pyjlib

Release 0.0.0

Mohammad Abouali (maboualidev@gmail.com)

Sep 24, 2019

CONTENTS:

Python Implementation of some of the Java classes.

The objects in this library, try to mimic the behavior of their Java counter-part. However, some Python flavor is also added to them. It is possible that certain extra features are added to the object.

CHAPTER

ONE

UTIL

1.1 StringJoiner

class pyjlib.util.stringjoiner.**StringJoiner**(*separator=', '*, *prefix=''*, *suffix=''*)

Bases: object

`StringJoiner` mimics the `java.util.StringJoiner` class. So, Make sure to also have a look at the `Java` documentations available here

It is supposed to be similar to the `Java` version but it would also have some `Python` feeling to it.

As elements are added to the `StringJoiner` instance, they are not getting immediately joined. The joining happens only when the object is passed to `str()` or its `.toString()` method is called.

Examples

- Creating a default `StringJoiner`:

```
>>> from pyjlib.util.stringjoiner import StringJoiner
>>> sj = StringJoiner()
>>> repr(sj)
'{"separator": ",", "prefix": "", "suffix": "", "nelem": 0, "str_length": 0,
↪"elements": []}'
>>> str(sj)
''
```

- Accessing readonly attributes:

```
>>> sj = StringJoiner(prefix="(",suffix=")")
>>> sj.separator
','
>>> sj.prefix
'('
>>> sj.suffix
')'
>>> sj.suffix = "["
>>> sj.suffix
')'
```

- adding few elements:

```
>>> sj.add("e1")
{"separator": ",", "prefix": "", "suffix": "", "nelem": 1, "str_length": 2,
↪"elements": ["e1"]}
>>> sj.add("e2")
{"separator": ",", "prefix": "", "suffix": "", "nelem": 2, "str_length": 5,
↪"elements": ["e1", "e2"]}
```

- adding an element at a specific position:

```
>>> sj.add("e1-2",1)
{"separator": ",", "prefix": "", "suffix": "", "nelem": 3, "str_length": 10,
↪"elements": ["e1", "e1-2", "e2"]}
```

- you could check how big would be the resulting string by using `len()` or you could see how many elements are currently within the `StringJoiner`:

```
>>> sj = StringJoiner(prefix="(",suffix=")")
>>> sj.add_multi("e1","e2","e3")
{"separator": ",", "prefix": "(", "suffix": ")", "nelem": 3, "str_length": 10,
↪"elements": ["e1", "e2", "e3"]}
>>> s = str(sj)
>>> print(s)
(e1,e2,e3)
>>> len(s)
10
>>> len(sj)
10
>>> sj.nelem
3
```

- you could check if something exists in the `StringJoiner`:

```
>>> sj = StringJoiner()
>>> sj.add_multi("e1", "e2", "e3")
{"separator": ",", "prefix": "", "suffix": "", "nelem": 3, "str_length": 8,
↪"elements": ["e1", "e2", "e3"]}
>>> "e2" in sj
True
>>> "non-existing" in sj
False
```

- You could access each element separately:

```
>>> sj = StringJoiner(prefix="(",suffix=")")
>>> sj.add_multi("e1","e2","e3")
{"separator": ",", "prefix": "(", "suffix": ")", "nelem": 3, "str_length": 10,
↪"elements": ["e1", "e2", "e3"]}
>>> sj[0]
'e1'
>>> sj[1]
'e2'
>>> sj[2]
'e3'
```

(continues on next page)

(continued from previous page)

```
>>> sj[3]
Traceback (most recent call last):
...
IndexError: index out of bound. Expected an integer between 0 and 2; got 3.
```

- a merge example:

```
>>> o_sj.add_multi("o1","o2")
{"separator": ",", "prefix": "[", "suffix": "]", "nelem": 2, "str_length": 7,
↪"elements": ["o1", "o2"]}
>>> str(o_sj)
'[o1,o2]'
>>> str(sj)
'e1,e1-2,e2'
>>> o_sj.merge(sj,1)
>>> str(o_sj)
'[o1,e1,e1-2,e2,o2]'
```

