natsort Documentation

Release 8.3.1

Seth M. Morton

Mar 02, 2023

Contents

- Source Code: https://github.com/SethMMorton/natsort
- Downloads: https://pypi.org/project/natsort/
- Documentation: https://natsort.readthedocs.io/

Please see the GitHub main page for everything else, including

- Quick description
- Basic examples
- FAQ
- Requirements and optional dependencies
- Installation instructions
- Testing instructions
- Deprecation schedule

CHAPTER 1

How Does Natsort Work?

This page has been moved to the natsort wiki.

1.1 Special Cases Everywhere!

This page has been moved to the natsort wiki.

CHAPTER 2

Examples and Recipes

This page has been moved to the natsort wiki.

CHAPTER 3

natsort API

3.1 Standard API

3.1.1 natsorted()

natsort.**natsorted**(*seq: Iterable[T], key: Optional[Callable[[T], Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT, None]]] = None, reverse: bool = False, alg: Union[natsort.ns_enum.ns, int] = <ns.DEFAULT: 0>*) → List[T]

Sorts an iterable naturally.

Parameters

- **seq** (*iterable*) – The input to sort.
- **key** (*callable, optional*) – A key used to determine how to sort each element of the iterable. It is **not** applied recursively. It should accept a single argument and return a single value.
- **reverse** (*{True, False}, optional*) – Return the list in reversed sorted order. The default is *False*.
- **alg** (*ns enum, optional*) – This option is used to control which algorithm *natsort* uses when sorting. For details into these options, please see the *ns* class documentation. The default is *ns.INT*.

Returns **out** – The sorted input.

Return type list

See also:

natsort_keygen() Generates the key that makes natural sorting possible.

realsorted() A wrapper for `natsorted(seq, alg=ns.REAL)`.

humansorted() A wrapper for `natsorted(seq, alg=ns.LOCALE)`.

index_natsorted() Returns the sorted indexes from *natsorted*.

os_sorted() Sort according to your operating system's rules.

Examples

Use *natsorted* just like the builtin *sorted*:

```
>>> a = ['num3', 'num5', 'num2']
>>> natsorted(a)
['num2', 'num3', 'num5']
```

3.1.2 The ns enum

natsort.**ns**

Enum to control the *natsort* algorithm.

This class acts like an enum to control the *natsort* algorithm. The user may select several options simultaneously by or'ing the options together. For example, to choose `ns.INT`, `ns.PATH`, and `ns.LOCALE`, you could do `ns.INT | ns.LOCALE | ns.PATH`. Each function in the `natsort` package has an *alg* option that accepts this enum to allow fine control over how your input is sorted.

Each option has a shortened 1- or 2-letter form.

Note: Please read *Possible Issues with humansorted() or ns.LOCALE* before using `ns.LOCALE`.

INT, I (default)
The default - parse numbers as integers.

FLOAT, F
Tell *natsort* to parse numbers as floats.

UNSIGNED, U (default)
Tell *natsort* to ignore any sign (i.e. "-" or "+") to the immediate left of a number. This is the default.

SIGNED, S
Tell *natsort* to take into account any sign (i.e. "-" or "+") to the immediate left of a number.

REAL, R
This is a shortcut for `ns.FLOAT | ns.SIGNED`, which is useful when attempting to sort real numbers.

NOEXP, N
Tell *natsort* to not search for exponents as part of a float number. For example, with *NOEXP* the number "5.6E5" would be interpreted as *5.6*, *"E"*, and *5* instead of *560000*.

NUMAFTER, NA
Tell *natsort* to sort numbers after non-numbers. By default numbers will be ordered before non-numbers.

PATH, P
Tell *natsort* to interpret strings as filesystem paths, so they will be split according to the filesystem separator (i.e. '/' on UNIX, '' on Windows), as well as splitting on the file extension, if any. Without this, lists of file paths like `['Folder/', 'Folder (1)/', 'Folder (10)/']` will not be sorted properly; 'Folder/' will be placed at the end, not at the front. It is the same as setting the old *as_path* option to *True*.

COMPATIBILITYNORMALIZE, CN
Use the "NFKD" unicode normalization form on input rather than the default "NFD". This will transform characters such as '' into '7'. Please see https://stackoverflow.com/a/7934397/1399279, https://stackoverflow.com/a/7931547/1399279, and https://unicode.org/reports/tr15/ for full details into unicode normalization.

LOCALE, L
Tell *natsort* to be locale-aware when sorting. This includes both proper sorting of alphabetical characters as well as proper handling of locale-dependent decimal separators and thousands separators. This is a shortcut for `ns.LOCALEALPHA | ns.LOCALENUM`. Your sorting results will vary depending on your current locale.

LOCALEALPHA, LA
Tell *natsort* to be locale-aware when sorting, but only for alphabetical characters.

LOCALENUM, LN
Tell *natsort* to be locale-aware when sorting, but only for decimal separators and thousands separators.

IGNORECASE, IC
Tell *natsort* to ignore case when sorting. For example, `['Banana', 'apple', 'banana', 'Apple']` would be sorted as `['apple', 'Apple', 'Banana', 'banana']`.

