), (12756, 9.8, 15), (6792, 3.7, -
↪65))
>>> f = sf.Frame.from_records(records, index=index, columns=columns,␣
↪dtypes=dict(diameter=np.int64, temperature=np.int64))
>>> f
<Frame>
<Index> diameter gravity   temperature <<U11>
<Index>
Mercury 4879     3.7       167
Venus   12104    8.9       464
Earth   12756    9.8       15
Mars    6792     3.7       -65
```

(continues on next page)

(continued from previous page)

```
<<U7>   <int64>  <float64> <int64>

>>> f['gravity']
<Series: gravity>
<Index>
Mercury           3.7
Venus             8.9
Earth             9.8
Mars              3.7
<<U7>             <float64>
>>> f['gravity':]
<Frame>
<Index> gravity   temperature <<U11>
<Index>
Mercury 3.7       167
Venus   8.9       464
Earth   9.8       15
Mars    3.7       -65
<<U7>   <float64> <int64>
>>> f[['diameter', 'temperature']]
<Frame>
<Index> diameter temperature <<U11>
<Index>
Mercury 4879     167
Venus   12104    464
Earth   12756    15
Mars    6792     -65
<<U7>   <int64>  <int64>

>>> f.loc['Earth', 'temperature']
15
>>> f.loc['Earth':, 'temperature']
<Series: temperature>
<Index>
Earth                15
Mars                 -65
<<U7>                <int64>
>>> f.loc[f['temperature'] > 100, 'diameter']
<Series: diameter>
<Index>
Mercury           4879
Venus             12104
<<U7>             <int64>
>>> f.loc[sf.ILoc[-1], ['gravity', 'temperature']]
<Series: Mars>
<Index>
gravity        3.7
temperature    -65.0
<<U11>         <float64>

>>> f.iloc[-2:, -1]
<Series: temperature>
<Index>
Earth                15
Mars                 -65
<<U7>                <int64>
```

CHAPTER

SIXTEEN

SELECTION MODIFIERS

StaticFrame permits using selection modifiers in `loc` selectors. These modifiers permit encapsulating, per axis, a different kind of selection.

`ILoc[key]`
: A wrapper for embedding `iloc` specificiations within a single axis argument of a `loc` selection.

`HLoc[key]`
: A wrapper for embedding hierarchical specificiations for *`static_frame.IndexHierarchy`* within a single axis argument of a `loc` selection.

CHAPTER

SEVENTEEN

DICTIONARY-LIKE INTERFACE

`Series` and `Frame` provide dictionary-like interfaces.

For more flexible iteration of keys and values, see Iterators, below.

17.1 Series

`Series.`**`keys`**`()` → static_frame.core.index.Index
Iterator of index labels.

`Series.`**`__iter__`**`()`
Iterator of index labels, same as *`Series.keys()`*.

`Series.`**`__contains__`**`(`*value*`)` → bool
Inclusion of value in index labels.

`Series.`**`values`**
1D NumPy array of values

`Series.`**`items`**`()` → Generator[Tuple[Any, Any], None, None]
Iterator of pairs of index label and value.

`Series.`**`__len__`**`()` → int
Length of values.

`Series.`**`get`**`(`*key: Hashable*, *default=None*`)` → Any
Return the value found at the index key, else the default if the key is not found.

17.1.1 Examples

```
>>> s = sf.Series((1, 2, 67, 62, 27, 14), index=('Earth', 'Mars', 'Jupiter', 'Saturn',
↪ 'Uranus', 'Neptune'), dtype=np.int64)
>>> s
<Series>
<Index>
Earth    1
Mars     2
Jupiter  67
Saturn   62
Uranus   27
Neptune  14
<<U7>    <int64>
>>> len(s)
```

(continues on next page)

(continued from previous page)

```
6
>>> [k for k, v in s.items() if v > 60]
['Jupiter', 'Saturn']
>>> [s.get(k, None) for k in ('Mercury', 'Neptune', 'Pluto')]
[None, 14, None]
>>> 'Pluto' in s
False
>>> s.values.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : False
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False
```

17.2 Frame

`Frame.`**`keys`**`()`
Iterator of column labels.

`Frame.`**`__iter__`**`()`
Iterator of column labels, same as *`Frame.keys()`*.

`Frame.`**`__contains__`**(*value*) → bool
Inclusion of value in column labels.

`Frame.`**`values`**
2D NumPy array of `Frame` values

`Frame.`**`items`**`()` → Generator[Tuple[Any, static_frame.core.series.Series], None, None]
Iterator of pairs of column label and corresponding column *`Series`*.

`Frame.`**`__len__`**`()` → int
Length of rows in values.

`Frame.`**`get`**(*key*, *default=None*)
Return the value found at the columns key, else the default if the key is not found. This method is implemented to complete the dictionary-like interface.

17.2.1 Examples

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 142984, 120536), temperature=(15, -
↪110, -140)), index=('Earth', 'Jupiter', 'Saturn'), dtypes=dict(temperature=np.int64,
↪ diameter=np.int64))
>>> f
<Frame>
<Index> diameter temperature <<U11>
<Index>
Earth   12756    15
Jupiter 142984   -110
Saturn  120536   -140
<<U7>   <int64>  <int64>
```

(continues on next page)

(continued from previous page)

```
>>> len(f)
3
>>> [k for k, v in f.items() if (v < 0).any()]
['temperature']
>>> f.get('diameter')
<Series: diameter>
<Index>
Earth              12756
Jupiter            142984
Saturn             120536
<<U7>              <int64>
>>> f.get('mass', np.nan)
nan
>>> 'temperature' in f
True
>>> f.values.tolist()
[[12756, 15], [142984, -110], [120536, -140]]

>>> f.values.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : False
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False
```

Deviations from Pandas

For consistency, the iterator returned by `Series.keys()` and `Frame.keys()` is the same as the iterator returned by iterating the object itself. This deviates from Pandas, where iterating a Series iterates `pd.Series.values` while iterating a DataFrame iterates `pd.DataFrame.keys()`.

CHAPTER

EIGHTEEN

ASSIGNMENT / DROPPING / MASKING

`Series` and `Frame` provide interface attributes for exposing assignment-like operations, dropping data, and producing masks. Each interface attribute exposes a root `__getitem__` interface, as well as `__getitem__` interfaces on `loc` and `iloc` attributes, exposing the full range selection approaches.

18.1 Assignment

The assign-to-copy interfaces permit expressive assignment to new containers with the same flexibility as Pandas and NumPy. As all underlying data is immutable, the caller will not be mutated. With `Frame` objects, the minimum amount of data will be copied to the new `Frame`, depending on the type of assignment and the organization of the underlying `TypeBlocks`.

18.1.1 Series

Series.assign[key](value)

Series.assign.loc[key](value)

Series.assign.iloc[key](value)

Replace the values specified by the `key` with `value`.

Parameters

- **key** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root `__getitem__` takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices.
- **value** – The value to be assigned. Can be a single value, an iterable of values, or a `Series`.

Returns *static_frame.Series*

```
>>> s = sf.Series.from_items((('Venus', 108.2), ('Earth', 149.6), ('Saturn', 1433.5)))
>>> s
<Series>
<Index>
Venus    108.2
Earth    149.6
Saturn   1433.5
<<U6>    <float64>
>>> s.assign['Earth'](150)
<Series>
<Index>
Venus    108.2
```

(continues on next page)

(continued from previous page)

```
Earth    150.0
Saturn   1433.5
<<U6>    <float64>
>>> s.assign['Earth':](0)
<Series>
<Index>
Venus    108.2
Earth    0.0
Saturn   0.0
<<U6>    <float64>
>>> s.assign.loc[s < 150](0)
<Series>
<Index>
Venus    0.0
Earth    0.0
Saturn   1433.5
<<U6>    <float64>
>>> s.assign.iloc[-1](0)
<Series>
<Index>
Venus    108.2
Earth    149.6
Saturn   0.0
<<U6>    <float64>
```

18.1.2 Frame

Frame.assign[key](value)

Frame.assign.loc[key](value)

Frame.assign.iloc[key](value)

Replace the values specified by the `key` with `value`.

Parameters

- **key** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root `__getitem__` takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices. The root `__getitem__` interface is a column selector; `loc` and `iloc` interfaces accept one or two arguments, for either row selection or row and column selection (respectively).
- **value** – The value to be assigned. Can be a single value, an iterable of values, a `Series`, or a `Frame`.

