# natsort Documentation

***Release 8.4.0***

**Seth M. Morton**

**Jun 20, 2023**

# Contents


**1 How Does Natsort Work? 3**
- 1.1 Special Cases Everywhere! . . . 3

**2 Examples and Recipes 5**

**3 natsort API 7**
- 3.1 Standard API . . . 8
- 3.2 Convenience Functions . . . 12

**4 Possible Issues with `humansorted()` or `ns.LOCALE` 21**

**5 Shell Script 23**

**6 Changelog 25**

**7 Indices and tables 27**

**Index 29**

- 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 as well.

CHAPTER 2

# Examples and Recipes

This page has been moved to the natsort wiki.

# CHAPTER 3

# natsort API


- *Standard API*
  - `natsorted()`
  - *The* `ns` *enum*
  - `natsort_key()`
  - `natsort_keygen()`
  - `os_sort_key()`
  - `os_sort_keygen()`
- *Convenience Functions*
  - `os_sorted()`
  - `realsorted()`
  - `humansorted()`
  - `index_natsorted()`
  - `index_realsorted()`
  - `index_humansorted()`
  - `order_by_index()`
  - *Help With Bytes*
  - *Help With Creating Function Keys*
  - *Help With Type Hinting*

## 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

Please visit https://github.com/SethMMorton/natsort/blob/main/CHANGELOG.md.

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Index


## A

as_ascii() (*in module natsort*), 18
as_utf8() (*in module natsort*), 18

## C

chain_functions() (*in module natsort*), 19

## D

decoder() (*in module natsort*), 18

## H

humansorted() (*in module natsort*), 14

## I

index_humansorted() (*in module natsort*), 16
index_natsorted() (*in module natsort*), 15
index_realsorted() (*in module natsort*), 15

## N

natsort_key() (*in module natsort*), 10
natsort_keygen() (*in module natsort*), 11
natsorted() (*in module natsort*), 8
ns (*in module natsort*), 8
numeric_regex_chooser() (*in module natsort*), 19

## O

order_by_index() (*in module natsort*), 17
os_sort_key() (*in module natsort*), 11
os_sort_keygen() (*in module natsort*), 12
os_sorted() (*in module natsort*), 12

## R

realsorted() (*in module natsort*), 13