LOWERCASEFIRST, LF
Tell *natsort* to put lowercase letters before uppercase letters when sorting. For example, `['Banana', 'apple', 'banana', 'Apple']` would be sorted as `['apple', 'banana', 'Apple', 'Banana']` (the default order would be `['Apple', 'Banana', 'apple', 'banana']` which is the order from a purely ordinal sort). Useless when used with *IGNORECASE*. Please note that if used

with `LOCALE`, this actually has the reverse effect and will put uppercase first (this is because `LOCALE` already puts lowercase first); you may use this to your advantage if you need to modify the order returned with `LOCALE`.

GROUPLETTERS, G
Tell *natsort* to group lowercase and uppercase letters together when sorting. For example, `['Banana', 'apple', 'banana', 'Apple']` would be sorted as `['Apple', 'apple', 'Banana', 'banana']`. Useless when used with *IGNORECASE*; use with *LOWERCASEFIRST* to reverse the order of upper and lower case. Generally not needed with *LOCALE*.

CAPITALFIRST, C
Only used when *LOCALE* is enabled. Tell *natsort* to put all capitalized words before non-capitalized words. This is essentially the inverse of *GROUPLETTERS*, and is the default Python sorting behavior without *LOCALE*.

UNGROUPLETTERS, UG
An alias for *CAPITALFIRST*.

NANLAST, NL
If an NaN shows up in the input, this instructs *natsort* to treat these as +Infinity and place them after all the other numbers. By default, an NaN be treated as -Infinity and be placed first. Note that this `None` is treated like NaN internally.

PRESORT, PS
Sort the input as strings before sorting with the *nasort* algorithm. This can help eliminate inconsistent sorting in cases where two different strings represent the same number. For example, "a1" and "a01" both are internally represented as ("a", "1), so without *PRESORT* the order of these two values would depend on the order they appeared in the input (because Python's *sorted* is a stable sorting algorithm).

Notes

If you prefer to use *import natsort as ns* as opposed to *from natsort import natsorted, ns*, the *ns* options are available as top-level imports.

```
>>> import natsort as ns
>>> a = ['num5.10', 'num-3', 'num5.3', 'num2']
>>> ns.natsorted(a, alg=ns.REAL) == ns.natsorted(a, alg=ns.ns.REAL)
True
```

3.1.3 natsort_key()

natsort.**natsort_key**(*val*)
The default natural sorting key.

This is the output of *natsort_keygen()* with default values.

See also:

natsort_keygen()

3.1.4 natsort_keygen()

natsort.**natsort_keygen**(*key: Optional[Callable[[Any], Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT, None]]] = None, alg: Union[natsort.ns_enum.ns, int] = <ns.DEFAULT: 0>*) → Callable[[Any], Tuple[Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT], ...]]

Generate a key to sort strings and numbers naturally.

This key is designed for use as the *key* argument to functions such as the *sorted* builtin.

The user may customize the generated function with the arguments to *natsort_keygen*, including an optional *key* function.

Parameters

- **key** (*callable, optional*) – A key used to manipulate the input value before parsing for numbers. It is **not** applied recursively. It should accept a single argument and return a single value.
- **alg** (*ns enum, optional*) – This option is used to control which algorithm *natsort* uses when sorting. For details into these options, please see the *ns* class documentation. The default is *ns.INT*.

Returns **out** – A function that parses input for natural sorting that is suitable for passing as the *key* argument to functions such as *sorted*.

Return type function

See also:

natsorted(), *natsort_key()*

Examples

natsort_keygen is a convenient way to create a custom key to sort lists in-place (for example).:

```
>>> a = ['num5.10', 'num-3', 'num5.3', 'num2']
>>> a.sort(key=natsort_keygen(alg=ns.REAL))
>>> a
['num-3', 'num2', 'num5.10', 'num5.3']
```

3.1.5 os_sort_key()

natsort.**os_sort_key**(*val*)

The default key to replicate your file browser's sort order

This is the output of *os_sort_keygen()* with default values.

See also:

os_sort_keygen()

3.1.6 os_sort_keygen()

natsort.**os_sort_keygen**(*key: Optional[Callable[[Any], Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT, None]]] = None*) → Callable[[Any], Tuple[Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT], ...]]

Generate a sorting key to replicate your file browser's sort order

See *os_sorted()* for description and caveats.

Returns **out** – A function that parses input for OS path sorting that is suitable for passing as the *key* argument to functions such as *sorted*.

Return type function

See also:

os_sort_key(), *os_sorted()*

Notes

On Windows, this will implicitly coerce all inputs to str before collating.

3.2 Convenience Functions

3.2.1 os_sorted()

natsort.**os_sorted**(*seq: Iterable[T], key: Optional[Callable[[T], Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT, None]]] = None, reverse: bool = False, presort: bool = False*) → List[T]

Sort elements in the same order as your operating system's file browser

> **Warning:** The resulting function will generate results that will be different depending on your platform. This is intentional.

On Windows, this will sort with the same order as Windows Explorer.

On MacOS/Linux, you will get different results depending on whether or not you have `pyicu` installed.

- If you have `pyicu` installed, you will get results that are the same as (or very close to) the same order as your operating system's file browser.
- If you do not have `pyicu` installed, then this will give the same results as if you used `ns.LOCALE`, `ns.PATH`, and `ns.IGNORECASE` with *natsorted()*. If you do not have special characters this will give correct results, but once special characters are added you should lower your expectations.

It is *strongly* recommended to have `pyicu` installed on MacOS/Linux if you want correct sort results.

It does *not* take into account if a path is a directory or a file when sorting.