Returns *static_frame.Frame*

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'), dtypes=dict(diameter=np.int64))
>>> f
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Mars    6792     0.642
Jupiter 142984   1898.0
<<U7>   <int64>  <float64>
```

(continues on next page)

(continued from previous page)

```
>>> f.assign['mass'](f['mass'] * .001)
<Frame>
<Index> diameter mass                <<U8>
<Index>
Earth   12756    0.00597
Mars    6792     0.000642
Jupiter 142984   1.8980000000000001
<<U7>   <int64>  <float64>
>>> f.assign.loc['Mars', 'mass'](0)
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Mars    6792     0.0
Jupiter 142984   1898.0
<<U7>   <int64>  <float64>
>>> f.assign.loc['Mars':, 'diameter'](0)
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Mars    0        0.642
Jupiter 0        1898.0
<<U7>   <int64>  <float64>
>>> f.assign.loc[f['diameter'] > 10000, 'mass'](0)
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    0.0
Mars    6792     0.642
Jupiter 142984   0.0
<<U7>   <int64>  <float64>
```

18.2 Dropping Data

While data from a `Series` or `Frame` can be excluded through common selection interfaces, in some cases it is more efficient and readable to specify what to drop rather than what to keep. The drop interface return new containers, efficiently removing the values specified by the key. For `Frame`, removal of rows and columns can happen simultaneously.

18.2.1 Series

`Series.drop[key]`

`Series.drop.loc[key]`

`Series.drop.iloc[key]`

Remove the values specified by the `key`.

Parameters **`key`** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root `__getitem__` takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices.

Returns *static_frame.Series*

```
>>> s = sf.Series((0, 0, 1, 2), index=('Mercury', 'Venus', 'Earth', 'Mars'), dtype=np.
↪int64)
>>> s
<Series>
<Index>
Mercury  0
Venus    0
Earth    1
Mars     2
<<U7>    <int64>
>>> s.drop[s < 1]
<Series>
<Index>
Earth    1
Mars     2
<<U7>    <int64>
>>> s.drop[['Mercury', 'Mars']]
<Series>
<Index>
Venus    0
Earth    1
<<U7>    <int64>
>>> s.drop.iloc[-2:]
<Series>
<Index>
Mercury  0
Venus    0
<<U7>    <int64>
```

18.2.2 Frame

Frame.drop[key]

Frame.drop.loc[key]

Frame.drop.iloc[key]

Remove the values specified by the `key`.

Parameters **key** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root __getitem__ takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices. The root __getitem__ interface is a column selector; `loc` and `iloc` interfaces accept one or two arguments, for either row selection or row and column selection (respectively).

Returns *static_frame.Frame*

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 142984, 120536), temperature=(15, -
↪110, -140)), index=('Earth', 'Jupiter', 'Saturn'), dtypes=dict(diameter=np.int64,␣
↪temperature=np.int64))
>>> f
<Frame>
<Index> diameter temperature <<U11>
<Index>
Earth   12756    15
Jupiter 142984   -110
Saturn  120536   -140
<<U7>   <int64>  <int64>
```

(continues on next page)

(continued from previous page)

```
>>> f.drop['diameter']
<Frame>
<Index> temperature <<U11>
<Index>
Earth   15
Jupiter -110
Saturn  -140
<<U7>   <int64>
>>> f.drop.loc[f['temperature'] < 0]
<Frame>
<Index> diameter temperature <<U11>
<Index>
Earth   12756    15
<<U7>   <int64>  <int64>
>>> f.drop.iloc[-1, -1]
<Frame>
<Index> diameter <<U11>
<Index>
Earth   12756
Jupiter 142984
<<U7>   <int64>
```

18.3 Masking Data

While Boolean `Series` and `Frame` can be created directly or with comparison operators (or functions like `isin()`), in some cases it is desirable to directly specify a mask through the common selection idioms.

18.3.1 Series

`Series.mask[key]`

`Series.mask.loc[key]`

`Series.mask.iloc[key]`

Mask (set to `True`) the values specified by the key and return a Boolean `Series`.

Parameters **`key`** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root `__getitem__` takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices.

Returns *static_frame.Series*

18.3.2 Frame

`Frame.mask[key]`

`Frame.mask.loc[key]`

`Frame.mask.iloc[key]`

Mask (set to `True`) the values specified by the key and return a Boolean `Frame`.

Parameters **`key`** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root `__getitem__` takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices.

The root __getitem__ interface is a column selector; `loc` and `iloc` interfaces accept one or two arguments, for either row selection or row and column selection (respectively).

Returns *static_frame.Frame*

18.4 Creating a Masked Array

NumPy masked arrays permit blocking out problematic data (i.e., NaNs) while maintaining compatibility with nearly all NumPy operations.

https://docs.scipy.org/doc/numpy/reference/maskedarray.generic.html

18.4.1 Series

Series.masked_array[key]

Series.masked_array.loc[key]

Series.masked_array.iloc[key]

Mask (set to `True`) the values specified by the key and return a NumPy `MaskedArray`.

Parameters **key** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root __getitem__ takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices.

Returns `np.ma.MaskedArray`

18.4.2 Frame

Frame.masked_array[key]

Frame.masked_array.loc[key]

Frame.masked_array.iloc[key]

Mask (set to `True`) the values specified by the key and return a NumPy `MaskedArray`.

Parameters **key** – A selector, either a label, a list of labels, a slice of labels, or a Boolean array. The root __getitem__ takes loc labels, `loc` takes loc labels, and `iloc` takes integer indices. The root __getitem__ interface is a column selector; `loc` and `iloc` interfaces accept one or two arguments, for either row selection or row and column selection (respectively).

Returns `np.ma.MaskedArray`

CHAPTER

NINETEEN

INDEX MANIPULATION

19.1 Series

Series.**reindex**(*index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]], fill_value=nan, own_index: bool = False*) → static_frame.core.series.Series

Return a new Series with labels defined by the provided index. The size and ordering of the data is determined by the newly provided index, where data will continue to be aligned under labels found in both the new and the old index. Labels found only in the new index will be filled with fill_value.

Parameters

- **index** – An iterable of unique, hashable values, or another Index or IndexHierarchy, to be used as the labels of the index.
- **columns** – An iterable of unique, hashable values, or another Index or IndexHierarchy, to be used as the labels of the index.
- **fill_value** – A value to be used to fill space created by a new index that has values not found in the previous index.
- **own_index** – Flag the passed index as ownable by this Series. Primarily used by internal clients.

```
>>> s = sf.Series((0, 62, 13), index=('Venus', 'Saturn', 'Neptune'))

>>> s.reindex(('Venus', 'Earth', 'Mars', 'Neptune'))
<Series>
<Index>
Venus    0.0
Earth    nan
Mars     nan
Neptune  13.0
<<U7>    <float64>
```

Series.**relabel**(*index: Union[Callable[[...], Any], Mapping[Hashable, Any], Series, Type[static_frame.core.index_auto.IndexAutoFactory], IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None]]*) → static_frame.core.series.Series

Return a new Series with transformed labels on the index. The size and ordering of the data is never chagned in a relabeling operation. The resulting index must be unique.

Parameters **index** – One of the following types, used to create a new Index with the same size as the previous index. (a) A mapping (as a dictionary or Series), used to lookup and transform the labels in the previous index. Previous labels not found in the mapping will be reused.

(b) A function, returning a hashable, that is applied to each label in the previous index. (c) The `IndexAutoFactory` type, to apply an auto-incremented integer index. (d) An index initializer, i.e., either an iterable of hashables or an `Index` instance.

```
>>> s = sf.Series((0, 62, 13), index=('Venus', 'Saturn', 'Neptune'), dtype=np.int64)

>>> s.relabel({'Venus': 'Mercury'})
<Series>
<Index>
Mercury  0
Saturn   62
Neptune  13
<object> <int64>
>>> s.relabel(lambda x: x[:2].upper())
<Series>
<Index>
VE       0
SA       62
NE       13
<object> <int64>
```

`Series.`**`relabel_flat`**`()` → static_frame.core.series.Series

Return a new `Series`, where an `IndexHierarchy` (if defined) is replaced with a flat, one-dimension index of tuples.

`Series.`**`relabel_add_level`**`(`*level: Hashable*`)` → static_frame.core.series.Series

Return a new `Series`, adding a new root level to an existing `IndexHierarchy`, or creating an `IndexHierarchy` if one is not yet defined.

Parameters **`level`** – A hashable value to be used as a new root level, extending or creating an `IndexHierarchy`

`Series.`**`relabel_drop_level`**`(`*count: int = 1*`)` → static_frame.core.series.Series

Return a new `Series`, dropping one or more levels from a either the root or the leaves of an `IndexHierarchy`. The resulting index must be unique.

Parameters **`count`** – A positive integer drops that many outer-most (root) levels; a negative integer drops that many inner-most (leaf)levels.

19.2 Frame

`Frame.`**`reindex`**`(`*index: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], None] = None, columns: Union[IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], None] = None, fill_value=nan, own_index: bool = False, own_columns: bool = False*`)` → static_frame.core.frame.Frame

Return a new `Frame` with labels defined by the provided index. The size and ordering of the data is determined by the newly provided index, where data will continue to be aligned under labels found in both the new and the old index. Labels found only in the new index will be filled with `fill_value`.

Parameters

- **`index`** – An iterable of unique, hashable values, or another `Index` or `IndexHierarchy`, to be used as the labels of the index.
- **`columns`** – An iterable of unique, hashable values, or another `Index` or `IndexHierarchy`, to be used as the labels of the index.

- **fill_value** – A value to be used to fill space created by a new index that has values not found in the previous index.
- **own_index** – Flag the passed index as ownable by this Frame. Primarily used by internal clients.
- **own_columns** – Flag the passed columns as ownable by this Frame. Primarily used by internal clients.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'), dtypes=dict(diameter=np.int64))
>>> f
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Mars    6792     0.642
Jupiter 142984   1898.0
<<U7>   <int64>  <float64>

>>> f.reindex(index=('Jupiter', 'Mars', 'Mercury'), columns=('density', 'mass'))
<Frame>
<Index> density   mass      <<U7>
<Index>
Jupiter nan       1898.0
Mars    nan       0.642
Mercury nan       nan
<<U7>   <float64> <float64>
```

Frame.**relabel**(*index: Union[Callable[[...], Any], Mapping[Hashable, Any], Series, Type[static_frame.core.index_auto.IndexAutoFactory], IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], None] = None, columns: Union[Callable[[...], Any], Mapping[Hashable, Any], Series, Type[static_frame.core.index_auto.IndexAutoFactory], IndexBase, Iterable[Hashable], Iterable[Sequence[Hashable]], Generator[Hashable, None, None], None] = None*) → static_frame.core.frame.Frame

Return a new Frame with transformed labels on the index. The size and ordering of the data is never chagned in a relabeling operation. The resulting index must be unique.

Parameters

- **index** – One of the following types, used to create a new Index with the same size as the previous index. (a) A mapping (as a dictionary or Series), used to lookup and transform the labels in the previous index. Previous labels not found in the mapping will be reused. (b) A function, returning a hashable, that is applied to each label in the previous index. (c) The IndexAutoFactory type, to apply an auto-incremented integer index. (d) An index initializer, i.e., either an iterable of hashables or an Index instance.
- **columns** – One of the following types, used to create a new Index with the same size as the previous index. (a) A mapping (as a dictionary or Series), used to lookup and transform the labels in the previous index. Previous labels not found in the mapping will be reused. (b) A function, returning a hashable, that is applied to each label in the previous index. (c) The IndexAutoFactory type, to apply an auto-incremented integer index. (d) An index initializer, i.e., either an iterable of hashables or an Index instance.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'), dtypes=dict(diameter=np.int64))
```

(continues on next page)

(continued from previous page)

```
>>> f.relabel(index=lambda x: x[:2].upper(), columns={'mass': 'mass(1e24kg)'})
<Frame>
<Index>  diameter mass(1e24kg) <object>
<Index>
EA       12756    5.97
MA       6792     0.642
JU       142984   1898.0
<object> <int64>  <float64>
```

Frame.**relabel_flat**(*index: bool = False*, *columns: bool = False*) → static_frame.core.frame.Frame

Return a new Frame, where an IndexHierarchy (if defined) is replaced with a flat, one-dimension index of tuples.

