# intspan Documentation

***Release 1.5.15***

**Jonathan Eunice**

**Jan 25, 2019**

# Contents

`intspan` is a `set` subclass that conveniently represents sets of integers. Sets can be created from and displayed as integer spans such as `1-3,14,29,92-97` rather than exhaustive member listings. Compare:

```
intspan('1-3,14,29,92-97')
[1, 2, 3, 14, 29, 92, 93, 94, 95, 96, 97]
```

Or worse, the unsorted, non-intuitive listings that crop up with Python's native unordered sets, such as:

```
set([96, 1, 2, 3, 97, 14, 93, 92, 29, 94, 95])
```

While they all indicate the same values, `intspan` output is much more compact and comprehensible. It better divulges the contiguous nature of segments of the collection, making it easier for humans to quickly determine the "shape" of the data and ascertain "what's missing?"

When iterating, `pop()`-ing an item, or converting to a list, `intspan` behaves as if it were an ordered–in fact, sorted–collection. A key implication is that, regardless of the order in which items are added, an `intspan` will always be rendered in the most compact, organized form possible.

The main draw is having a convenient way to specify, manage, and see output in terms of ranges–for example, rows to process in a spreadsheet. It can also help you quickly identify or report which items were *not* successfully processed in a large dataset.

CHAPTER 1

# Usage

```
from __future__ import print_function   # Python 2 and 3 compatibility
from intspan import intspan

s = intspan('1-3,14,29,92-97')
s.discard('2,13,92')
print(s)
print(repr(s))
print(list(s))
```

yields:

```
1,3,14,29,93-97
intspan('1,3,14,29,93-97')
[1, 3, 14, 29, 93, 94, 95, 96, 97]
```

While:

```
>>> for n in intspan('1-3,5'):
>>>     print(n)
1
2
3
5
```

Most set operations such as intersection, union, and so on are available just as they are in Python's `set`. In addition, if you wish to extract the contiguous ranges:

```
>>> for r in intspan('1-3,5,7-9,10,21-22,23,24').ranges():
>>>     print(r)
(1, 3)
(5, 5)
(7, 10)
(21, 24)
```

Note that these endpoints represent closed intervals, rather than the half-open intervals commonly used with Python's `range()`. If you combine `intspan` ranges with Python generators, you'll have to increment the stop value by one yourself to create the suitable "half-open interval."

There is a corresponding range-oriented constructor:

```
>>> intspan.from_ranges([ (4,6), (10,12) ])
intspan('4-6,10-12')
```

A convenience `from_range` method creates a contiguous `intspan` from a given low to a high value.:

```
>>> intspan.from_range(8, 12)
intspan('8-12')
```

The `universe` method returns the covering set or "implied universe" of an `intspan`:

```
>> intspan('1,3,5,7').universe()
intspan('1-7')
```

To find the elements *not* included, you can use the `complement` method:

```
>>> items = intspan('1-3,5,7-9,10,21-24')
>>> items.complement()
intspan('4,6,11-20')
```

The "missing" elements are computed as any integers between the `intspan`'s minimum and maximum values that aren't included. If you'd like to customize the intended `low` and `high` bounds, you can give those explicitly.:

```
>>> items.complement(high=30)
intspan('4,6,11-20,25-30')
```

You can use the `difference` method or `-` operator to find the complement with respect to an arbitrary set, rather than just an expected contiguous range.

# CHAPTER 2

# intspanlist

As of version 1.2, a new function `spanlist` is provided. It returns a list from the same kind of specification string `intspan` does, but ordered as given rather than fully sorted. A corresponding `intspanlist` class subclasses `list` in the same way that `intspan` subclasses `set`.

```
>>> intspanlist('4,1-5,5')   # note order preserved
intspanlist('4,1-3,5')

>>> list(intspanlist('4,1-5,5'))
[4, 1, 2, 3, 5]

>>> spanlist('4,1-5,5')
[4, 1, 2, 3, 5]
```

So `spanlist` the function creates a `list`, whereas `intspanlist` creates a similar object–but one that has a more sophisticated representation and more specific update methods. Both of them have somewhat set-like behavior, in that they seek to not have excess duplication of members.