Parameters

- **seq** (*iterable*) – The input to sort. Each element must be of type str.
- **key** (*callable, optional*) – A key used to determine how to sort each element of the sequence. It should accept a single argument and return a single value.

- **reverse** (*{{True, False}, optional*) – Return the list in reversed sorted order. The default is *False*.
- **presort** (*{{True, False}, optional*) – Equivalent to adding `ns.PRESORT`, see *ns* for documentation. The default is *False*.

Returns **out** – The sorted input.

Return type list

See also:

natsorted(), *os_sort_keygen()*

Notes

This will implicitly coerce all inputs to str before collating.

3.2.2 realsorted()

natsort.**realsorted**(*seq: Iterable[T], key: Optional[Callable[[T], Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT, None]]] = None, reverse: bool = False, alg: Union[natsort.ns_enum.ns, int] = <ns.DEFAULT: 0>*) → List[T]

Convenience function to properly sort signed floats.

A signed float in a string could be "a-5.7". This is a wrapper around `natsorted(seq, alg=ns.REAL)`.

The behavior of *realsorted()* for *natsort* version >= 4.0.0 was the default behavior of *natsorted()* for *natsort* version < 4.0.0.

Parameters

- **seq** (*iterable*) – The input to sort.
- **key** (*callable, optional*) – A key used to determine how to sort each element of the sequence. It is **not** applied recursively. It should accept a single argument and return a single value.
- **reverse** (*{{True, False}, optional*) – Return the list in reversed sorted order. The default is *False*.
- **alg** (*ns enum, optional*) – This option is used to control which algorithm *natsort* uses when sorting. For details into these options, please see the *ns* class documentation. The default is *ns.REAL*.

Returns **out** – The sorted input.

Return type list

See also:

index_realsorted() Returns the sorted indexes from *realsorted*.

Examples

Use *realsorted* just like the builtin *sorted*:

```
>>> a = ['num5.10', 'num-3', 'num5.3', 'num2']
>>> natsorted(a)
['num2', 'num5.3', 'num5.10', 'num-3']
>>> realsorted(a)
['num-3', 'num2', 'num5.10', 'num5.3']
```

3.2.3 humansorted()

natsort.**humansorted**(*seq: Iterable[T], key: Optional[Callable[[T], Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT, None]]] = None, reverse: bool = False, alg: Union[natsort.ns_enum.ns, int] = <ns.DEFAULT: 0>*) → List[T]

Convenience function to properly sort non-numeric characters.

This is a wrapper around `natsorted(seq, alg=ns.LOCALE)`.

Parameters

- **seq** (*iterable*) – The input to sort.
- **key** (*callable, optional*) – A key used to determine how to sort each element of the sequence. It is **not** applied recursively. It should accept a single argument and return a single value.
- **reverse** (*{{True, False}}, optional*) – Return the list in reversed sorted order. The default is *False*.
- **alg** (*ns enum, optional*) – This option is used to control which algorithm *natsort* uses when sorting. For details into these options, please see the *ns* class documentation. The default is *ns.LOCALE*.

Returns **out** – The sorted input.

Return type list

See also:

index_humansorted() Returns the sorted indexes from *humansorted*.

Notes

Please read *Possible Issues with humansorted() or ns.LOCALE* before using *humansorted*.

Examples

Use *humansorted* just like the builtin *sorted*:

```
>>> a = ['Apple', 'Banana', 'apple', 'banana']
>>> natsorted(a)
['Apple', 'Banana', 'apple', 'banana']
>>> humansorted(a)
['apple', 'Apple', 'banana', 'Banana']
```

3.2.4 index_natsorted()

natsort.**index_natsorted**(*seq: Iterable[T], key: Optional[Callable[[T], Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT, None]]] = None, reverse: bool = False, alg: Union[natsort.ns_enum.ns, int] = <ns.DEFAULT: 0>*) → List[int]

Determine the list of the indexes used to sort the input sequence.

Sorts a sequence naturally, but returns a list of sorted the indexes and not the sorted list itself. This list of indexes can be used to sort multiple lists by the sorted order of the given sequence.

Parameters

- **seq** (*iterable*) – The input to sort.
- **key** (*callable, optional*) – A key used to determine how to sort each element of the sequence. It is **not** applied recursively. It should accept a single argument and return a single value.
- **reverse** (*{{True, False}}, optional*) – Return the list in reversed sorted order. The default is *False*.
- **alg** (*ns enum, optional*) – This option is used to control which algorithm *natsort* uses when sorting. For details into these options, please see the *ns* class documentation. The default is *ns.INT*.

Returns **out** – The ordered indexes of the input.

Return type tuple

See also:

natsorted(), *order_by_index()*

Examples

Use index_natsorted if you want to sort multiple lists by the sorted order of one list:

```
>>> a = ['num3', 'num5', 'num2']
>>> b = ['foo', 'bar', 'baz']
>>> index = index_natsorted(a)
>>> index
[2, 0, 1]
>>> # Sort both lists by the sort order of a
>>> order_by_index(a, index)
['num2', 'num3', 'num5']
>>> order_by_index(b, index)
['baz', 'foo', 'bar']
```

3.2.5 index_realsorted()

natsort.**index_realsorted**(*seq: Iterable[T], key: Optional[Callable[[T], Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT, None]]] = None, reverse: bool = False, alg: Union[natsort.ns_enum.ns, int] = <ns.DEFAULT: 0>*) → List[int]

This is a wrapper around `index_natsorted(seq, alg=ns.REAL)`.