Parameters

- **index** – Boolean to flag flatening on the index.
- **columns** – Boolean to flag flatening on the columns.

Frame.**relabel_add_level**(*index: Hashable = None*, *columns: Hashable = None*) → static_frame.core.frame.Frame

Return a new Frame, adding a new root level to an existing IndexHierarchy, or creating an IndexHierarchy if one is not yet defined.

Parameters

- **index** – A hashable value to be used as a new root level, extending or creating an IndexHierarchy
- **columns** – A hashable value to be used as a new root level, extending or creating an IndexHierarchy

Frame.**relabel_drop_level**(*index: int = 0*, *columns: int = 0*) → static_frame.core.frame.Frame

Return a new Frame, dropping one or more levels from a either the root or the leaves of an IndexHierarchy. The resulting index must be unique.

Parameters

- **index** – A positive integer drops that many outer-most (root) levels; a negative integer drops that many inner-most (leaf)levels. Default is zero.
- **columns** – A positive integer drops that many outer-most (root) levels; a negative integer drops that many inner-most (leaf)levels. Default is zero.

Frame.**set_index**(*column: Union[int, numpy.integer, slice, List[Any], None, Index, Series, numpy.ndarray]*, ***, *drop: bool = False*, *index_constructor=<class 'static_frame.core.index.Index'>*) → static_frame.core.frame.Frame

Return a new frame produced by setting the given column as the index, optionally removing that column from the new Frame.

Frame.**set_index_hierarchy**(*columns: Union[int, numpy.integer, slice, List[Any], None, Index, Series, numpy.ndarray]*, ***, *drop: bool = False*, *index_constructors: Optional[Sequence[Callable[[...], IndexBase]]] = None*) → static_frame.core.frame.Frame

Given an iterable of column labels, return a new Frame with those columns as an IndexHierarchy on the index.

Parameters

- **columns** – Iterable of column labels.

- **drop** – Boolean to determine if selected columns should be removed from the data.
- **index_constructors** – Optionally provide a sequence of `Index` constructors, of length equal to depth, to be used in converting columns Index components in the `IndexHierarchy`.

Returns *Frame*

Deviations from Pandas

The functionality of the Pandas `pd.DataFrame.rename()` and `pd.Series.rename()` is available with `Frame.relabel()` and `Series.relabel()`, respectively. The functionality of the Pandas `pd.DataFrame.reset_index()` and `pd.Series.reset_index()` is available by providing the `IndexAutoFactory` type to `Frame.relabel()` and `Series.relabel()`, respectively.

19.3 Index

`Index.`**`relabel`**(*mapper: Union[Callable[[...], Any], Mapping[Hashable, Any], Series]*) → static_frame.core.index.Index

Return a new Index with labels replaced by the callable or mapping; order will be retained. If a mapping is used, the mapping need not map all origin keys.

```
>>> index = sf.Index(('Venus', 'Saturn', 'Neptune'))
>>> index.relabel({'Venus': 'Mars'})
<Index>
Mars
Saturn
Neptune
<object>

>>> index = sf.Index(('Venus', 'Saturn', 'Neptune'))
>>> index.relabel({'Neptune': 'Uranus'})
<Index>
Venus
Saturn
Uranus
<object>

>>> index.relabel(lambda x: x[:2].upper())
<Index>
VE
SA
NE
<object>
```

CHAPTER

TWENTY

ITERATORS

Both `Series` and `Frame` offer a variety of iterators (all generators) for flexible transversal of axis and values. In addition, all iterators have a family of apply methods for applying functions to the values iterated. In all cases, alternate "items" versions of iterators are provided; these methods return pairs of (index, value).

20.1 Element Iterators

20.1.1 Series

`Series.`**`iter_element`**`()`

`Series.`**`iter_element`**`().apply(`*func*, *dtype*`)`

`Series.`**`iter_element`**`().apply_pool(`*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*`)`

`Series.`**`iter_element`**`().apply_iter(`*func*`)`

`Series.`**`iter_element`**`().apply_iter_items(`*func*`)`

Iterate over the values of the Series, or expose *`static_frame.IterNodeDelegate`* for function application.

```
>>> s = sf.Series((1, 2, 67, 62, 27, 14), index=('Earth', 'Mars', 'Jupiter', 'Saturn',
↪ 'Uranus', 'Neptune'))
>>> [x for x in s.iter_element()]
[1, 2, 67, 62, 27, 14]

>>> s.iter_element().apply(lambda v: v > 20)
<Series>
<Index>
Earth    False
Mars     False
Jupiter  True
Saturn   True
Uranus   True
Neptune  False
<<U7>    <bool>
>>> [x for x in s.iter_element().apply_iter(lambda v: v > 20)]
[False, False, True, True, True, False]
```

`Series.`**`iter_element_items`**`()`

`Series.`**`iter_element_items`**`().apply(`*func*`)`

`Series.`**`iter_element_items`**`().apply_pool(`*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*`)`

Series.**iter_element_items**().*apply_iter(func)*

Series.**iter_element_items**().*apply_iter_items(func)*

Iterate over pairs of index and values of the Series, or expose *static_frame.IterNodeDelegate* for function application.

```
>>> s = sf.Series((1, 2, 67, 62, 27, 14), index=('Earth', 'Mars', 'Jupiter', 'Saturn',
↪ 'Uranus', 'Neptune'))

>>> [x for x in s.iter_element_items()]
[('Earth', 1), ('Mars', 2), ('Jupiter', 67), ('Saturn', 62), ('Uranus', 27), ('Neptune
↪', 14)]

>>> s.iter_element_items().apply(lambda k, v: v if 'u' in k else None)
<Series>
<Index>
Earth    None
Mars     None
Jupiter  67
Saturn   62
Uranus   27
Neptune  14
<<U7>    <object>

>>> [x for x in s.iter_element_items().apply_iter_items(lambda k, v: k.upper() if v >␣
↪20 else None)]
[('Earth', None), ('Mars', None), ('Jupiter', 'JUPITER'), ('Saturn', 'SATURN'), (
↪'Uranus', 'URANUS'), ('Neptune', None)]
```

Deviations from Pandas

The functionality of Pandas `pd.Series.map()` and `pd.Series.apply()` can both be obtained with `Series.iter_element().apply()`. When given a mapping, `Series.iter_element().apply()` will pass original values unchanged if they are not found in the mapping. This deviates from `pd.Series.map()`, which fills unmapped values with NaN.

20.1.2 Frame

Frame.**iter_element**()

Frame.**iter_element**().*apply(func)*

Frame.**iter_element**().*apply_pool(func, dtype, max_workers, chunksize, use_threads)*

Frame.**iter_element**().*apply_iter(func)*

Frame.**iter_element**().*apply_iter_items(func)*

Iterate over the values of the Frame, or expose *static_frame.IterNodeDelegate* for function application.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'), dtypes=dict(diameter=np.int64))

>>> [x for x in f.iter_element()]
[12756, 5.97, 6792, 0.642, 142984, 1898.0]
```

(continues on next page)

(continued from previous page)

```
>>> f.iter_element().apply(lambda x: x ** 2)
<Frame>
<Index> diameter    mass                <<U8>
<Index>
Earth   162715536   35.640899999999995
Mars    46131264    0.41216400000000003
Jupiter 20444424256 3602404.0
<<U7>   <object>    <object>
```

`Frame.`**`iter_element_items`**`()`

`Frame.`**`iter_element_items`**`().apply(`*func*`)`

`Frame.`**`iter_element_items`**`().apply_pool(`*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*`)`

`Frame.`**`iter_element_items`**`().apply_iter(`*func*`)`

`Frame.`**`iter_element_items`**`().apply_iter_items(`*func*`)`

Iterate over pairs of index / column coordinates and values of the Frame, or expose *static_frame.IterNodeDelegate* for function application.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'))

>>> [x for x in f.iter_element_items()]
[(('Earth', 'diameter'), 12756), (('Earth', 'mass'), 5.97), (('Mars', 'diameter'),␣
↪6792), (('Mars', 'mass'), 0.642), (('Jupiter', 'diameter'), 142984), (('Jupiter',
↪'mass'), 1898.0)]

>>> f.iter_element_items().apply(lambda k, v: v ** 2 if k[0] == 'Mars' else None)
<Frame>
<Index> diameter mass                <<U8>
<Index>
Earth   None     None
Mars    46131264 0.41216400000000003
Jupiter None     None
<<U7>   <object> <object>
```

Deviations from Pandas

The functionality of Pandas `pd.DataFrame.applymap()` can be obtained with `Frame.iter_element().apply()`, though the latter accepts both callables and mapping objects.

