pandas.StringDtype

class pandas.**StringDtype**
Extension dtype for string data.

New in version 1.0.0.

> **Warning:** StringDtype is considered experimental. The implementation and parts of the API may change without warning.
>
> In particular, StringDtype.na_value may change to no longer be `numpy.nan`.

Examples

```
>>> pd.StringDtype()
StringDtype
```

Attributes

None	

Methods

None	

The `Series.str` accessor is available for `Series` backed by a *arrays.StringArray*. See *String handling* for more.

3.5.11 Boolean data with missing values

The boolean dtype (with the alias `"boolean"`) provides support for storing boolean data (True, False values) with missing values, which is not possible with a bool `numpy.ndarray`.

arrays.BooleanArray(values, mask, copy)	Array of boolean (True/False) data with missing values.

pandas.arrays.BooleanArray

class pandas.arrays.**BooleanArray**(*values: numpy.ndarray*, *mask: numpy.ndarray*, *copy: bool = False*)
Array of boolean (True/False) data with missing values.

This is a pandas Extension array for boolean data, under the hood represented by 2 numpy arrays: a boolean array with the data and a boolean array with the mask (True indicating missing).

BooleanArray implements Kleene logic (sometimes called three-value logic) for logical operations. See *Kleene Logical Operations* for more.

To construct an BooleanArray from generic array-like input, use *pandas.array()* specifying `dtype=`

`"boolean"` (see examples below).

New in version 1.0.0.

Warning: BooleanArray is considered experimental. The implementation and parts of the API may change without warning.

Parameters

values [numpy.ndarray] A 1-d boolean-dtype array with the data.

mask [numpy.ndarray] A 1-d boolean-dtype array indicating missing values (True indicates missing).

copy [bool, default False] Whether to copy the *values* and *mask* arrays.

Returns

BooleanArray

Examples

Create an BooleanArray with *pandas.array()*:

```
>>> pd.array([True, False, None], dtype="boolean")
<BooleanArray>
[True, False, <NA>]
Length: 3, dtype: boolean
```

Attributes

None	

Methods

None	

BooleanDtype()	Extension dtype for boolean data.

pandas.BooleanDtype

class pandas.**BooleanDtype**

Extension dtype for boolean data.

New in version 1.0.0.

Warning: BooleanDtype is considered experimental. The implementation and parts of the API may change without warning.

Examples

```
>>> pd.BooleanDtype()
BooleanDtype
```

Attributes

None	

Methods

None	

3.6 Panel

Panel was removed in 0.25.0. For prior documentation, see the 0.24 documentation

3.7 Index objects

3.7.1 Index

Many of these methods or variants thereof are available on the objects that contain an index (Series/DataFrame) and those should most likely be used before calling these methods directly.

Index([data, dtype, copy, name, tupleize_cols])	Immutable ndarray implementing an ordered, sliceable set.

pandas.Index

class pandas.**Index**(*data=None*, *dtype=None*, *copy=False*, *name=None*, *tupleize_cols=True*, ***kwargs*)

Immutable ndarray implementing an ordered, sliceable set. The basic object storing axis labels for all pandas objects.

Parameters

data [array-like (1-dimensional)]

dtype [NumPy dtype (default: object)] If dtype is None, we find the dtype that best fits the data. If an actual dtype is provided, we coerce to that dtype if it's safe. Otherwise, an error will be raised.

copy [bool] Make a copy of input ndarray.

name [object] Name to be stored in the index.

tupleize_cols [bool (default: True)] When True, attempt to create a MultiIndex if possible.

See also:

RangeIndex Index implementing a monotonic integer range.
CategoricalIndex Index of *Categorical* s.
MultiIndex A multi-level, or hierarchical, Index.
IntervalIndex An Index of *Interval* s.
DatetimeIndex, ***TimedeltaIndex***, ***PeriodIndex***
Int64Index, ***UInt64Index***, ***Float64Index***

Notes

An Index instance can **only** contain hashable objects

Examples

```
>>> pd.Index([1, 2, 3])
Int64Index([1, 2, 3], dtype='int64')
```

```
>>> pd.Index(list('abc'))
Index(['a', 'b', 'c'], dtype='object')
```