Parameters

- **seq** (*iterable*) – The input to sort.
- **key** (*callable, optional*) – A key used to determine how to sort each element of the sequence. It is **not** applied recursively. It should accept a single argument and return a single value.
- **reverse** (*{True, False}, optional*) – Return the list in reversed sorted order. The default is *False*.
- **alg** (*ns enum, optional*) – This option is used to control which algorithm *natsort* uses when sorting. For details into these options, please see the *ns* class documentation. The default is *ns.REAL*.

Returns **out** – The ordered indexes of the input.

Return type tuple

See also:

realsorted(), *order_by_index()*

Examples

Use *index_realsorted* just like the builtin *sorted*:

```
>>> a = ['num5.10', 'num-3', 'num5.3', 'num2']
>>> index_realsorted(a)
[1, 3, 0, 2]
```

3.2.6 index_humansorted()

natsort.**index_humansorted**(*seq: Iterable[T], key: Optional[Callable[[T], Union[natsort.utils.SupportsDunderLT, natsort.utils.SupportsDunderGT, None]]] = None, reverse: bool = False, alg: Union[natsort.ns_enum.ns, int] = <ns.DEFAULT: 0>*) → List[int]

This is a wrapper around `index_natsorted(seq, alg=ns.LOCALE)`.

Parameters

- **seq** (*iterable*) – The input to sort.
- **key** (*callable, optional*) – A key used to determine how to sort each element of the sequence. It is **not** applied recursively. It should accept a single argument and return a single value.
- **reverse** (*{True, False}, optional*) – Return the list in reversed sorted order. The default is *False*.
- **alg** (*ns enum, optional*) – This option is used to control which algorithm *natsort* uses when sorting. For details into these options, please see the *ns* class documentation. The default is *ns.LOCALE*.

Returns **out** – The ordered indexes of the input.

Return type tuple

See also:

humansorted(), *order_by_index()*

Notes

Please read *Possible Issues with humansorted() or ns.LOCALE* before using *humansorted*.

Examples

Use *index_humansorted* just like the builtin *sorted*:

```
>>> a = ['Apple', 'Banana', 'apple', 'banana']
>>> index_humansorted(a)
[2, 0, 3, 1]
```

3.2.7 order_by_index()

natsort.**order_by_index**(*seq: Sequence[Any], index: Iterable[int], iter: bool = False*) → Iterable[Any]

Order a given sequence by an index sequence.

The output of *index_natsorted* is a sequence of integers (index) that correspond to how its input sequence **would** be sorted. The idea is that this index can be used to reorder multiple sequences by the sorted order of the first sequence. This function is a convenient wrapper to apply this ordering to a sequence.

Parameters

- **seq** (*sequence*) – The sequence to order.
- **index** (*iterable*) – The iterable that indicates how to order *seq*. It should be the same length as *seq* and consist of integers only.
- **iter** (*{{True, False}}, optional*) – If *True*, the ordered sequence is returned as a iterator; otherwise it is returned as a list. The default is *False*.

Returns **out** – The sequence ordered by *index*, as a *list* or as an iterator (depending on the value of *iter*).

Return type {{list, iterator}}

See also:

index_natsorted(), *index_humansorted()*, *index_realsorted()*

Examples

order_by_index is a convenience function that helps you apply the result of *index_natsorted*:

```
>>> a = ['num3', 'num5', 'num2']
>>> b = ['foo', 'bar', 'baz']
>>> index = index_natsorted(a)
>>> index
[2, 0, 1]
>>> # Sort both lists by the sort order of a
>>> order_by_index(a, index)
['num2', 'num3', 'num5']
>>> order_by_index(b, index)
['baz', 'foo', 'bar']
```

3.2.8 Help With Bytes

The official stance of `natsort` is to not support *bytes* for sorting; there is just too much that can go wrong when trying to automate conversion between *bytes* and *str*. But rather than completely give up on *bytes*, `natsort` provides three functions that make it easy to quickly decode *bytes* to *str* so that sorting is possible.

`natsort.`**`decoder`**(*encoding: str*) → Callable[[Any], Any]
Return a function that can be used to decode bytes to unicode.

Parameters **`encoding`** (*str*) – The codec to use for decoding. This must be a valid unicode codec.

Returns A function that takes a single argument and attempts to decode it using the supplied codec. Any *UnicodeErrors* are raised. If the argument was not of *bytes* type, it is simply returned as-is.

Return type decode_function

See also:

as_ascii(), *as_utf8()*

Examples

```
>>> f = decoder('utf8')
>>> f(b'bytes') == 'bytes'
True
>>> f(12345) == 12345
True
>>> # On Python 3, without decoder this would return [b'a10', b'a2']
>>> natsorted([b'a10', b'a2'], key=decoder('utf8')) == [b'a2', b'a10']
True
>>> # On Python 3, without decoder this would raise a TypeError.
>>> natsorted([b'a10', 'a2'], key=decoder('utf8')) == ['a2', b'a10']
True
```

`natsort.`**`as_ascii`**(*s: Any*) → Any
Function to decode an input with the ASCII codec, or return as-is.

Parameters **`s`** (*object*) –

Returns If the input was of type *bytes*, the return value is a *str* decoded with the ASCII codec. Otherwise, the return value is identically the input.