20.2 Axis Iterators

Axis iterators are available on `Frame` to support iterating on rows or columns as NumPy arrays, named tuples, or `Series`. Alternative items functions are also available to pair values with the appropriate axis label (either columns or index).

`Frame.`**`iter_array`**`(`*axis*`)`

`Frame.`**`iter_array`**`(`*axis*`).apply(`*func*`)`

Frame.**iter_array**(*axis*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

Frame.**iter_array**(*axis*).*apply_iter*(*func*)

Frame.**iter_array**(*axis*).*apply_iter_items*(*func*)

Iterate over NumPy arrays of Frame axis, where axis 0 iterates column data and axis 1 iterates row data. The returned *static_frame.IterNodeDelegate* exposes interfaces for function application.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'), dtypes=dict(diameter=np.int64))
>>> f
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Mars    6792     0.642
Jupiter 142984   1898.0
<<U7>   <int64>  <float64>

>>> [x.tolist() for x in f.iter_array(axis=0)]
[[12756, 6792, 142984], [5.97, 0.642, 1898.0]]

>>> [x.tolist() for x in f.iter_array(axis=1)]
[[12756.0, 5.97], [6792.0, 0.642], [142984.0, 1898.0]]

>>> f.iter_array(axis=0).apply(np.sum)
<Series>
<Index>
diameter 162532.0
mass     1904.612
<<U8>    <float64>
```

Frame.**iter_array_items**(*axis*)

Frame.**iter_array_items**(*axis*).*apply*(*func*)

Frame.**iter_array_items**(*axis*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

Frame.**iter_array_items**(*axis*).*apply_iter*(*func*)

Frame.**iter_array_items**(*axis*).*apply_iter_items*(*func*)

Iterate over pairs of label, NumPy array, per Frame axis, where axis 0 iterates column data and axis 1 iterates row data. The returned *static_frame.IterNodeDelegate* exposes interfaces for function application.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'))

>>> [x for x in f.iter_array_items(axis=0)]
[('diameter', array([ 12756,   6792, 142984])), ('mass', array([5.970e+00, 6.420e-01,␣
↪1.898e+03]))]

>>> [x for x in f.iter_array_items(axis=1)]
[('Earth', array([1.2756e+04, 5.9700e+00])), ('Mars', array([6.792e+03, 6.420e-01])),␣
↪('Jupiter', array([142984.,   1898.]))]

>>> f.iter_array_items(axis=1).apply(lambda k, v: v.sum() if k == 'Earth' else 0)
<Series>
<Index>
Earth    12761.97
```

(continues on next page)

(continued from previous page)

```
Mars      0.0
Jupiter   0.0
<<U7>     <float64>
```

`Frame.`**`iter_tuple`**(*axis*)

`Frame.`**`iter_tuple`**(*axis*).*apply*(*func*)

`Frame.`**`iter_tuple`**(*axis*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

`Frame.`**`iter_tuple`**(*axis*).*apply_iter*(*func*)

`Frame.`**`iter_tuple`**(*axis*).*apply_iter_items*(*func*)

Iterate over NamedTuples of Frame axis, where axis 0 iterates column data and axis 1 iterates row data. The returned *static_frame.IterNodeDelegate* exposes interfaces for function application.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'))

>>> [x for x in f.iter_tuple(axis=0)]
[Axis(Earth=12756, Mars=6792, Jupiter=142984), Axis(Earth=5.97, Mars=0.642,␣
↪Jupiter=1898.0)]

>>> [x for x in f.iter_tuple(axis=1)]
[Axis(diameter=12756.0, mass=5.97), Axis(diameter=6792.0, mass=0.642),␣
↪Axis(diameter=142984.0, mass=1898.0)]

>>> f.iter_tuple(1).apply(lambda nt: nt.mass / (4 / 3 * np.pi * (nt.diameter * 0.5)␣
↪** 3))
<Series>
<Index>
Earth     5.49328558e-12
Mars      3.91330208e-12
Jupiter   1.24003876e-12
<<U7>     <float64>
```

`Frame.`**`iter_tuple_items`**(*axis*)

`Frame.`**`iter_tuple_items`**(*axis*).*apply*(*func*)

`Frame.`**`iter_tuple_items`**(*axis*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

`Frame.`**`iter_tuple_items`**(*axis*).*apply_iter*(*func*)

`Frame.`**`iter_tuple_items`**(*axis*).*apply_iter_items*(*func*)

Iterate over pairs of label, NamedTuple, per Frame axis, where axis 0 iterates column data and axis 1 iterates row data. The returned *static_frame.IterNodeDelegate* exposes interfaces for function application.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'))

>>> [x for x in f.iter_tuple_items(axis=0)]
[('diameter', Axis(Earth=12756, Mars=6792, Jupiter=142984)), ('mass', Axis(Earth=5.97,
↪ Mars=0.642, Jupiter=1898.0))]

>>> [x for x in f.iter_tuple_items(axis=1)]
[('Earth', Axis(diameter=12756.0, mass=5.97)), ('Mars', Axis(diameter=6792.0, mass=0.
↪642)), ('Jupiter', Axis(diameter=142984.0, mass=1898.0))]
```

(continues on next page)

(continued from previous page)

```
>>> f.iter_tuple_items(axis=1).apply(lambda k, v: v.diameter if k == 'Earth' else 0)
<Series>
<Index>
Earth    12756.0
Mars     0.0
Jupiter  0.0
<<U7>    <float64>
```

Frame.**iter_series**(*axis*)

Frame.**iter_series**(*axis*).*apply*(*func*)

Frame.**iter_series**(*axis*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

Frame.**iter_series**(*axis*).*apply_iter*(*func*)

Frame.**iter_series**(*axis*).*apply_iter_items*(*func*)

Iterate over Series of Frame axis, where axis 0 iterates column data and axis 1 iterates row data. The returned *static_frame.IterNodeDelegate* exposes interfaces for function application.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'), dtypes=dict(diameter=np.int64))

>>> next(iter(f.iter_series(axis=0)))
<Series>
<Index>
Earth    12756
Mars     6792
Jupiter  142984
<<U7>    <int64>

>>> next(iter(f.iter_series(axis=1)))
<Series>
<Index>
diameter 12756.0
mass     5.97
<<U8>    <float64>

>>> f.iter_series(0).apply(lambda s: s.mean())
<Series>
<Index>
diameter 54177.333333333336
mass     634.8706666666667
<<U8>    <float64>
```

Frame.**iter_series_items**(*axis*)

Frame.**iter_series_items**(*axis*).*apply*(*func*)

Frame.**iter_series_items**(*axis*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

Frame.**iter_series_items**(*axis*).*apply_iter*(*func*)

Frame.**iter_series_items**(*axis*).*apply_iter_items*(*func*)

Iterate over pairs of label, Series, per Frame axis, where axis 0 iterates column data and axis 1 iterates row data. The returned *static_frame.IterNodeDelegate* exposes interfaces for function application.

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'))

>>> [(k, v.mean()) for k, v in f.iter_series_items(0)]
[('diameter', 54177.333333333336), ('mass', 634.8706666666667)]

>>> [(k, v.max()) for k, v in f.iter_series_items(1)]
[('Earth', 12756.0), ('Mars', 6792.0), ('Jupiter', 142984.0)]

>>> f.iter_series_items(0).apply(lambda k, v: v.mean() if k == 'diameter' else v.
↪sum())
<Series>
<Index>
diameter 54177.333333333336
mass     1904.612
<<U8>    <float64>
```

Deviations from Pandas

The functionality of Pandas `pd.DataFrame.itertuples()` can be obtained with `Frame.iter_tuple(axis=0)`. The functionality of Pandas `pd.DataFrame.iterrows()` can be obtained with `Frame.iter_series(axis=0)`. The functionality of Pandas `pd.DataFrame.iteritems()` can be obtained with `Frame.iter_series_items(axis=1)`. The functionality of Pandas `pd.DataFrame.apply(axis)` can be obtained with `Frame.iter_series(axis).apply()`.

20.3 Group Iterators

20.3.1 Series

`Series.`**`iter_group`**(*key*)

`Series.`**`iter_group`**(*key*)*.apply(func*)

`Series.`**`iter_group`**(*key*)*.apply_pool(func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

`Series.`**`iter_group`**(*key*)*.apply_iter(func*)

`Series.`**`iter_group`**(*key*)*.apply_iter_items(func*)

Iterator of `Series` formed from groups of unique values in a Series.

```
>>> s = sf.Series((0, 0, 1, 2), index=('Mercury', 'Venus', 'Earth', 'Mars'), dtype=np.
↪int64)
>>> next(iter(s.iter_group()))
<Series>
<Index>
Mercury  0
Venus    0
<<U7>    <int64>
>>> [x.values.tolist() for x in s.iter_group()]
[[0, 0], [1], [2]]
```

`Series.`**`iter_group_items`**(*key*)

Series.**iter_group_items**(*key*).*apply*(*func*)

Series.**iter_group_items**(*key*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

Series.**iter_group_items**(*key*).*apply_iter*(*func*)

Series.**iter_group_items**(*key*).*apply_iter_items*(*func*)
Iterator of pairs of group value and the `Series` formed from groups of unique values in a Series.

```
>>> s = sf.Series((0, 0, 1, 2), index=('Mercury', 'Venus', 'Earth', 'Mars'))
>>> [(k, v.index.values.tolist()) for k, v in iter(s.iter_group_items()) if k > 0]
[(1, ['Earth']), (2, ['Mars'])]
```

Series.**iter_group_index**(*depth_level*)

Series.**iter_group_index**(*depth_level*).*apply*(*func*)

Series.**iter_group_index**(*depth_level*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

Series.**iter_group_index**(*depth_level*).*apply_iter*(*func*)

Series.**iter_group_index**(*depth_level*).*apply_iter_items*(*func*)
Iterator of `Series` formed from groups of unique `Index` labels.