Attributes

T	Return the transpose, which is by definition self.
array	The ExtensionArray of the data backing this Series or Index.
asi8	Integer representation of the values.
dtype	Return the dtype object of the underlying data.
hasnans	Return if I have any nans; enables various perf speedups.
inferred_type	Return a string of the type inferred from the values.
is_monotonic	Alias for is_monotonic_increasing.
is_monotonic_decreasing	Return if the index is monotonic decreasing (only equal or decreasing) values.
is_monotonic_increasing	Return if the index is monotonic increasing (only equal or increasing) values.
is_unique	Return if the index has unique values.
nbytes	Return the number of bytes in the underlying data.
ndim	Number of dimensions of the underlying data, by definition 1.
nlevels	Number of levels.
shape	Return a tuple of the shape of the underlying data.
size	Return the number of elements in the underlying data.
values	Return an array representing the data in the Index.

pandas.Index.T

property Index.**T**
Return the transpose, which is by definition self.

pandas.Index.array

Index.**array**
The ExtensionArray of the data backing this Series or Index.

New in version 0.24.0.

Returns

ExtensionArray An ExtensionArray of the values stored within. For extension types, this is the actual array. For NumPy native types, this is a thin (no copy) wrapper around `numpy.ndarray`.

`.array` differs `.values` which may require converting the data to a different form.

See also:

Index.to_numpy Similar method that always returns a NumPy array.

Series.to_numpy Similar method that always returns a NumPy array.

Notes

This table lays out the different array types for each extension dtype within pandas.

dtype	array type
category	Categorical
period	PeriodArray
interval	IntervalArray
IntegerNA	IntegerArray
string	StringArray
boolean	BooleanArray
datetime64[ns, tz]	DatetimeArray

For any 3rd-party extension types, the array type will be an ExtensionArray.

For all remaining dtypes `.array` will be a `arrays.NumpyExtensionArray` wrapping the actual ndarray stored within. If you absolutely need a NumPy array (possibly with copying / coercing data), then use *`Series.to_numpy()`* instead.

Examples

For regular NumPy types like int, and float, a PandasArray is returned.

```
>>> pd.Series([1, 2, 3]).array
<PandasArray>
[1, 2, 3]
Length: 3, dtype: int64
```

For extension types, like Categorical, the actual ExtensionArray is returned

```
>>> ser = pd.Series(pd.Categorical(['a', 'b', 'a']))
>>> ser.array
[a, b, a]
Categories (2, object): [a, b]
```

pandas.Index.asi8

property Index.**asi8**
Integer representation of the values.

Returns

ndarray An ndarray with int64 dtype.

pandas.Index.dtype

Index.**dtype**
Return the dtype object of the underlying data.

pandas.Index.hasnans

Index.**hasnans**
Return if I have any nans; enables various perf speedups.

pandas.Index.inferred_type

Index.**inferred_type**
Return a string of the type inferred from the values.

pandas.Index.is_monotonic

property Index.**is_monotonic**
Alias for is_monotonic_increasing.

pandas.Index.is_monotonic_decreasing

property Index.**is_monotonic_decreasing**
Return if the index is monotonic decreasing (only equal or decreasing) values.

Examples

```
>>> Index([3, 2, 1]).is_monotonic_decreasing
True
>>> Index([3, 2, 2]).is_monotonic_decreasing
True
>>> Index([3, 1, 2]).is_monotonic_decreasing
False
```

pandas.Index.is_monotonic_increasing

property Index.**is_monotonic_increasing**
Return if the index is monotonic increasing (only equal or increasing) values.

Examples

```
>>> Index([1, 2, 3]).is_monotonic_increasing
True
>>> Index([1, 2, 2]).is_monotonic_increasing
True
>>> Index([1, 3, 2]).is_monotonic_increasing
False
```

pandas.Index.is_unique

Index.**is_unique**
Return if the index has unique values.

pandas.Index.nbytes

property Index.**nbytes**
Return the number of bytes in the underlying data.

pandas.Index.ndim

property Index.**ndim**
Number of dimensions of the underlying data, by definition 1.

pandas.Index.nlevels

property Index.**nlevels**
Number of levels.

pandas.Index.shape

property Index.**shape**
Return a tuple of the shape of the underlying data.

pandas.Index.size

property Index.**size**
Return the number of elements in the underlying data.

pandas.Index.values

property Index.**values**
Return an array representing the data in the Index.

> **Warning:** We recommend using *Index.array* or *Index.to_numpy()*, depending on whether you need a reference to the underlying data or a NumPy array.

Returns

array: numpy.ndarray or ExtensionArray

See also:

Index.array Reference to the underlying data.