The intended use for this strictly-ordered version of `intspan` is to specify an ordering of elements. For example, a program might have 20 items, 1-20. If you wanted to process item 7, then item 3, then "all the rest," `intspanlist('7,3,1-20')` would be a convenient way to specify this. You could loop over that object in the desired order.(See below for a different formulation, `intspanlist('7,3,*')`, in which the `*` is a symbolic "all the rest" marker, and the universe set can be specified either immediately or later.)

Note that `intspanlist` objects do not necessarily display as they are entered:

```
>>> intspanlist('7,3,1-20')
intspanlist('7,3,1-2,4-6,8-20')
```

This is an equivalent representation–though lower-level, more explicit, and more verbose.

Many other `list` methods are available to `intspanlist`, especially including iteration. Note however that while `intspan` attempts to faithfully implement the complete methods of a Python `set` , `intspanlist` is a thinner shim over `list`. It works well as an immutable type, but modifications such as `pop`, `insert`, and slicing are more problematic. `append` and `extend` work to maintain a "set-ish," no-repeats nature–by discarding any additions that are already in the container. Whatever was seen first is considered to be in its "right" position. `insert` and other `list` update methods, however, provide no such promises.

Indeed, it's not entirely clear what update behavior *should be*, given the use case. If a duplicate is appended or inserted somewhere, should an exception be raised? Should the code silently refuse to add items already seen? Or something else? Maybe even duplicates should be allowed? Silent denial is the current default, which is compatible with set behavior and `intspan`; whether that's the "right" or best choice for a fully ordered variant is unclear. (If you have thoughts on this or relevant use cases to discuss, open an issue on Bitbucket or ping the author.)

## 2.1 Symbolic Rest

As a final trick, `intspanlist` instances can contain a special value, rendered as an asterisk (`*`), meaning "the rest of the list." Under the covers, this is converted into the singleton object `TheRest`.

```
>>> intspanlist('1-4,*,8')
intspanlist('1-4,*,8')
```

This symbolic "everything else" can be a convenience, but eventually it must be "resolved."

`intspanlist` objects may be created with an optional second parameter which provides "the universe of all items" against which "the rest" may be evaluated. For example:

```
>>> intspanlist('1-4,*,8', '1-9')
intspanlist('1-7,9,8')
```

Whatever items are "left over" from the universe set are included wherever the asterisk appears. Like the rest of `intspan` and `intspanlist` constructors, duplicates are inherently removed.

If the universe is not given immediately, you may later update the `intspanlist` with it:

```
>>> i = intspanlist('1-4,*,8')
>>> i.therest_update('1-9')
intspanlist('1-7,9,8')
```

If you don't wish to modify the original list (leaving its abstract marker in place), a copy may be created by setting the `inplace=False` kwarg.

The abstract "and the rest" markers are intended to make `intspanlist` more convenient for specifying complex partial orderings.

# CHAPTER 3

# Performance and Alternatives

`intspan` piggybacks Python's `set` type. `inspanlist` piggybacks `list`. So they both store every integer individually. Unlike Perl's `Set::IntSpan` these types are not optimized for long contiguous runs. For sets of several hundred or even thousands of members, you'll probably never notice the difference.

But if you're doing extensive processing of large sets (e.g. with 100K, 1M, or more elements), or doing numerous set operations on them (e.g. union or intersection), a data structure based on lists of ranges, run length encoding, or Judy arrays might perform and scale better. Horses for courses.