- adding two `StringJoiner` instances: note that the + operator is not cumulative, also, the addition would not change the original instances:

```
>>> sj1 = StringJoiner()
>>> repr(sj1)
'{"separator": ",", "prefix": "", "suffix": "", "nelem": 0, "str_length": 0,
↪"elements": []}'
>>> sj2 = StringJoiner(prefix="(",suffix=")")
>>> repr(sj2)
'{"separator": ",", "prefix": "(", "suffix": ")", "nelem": 0, "str_length": 2,
↪"elements": []}'
>>> sj1.add_multi("sj1_e1","sj1_e2")
{"separator": ",", "prefix": "", "suffix": "", "nelem": 2, "str_length": 13,
↪"elements": ["sj1_e1", "sj1_e2"]}
>>> sj2.add_multi("sj2_e1","sj2_e2")
{"separator": ",", "prefix": "(", "suffix": ")", "nelem": 2, "str_length": 15,
↪"elements": ["sj2_e1", "sj2_e2"]}
>>> str(sj1)
'sj1_e1,sj1_e2'
>>> str(sj2)
'(sj2_e1,sj2_e2)'
>>> sj1_2 = sj1 + sj2 # the sum is not cumulative. You get a different results if␣
↪you change the order.
>>> str(sj1_2)
'sj1_e1,sj1_e2,sj2_e1,sj2_e2'
>>> sj2_1 = sj2 + sj1
>>> str(sj2_1)
'(sj2_e1,sj2_e2,sj1_e1,sj1_e2)'
>>> str(sj1) # after addition, the original instances do not change.
'sj1_e1,sj1_e2'
>>> str(sj2)
'(sj2_e1,sj2_e2)'
```

- you could subtract two `StringJoiner`:

```
>>> sj1 = StringJoiner()
>>> sj1.add_multi("e1", "e2", "e3")
{"separator": ",", "prefix": "", "suffix": "", "nelem": 3, "str_length": 8,
↪"elements": ["e1", "e2", "e3"]}
>>>
>>> sj2 = StringJoiner()
>>> sj2.add_multi("e2", "e3", "e4")
{"separator": ",", "prefix": "", "suffix": "", "nelem": 3, "str_length": 8,
↪"elements": ["e2", "e3", "e4"]}
>>> sj1_2 = sj1 - sj2
>>> str(sj1_2)
'e1'
>>> sj2_1 = sj2 - sj1
>>> str(sj2_1)
'e4'
```

__init__(*separator=', ', prefix='', suffix=''*) → None
Initializes a StringJoiner instance. :param separator: The separator to be used. Default value is comma :param prefix: The prefix to be used. Default value is empty :param suffix: The suffix to be used. Default value is empty

Raise TypeError: if any of the separator, prefix, or suffix is not of type str.

property separator
provides access to separator. Readonly attribute.

Returns the string used as separator

property prefix
provides access to the prefix. Readonly attribute.

Returns the string used as prefix

property suffix
provides access to the suffix. Readonly attribue.

Returns the string used as suffix

classmethod _isReadOnly(*name: str*) → bool

property nelem
number of elements that would be join together. Readonly attribute.

Returns an integer showing number of elements that would be joined together.

toString()
Return str(self).

_setIdx(*idx: int*) → int

add(*value: Any*, *idx=None*)
adds an element to the list that are going to be joined.

Parameters

- **value** – the value that is going to be added to the list. Note that regardless of the type of the value the list would always add str(value) to the list and not the value itself. So, always a string representation of the value is added.
- **idx** – An optional integer suggesting where in the list this value should be added. If not provided, the value is added at the end of the list.

Returns the same instance of the `StringJoiner`, so you could use it in a chain call.

Example

Using Chain add:

```
>>> sj = StringJoiner()
>>> sj.add("first element").add("second element")
```

---

add_multi(**values*, *idx=None*)

The same as `sj.add`, however, you would be able to provide multiple entry at the same time.

Parameters

- **values** – the values that are going to be added to the list
- **idx** – the index to start inserting the values.

Returns the same instance of the `StringJoiner`, so you could use it in a chain call.

remove(*value*) → None

removes a value, if it is in the list.

Parameters **value** – The value, whose string representation needs to be removed.

Raise ValueError: if the value is not found in the list.

remove_multi(**values*)

The same as `.remove`, this function tries to remove the value that are provided. Note that this function does not raises a `ValueError` if the value is not in the list.

Parameters **values** – the values that are supposed to be removed.

Returns a list of string, with those elements that were removed from the `StringJoiner`

remove_by_index(**indices*)

Could be used to remove multiple elements by their index. This does not produces an `IndexError`. Also note that you could remove one element at a time as you do with regular lists.

Example

- removing multiple elements:

```
>>> sj.remove_by_index(1,3,4)
```

- removing one element at a time:

```
>>> del sj[1]
>>> del sj[3]
>>> del sj[4]
```

Parameters **indices** – list of indices that are supposed to be removed.

Returns list of indices that were successfully removed.

Raise TypeError: if any of the indices are not integer.

merge(*o_iterable*, *idx=None*) → None

merges the elements of another iterable object or another `StringJoiner` to this instance.

Parameters

- **o_iterable** – The other iterable object that its elements are supposed to be merged with this one.
- **idx** – The starting index to merge. This could be used to merge another iterable in the middle of the current one. If not provided, the merge happens at the end of the current elements.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

M

N

P

R

S

T