Index.to_numpy A NumPy array representing the underlying data.

empty	
has_duplicates	
is_all_dates	
name	
names	

Methods

all(self, *args, **kwargs)	Return whether all elements are True.
any(self, *args, **kwargs)	Return whether any element is True.
append(self, other)	Append a collection of Index options together.
argmax(self[, axis, skipna])	Return an ndarray of the maximum argument indexer.
argmin(self[, axis, skipna])	Return a ndarray of the minimum argument indexer.
argsort(self, *args, **kwargs)	Return the integer indices that would sort the index.
asof(self, label)	Return the label from the index, or, if not present, the previous one.
asof_locs(self, where, mask)	Find the locations (indices) of the labels from the index for every entry in the *where* argument.
astype(self, dtype[, copy])	Create an Index with values cast to dtypes.
copy(self[, name, deep, dtype])	Make a copy of this object.
delete(self, loc)	Make new Index with passed location(-s) deleted.
difference(self, other[, sort])	Return a new Index with elements from the index that are not in *other*.
drop(self, labels[, errors])	Make new Index with passed list of labels deleted.
drop_duplicates(self[, keep])	Return Index with duplicate values removed.
droplevel(self[, level])	Return index with requested level(s) removed.
dropna(self[, how])	Return Index without NA/NaN values.
duplicated(self[, keep])	Indicate duplicate index values.
equals(self, other)	Determine if two Index objects contain the same elements.
factorize(self[, sort, na_sentinel])	Encode the object as an enumerated type or categorical variable.
fillna(self[, value, downcast])	Fill NA/NaN values with the specified value.
format(self[, name, formatter])	Render a string representation of the Index.
get_indexer(self, target[, method, limit, ...])	Compute indexer and mask for new index given the current index.
get_indexer_for(self, target, **kwargs)	Guaranteed return of an indexer even when non-unique.
get_indexer_non_unique(self, target)	Compute indexer and mask for new index given the current index.
get_level_values(self, level)	Return an Index of values for requested level.
get_loc(self, key[, method, tolerance])	Get integer location, slice or boolean mask for requested label.
get_slice_bound(self, label, side, kind)	Calculate slice bound that corresponds to given label.
get_value(self, series, key)	Fast lookup of value from 1-dimensional ndarray.
groupby(self, values)	Group the index labels by a given array of values.
holds_integer(self)	Whether the type is an integer type.
identical(self, other)	Similar to equals, but check that other comparable attributes are also equal.
insert(self, loc, item)	Make new Index inserting new item at location.
intersection(self, other[, sort])	Form the intersection of two Index objects.
*is_*(self, other)	More flexible, faster check like `is` but that works through views.
is_categorical(self)	Check if the Index holds categorical data.
is_type_compatible(self, kind)	Whether the index type is compatible with the provided type.

continues on next page

Table 130 – continued from previous page

isin(self, values[, level])	Return a boolean array where the index values are in *values*.
isna(self)	Detect missing values.
isnull(self)	Detect missing values.
item(self)	Return the first element of the underlying data as a python scalar.
join(self, other[, how, level, ...])	Compute join_index and indexers to conform data structures to the new index.
map(self, mapper[, na_action])	Map values using input correspondence (a dict, Series, or function).
max(self[, axis, skipna])	Return the maximum value of the Index.
memory_usage(self[, deep])	Memory usage of the values.
min(self[, axis, skipna])	Return the minimum value of the Index.
notna(self)	Detect existing (non-missing) values.
notnull(self)	Detect existing (non-missing) values.
nunique(self[, dropna])	Return number of unique elements in the object.
putmask(self, mask, value)	Return a new Index of the values set with the mask.
ravel(self[, order])	Return an ndarray of the flattened values of the underlying data.
reindex(self, target[, method, level, ...])	Create index with target's values (move/add/delete values as necessary).
rename(self, name[, inplace])	Alter Index or MultiIndex name.
repeat(self, repeats[, axis])	Repeat elements of a Index.
searchsorted(self, value[, side, sorter])	Find indices where elements should be inserted to maintain order.
set_names(self, names[, level, inplace])	Set Index or MultiIndex name.
set_value(self, arr, key, value)	(DEPRECATED) Fast lookup of value from 1-dimensional ndarray.
shift(self[, periods, freq])	Shift index by desired number of time frequency increments.
slice_indexer(self[, start, end, step, kind])	For an ordered or unique index, compute the slice indexer for input labels and step.
slice_locs(self[, start, end, step, kind])	Compute slice locations for input labels.
sort(self, *args, **kwargs)	Use sort_values instead.
sort_values(self[, return_indexer, ascending])	Return a sorted copy of the index.
sortlevel(self[, level, ascending, ...])	For internal compatibility with with the Index API.
str	alias of `pandas.core.strings.StringMethods`
symmetric_difference(self, other[, ...])	Compute the symmetric difference of two Index objects.
take(self, indices[, axis, allow_fill, ...])	Return a new Index of the values selected by the indices.
to_flat_index(self)	Identity method.
to_frame(self[, index, name])	Create a DataFrame with a column containing the Index.
to_list(self)	Return a list of the values.
to_native_types(self[, slicer])	Format specified values of *self* and return them.
to_numpy(self[, dtype, copy, na_value])	A NumPy ndarray representing the values in this Series or Index.
to_series(self[, index, name])	Create a Series with both index and values equal to the index keys.