Series.**iter_group_index_items**(*depth_level*)

Series.**iter_group_index_items**(*depth_level*).*apply*(*func*)

Series.**iter_group_index_items**(*depth_level*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

Series.**iter_group_index_items**(*depth_level*).*apply_iter*(*func*)

Series.**iter_group_index_items**(*depth_level*).*apply_iter_items*(*func*)
Iterator of pairs of group value and `Series` formed from groups of unique `Index` labels.

20.3.2 Frame

Frame.**iter_group**(*key*, *axis*)

Frame.**iter_group**(*key*, *axis*).*apply*(*func*)

Frame.**iter_group**(*key*, *axis*).*apply_pool*(*func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

Frame.**iter_group**(*key*, *axis*).*apply_iter*(*func*)

Frame.**iter_group**(*key*, *axis*).*apply_iter_items*(*func*)
Iterate over groups (as Frames) based on unique values found in the column specified by `key`. If axis is 0, subgroups of rows are retuned and key selects columns; If axis is 1, subgroups of columns are returned and key selects rows.

```
>>> f = sf.Frame.from_dict(dict(mass=(0.33, 4.87, 5.97, 0.642), moons=(0, 0, 1, 2)),␣
↪index=('Mercury', 'Venus', 'Earth', 'Mars'), dtypes=dict(moons=np.int64))
>>> next(iter(f.iter_group('moons')))
<Frame>
<Index> mass      moons   <<U5>
<Index>
Mercury 0.33      0
Venus   4.87      0
<<U7>   <float64> <int64>
```

(continues on next page)

(continued from previous page)

```
>>> [x.shape for x in f.iter_group('moons')]
[(2, 2), (1, 2), (1, 2)]
```

`Frame.`**`iter_group_items`**(*key*, *axis*)

`Frame.`**`iter_group_items`**(*key*, *axis).apply(func*)

`Frame.`**`iter_group_items`**(*key*, *axis).apply_pool(func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

`Frame.`**`iter_group_items`**(*key*, *axis).apply_iter(func*)

`Frame.`**`iter_group_items`**(*key*, *axis).apply_iter_items(func*)
Iterator over pairs of group value and groups (as `Frame`) based on unique values found in the column specified by `key`. If axis is 0, subgroups of rows are retuned and key selects columns; If axis is 1, subgroups of columns are returned and key selects rows.

```
>>> f = sf.Frame.from_dict(dict(mass=(0.33, 4.87, 5.97, 0.642), moons=(0, 0, 1, 2)),␣
↪index=('Mercury', 'Venus', 'Earth', 'Mars'))
>>> [(k, v.index.values.tolist(), v['mass'].mean()) for k, v in f.iter_group_items(
↪'moons')]
[(0, ['Mercury', 'Venus'], 2.6), (1, ['Earth'], 5.97), (2, ['Mars'], 0.642)]
```

`Frame.`**`iter_group_index`**(*depth_level*, *axis*)

`Frame.`**`iter_group_index`**(*depth_level*, *axis).apply(func*)

`Frame.`**`iter_group_index`**(*depth_level*, *axis).apply_pool(func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

`Frame.`**`iter_group_index`**(*depth_level*, *axis).apply_iter(func*)

`Frame.`**`iter_group_index`**(*depth_level*, *axis).apply_iter_items(func*)
Iterate over groups (as `Frame`) based on unique labels found in the index specified by `depth_level`. If axis is 0, subgroups of rows are retuned and `depth_level` selects columns; If axis is 1, subgroups of columns are returned and `depth_level` selects rows.

`Frame.`**`iter_group_index_items`**(*depth_level*, *axis*)

`Frame.`**`iter_group_index_items`**(*depth_level*, *axis).apply(func*)

`Frame.`**`iter_group_index_items`**(*depth_level*, *axis).apply_pool(func*, *dtype*, *max_workers*, *chunksize*, *use_threads*)

`Frame.`**`iter_group_index_items`**(*depth_level*, *axis).apply_iter(func*)

`Frame.`**`iter_group_index_items`**(*depth_level*, *axis).apply_iter_items(func*)
Iterator over pairs of group value and groups (as `Frame`) based on unique labels found in the index specified by `depth_level`. If axis is 0, subgroups of rows are retuned and `depth_level` selects columns; If axis is 1, subgroups of columns are returned and `depth_level` selects rows.

CHAPTER

TWENTYONE

FUNCTION APPLICATION TO ITERATORS

static_frame.Frame and *static_frame.Series* *static_frame.IterNode* attributes return, when called, *static_frame.IterNodeDelegate* instances. These instances are prepared for iteration via *static_frame.IterNodeDelegate.__iter__()*, and expose a number of methods for function application.

class IterNode(**, container: FrameOrSeries, function_values: Callable[[...], Iterable[Any]], function_items: Callable[[...], Iterable[Tuple[Any, Any]]], yield_type: static_frame.core.iter_node.IterNodeType, apply_type: static_frame.core.iter_node.IterNodeApplyType = <IterNodeApplyType.SERIES_ITEMS: 1>*)

Interface to a type of iteration on *static_frame.Series* and *static_frame.Frame*.

class IterNodeDelegate(*func_values: Callable[[...], Iterable[Any]], func_items: Callable[[...], Iterable[Tuple[Any, Any]]], yield_type: static_frame.core.iter_node.IterNodeType, apply_constructor: Callable[[...], FrameOrSeries]*)

Delegate returned from *static_frame.IterNode*, providing iteration as well as a family of apply methods.

IterNodeDelegate.**__iter__**() → Union[Iterator[Any], Iterator[Tuple[Any, Any]]]

Return a generator based on the yield type.

IterNodeDelegate.**apply**(*func: Union[Callable[[...], Any], Mapping[Hashable, Any], Series], *, dtype: Union[str, numpy.dtype, type, None] = None*) → FrameOrSeries

Apply passed function to each object iterated, where the object depends on the creation of this instance.

Parameters

- **func** – A function, or a mapping object that defines __getitem__. If a mapping is given, all values must be found in the mapping.
- **dtype** – Type used to create the returned array.

IterNodeDelegate.**apply_pool**(*func: Union[Callable[[...], Any], Mapping[Hashable, Any], Series], *, dtype: Union[str, numpy.dtype, type, None] = None, max_workers: Optional[int] = None, chunksize: int = 1, use_threads: bool = False*) → FrameOrSeries

Apply passed function to each object iterated, where the object depends on the creation of this instance. Employ parallel processing with either the ProcessPoolExecutor or ThreadPoolExecutor.

Parameters

- **func** – A function, or a mapping object that defines __getitem__. If a mapping is given, all values must be found in the mapping.
- **dtype** – Type used to create the returned array.

- **max_workers** – Passed to the pool_executor, where None defaults to the max number of machine processes.
- **chunksize** – Passed to the pool executor.
- **use_thread** – When True, the ThreadPoolExecutor will be used rather than the default ProcessPoolExecutor.

IterNodeDelegate.**apply_iter**(*func: Union[Callable[[...], Any], Mapping[Hashable, Any], Series]*) → Generator[Any, None, None]

Generator that applies the passed function to each element iterated and yields the result.

Parameters **func** – A function, or a mapping object that defines __getitem__. If a mapping is given, all values must be found in the mapping.

IterNodeDelegate.**apply_iter_items**(*func: Union[Callable[[...], Any], Mapping[Hashable, Any], Series]*) → Generator[Tuple[Any, Any], None, None]

Generator that applies function to each element iterated and yields the pair of element and the result.

Parameters **func** – A function or a mapping object that defines `__getitem__` and `__contains__`. If a mpping is given and a value is not found in the mapping, the value is returned unchanged (this deviates from Pandas `Series.map`, which inserts NaNs)

CHAPTER

TWENTYTWO

MISSING VALUE HANDLING

static_frame.Series and *static_frame.Frame* provide convenient funcions for finding, dropping, and replacing missing values. In the tradition of Pandas, NaN and None values are treated as both missing, regardless of the dtype in which they are contained.

22.1 Series

Series.**isna**() → static_frame.core.series.Series
Return a same-indexed, Boolean Series indicating which values are NaN or None.

Series.**notna**() → static_frame.core.series.Series
Return a same-indexed, Boolean Series indicating which values are NaN or None.

Series.**dropna**() → static_frame.core.series.Series
Return a new Series after removing values of NaN or None.

Series.**fillna**(*value*) → static_frame.core.series.Series
Return a new Series after replacing null (NaN or None) with the supplied value.

Parameters **value** – Value to be used to replace missing values (NaN or None).

Series.**fillna_forward**(*limit: int = 0*) → static_frame.core.series.Series
Return a new Series after feeding forward the last non-null (NaN or None) observation across contiguous nulls.

Parameters **limit** – Set the maximum count of missing values (NaN or None) to be filled per contiguous region of missing vlaues. A value of 0 is equivalent to no limit.

Series.**fillna_backward**(*limit: int = 0*) → static_frame.core.series.Series
Return a new Series after feeding backward the last non-null (NaN or None) observation across contiguous nulls.

Parameters **limit** – Set the maximum count of missing values (NaN or None) to be filled per contiguous region of missing vlaues. A value of 0 is equivalent to no limit.

Series.**fillna_leading**(*value: Any*) → static_frame.core.series.Series
Return a new Series after filling leading (and only leading) null (NaN or None) with the supplied value.