There are several modules you might want to consider as alternatives or supplements. AFAIK, none of them provide the convenient integer span specification `intspan` does, but they have other virtues:

- cowboy provides generalized ranges and multi-ranges. Bonus points for the tagline: "It works on ranges"
- spans provides several different kinds of ranges and then sets for those ranges. Includes nice `datetime` based intervals similar to PostgreSQL time intervals, and `float` ranges/sets. More ambitious and general than `intspan`, but lacks truly convenient input or output methods akin to `intspan`.
- ranger is a generalized range and range set module. It supports open and closed ranges, and includes mapping objects that attach one or more objects to range sets.
- rangeset is a generalized range set module. It also supports infinite ranges.
- judy a Python wrapper around Judy arrays that are implemented in C. No docs or tests to speak of.
- RoaringBitmap, a hybrid array and bitmap structure designed for efficient compression and fast operations on sets of 32-bit integers.

CHAPTER 4

# Notes

- Though inspired by Perl's Set::IntSpan, that's where the similarity stops. `intspan` supports only finite sets, and it follows the methods and conventions of Python's `set`.
- `intspan` methods and operations such as `add()` `discard()`, and `>=` take integer span strings, lists, and sets as arguments, changing facilities that used to take only one item into ones that take multiple, including arguments that are technically string specifications rather than proper `intspan` objects.
- A version of `intspanlist` that doesn't discard duplicates is under consideration.
- String representation and `ranges()` method based on Jeff Mercado's concise answer to this StackOverflow question. Thank you, Jeff!
- Automated multi-version testing managed with pytest, pytest-cov, coverage and tox. Continuous integration testing with Travis-CI. Packaging linting with pyroma.

  Successfully packaged for, and tested against, all current versions of Python including latest builds of 2.6, 2.7, 3.2, 3.3, 3.4, 3.5, 3.6, and 3.7 as well as latest PyPy and PyPy3. Test line coverage 100%.
- The author Jonathan Eunice or @jeunice on Twitter welcomes your comments and suggestions.

# CHAPTER 5

# API Reference

In addition to the methods explicitly documented here, `intspan` attempts to implement the complete list of `set` methods.

**class** `intspan.`**`intspan`**(*initial=None*)

A set of integers, expressed as an ordered sequence of spans. Because `intspan('1-3,14,29,92-97')` is better than `[1, 2, 3, 14, 29, 92, 93, 94, 95, 96, 97]`.

**`add`**(*items*)

Add items.

**Parameters** **`items`** (*`iterable | str`*) – Items to add. May be an intspan-style string.

**`complement`**(*low=None*, *high=None*)

Return the complement of the given intspan–that is, all of the 'missing' elements between its minimum and missing values. Optionally allows the universe set to be manually specified.

**Parameters**

- **`low`** (*`int`*) – Low bound of universe to complement against.
- **`high`** (*`int`*) – High bound of universe to complement against.

**Returns** the complement set

**Return type** *intspan*

**Raises** **`ValueError`** – if the set is empty (thus the compelement is infinite)

**`copy`**()

Return a new set with the same members.

**`difference`**(*items*)

Return the difference of two or more sets as a new set.

(i.e. all elements that are in this set but not the others.)

**`difference_update`**(*items*)

Remove all elements of another set from this set.

**discard**(*items*)

Discard items.

**Parameters** **items** (*iterable|str*) – Items to remove. May be an intspan-style string.

**classmethod from_range**(*low*, *high*)

Construct an intspan from the low value to the high value, inclusive. I.e., closed range, not the more typical Python half-open range.

**Parameters**

- **low** (*int*) – Low bound of set to construct.
- **high** (*int*) – High bound of set to construct.

**Returns** New intspan low-high.

**Return type** *intspan*

**classmethod from_ranges**(*ranges*)

Construct an intspan from a sequence of (low, high) value sequences (lists or tuples, say). Note that these values are inclusive, closed ranges, not the more typical Python half-open ranges.

**Parameters** **ranges** (*list*) – List of closed/inclusive ranges, each a tuple.

**Returns** intspan combining the ranges

**Return type** *intspan*

**intersection**(*items*)

Return the intersection of two or more sets as a new set.

(i.e. elements that are common to all of the sets.)

**intersection_update**(*items*)

Update a set with the intersection of itself and another.

**issubset**(*items*)

Report whether another set contains this set.

**issuperset**(*items*)

Report whether this set contains another set.