continues on next page

Table 130 – continued from previous page

tolist(self)	Return a list of the values.
transpose(self, *args, **kwargs)	Return the transpose, which is by definition self.
union(self, other[, sort])	Form the union of two Index objects.
unique(self[, level])	Return unique values in the index.
value_counts(self[, normalize, sort, ...])	Return a Series containing counts of unique values.
where(self, cond[, other])	Return an Index of same shape as self and whose corresponding entries are from self where cond is True and otherwise are from other.

pandas.Index.all

Index.**all**(*self*, **args*, ***kwargs*)

Return whether all elements are True.

Parameters

***args** These parameters will be passed to numpy.all.

****kwargs** These parameters will be passed to numpy.all.

Returns

all [bool or array_like (if axis is specified)] A single element array_like may be converted to bool.

See also:

Index.any Return whether any element in an Index is True.

Series.any Return whether any element in a Series is True.

Series.all Return whether all elements in a Series are True.

Notes

Not a Number (NaN), positive infinity and negative infinity evaluate to True because these are not equal to zero.

Examples

all

True, because nonzero integers are considered True.

```
>>> pd.Index([1, 2, 3]).all()
True
```

False, because `0` is considered False.

```
>>> pd.Index([0, 1, 2]).all()
False
```

any

True, because `1` is considered True.

```
>>> pd.Index([0, 0, 1]).any()
True
```

False, because `0` is considered False.

```
>>> pd.Index([0, 0, 0]).any()
False
```

pandas.Index.any

Index.**any**(*self*, **args*, ***kwargs*)

Return whether any element is True.

Parameters

***args** These parameters will be passed to numpy.any.

****kwargs** These parameters will be passed to numpy.any.

Returns

any [bool or array_like (if axis is specified)] A single element array_like may be converted to bool.

See also:

Index.all Return whether all elements are True.

Series.all Return whether all elements are True.

Notes

Not a Number (NaN), positive infinity and negative infinity evaluate to True because these are not equal to zero.

Examples

```
>>> index = pd.Index([0, 1, 2])
>>> index.any()
True
```

```
>>> index = pd.Index([0, 0, 0])
>>> index.any()
False
```

pandas.Index.append

Index.**append**(*self*, *other*)

Append a collection of Index options together.

Parameters

other [Index or list/tuple of indices]

Returns

appended [Index]

pandas.Index.argmax

Index.**argmax**(*self*, *axis=None*, *skipna=True*, **args*, ***kwargs*)

Return an ndarray of the maximum argument indexer.

Parameters

axis [{None}] Dummy argument for consistency with Series.

skipna [bool, default True]

Returns

numpy.ndarray Indices of the maximum values.

See also:

numpy.ndarray.argmax

pandas.Index.argmin

Index.**argmin**(*self*, *axis=None*, *skipna=True*, **args*, ***kwargs*)

Return a ndarray of the minimum argument indexer.

Parameters

axis [{None}] Dummy argument for consistency with Series.

skipna [bool, default True]

Returns

numpy.ndarray

See also:

numpy.ndarray.argmin

pandas.Index.argsort

Index.**argsort**(*self*, **args*, ***kwargs*)

Return the integer indices that would sort the index.

Parameters

***args** Passed to *numpy.ndarray.argsort*.

****kwargs** Passed to *numpy.ndarray.argsort*.

Returns

numpy.ndarray Integer indices that would sort the index if used as an indexer.