Return type output

See also:

decoder()

`natsort.`**`as_utf8`**(*s: Any*) → Any
Function to decode an input with the UTF-8 codec, or return as-is.

Parameters **`s`** (*object*) –

Returns If the input was of type *bytes*, the return value is a *str* decoded with the UTF-8 codec. Otherwise, the return value is identically the input.

Return type output

See also:

decoder()

3.2.9 Help With Creating Function Keys

If you need to create a complicated *key* argument to (for example) *natsorted()* that is actually multiple functions called one after the other, the following function can help you easily perform this action. It is used internally to natsort, and has been exposed publicly for the convenience of the user.

natsort.**chain_functions**(*functions: Iterable[Callable[[Any], Any]]*) → Callable[[Any], Any]
Chain a list of single-argument functions together and return.

The functions are applied in list order, and the output of the previous functions is passed to the next function.

Parameters **functions** (*list*) – A list of single-argument functions to chain together.

Returns **func** – A single argument function.

Return type callable

Examples

Chain several functions together!

```
>>> funcs = [lambda x: x * 4, len, lambda x: x + 5]
>>> func = chain_functions(funcs)
>>> func('hey')
17
```

If you need to be able to search your input for numbers using the same definition as natsort, you can do so using the following function. Given your chosen algorithm (selected using the *ns* enum), the corresponding regular expression to locate numbers will be returned.

natsort.**numeric_regex_chooser**(*alg: Union[natsort.ns_enum.ns, int]*) → str
Select an appropriate regex for the type of number of interest.

Parameters **alg** (*ns enum*) – Used to indicate the regular expression to select.

Returns **regex** – Regular expression string that matches the desired number type.

Return type str

3.2.10 Help With Type Hinting

If you need to explicitly specify the types that natsort accepts or returns in your code, the following types have been exposed for your convenience.

Type	Purpose
natsort.NatsortKeyType	Returned by *natsort.natsort_keygen()*, and type of *natsort.natsort_key*
natsort.OSSortKeyType	Returned by *natsort.os_sort_keygen()*, and type of *natsort.os_sort_key*
natsort.KeyType	Type of *key* argument to *natsort.natsorted()* and *natsort.natsort_keygen()*
natsort.NatsortInType	The input type of natsort.NatsortKeyType
natsort.NatsortOutType	The output type of natsort.NatsortKeyType
natsort.NSType	The type of the *ns* enum

CHAPTER 4

Possible Issues with `humansorted()` or `ns.LOCALE`

This page has been moved to the natsort wiki.

CHAPTER 5

Shell Script

This page has been moved to the natsort wiki.

CHAPTER 6

Changelog

6.1 Unreleased

6.2 8.3.1 - 2023-03-01

6.2.1 Fixed

- Broken test on FreeBSD due to a broken `locale.strxfrm`. **This change has no effect outside fixing tests** (Issue #161)

6.3 8.3.0 - 2023-02-27

6.3.1 Added

- The `PRESORT` option to the `ns` enum to attain consistent sort order in certain corner cases (Issue #149)
- Logic to ensure `None` and NaN are sorted in a consistent order (Issue #149)
- Explict Python 3.11 support

6.3.2 Changed

- Only convert to `str` if necessary in `os_sorted` (@Dobatymo, issues #157 and #158)
- Attempt to use new `fastnumbers` functionality if available
- Move non-API documentation to the GitHub wiki

6.3.3 Removed

- Support for EOL Python 3.6

6.4 8.2.0 - 2022-09-01

6.4.1 Changed

- Auto-coerce `pathlib.Path` objects to `str` since it is the least astonishing behavior (@Gilthans, issues #152, #153)
- Reduce strictness of type hints to avoid over-constraining client code (issues #154, #155)

6.5 8.1.0 - 2022-01-30

6.5.1 Changed

- When using `ns.PATH`, only split off a maximum of two suffixes from a file name (issues #145, #146).

6.6 8.0.2 - 2021-12-14

6.6.1 Fixed

- Bug where sorting paths fail if one of the paths is '.' (issues #142, #143)

6.7 8.0.1 - 2021-12-10

6.7.1 Fixed

- Compose unicode characters when using locale to ensure sorting is correct across all locales (issues #140, #141)

6.8 8.0.0 - 2021-11-03

- Re-release 7.2.0 as 8.0.0 because introduction of type hints can break CI builds (issue #139)

6.9 7.2.0 - 2021-11-02 (Yanked)

6.9.1 Added

- Type hints (contributions from @thethiny and @domdfcoding, issues #132, #135, and #138)
- Explicit testing for Python 3.10

6.9.2 Removed

- Support for Python 3.4 and Python 3.5

6.10 7.1.1 - 2021-01-24

6.10.1 Changed

- Use GitHub Actions instead of Travis-CI (issue #125)
- No longer pin testing dependencies (issue #126)

6.10.2 Fixed

- Correct a minor typo (@madphysicist, issue #127)

6.11 7.1.0 - 2020-11-19

6.11.1 Added

- `os_sorted`, `os_sort_keygen`, and `os_sort_key` to better support sorting like the file browser on the current operating system - this closes the long-standing issue #41
- Support for Python 3.9 (@swt2c, issue #119)

6.11.2 Changed

- MacOS unit tests run on native Python
- Treat `None` like `NaN` internally to avoid `TypeError` (issue #117)
- No longer fail tests every time a new Python version is released (issue #122)

6.11.3 Fixed

- Various typos, missing figures, and out-of-date information in the "How it works"
- Fix typo in CHANGELOG (@graingert, issue #113)
- Updated "How it works" to account for Pandas updates (@kuraga, issue #116)

6.12 7.0.1 - 2020-01-27

6.12.1 Fixed

- Bug where that caused incorrect sorting when using locales that have a `"."` character as the thousands separator.