Parameters **value** – Value to be used to replace missing values (NaN or None).

Series.**fillna_trailing**(*value: Any*) → static_frame.core.series.Series
Return a new Series after filling trailing (and only trailing) null (NaN or None) with the supplied value.

Parameters **value** – Value to be used to replace missing values (NaN or None).

22.2 Frame

`Frame.`**`isna`**`()` → static_frame.core.frame.Frame
Return a same-indexed, Boolean Frame indicating True which values are NaN or None.

`Frame.`**`notna`**`()` → static_frame.core.frame.Frame
Return a same-indexed, Boolean Frame indicating True which values are not NaN or None.

`Frame.`**`dropna`**`(`*axis: int = 0, condition: Callable[[numpy.ndarray], bool] = <function all>*`)` → static_frame.core.frame.Frame
Return a new Frame after removing rows (axis 0) or columns (axis 1) where condition is True, where condition is an NumPy ufunc that process the Boolean array returned by isna().

`Frame.`**`fillna`**`(`*value: Any*`)` → static_frame.core.frame.Frame
Return a new `Frame` after replacing null (NaN or None) with the supplied value.

Parameters **`value`** – Value to be used to replace missing values (NaN or None).

`Frame.`**`fillna_forward`**`(`*limit: int = 0, *, axis: int = 0*`)` → static_frame.core.frame.Frame
Return a new `Frame` after filling forward null (NaN or None) with the supplied value.

Parameters

- **`limit`** – Set the maximum count of missing values (NaN or None) to be filled per contiguous region of missing vlaues. A value of 0 is equivalent to no limit.
- **`axis`** – Axis upon which to evaluate contiguous missing values, where 0 is vertically (between row values) and 1 is horizontally (between column values).

`Frame.`**`fillna_backward`**`(`*limit: int = 0, *, axis: int = 0*`)` → static_frame.core.frame.Frame
Return a new `Frame` after filling backward null (NaN or None) with the supplied value.

Parameters

- **`limit`** – Set the maximum count of missing values (NaN or None) to be filled per contiguous region of missing vlaues. A value of 0 is equivalent to no limit.
- **`axis`** – Axis upon which to evaluate contiguous missing values, where 0 is vertically (between row values) and 1 is horizontally (between column values).

`Frame.`**`fillna_leading`**`(`*value: Any, *, axis: int = 0*`)` → static_frame.core.frame.Frame
Return a new `Frame` after filling leading (and only leading) null (NaN or None) with the supplied value.

Parameters

- **`value`** – Value to be used to replace missing values (NaN or None).
- **`axis`** – Axis upon which to evaluate contiguous missing values, where 0 is vertically (between row values) and 1 is horizontally (between column values).

`Frame.`**`fillna_trailing`**`(`*value: Any, *, axis: int = 0*`)` → static_frame.core.frame.Frame
Return a new `Frame` after filling trailing (and only trailing) null (NaN or None) with the supplied value.

Parameters

- **`value`** – Value to be used to replace missing values (NaN or None).
- **`axis`** – Axis upon which to evaluate contiguous missing values, where 0 is vertically (between row values) and 1 is horizontally (between column values).

Deviations from Pandas

dropna() takes a `condition` argument, which is a NumPy ufunc that accepts an axis argument. This differs from Pandas `how` argument. A `how` of "all" is equivalent to a `condition` of `np.all`; A `how` of "any" is equivalent to a `condition` of `np.any`.

CHAPTER

TWENTYTHREE

SORTING

`Index`, `Series` and `Frame` provide sorting. In all cases, a new object is returned.

23.1 Index

`Index`.**`sort`**(*ascending: bool = True*, *kind: str = 'mergesort'*) → static_frame.core.index.Index
Return a new Index with the labels sorted.

Parameters **`kind`** – Sort algorithm passed to NumPy.

23.2 Series

`Series`.**`sort_index`**(*ascending: bool = True*, *kind: str = 'mergesort'*) → static_frame.core.series.Series
Return a new Series ordered by the sorted Index.

`Series`.**`sort_values`**(*ascending: bool = True*, *kind: str = 'mergesort'*) → static_frame.core.series.Series
Return a new Series ordered by the sorted values.

23.3 Frame

`Frame`.**`sort_index`**(*ascending: bool = True*, *kind: str = 'mergesort'*) → static_frame.core.frame.Frame
Return a new Frame ordered by the sorted Index.

`Frame`.**`sort_columns`**(*ascending: bool = True*, *kind: str = 'mergesort'*) → static_frame.core.frame.Frame
Return a new Frame ordered by the sorted Columns.

`Frame`.**`sort_values`**(*key: Union[Hashable, Iterable[Hashable]], ascending: bool = True, axis: int = 1, kind='mergesort'*) → static_frame.core.frame.Frame
Return a new Frame ordered by the sorted values, where values is given by single column or iterable of columns.

Parameters **`key`** – a key or tuple of keys. Presently a list is not supported.

Deviations from Pandas

The default sort kind, delegated to NumPy sorting routines, is merge sort, a stable sort. In some versions of Pandas the default sort kind is quicksort.

CHAPTER

TWENTYFOUR

TRANSFORMATIONS & UTILITIES

The following utilites transform a container into a container of similar size.

24.1 Index

`Index.`**`isin`**(*other: Iterable[Any]*) → numpy.ndarray
Return a Boolean array showing True where a label is found in other. If other is a multidimensional array, it is flattened.

`Index.`**`roll`**(*shift: int*) → static_frame.core.index.Index
Return an Index with values rotated forward and wrapped around (with a postive shift) or backward and wrapped around (with a negative shift).

24.2 Series

`Series.`**`astype`**(*dtype: Union[str, numpy.dtype, type, None]*) → static_frame.core.series.Series
Return a Series with type determined by *dtype* argument. Note that for Series, this is a simple function, whereas for Frame, this is an interface exposing both a callable and a getitem interface.

`Series.`**`clip`**(*lower=None*, *upper=None*)
Apply a clip opertion to this `Series`. Note that clip operations can be applied to object types, but cannot be applied to non-numerical objects (e.g., strings, None)

Parameters

- **`lower`** – value or `Series` to define the inclusive lower bound.
- **`upper`** – value or `Series` to define the inclusive upper bound.

`Series.`**`isin`**(*other*) → static_frame.core.series.Series
Return a same-sized Boolean Series that shows if the same-positoined element is in the iterable passed to the function.

`Series.`**`transpose`**() → static_frame.core.series.Series
The transpositon of a Series is itself.

`Series.`**`unique`**() → numpy.ndarray
Return a NumPy array of unqiue values.

`Series.`**`duplicated`**(*exclude_first=False*, *exclude_last=False*) → numpy.ndarray
Return a same-sized Boolean Series that shows True for all b values that are duplicated.

Series.**drop_duplicated**(*exclude_first: bool = False*, *exclude_last: bool = False*) → static_frame.core.series.Series
Return a Series with duplicated values removed.

Series.**roll**(*shift: int*, *include_index: bool = False*) → static_frame.core.series.Series
Return a Series with values rotated forward and wrapped around the index (with a postive shift) or backward and wrapped around the index (with a negative shift).

Parameters

- **shift** – Postive or negative integer shift.
- **include_index** – Determine if the Index is shifted with the underlying data.

Series.**shift**(*shift: int*, *fill_value=nan*) → static_frame.core.series.Series
Return a Series with values shifted forward on the index (with a postive shift) or backward on the index (with a negative shift).

Parameters

- **shift** – Postive or negative integer shift.
- **fill_value** – Value to be used to fill data missing after the shift.

Series.**head**(*count: int = 5*) → static_frame.core.series.Series
Return a Series consisting only of the top elements as specified by `count`.

Parameters **count** – Number of elements to be returned from the top of the Series.

Series.**tail**(*count: int = 5*) → static_frame.core.series.Series
Return a Series consisting only of the bottom elements as specified by `count`.

Parameters **count** – Number of elements to be returned from the bottom of the Series.

24.3 Frame

Series.**astype**(*dtype*)
Replace the values specified by the key with values casted to the provided dtype.

Series.astype[key](dtype)
Given a column key (either a column label, list of column lables, slice of colum labels, or Boolean array), replace the values specified by the column key with values casted to the provided `dtype`.

Frame.**clip**(*lower=None*, *upper=None*, *axis: Optional[int] = None*)
Apply a clip opertion to this `Frame`. Note that clip operations can be applied to object types, but cannot be applied to non-numerical objects (e.g., strings, None)

Parameters

- **lower** – value, `Series`, `Frame`
- **upper** – value, `Series`, `Frame`
- **axis** – required if `lower` or `upper` are given as a `Series`.

Frame.**isin**(*other*) → static_frame.core.frame.Frame
Return a same-sized Boolean Frame that shows if the same-positioned element is in the iterable passed to the function.

Frame.**transpose**() → static_frame.core.frame.Frame
Return a tansposed version of the `Frame`.

`Frame.`**`unique`**(*axis: Optional[int] = None*) → numpy.ndarray
Return a NumPy array of unqiue values. If the axis argument is provied, uniqueness is determined by columns or row.

`Frame.`**`duplicated`**(*axis=0*, *exclude_first=False*, *exclude_last=False*) → static_frame.core.series.Series
Return an axis-sized Boolean Series that shows True for all rows (axis 0) or columns (axis 1) duplicated.

`Frame.`**`drop_duplicated`**(*axis=0*, *exclude_first: bool = False*, *exclude_last: bool = False*) → static_frame.core.frame.Frame
Return a Frame with duplicated values removed.

`Frame.`**`roll`**(*index: int = 0*, *columns: int = 0*, *include_index: bool = False*, *include_columns: bool = False*) → static_frame.core.frame.Frame

Parameters

- **`include_index`** – Determine if index is included in index-wise rotation.
- **`include_columns`** – Determine if column index is included in index-wise rotation.