See also:

numpy.argsort Similar method for NumPy arrays.

Index.sort_values Return sorted copy of Index.

Examples

```
>>> idx = pd.Index(['b', 'a', 'd', 'c'])
>>> idx
Index(['b', 'a', 'd', 'c'], dtype='object')
```

```
>>> order = idx.argsort()
>>> order
array([1, 0, 3, 2])
```

```
>>> idx[order]
Index(['a', 'b', 'c', 'd'], dtype='object')
```

pandas.Index.asof

Index.**asof**(*self*, *label*)

Return the label from the index, or, if not present, the previous one.

Assuming that the index is sorted, return the passed index label if it is in the index, or return the previous index label if the passed one is not in the index.

Parameters

label [object] The label up to which the method returns the latest index label.

Returns

object The passed label if it is in the index. The previous label if the passed label is not in the sorted index or *NaN* if there is no such label.

See also:

Series.asof Return the latest value in a Series up to the passed index.

merge_asof Perform an asof merge (similar to left join but it matches on nearest key rather than equal key).

Index.get_loc An *asof* is a thin wrapper around *get_loc* with method='pad'.

Examples

Index.asof returns the latest index label up to the passed label.

```
>>> idx = pd.Index(['2013-12-31', '2014-01-02', '2014-01-03'])
>>> idx.asof('2014-01-01')
'2013-12-31'
```

If the label is in the index, the method returns the passed label.

```
>>> idx.asof('2014-01-02')
'2014-01-02'
```

If all of the labels in the index are later than the passed label, NaN is returned.

```
>>> idx.asof('1999-01-02')
nan
```

If the index is not sorted, an error is raised.

```
>>> idx_not_sorted = pd.Index(['2013-12-31', '2015-01-02',
...                            '2014-01-03'])
>>> idx_not_sorted.asof('2013-12-31')
Traceback (most recent call last):
ValueError: index must be monotonic increasing or decreasing
```

pandas.Index.asof_locs

Index.**asof_locs**(*self*, *where*, *mask*)

Find the locations (indices) of the labels from the index for every entry in the *where* argument.

As in the *asof* function, if the label (a particular entry in *where*) is not in the index, the latest index label up to the passed label is chosen and its index returned.

If all of the labels in the index are later than a label in *where*, -1 is returned.

mask is used to ignore NA values in the index during calculation.

Parameters

where [Index] An Index consisting of an array of timestamps.

mask [array-like] Array of booleans denoting where values in the original data are not NA.

Returns

numpy.ndarray An array of locations (indices) of the labels from the Index which correspond to the return values of the *asof* function for every element in *where*.

pandas.Index.astype

Index.**astype**(*self*, *dtype*, *copy=True*)

Create an Index with values cast to dtypes. The class of a new Index is determined by dtype. When conversion is impossible, a ValueError exception is raised.

Parameters

dtype [numpy dtype or pandas type] Note that any signed integer *dtype* is treated as `'int64'`, and any unsigned integer *dtype* is treated as `'uint64'`, regardless of the size.

copy [bool, default True] By default, astype always returns a newly allocated object. If copy is set to False and internal requirements on dtype are satisfied, the original data is used to create a new Index or the original Index is returned.

Returns

Index Index with values cast to specified dtype.

pandas.Index.copy

Index.**copy**(*self*, *name=None*, *deep=False*, *dtype=None*, ***kwargs*)

Make a copy of this object. Name and dtype sets those attributes on the new object.

Parameters

name [str, optional]

deep [bool, default False]

dtype [numpy dtype or pandas type]

Returns

copy [Index]

Notes

In most cases, there should be no functional difference from using `deep`, but if `deep` is passed it will attempt to deepcopy.

pandas.Index.delete

Index.**delete**(*self*, *loc*)

Make new Index with passed location(-s) deleted.

Returns

new_index [Index]

pandas.Index.difference

Index.**difference**(*self*, *other*, *sort=None*)

Return a new Index with elements from the index that are not in *other*.

This is the set difference of two Index objects.

Parameters

other [Index or array-like]

sort [False or None, default None] Whether to sort the resulting index. By default, the values are attempted to be sorted, but any TypeError from incomparable elements is caught by pandas.

- None : Attempt to sort the result, but catch any TypeErrors from comparing incomparable elements.
- False : Do not sort the result.

New in version 0.24.0.

Changed in version 0.24.1: Changed the default value from `True` to `None` (without change in behaviour).