**pop**()

Remove and return an arbitrary element; raises KeyError if empty.

**Returns** Arbitrary member of the set (which is removed)

**Return type** int

**Raises** **KeyError** – If the set is empty.

**ranges**()

Return a list of the set's contiguous (inclusive) ranges.

**Returns** List of all contained ranges.

**Return type** list

**remove**(*items*)

Remove an element from a set; it must be a member.

If the element is not a member, raise a KeyError.

**symmetric_difference**(*items*)

Return the symmetric difference of two sets as a new set.

(i.e. all elements that are in exactly one of the sets.)

**`symmetric_difference_update`**(*items*)
Update a set with the symmetric difference of itself and another.

**`union`**(*items*)
Return the union of sets as a new set.

(i.e. all elements that are in either set.)

**`universe`**(*low=None*, *high=None*)
Return the "universe" or "covering set" of the given intspan–that is, all of the integers between its minimum and missing values. Optionally allows the bounds of the universe set to be manually specified.

**Parameters**

- **`low`** (*`int`*) – Low bound of universe.
- **`high`** (*`int`*) – High bound of universe.

**Returns** the universe or covering set

**Return type** *intspan*

**`update`**(*items*)
Add multiple items.

**Parameters** **`items`** (*`iterable | str`*) – Items to add. May be an intspan-style string.

**`class`** `intspan.`**`intspanlist`**(*initial=None*, *universe=None*)
An ordered version of `intspan`. Is to `list` what `intspan` is to `set`, except that it is somewhat set-like, in that items are not intended to be repeated. Works fine as an immutable data structure. Still some issues if one mutates an instance. Not terrible problems, but the set-like nature where there is only one entry for each included integer may be broken.

**`append`**(*item*)
Add to the end of the intspanlist

**Parameters** **`item`** (*`int`*) – Item to add

**`complement`**(*low=None*, *high=None*)
Return the complement of the given intspanlist–that is, all of the 'missing' elements between its minimum and missing values. Optionally allows the universe set to be manually specified.

**Parameters**

- **`low`** (*`int`*) – Low bound of universe to complement against.
- **`high`** (*`int`*) – High bound of universe to complement against.

**Returns** the complement set

**Return type** *intspanlist*

**Raises** **`ValueError`** – if the set is empty (thus the compelement is infinite)

**`copy`**()
Return a copy of the intspanlist.

**`extend`**(*items*)
Add a collection to the end of the intspanlist

**Parameters** **`items`** (*`iterable`*) – integers to add

**classmethod from_range**(*low*, *high*)

Construct an intspanlist from the low value to the high value, inclusive. I.e., closed range, not the more typical Python half-open range.

**Parameters**

- **low** (*int*) – Low bound.
- **high** (*int*) – High bound.

**Returns** New intspanlist low-high.

**Return type** *intspanlist*

**classmethod from_ranges**(*ranges*)

Construct an intspanlist from a sequence of (low, high) value sequences (lists or tuples, say). Note that these values are inclusive, closed ranges, not the more typical Python half-open ranges.

**Parameters** **ranges** (*list*) – List of closed/inclusive ranges, each a tuple.

**Returns** intspanlist combining the ranges

**Return type** *intspanlist*

**ranges**()

Return a list of the set's contiguous (inclusive) ranges.

**therest_update**(*universe*, *inplace=True*)

If the receiving `intspanlist` contains a `TheRest` marker, replace it with the contents of the universe. Generally done *in situ*, but if value of `inplace` kwarg false, returns an edited copy.

`intspan.`**spanlist**(*spec=None*)

Given a string specification like the ones given to `intspan`, return a list of the included items, in the same item given. Thus, `spanlist("3,1-4")` yields `[3, 1, 2, 4]`. Experimental partial implementation of ability to have ordered intspans.