`Frame.`**`shift`**(*index: int = 0*, *columns: int = 0*, *fill_value=nan*) → static_frame.core.frame.Frame

`Frame.`**`head`**(*count: int = 5*) → static_frame.core.frame.Frame
Return a Frame consisting only of the top rows as specified by `count`.

`Frame.`**`tail`**(*count: int = 5*) → static_frame.core.frame.Frame
Return a Frame consisting only of the bottom rows as specified by `count`.

Deviations from Pandas

Pandas `pd.DataFrame.duplicated()` is equivalent to `Frame.duplicated(exclude_first=True)`. Pandas `pd.DataFrame.drop_duplicates()` is equivalent to `Frame.drop_duplicated(exclude_first=True)`.

CHAPTER

TWENTYFIVE

MATHEMATICAL / LOGICAL / STATISTICAL UTILITIES

`Series`, `Frame`, and `Index`, as well as their derived classes, provide support for common mathematical and statisticasl operations with NumPy.

25.1 Index

Mathematical and statistical operations, when applied on an Index, apply to the index labels.

`Index.`**`all`**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Logical and over values along the specified axis.

Parameters

- **`axis`** – Axis, defaulting to axis 0.
- **`skipna`** – Skip missing (NaN) values, defaulting to True.

`Index.`**`any`**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Logical or over values along the specified axis.

Parameters

- **`axis`** – Axis, defaulting to axis 0.
- **`skipna`** – Skip missing (NaN) values, defaulting to True.

`Index.`**`max`**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the maximum along the specified axis.

Parameters

- **`axis`** – Axis, defaulting to axis 0.
- **`skipna`** – Skip missing (NaN) values, defaulting to True.

`Index.`**`mean`**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the mean along the specified axis.

Parameters

- **`axis`** – Axis, defaulting to axis 0.
- **`skipna`** – Skip missing (NaN) values, defaulting to True.

`Index.`**`median`**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the median along the specified axis.

Parameters

- **`axis`** – Axis, defaulting to axis 0.

- **skipna** – Skip missing (NaN) values, defaulting to True.

Index.**min**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the minimum along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Index.**prod**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the product along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Index.**std**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the standard deviaton along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Index.**sum**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Sum values along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Index.**var**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the variance along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

25.2 Series

Series.**all**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Logical and over values along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**any**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Logical or over values along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**max**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the maximum along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**mean**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the mean along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**median**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the median along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**min**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the minimum along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**prod**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the product along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**std**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the standard deviaton along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**sum**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Sum values along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**var**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the variance along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.

- **skipna** – Skip missing (NaN) values, defaulting to True.

Series.**loc_min**(*, *skipna: bool = True*) → Hashable
Return the label corresponding to the minimum value found.

Parameters **skipna** – if True, NaN or None values will be ignored; if False, a found NaN will propagate.

Series.**loc_max**(*, *skipna: bool = True*) → Hashable
Return the label corresponding to the maximum value found.

Parameters **skipna** – if True, NaN or None values will be ignored; if False, a found NaN will propagate.

Series.**iloc_min**(*, *skipna: bool = True*) → int
Return the integer index corresponding to the minimum value found.

Parameters **skipna** – if True, NaN or None values will be ignored; if False, a found NaN will propagate.

Series.**iloc_max**(*, *skipna: bool = True*) → int
Return the integer index corresponding to the maximum value.

Parameters **skipna** – if True, NaN or None values will be ignored; if False, a found NaN will propagate.

25.3 Frame

Frame.**all**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Logical and over values along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**any**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Logical or over values along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**max**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the maximum along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**mean**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the mean along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**median**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the median along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**min**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the minimum along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**prod**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the product along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**std**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the standard deviaton along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**sum**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Sum values along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**var**(*axis: int = 0*, *skipna: bool = True*, ***_: object*) → Any
Return the variance along the specified axis.

Parameters

- **axis** – Axis, defaulting to axis 0.
- **skipna** – Skip missing (NaN) values, defaulting to True.

Frame.**loc_min**(*, *skipna: bool = True*, *axis: int = 0*) → static_frame.core.series.Series
Return the labels corresponding to the minimum value found.

Parameters

- **skipna** – if True, NaN or None values will be ignored; if False, a found NaN will propagate.
- **axis** – Axis upon which to evaluate contiguous missing values, where 0 is vertically (between row values) and 1 is horizontally (between column values).

Frame.**loc_max**(*, *skipna: bool = True*, *axis: int = 0*) → static_frame.core.series.Series
Return the labels corresponding to the maximum values found.

Parameters

- **skipna** – if True, NaN or None values will be ignored; if False, a found NaN will propagate.
- **axis** – Axis upon which to evaluate contiguous missing values, where 0 is vertically (between row values) and 1 is horizontally (between column values).

Frame.**iloc_min**(*, *skipna: bool = True*, *axis: int = 0*) → static_frame.core.series.Series

Return the integer indices corresponding to the minimum values found.

Parameters

- **skipna** – if True, NaN or None values will be ignored; if False, a found NaN will propagate.
- **axis** – Axis upon which to evaluate contiguous missing values, where 0 is vertically (between row values) and 1 is horizontally (between column values).

Frame.**iloc_max**(*, *skipna: bool = True*, *axis: int = 0*) → static_frame.core.series.Series

Return the integer indices corresponding to the maximum values found.

Parameters

- **skipna** – if True, NaN or None values will be ignored; if False, a found NaN will propagate.
- **axis** – Axis upon which to evaluate contiguous missing values, where 0 is vertically (between row values) and 1 is horizontally (between column values).

25.3.1 Examples

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 142984, 120536), mass=(5.97, 1898,␣
↪568)), index=('Earth', 'Jupiter', 'Saturn'), dtypes=dict(diameter=np.int64))
>>> f
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Jupiter 142984   1898.0
Saturn  120536   568.0
<<U7>   <int64>  <float64>

>>> f.max()
<Series>
<Index>
diameter 142984.0
mass     1898.0
<<U8>    <float64>
>>> f.min()
<Series>
<Index>
diameter 12756.0
mass     5.97
<<U8>    <float64>
>>> f.std()
<Series>
<Index>
diameter 56842.64155250587
```

(continues on next page)

(continued from previous page)

```
mass     793.344204533358
<<U8>    <float64>
>>> f.sum()
<Series>
<Index>
diameter 276276.0
mass     2471.9700000000003
<<U8>    <float64>
>>> f.mean()
<Series>
<Index>
diameter 92092.0
mass     823.9900000000001
<<U8>    <float64>
```

CHAPTER

TWENTYSIX

OPERATORS

`Series`, `Frame`, and `Index`, as well as their derived classes, provide support for the full range of operators available with NumPy. In addition, `Series` and `Frame` feature index-alignment and automatic index expansion when both operands are StaticFrame objects.

26.1 Index

Index operators operate on the Index labels. In all cases, an immutable NumPy array is returned rather than a new Index instance.

26.1.1 Unary Operators

`Index.__abs__()`
Same as abs(a).

`Index.__invert__()`
Same as ~a.

`Index.__neg__()`
Same as -a.

`Index.__pos__()`
Same as +a.

26.1.2 Binary Operators

`Index.__add__(other)`
Same as a + b.

`Index.__and__(other)`
Same as a & b.

`Index.__eq__(other)`
Same as a == b.

`Index.__floordiv__(other)`
Same as a // b.

`Index.__ge__(other)`
Same as a >= b.

`Index.__gt__(other)`
Same as a > b.

Index.__le__(*other*)
Same as a <= b.

Index.__lshift__(*other*)
Same as a << b.

Index.__lt__(*other*)
Same as a < b.

Index.__matmul__(*other*)
Same as a @ b.

Index.__mod__(*other*)
Same as a % b.

Index.__mul__(*other*)
Same as a * b.

Index.__ne__(*other*)
Same as a != b.

Index.__or__(*other*)
Same as a | b.

Index.__pow__(*other*)
Same as a ** b.

Index.__rshift__(*other*)
Same as a >> b.

Index.__sub__(*other*)
Same as a - b.

Index.__truediv__(*other*)
Same as a / b.

Index.__xor__(*other*)
Same as a ^ b.

26.2 Series

Series operators operate on the Series values. In all cases, a new Series is returned. Operations on two Series always return a new Series with a union Index.

26.2.1 Unary Operators

Series.__abs__()
Same as abs(a).

Series.__invert__()
Same as ~a.

Series.__neg__()
Same as -a.

Series.__pos__()
Same as +a.

26.2.2 Binary Operators

Series.**__add__**(*other*)
Same as a + b.

Series.**__and__**(*other*)
Same as a & b.

Series.**__eq__**(*other*)
Same as a == b.

Series.**__floordiv__**(*other*)
Same as a // b.

Series.**__ge__**(*other*)
Same as a >= b.

Series.**__gt__**(*other*)
Same as a > b.

Series.**__le__**(*other*)
Same as a <= b.

Series.**__lshift__**(*other*)
Same as a << b.

Series.**__lt__**(*other*)
Same as a < b.

Series.**__matmul__**(*other*)
Same as a @ b.

Series.**__mod__**(*other*)
Same as a % b.

Series.**__mul__**(*other*)
Same as a * b.

Series.**__ne__**(*other*)
Same as a != b.

Series.**__or__**(*other*)
Same as a | b.

Series.**__pow__**(*other*)
Same as a ** b.

Series.**__rshift__**(*other*)
Same as a >> b.

Series.**__sub__**(*other*)
Same as a - b.

Series.**__truediv__**(*other*)
Same as a / b.

Series.**__xor__**(*other*)
Same as a ^ b.

26.2.3 Examples