Returns

difference [Index]

Examples

```
>>> idx1 = pd.Index([2, 1, 3, 4])
>>> idx2 = pd.Index([3, 4, 5, 6])
>>> idx1.difference(idx2)
Int64Index([1, 2], dtype='int64')
>>> idx1.difference(idx2, sort=False)
Int64Index([2, 1], dtype='int64')
```

pandas.Index.drop

Index.**drop**(*self*, *labels*, *errors='raise'*)

Make new Index with passed list of labels deleted.

Parameters

labels [array-like]

errors [{'ignore', 'raise'}, default 'raise'] If 'ignore', suppress error and existing labels are dropped.

Returns

dropped [Index]

Raises

KeyError If not all of the labels are found in the selected axis

pandas.Index.drop_duplicates

Index.**drop_duplicates**(*self*, *keep='first'*)

Return Index with duplicate values removed.

Parameters

keep [{'first', 'last', False}, default 'first']

- 'first' : Drop duplicates except for the first occurrence.
- 'last' : Drop duplicates except for the last occurrence.
- False : Drop all duplicates.

Returns

deduplicated [Index]

See also:

Series.drop_duplicates Equivalent method on Series.

DataFrame.drop_duplicates Equivalent method on DataFrame.

Index.duplicated Related method on Index, indicating duplicate Index values.

Examples

Generate an pandas.Index with duplicate values.

```
>>> idx = pd.Index(['lama', 'cow', 'lama', 'beetle', 'lama', 'hippo'])
```

The *keep* parameter controls which duplicate values are removed. The value 'first' keeps the first occurrence for each set of duplicated entries. The default value of keep is 'first'.

```
>>> idx.drop_duplicates(keep='first')
Index(['lama', 'cow', 'beetle', 'hippo'], dtype='object')
```

The value 'last' keeps the last occurrence for each set of duplicated entries.

```
>>> idx.drop_duplicates(keep='last')
Index(['cow', 'beetle', 'lama', 'hippo'], dtype='object')
```

The value False discards all sets of duplicated entries.

```
>>> idx.drop_duplicates(keep=False)
Index(['cow', 'beetle', 'hippo'], dtype='object')
```

pandas.Index.droplevel

Index.**droplevel**(*self*, *level=0*)

Return index with requested level(s) removed.

If resulting index has only 1 level left, the result will be of Index type, not MultiIndex.

New in version 0.23.1: (support for non-MultiIndex)

Parameters

level [int, str, or list-like, default 0] If a string is given, must be the name of a level If list-like, elements must be names or indexes of levels.

Returns

Index or MultiIndex

pandas.Index.dropna

Index.**dropna**(*self*, *how='any'*)

Return Index without NA/NaN values.

Parameters

how [{'any', 'all'}, default 'any'] If the Index is a MultiIndex, drop the value when any or all levels are NaN.

Returns

valid [Index]

pandas.Index.duplicated

Index.**duplicated**(*self*, *keep='first'*)

Indicate duplicate index values.

Duplicated values are indicated as `True` values in the resulting array. Either all duplicates, all except the first, or all except the last occurrence of duplicates can be indicated.

Parameters

keep [{'first', 'last', False}, default 'first'] The value or values in a set of duplicates to mark as missing.

- 'first' : Mark duplicates as `True` except for the first occurrence.
- 'last' : Mark duplicates as `True` except for the last occurrence.
- `False` : Mark all duplicates as `True`.

Returns

numpy.ndarray

See also:

Series.duplicated Equivalent method on pandas.Series.

DataFrame.duplicated Equivalent method on pandas.DataFrame.

Index.drop_duplicates Remove duplicate values from Index.

Examples

By default, for each set of duplicated values, the first occurrence is set to False and all others to True:

```
>>> idx = pd.Index(['lama', 'cow', 'lama', 'beetle', 'lama'])
>>> idx.duplicated()
array([False, False,  True, False,  True])
```

which is equivalent to

```
>>> idx.duplicated(keep='first')
array([False, False,  True, False,  True])
```

By using 'last', the last occurrence of each set of duplicated values is set on False and all others on True:

```
>>> idx.duplicated(keep='last')
array([ True, False,  True, False, False])
```

By setting keep on `False`, all duplicates are True:

```
>>> idx.duplicated(keep=False)
array([ True, False,  True, False,  True])
```

pandas.Index.equals

`Index.`**`equals`**(*self*, *other*) → bool

Determine if two Index objects contain the same elements.