6.13 7.0.0 - 2020-01-08

6.13.1 Added

- Ability to deploy directly from TravisCI (@hugovk, issue #106)
- Release checklist in `RELEASING.md` (@hugovk, issue #106)

6.13.2 Changed

- Updated auxiliary shell scripts to be written in python, and added ability to call these from `tox`
- Improved Travis-CI experience
- Update testing dependency versions

6.13.3 Removed

- Support for Python 2

6.14 6.2.0 - 2019-11-13

6.14.1 Added

- Support for Python 3.8 (@hugovk, issue #104)

6.14.2 Changed

- `index_natsorted` internally now uses tuples for index-element pairs instead of lists
- Added a TOC to the README
- Python 3.4 is no longer included in testing

6.14.3 Fixed

- Pin testing dependencies to prevent CI breaking due to third-party library changes

6.14.4 Removed

- Introduction page in documentation

6.15 6.1.0 - 2019-11-09

6.15.1 Added

- Expose `numeric_regex_chooser` as a public function for ease in making key functions

- Example in the documentation on how to sort numbers with units
- Automated testing support for macos and Windows (issue #91)

6.15.2 Changed

- Update CHANGELOG format to style from https://keepachangelog.com/ (issue #92)

6.15.3 Fixed

- Removed dependency on `sudo` in TravisCI configuration (@hugovk, issue #99)
- Documentation typos (@jdufresne, issue #94) (@cpburnz, issue #95)

6.16 6.0.0 - 2019-02-04

6.16.1 Changed

- Simply Travis-CI configuration (@jdufresne, issue #88)

6.16.2 Fixed

- Fix README rendering in PyPI (@altendky, issue #89)

6.16.3 Removed

- Drop support for Python 2.6 and 3.3 (@jdufresne, issue #70)
- Remove deprecated APIs (kwargs `number_type`, `signed`, `exp`, `as_path`, `py3_safe`; enums `ns.TYPESAFE`, `ns.DIGIT`, `ns.VERSION`; functions `versorted`, `index_versorted`) (issue #81)
- Remove `pipenv` as a dependency for building (issue #86)

6.17 5.5.0 - 2018-11-18

6.17.1 Added

- `CHANGELOG.rst` to the top-level of the repository (issue #85)

6.17.2 Changed

- Documentation, packaging, and CI cleanup (@jdufresne, issues #69, #71-#80)
- Consolidate API documentation into a single page (issue #82)

6.17.3 Deprecated

- Formally deprecated old or misleading APIs (issue #83)

6.17.4 Fixed

- Add back support for very old versions of setuptools (issue #84)

6.18 5.4.1 - 2018-09-09

6.18.1 Changed

- Code format and quality checking infrastructure (issue #68)

6.18.2 Fixed

- Error in a newly added test (issues #65, #67)

6.19 5.4.0 - 2018-09-06

6.19.1 Changed

- Re-expose `natsort_key` as "public" and remove the associated `DeprecationWarning`
- Better developer documentation
- Refactor tests (issue #66)
- Bump allowed *``fastnumbers`* <https://github.com/SethMMorton/fastnumbers>`_ version

6.20 5.3.3 - 2018-07-07

6.20.1 Added

- Enable Python 3.7 support in Travis-CI (issue #61)

6.20.2 Changed

- Update docs with a FAQ and quick how-it-works (issue #60)

6.20.3 Fixed

- `StopIteration` error in the testing code

6.21 5.3.2 - 2018-05-17

6.21.1 Fixed

- Bug that prevented install on old versions of `setuptools` (issues #55, #56)
- Revert layout from `src/natsort/` back to `natsort/` to make user testing simpler (issues #57, #58)

6.22 5.3.1 - 2018-05-14

6.22.1 Added

- *‘‘bumpversion‘* <https://github.com/c4urself/bump2version>‘_ infrastructure
- Extras can be installed by “[]” notation

6.22.2 Changed

- No bugfixes or features, just infrastructure and installation updates
- Move to defining dependencies with `Pipfile`
- Development layout is now `src/natsort/` instead of `natsort/`

6.23 5.3.0 - 2018-04-20

6.23.1 Added

- Ability to consider unicode-decimal numbers as numbers (issues #52, #54)

6.23.2 Fixed

- Bug in assessing *‘‘fastnumbers‘* <https://github.com/SethMMorton/fastnumbers>‘_ version at import-time (@hholzgra, issues #51, #53)

6.24 5.2.0 - 2018-02-14

6.24.1 Added

- `ns.NUMAFTER` to cause numbers to be placed after non-numbers (issues #48, #49)
- `natcmp` function (Python 2 only) (@rinslow, issue #47)

6.25 5.1.1 - 2017-11-11

6.25.1 Added

- Additional unicode number support for Python 3.7
- Information on how to install and test (issue #46)