```
>>> s = sf.Series.from_items((('Venus', 108.2), ('Earth', 149.6), ('Saturn', 1433.5)))
>>> s
<Series>
<Index>
Venus    108.2
Earth    149.6
Saturn   1433.5
<<U6>    <float64>

>>> abs(s - s['Earth'])
<Series>
<Index>
Venus    41.39999999999999
Earth    0.0
Saturn   1283.9
<<U6>    <float64>

>>> s > s['Earth']
<Series>
<Index>
Venus    False
Earth    False
Saturn   True
<<U6>    <bool>

>>> s / s['Earth']
<Series>
<Index>
Venus    0.7232620320855615
Earth    1.0
Saturn   9.582219251336898
<<U6>    <float64>
```

```
>>> s1 = sf.Series((1, 2), index=('Earth', 'Mars'))
>>> s2 = sf.Series((2, 0), index=('Mars', 'Mercury'))
>>> s1 * s2
<Series>
<Index>
Earth    nan
Mars     4.0
Mercury  nan
<<U7>    <float64>

>>> s1 == s2
<Series>
<Index>
Earth    False
Mars     True
Mercury  False
<<U7>    <bool>
```

26.3 Frame

Frame operators operate on the Frame values. In all cases, a new Frame is returned.

26.3.1 Unary Operators

Frame.**__abs__**()
Same as abs(a).

Frame.**__invert__**()
Same as ~a.

Frame.**__neg__**()
Same as -a.

Frame.**__pos__**()
Same as +a.

26.3.2 Binary Operators

Frame.**__add__**(*other*)
Same as a + b.

Frame.**__and__**(*other*)
Same as a & b.

Frame.**__eq__**(*other*)
Same as a == b.

Frame.**__floordiv__**(*other*)
Same as a // b.

Frame.**__ge__**(*other*)
Same as a >= b.

Frame.**__gt__**(*other*)
Same as a > b.

Frame.**__le__**(*other*)
Same as a <= b.

Frame.**__lshift__**(*other*)
Same as a << b.

Frame.**__lt__**(*other*)
Same as a < b.

Frame.**__matmul__**(*other*)
Same as a @ b.

Frame.**__mod__**(*other*)
Same as a % b.

Frame.**__mul__**(*other*)
Same as a * b.

Frame.**__ne__**(*other*)
Same as a != b.

`Frame.__or__`(*other*)
Same as a | b.

`Frame.__pow__`(*other*)
Same as a ** b.

`Frame.__rshift__`(*other*)
Same as a >> b.

`Frame.__sub__`(*other*)
Same as a - b.

`Frame.__truediv__`(*other*)
Same as a / b.

`Frame.__xor__`(*other*)
Same as a ^ b.

Deviations from Pandas

For consistency in operator application and to insure index alignment, all operators return an union index when both opperrands are StaticFrame containers. This deviates from Pandas, where in some versions equality operators did not align on a union index, and behaved differently than other operators.

26.3.3 Examples

```
>>> f = sf.Frame.from_dict(dict(diameter=(12756, 6792, 142984), mass=(5.97, 0.642,␣
↪1898)), index=('Earth', 'Mars', 'Jupiter'), dtypes=dict(diameter=np.int64))
>>> f
<Frame>
<Index> diameter mass      <<U8>
<Index>
Earth   12756    5.97
Mars    6792     0.642
Jupiter 142984   1898.0
<<U7>   <int64>  <float64>

>>> f / f.loc['Earth']
<Frame>
<Index> diameter           mass                <<U8>
<Index>
Earth   1.0                1.0
Mars    0.5324553151458138 0.10753768844221107
Jupiter 11.209156475384132 317.92294807370183
<<U7>   <float64>          <float64>
```

CHAPTER

TWENTYSEVEN

CONTAINER EXPORT

Methods for exporting alternative representations from `Series` and `Frame`.

27.1 Index

`Index.`**`to_series`**`()`
Return a Series with values from this Index's labels.

`Index.`**`to_html`**`(`*config: Optional[static_frame.core.display.DisplayConfig] = None*`)` → str
Return an HTML table representation of this `Index` using standard TABLE, TR, and TD tags. This is not a complete HTML page.

Parameters **`config`** – Optional `static_frame.DisplayConfig` instance.

`Index.`**`to_html_datatables`**`(`*fp: Union[str, TextIO, None] = None*, *, *show: bool = True*, *config: Optional[static_frame.core.display.DisplayConfig] = None*`)` → Optional[str]
Return a complete HTML representation of this `Index` using the DataTables JS library for table naviagation and search. The page links to CDNs for JS resources, and thus will not fully render without an internet connection.

Parameters

- **`fp`** – optional file path to write; if not provided, a temporary file will be created. Note: the caller is responsible for deleting this file.
- **`show`** – if True, the file will be opened with a webbrowser.
- **`config`** – Optional `static_frame.DisplayConfig` instance.

Returns Absolute file path to the file written.

`Index.`**`to_pandas`**`()` → pd.Index
Return a Pandas Index.

27.2 Index Hierarchy

`IndexHierarchy.`**`to_frame`**`()` → Frame
Return the index as a Frame.

`IndexHierarchy.`**`to_html`**`(`*config: Optional[static_frame.core.display.DisplayConfig] = None*`)` → str
Return an HTML table representation of this `Index` using standard TABLE, TR, and TD tags. This is not a complete HTML page.

Parameters **`config`** – Optional `static_frame.DisplayConfig` instance.

IndexHierarchy.**to_html_datatables**(*fp: Union[str, TextIO, None] = None*, ***, *show: bool = True*, *config: Optional[static_frame.core.display.DisplayConfig] = None*) → Optional[str]

Return a complete HTML representation of this Index using the DataTables JS library for table naviagation and search. The page links to CDNs for JS resources, and thus will not fully render without an internet connection.

Parameters

- **fp** – optional file path to write; if not provided, a temporary file will be created. Note: the caller is responsible for deleting this file.
- **show** – if True, the file will be opened with a webbrowser.
- **config** – Optional static_frame.DisplayConfig instance.

Returns Absolute file path to the file written.

IndexHierarchy.**to_pandas**() → DataFrame

Return a Pandas MultiIndex.

27.3 Series

Series.**to_pairs**() → Iterable[Tuple[Hashable, Any]]

Return a tuple of tuples, where each inner tuple is a pair of index label, value.

Series.**to_frame**(*axis: int = 1*)

Return a *static_frame.Frame* view of this *static_frame.Series*. As underlying data is immutable, this is a no-copy operation.

Series.**to_frame_go**(*axis: int = 1*)

Return *static_frame.FrameGO* view of this *static_frame.Series*. As underlying data is immutable, this is a no-copy operation.

Series.**to_pandas**() → DataFrame

Return a Pandas Series.

Series.**to_html**(*config: Optional[static_frame.core.display.DisplayConfig] = None*)

Return an HTML table representation of this Series using standard TABLE, TR, and TD tags. This is not a complete HTML page.

Parameters **config** – Optional static_frame.DisplayConfig instance.

Series.**to_html_datatables**(*fp: Union[str, TextIO, None] = None*, *show: bool = True*, *config: Optional[static_frame.core.display.DisplayConfig] = None*) → str

Return a complete HTML representation of this Series using the DataTables JS library for table naviagation and search. The page links to CDNs for JS resources, and thus will not fully render without an internet connection.

Parameters

- **fp** – optional file path to write; if not provided, a temporary file will be created. Note: the caller is responsible for deleting this file.
- **show** – if True, the file will be opened with a webbrowser.
- **config** – Optional static_frame.DisplayConfig instance.

Returns Absolute file path to the file written.

27.4 Frame

`Frame.`**`to_pairs`**(*axis*) → Iterable[Tuple[Hashable, Iterable[Tuple[Hashable, Any]]]]
Return a tuple of major axis key, minor axis key vlaue pairs, where major axis is determined by the axis argument.

`Frame.`**`to_frame_go`**() → static_frame.core.frame.FrameGO
Return a FrameGO view of this Frame. As underlying data is immutable, this is a no-copy operation.

`Frame.`**`to_pandas`**()
Return a Pandas DataFrame.

`Frame.`**`to_csv`**(*fp: Union[str, TextIO], delimiter: str = ',', include_index: bool = True, include_columns: bool = True, encoding: Optional[str] = None, line_terminator: str = '\n'*)
Given a file path or file-like object, write the Frame as delimited text.

Parameters **`delimiter`** – character to be used for delimiterarating elements.

`Frame.`**`to_tsv`**(*fp: Union[str, TextIO], **kwargs*)
Given a file path or file-like object, write the Frame as tab-delimited text.

`Frame.`**`to_html`**(*config: Optional[static_frame.core.display.DisplayConfig] = None*)
Return an HTML table representation of this `Frame` using standard TABLE, TR, and TD tags. This is not a complete HTML page.

Parameters **`config`** – Optional `static_frame.DisplayConfig` instance.

`Frame.`**`to_html_datatables`**(*fp: Union[str, TextIO, None] = None, show: bool = True, config: Optional[static_frame.core.display.DisplayConfig] = None*) → str
Return a complete HTML representation of this `Frame` using the DataTables JS library for table naviagation and search. The page links to CDNs for JS resources, and thus will not fully render without an internet connection.

Parameters

- **`fp`** – optional file path to write; if not provided, a temporary file will be created. Note: the caller is responsible for deleting this file.
- **`show`** – if True, the file will be opened with a webbrowser.
- **`config`** – Optional `static_frame.DisplayConfig` instance.

Returns Absolute file path to the file written.

genindex

INDEX

Symbols

A

C

D

F

G

H

I

K

L

M

N

P

R

S

T

U

V