Returns

bool True if "other" is an Index and it has the same elements as calling index; False otherwise.

pandas.Index.factorize

`Index.`**`factorize`**(*self*, *sort=False*, *na_sentinel=- 1*)

Encode the object as an enumerated type or categorical variable.

This method is useful for obtaining a numeric representation of an array when all that matters is identifying distinct values. *factorize* is available as both a top-level function *pandas.factorize()*, and as a method *Series.factorize()* and *Index.factorize()*.

Parameters

sort [bool, default False] Sort *uniques* and shuffle *codes* to maintain the relationship.

na_sentinel [int, default -1] Value to mark "not found".

Returns

codes [ndarray] An integer ndarray that's an indexer into *uniques*. `uniques.take(codes)` will have the same values as *values*.

uniques [ndarray, Index, or Categorical] The unique valid values. When *values* is Categorical, *uniques* is a Categorical. When *values* is some other pandas object, an *Index* is returned. Otherwise, a 1-D ndarray is returned.

Note: Even if there's a missing value in *values*, *uniques* will *not* contain an entry for it.

See also:

cut Discretize continuous-valued array.

unique Find the unique value in an array.

Examples

These examples all show factorize as a top-level method like `pd.factorize(values)`. The results are identical for methods like *`Series.factorize()`*.

```
>>> codes, uniques = pd.factorize(['b', 'b', 'a', 'c', 'b'])
>>> codes
array([0, 0, 1, 2, 0])
>>> uniques
array(['b', 'a', 'c'], dtype=object)
```

With `sort=True`, the *uniques* will be sorted, and *codes* will be shuffled so that the relationship is the maintained.

```
>>> codes, uniques = pd.factorize(['b', 'b', 'a', 'c', 'b'], sort=True)
>>> codes
array([1, 1, 0, 2, 1])
>>> uniques
array(['a', 'b', 'c'], dtype=object)
```

Missing values are indicated in *codes* with *na_sentinel* (`-1` by default). Note that missing values are never included in *uniques*.

```
>>> codes, uniques = pd.factorize(['b', None, 'a', 'c', 'b'])
>>> codes
array([ 0, -1,  1,  2,  0])
>>> uniques
array(['b', 'a', 'c'], dtype=object)
```

Thus far, we've only factorized lists (which are internally coerced to NumPy arrays). When factorizing pandas objects, the type of *uniques* will differ. For Categoricals, a *Categorical* is returned.

```
>>> cat = pd.Categorical(['a', 'a', 'c'], categories=['a', 'b', 'c'])
>>> codes, uniques = pd.factorize(cat)
>>> codes
array([0, 0, 1])
>>> uniques
[a, c]
Categories (3, object): [a, b, c]
```

Notice that `'b'` is in `uniques.categories`, despite not being present in `cat.values`.

For all other pandas objects, an Index of the appropriate type is returned.

```
>>> cat = pd.Series(['a', 'a', 'c'])
>>> codes, uniques = pd.factorize(cat)
```

(continues on next page)

(continued from previous page)

```
>>> codes
array([0, 0, 1])
>>> uniques
Index(['a', 'c'], dtype='object')
```

pandas.Index.fillna

Index.**fillna**(*self*, *value=None*, *downcast=None*)

Fill NA/NaN values with the specified value.

Parameters

value [scalar] Scalar value to use to fill holes (e.g. 0). This value cannot be a list-likes.

downcast [dict, default is None] a dict of item->dtype of what to downcast if possible, or the string 'infer' which will try to downcast to an appropriate equal type (e.g. float64 to int64 if possible).

Returns

filled [Index]

pandas.Index.format

Index.**format**(*self*, *name=False*, *formatter=None*, ***kwargs*)

Render a string representation of the Index.

pandas.Index.get_indexer

Index.**get_indexer**(*self*, *target*, *method=None*, *limit=None*, *tolerance=None*)

Compute indexer and mask for new index given the current index. The indexer should be then used as an input to ndarray.take to align the current data to the new index.

Parameters

target [Index]

method [{None, 'pad'/'ffill', 'backfill'/'bfill', 'nearest'}, optional]

- default: exact matches only.
- pad / ffill: find the PREVIOUS index value if no exact match.
- backfill / bfill: use NEXT index value if no exact match
- nearest: use the NEAREST index value if no exact match. Tied distances are broken by preferring the larger index value.

limit [int, optional] Maximum number of consecutive labels in `target` to match for inexact matches.

tolerance [optional] Maximum distance between original and new labels for inexact matches. The values of the index at the matching locations most satisfy the equation `abs(index[indexer] - target) <= tolerance`.