6.26 5.1.0 - 2017-08-19

6.26.1 Changed

- All Unicode input is now normalized (issue #44, #45)

6.26.2 Fixed

- `StopIteration` warning on Python 3.6+ (@lykinsbd, issues #42, #43)

6.27 5.0.3 - 2017-04-30

- Improved development infrastructure
- Migrated documentation to ReadTheDocs

6.28 5.0.2 - 2017-01-02

6.28.1 Added

- Additional unicode number support for Python 3.6
- "how does it work?" section to the documentation

6.28.2 Changed

- Renamed several internal functions and variables to improve clarity
- Improved documentation examples

6.29 5.0.1 - 2016-06-04

6.29.1 Added

- The `ns` enum attributes can now be imported from the top-level namespace

6.29.2 Fixed

- Bug with the `from natsort import *` mechanism
- Bug with using `natsort` with `python -OO` (issues #38, #39)

6.30 5.0.0 - 2016-05-08

6.30.1 Added

- `chain_functions` function for convenience in creating a complex user-given `key` from several existing functions

6.30.2 Changed

- `ns.LOCALE`/`humansorted` now accounts for thousands separators (issue #36)
- Refactored entire codebase to be more functional (as in use functions as units). Previously, the code was rather monolithic and difficult to follow. The goal is that with the code existing in smaller units, contributing will be easier (issue #37)
- Increased speed of execution (came for free with the new functional approach because the new factory function paradigm eliminates most `if` branches during execution). For the most cases, the code is 30-40% faster than version 4.0.4. If using `ns.LOCALE` or `humansorted`, the code is 1100% faster than version 4.0.4
- Improved clarity of documentation with regards to locale-aware sorting

6.30.3 Deprecated

- `ns.TYPESAFE` option as it is now always on (due to a new iterator-based algorithm, the typesafe function is now cheap)

6.31 4.0.4 - 2015-11-01

6.31.1 Changed

- Improved coverage of unit tests
- Unit tests use new and improved hypothesis library

6.31.2 Fixed

- Compatibility issues with Python 3.5

6.32 4.0.3 - 2015-06-25

6.32.1 Fixed

- Bad install on last release (sorry guys!) (issue #30)

6.33 4.0.2 - 2015-06-24

6.33.1 Changed

- Consolidated under-the-hood compatibility functionality

6.33.2 Fixed

- Python 2.6 and Python 3.2 compatibility. Unit testing is now performed for these versions (@dpetzold, issue #29)

6.34 4.0.1 - 2015-06-04

6.34.1 Added

- Support for sorting NaN by internally converting to -Infinity or +Infinity (issue #27)

6.35 4.0.0 - 2015-05-17

6.35.1 Changed

- Made default behavior of `natsort` search for unsigned ints, rather than signed floats. This is a backwards-incompatible change but in 99% of use cases it should not require any end-user changes (issue #20)
- Improved handling of locale-aware sorting on systems where the underlying locale library is broken (issue #34))
- Greatly improved all unit tests by adding the `hypothesis` library

6.36 3.5.6 - 2015-04-06

6.36.1 Added

- `UNGROUPLETTERS` algorithm to get the case-grouping behavior of an ordinal sort when using `LOCALE` (issue #23)
- Convenience functions `decoder`, `as_ascii`, and `as_utf8` for dealing with bytes types

6.37 3.5.5 - 2015-04-04

6.37.1 Added

- `realsorted` and `index_realsorted` functions for forward-compatibility with >= 4.0.0

6.37.2 Changed

- Made explanation of when to use `TYPESAFE` more clear in the docs

6.38 3.5.4 - 2015-04-02

6.38.1 Fixed

- Bug where a `TypeError` was raised if a string containing a leading number was sorted with alpha-only strings when `LOCALE` is used (issue #22)

6.39 3.5.3 - 2015-03-26

6.39.1 Changed

- Documentation updates to better describe locale bug, and illustrate upcoming default behavior change
- Internal improvements, including making test suite more granular

6.39.2 Fixed

- Bug where `--reverse-filter` option in shell script was not getting checked for correctness

6.40 3.5.2 - 2015-01-13

6.40.1 Added

- A `pathlib.Path` object is converted to a `str` if `ns.PATH` is enabled (issue #16)

6.41 3.5.1 - 2014-09-25

6.41.1 Changed

- Refactored modules so that only the public API was in `natsort.py` and `ns_enum.py`
- Refactored all import statements to be absolute, not relative

6.41.2 Fixed

- Bug that caused list/tuples to fail when using `ns.LOWECASEFIRST` or `ns.IGNORECASE` (issue #15)

6.42 3.5.0 - 2014-09-02

6.42.1 Added

- `alg` argument to the `natsort` functions. This argument accepts an enum that is used to indicate the options the user wishes to use. The `number_type`, `signed`, `exp`, `as_path`, and `py3_safe` options are being deprecated and will become (undocumented) keyword-only options in `natsort` version 4.0.0
- The `humansorted` convenience function as a convenience to locale-aware sorting
- The user can now modify how `natsort` handles the case of non-numeric characters (issue #14)
- The user can now instruct `natsort` to use locale-aware sorting, which allows `natsort` to perform true "human sorting" (issue #14)
- Locale functionality to the shell script

6.43 3.4.1 - 2014-08-12

6.43.1 Changed

- `natsort` will now use the *'fastnumbers'* <https://github.com/SethMMorton/fastnumbers>'_ module if it is installed. This gives up to an extra 30% boost in speed over the previous performance enhancements
- Made documentation point to more `natsort` resources, and also added a new example in the examples section