CHAPTER 6

# Installation

To install or upgrade to the latest version:

```
pip install -U intspan
```

On some systems, you may need to use `pip2` to install under Python 2, or `pip3` to install under Python 3. You may also need to prefix these commands with `sudo` to authorize installation. In environments without super-user privileges, you may want to use `pip`'s `--user` option, to install only for a single user, rather than system-wide. Finally, in cases where `pip` is not well-configured to match a specific Python interpreter you want to use, a useful fallback:

```
python3.6 -m pip install -U intspan
```

## 6.1 Testing

`intspan` is tested twice before each release–once on the developer's workstation, and once by the Travis CI continuous integration service. If you'd like to also run the module tests locally, you'll need to install `pytest` and `tox`. For full testing, you will also need `pytest-cov` and `coverage`. Then run one of these commands:

```
tox                 # normal test run, across all currently-supported versions
tox -e py27         # run for a environment only (e.g. py27)
tox -c toxcov.ini   # run full coverage tests
```

# CHAPTER 7

# Change Log

**1.5.15** (January 25, 2019)

Documentation tweaked.

Dropped unused argument to `spanlist()`.

**1.5.13** (June 3, 2018)

Documentation tweaks.

**1.5.12** (June 2, 2018)

Updated testing matrix. Now includes Python 3.7 pre-release.

Tox tests trimmed in favor of Travis CI.

**1.5.11** (October 13, 2017)

Add pyproject.toml for PEP 508 conformance

**1.5.10** (May 30, 2017)

Update Python version compatibility strategy. Now Python 3 centric. More future-proofed.

Updated testing matrix. Tweaked docs.

**1.5.8** (February 9, 2017)

Changed from module to package builds.

**1.5.5** (February 1, 2017)

Documentation upgrades.

**1.5.4** (January 31, 2017)

Tested and certified for all early 2017 versions of Python including latest builds of 2.6, 2.7, 3.3, 3.4, 3.5, and 3.6, as well as latest PyPy and PyPy3.

**1.5.3** (June 21, 2016)

Tested and certified for Python 2.7.11, 3.4.4, 3.5.1, 3.6.0a2, PyPy 3.5.1 (based on Python 2.7.10), and PyPy3 5.2.0-alpha0 (based on Python 3.3.5). Python 3.2 is deprecated due to its advancing age and now-limited compatibility with my test rig. It still passes Travis CI tests–but as soon as it does not, it will be withdrawn from support.

**1.5.2** (September 23, 2015)

Tested and certified for PyPy 2.6.1 (based on Python 2.7.10)

**1.5.1** (September 14, 2015)

Updated testing for Python 3.5.0 final.

**1.5.0** (August 27, 2015)

Added `universe` method. Extended tests. Continuing to extend docs.

**1.4.4** (August 27, 2015)

Improving documentation, especially around API details. Will probably require a handful of incremental passes to get the API reference ship-shape.

**1.4.2** (August 26, 2015)

Reorganied documentation. Added API details and moved full docs to Read The Docs.

**1.4.1**

Achieves 100% *branch* coverage. *Hooah!*

**1.4.0**

Achieves 100% test coverage and integrated coverage testing across multiple versions with `tox`.

**1.3.10**

Improved test coverage and explicit coverage testing. Also tweaked docs.

**1.3.9**

Simplified `setup.py` and packaging.

**1.3.7**

Adds `bdist_wheel` packaging support.

**1.3.6**

Switches from BSD to Apache License 2.0 and integrates `tox` testing with `setup.py`

**1.3.0**

Adds `*` notation for abstract "the rest of the items" in an `intspanlist`.

**1.2.6**

Inaugurates continuous integration with Travis CI.

**1.2.0**

Adds an experimental `spanlist` constructor and `intspanlist` type.

**1.1.0**

Adds `from_range` and `complement` methods; improves error handling of `pop` on an empty set), and tweaks testing.

**1.0.0**

Immediately follows 0.73. Bumped to institute a cleaner "semantic versioning" scheme. Upgraded from "beta" to "production" status.

**0.73.0**

Updates testing to include the latest Python 3.4

**0.7.0**

Fixed parsing of spans including negative numbers, and added the `ranges()` method. As of 0.71, the `from_ranges()` constructor appeared.

# Index