Tolerance may be a scalar value, which applies the same tolerance to all values, or list-like, which applies variable tolerance per element. List-like includes list, tuple,

array, Series, and must be the same size as the index and its dtype must exactly match the index's type.

New in version 0.21.0: (list-like tolerance)

Returns

indexer [ndarray of int] Integers from 0 to n - 1 indicating that the index at these positions matches the corresponding target values. Missing values in the target are marked by -1.

Examples

```
>>> index = pd.Index(['c', 'a', 'b'])
>>> index.get_indexer(['a', 'b', 'x'])
array([ 1,  2, -1])
```

Notice that the return value is an array of locations in `index` and `x` is marked by -1, as it is not in `index`.

pandas.Index.get_indexer_for

`Index.`**`get_indexer_for`**(*self*, *target*, ***kwargs*)

Guaranteed return of an indexer even when non-unique.

This dispatches to get_indexer or get_indexer_non_unique as appropriate.

Returns

numpy.ndarray List of indices.

pandas.Index.get_indexer_non_unique

`Index.`**`get_indexer_non_unique`**(*self*, *target*)

Compute indexer and mask for new index given the current index. The indexer should be then used as an input to ndarray.take to align the current data to the new index.

Parameters

target [Index]

Returns

indexer [ndarray of int] Integers from 0 to n - 1 indicating that the index at these positions matches the corresponding target values. Missing values in the target are marked by -1.

missing [ndarray of int] An indexer into the target of the values not found. These correspond to the -1 in the indexer array.

pandas.Index.get_level_values

Index.**get_level_values**(*self*, *level*)

Return an Index of values for requested level.

This is primarily useful to get an individual level of values from a MultiIndex, but is provided on Index as well for compatibility.

Parameters

level [int or str] It is either the integer position or the name of the level.

Returns

Index Calling object, as there is only one level in the Index.

See also:

MultiIndex.get_level_values Get values for a level of a MultiIndex.

Notes

For Index, level should be 0, since there are no multiple levels.

Examples

```
>>> idx = pd.Index(list('abc'))
>>> idx
Index(['a', 'b', 'c'], dtype='object')
```

Get level values by supplying *level* as integer:

```
>>> idx.get_level_values(0)
Index(['a', 'b', 'c'], dtype='object')
```

pandas.Index.get_loc

Index.**get_loc**(*self*, *key*, *method=None*, *tolerance=None*)

Get integer location, slice or boolean mask for requested label.

Parameters

key [label]

method [{None, 'pad'/'ffill', 'backfill'/'bfill', 'nearest'}, optional]

- default: exact matches only.
- pad / ffill: find the PREVIOUS index value if no exact match.
- backfill / bfill: use NEXT index value if no exact match
- nearest: use the NEAREST index value if no exact match. Tied distances are broken by preferring the larger index value.

tolerance [int or float, optional] Maximum distance from index value for inexact matches. The value of the index at the matching location most satisfy the equation `abs(index[loc] - key) <= tolerance`.

New in version 0.21.0: (list-like tolerance)

Returns

loc [int if unique index, slice if monotonic index, else mask]

Examples

```
>>> unique_index = pd.Index(list('abc'))
>>> unique_index.get_loc('b')
1
```

```
>>> monotonic_index = pd.Index(list('abbc'))
>>> monotonic_index.get_loc('b')
slice(1, 3, None)
```

```
>>> non_monotonic_index = pd.Index(list('abcb'))
>>> non_monotonic_index.get_loc('b')
array([False,  True, False,  True], dtype=bool)
```

pandas.Index.get_slice_bound

`Index.`**`get_slice_bound`**(*self*, *label*, *side*, *kind*)

Calculate slice bound that corresponds to given label.

Returns leftmost (one-past-the-rightmost if `side=='right'`) position of given label.

Parameters

label [object]

side [{'left', 'right'}]

kind [{'ix', 'loc', 'getitem'}]

Returns

int Index of label.

pandas.Index.get_value

`Index.`**`get_value`**(*self*, *series*, *key*)

Fast lookup of value from 1-dimensional ndarray. Only use this if you know what you're doing.

Returns

scalar A value in the Series with the index of the key value in self.