6.44 3.4.0 - 2014-07-19

6.44.1 Added

- `natsort_keygen` function that will generate a wrapped version of `natsort_key` that is easier to call. `natsort_key` is now set to deprecate at natsort version 4.0.0
- `as_path` option to `natsorted` & co. that will try to treat input strings as filepaths. This will help yield correct results for OS-generated inputs like `['/p/q/o.x', '/p/q (1)/o.x', '/p/q (10)/o.x', '/p/q/o (1).x']` (issue #3)
- `order_by_index` function to help in using the output of `index_natsorted` and `index_versorted`
- `reverse` option to `natsorted` & co. to make it's API more similar to the builtin 'sorted'
- More unit tests
- Auxiliary test code that helps in profiling and stress-testing
- Support for coveralls.io

6.44.2 Changed

- Massive performance enhancements for string input (1.8x-2.0x), at the expense of reduction in speed for numeric input (~2.0x) - note that sorting numbersstill only takes 0.6x the time of sorting strings
- Entire codebase is now PyFlakes and PEP8 compliant
- Reworked the documentation, moving most of it to PyPI's hosting platform

6.44.3 Fixed

- Bug that caused user's options to the `natsort_key` to not be passed on to recursive calls of `natsort_key` (issue #12)

6.45 3.3.0 - 2014-06-28

6.45.1 Added

- `versorted` method for more convenient sorting of versions (issue #11)
- Unit test coverage (99%)

6.45.2 Changed

- Updated command-line tool `--number_type` option with 'version' and 'ver' to make it more clear how to sort version numbers
- Moved unit-testing mechanism from being docstring-based to actual unit tests in actual functions (issue #10)
- Made docstrings for public functions mirror the README API
- Connected `natsort` development to Travis-CI to help ensure quality releases

6.46 3.2.1 - 2014-06-20

6.46.1 Fixed

- Re-"Fixed" unorderable types issue on Python 3.x - this workaround is for when the problem occurs in the middle of the string (issue #7 again)

6.47 3.2.0 - 2014-05-07

6.47.1 Fixed

- "Fixed" unorderable types issue on Python 3.x with a workaround that attempts to replicate the Python 2.x behavior by putting all the numbers (or strings that begin with numbers) first (issue #7)

6.47.2 Removed

- Now explicitly excluding `__pycache__` from releases by adding a prune statement to MANIFEST.in

6.48 3.1.2 - 2014-05-05

6.48.1 Added

- `setup.cfg` to support universal wheels (issue #6)
- Python 3.0 and Python 3.1 as requiring the argparse module

6.49 3.1.1 - 2014-03-01

6.49.1 Added

- Ability to sort lists of lists (issue #5)

6.49.2 Changed

- Cleaned up import statements

6.50 3.1.0 - 2014-01-20

6.50.1 Added

- `signed` and `exp` options to allow finer tuning of the sorting
- Doctests
- New shell script options that correspond to `signed` and `exp`
- In the shell script the user can now specify multiple numbers to exclude or multiple ranges

6.50.2 Changed

- Entire codebase now works for both Python 2 and Python 3 without needing to run `2to3`
- Updated all doctests
- Further simplified the `natsort` base code by removing unneeded functions.
- Simplified documentation where possible
- Improved the shell script code
- Made the shell script documentation less "path"-centric to make it clear it is not just for sorting file paths

6.50.3 Removed

- The shell script filesystem-based options because these can be achieved better though a pipeline by which to filter

6.51 3.0.2 - 2013-10-01

6.51.1 Changed

- Made float, int, and digit searching algorithms all share the same base function
- Made the `__version__` variable available when importing the module

6.51.2 Fixed

- Outdated comments

6.52 3.0.1 - 2013-08-15

6.52.1 Added

- Support for unicode strings (issue #2)

6.52.2 Fixed

- Empty string removal function

6.52.3 Removed

- Extraneous `string2int` function

6.53 3.0.0 - 2013-07-13

6.53.1 Added

- A `number_type` argument to the sorting functions to specify how liberal to be when deciding what a number is

6.53.2 Changed

- Reworked the documentation

6.54 2.2.0 - 2013-06-25

6.54.1 Added

- `key` attribute to `natsorted` and `index_natsorted` so that it mimics the functionality of the built-in `sorted` (issue #1)
- Tests to reflect the new functionality, as well as tests demonstrating how to get similar functionality using `natsort_key`

6.55 2.1.0 - 2012-12-05

6.55.1 Changed

- Reorganized package
- Now using a platform independent shell script generator (`entry_points` from distribute)
- Can now execute `natsort` from command line with `python -m natsort` as well

6.56 2.0.2 - 2012-11-30

6.56.1 Added

- The `use_2to3` option to `setup.py`
- Include `distribute_setup.py` to the distribution
- Dependency to the `argparse` module (for python2.6)

6.57 2.0.1 - 2012-11-21

6.57.1 Added

- Tests into the natsort.py file itself

6.57.2 Changed

- Reorganized directory structure

6.58 2.0.0 - 2012-11-16

6.58.1 Added

- Better README documentation
- Doctests

6.58.2 Changed

- Sorting algorithm to support floats (including exponentials) and basic version number support

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index