sugar Documentation

Release 0.1.1

Bharadwaj Yarlagadda

November 24, 2016

Contents

Python utility library. Based on sugar Javascript Library.

CHAPTER 1

Links

- Project: https://github.com/bharadwajyarlagadda/sugar.py
- Documentation: http://sugarpy.readthedocs.io
- Pypi: https://pypi.python.org/pypi/sugar.py
- TravisCI: https://travis-ci.org/bharadwajyarlagadda/sugar.py

CHAPTER 2

Features

- Supported on Python 2.7 and Python 3.3+.

CHAPTER 3

Quickstart

Install using pip:

```
pip install sugar.py
```

CHAPTER 4

Guide

4.1 Installation

sugar.py requires Python 2.7 or 3.3+.

To install from PyPI:

```
pip install sugar.py
```

You can also install sugar.py with all the latest changes:

```
$ git clone git@github.com:bharadwajyarlagadda/sugar.py.git
$ cd sugar.py
$ python setup.py install
```

4.2 SugarJS Differences

4.2.1 Naming Conventions

sugar.py adheres to the following conventions:

- Function names use `snake_case` instead of `camelCase`.
- Any SugarJS function that shares its name with a reserved Python keyword will have an _ appended after it (e.g. `from` in sugarjs would be `from_` in sugar.py).

4.3 API Reference

4.3.1 Arrays

`sugar.arrays.`**`add`**(*array*, *item*, *index=None*)

Adds item to the array and returns the result as a new array. If item is also an array, it will be concatenated instead of inserted. index will control where item is added.

Parameters

- **`array`** (*list*) – List of values passed in by the user.
- **`item`** (*mixed*) – Value to be added to the array (passed in by the user).

- **index** (*int*) – Position at which the item will be added to the list.

Returns Adds item to the array and returns the result as a new array.

Return type list

Example

```
>>> add([11, 22, 33], 88)
[11, 22, 33, 88]
>>> add([11, 22, 33], 88, 1)
[11, 88, 22, 33]
>>> add([11, 22, 33], [44, 55])
[11, 22, 33, 44, 55]
>>> add([11, 22, 33], [44, 55, 66, 77], 1)
[11, 44, 55, 66, 77, 22, 33]
```

New in version TODO.

sugar.arrays.**append**(*array*, *item*, *index=None*)

Appends item to the array. If item is also an array, it will be concatenated instead of inserted.

Parameters

- **array** (*list*) – List of values passed in by the user.
- **item** (*mixed*) – Value to be added to the array (passed in by the user).
- **index** (*int*) – Position at which the item will be added to the list.

Returns Appends item to the array and returns the result.

Return type list

Example

```
>>> append([11, 22, 33], 88)
[11, 22, 33, 88]
>>> append([11, 22, 33], 88, 1)
[11, 88, 22, 33]
>>> append([11, 22, 33], [44, 55])
[11, 22, 33, 44, 55]
>>> append([11, 22, 33], [44, 55, 66, 77], 1)
[11, 44, 55, 66, 77, 22, 33]
```

New in version TODO.

sugar.arrays.**average**(*array*)

Returns the average for all the values in the given array.

Parameters **array** (*list*) – List of values.

Returns Average of all the values in the given list.

Return type int/float

Example

```
>>> float(average([1, 2, 3]))
2.0
```

New in version 0.1.0.

sugar.arrays.**clone**(*obj*)

Returns a shallow copy of the given list. This method can also be used for othe objects such as int/float/string.

Parameters **obj** (*list*) – List of values provided by the user.

Returns Shallow copy of the given array.

Return type list

Example

```
>>> clone([1, 2, 3])
[1, 2, 3]
>>> clone('foobar')
'foobar'
>>> clone(1234)
1234
```

New in version TODO.

sugar.arrays.**compact**(*array*, *all=False*)

Removes all instances of None, False, empty strings. This includes None, False, and empty strings.

Parameters

- **array** (*list*) – List of values provided by the user.
- **all** (*bool*) – Boolean value to remove all the instances of None, False and empty strings.

Returns List of values with all falsy elements removed.

Return type list

Example

```
>>> compact([1, None, 2, False, 3])
[1, 2, False, 3]
>>> compact([1, None, '', False, 2], all=True)
[1, 2]
```

New in version TODO.

sugar.arrays.**construct**(*var*, *callback*)

Constructs an array of `var` length from the values of `callback`.

Parameters

- **var** (*int*) – Length of the array intended.
- **callback** – A method that can take in each variable from the given range and return back a new value based on the method definition.

Returns A list of `callback` values.

Return type list

Example

```
>>> construct(4, lambda x: x * 2)
[0, 2, 4, 6]
```

New in version 0.1.0.

sugar.arrays.**count**(*array*, *value*)

Counts all elements in the `array` that match the given `value`.

Parameters

- **array** (*list*) – A list of values provided by the user to search for.
- **value** (*int/float/str*) – Value that needs to be counted.

Returns Count of the given value.

Return type int

Example

```
>>> count([1, 2, 3, 3], 3)
2
```

New in version 0.1.0.

sugar.arrays.**create**(*obj=None*, *copy=False*)

Creates an array from an unknown object.

Parameters

- **obj** (*mixed*) – Value passed in by the user.
- **copy** (*bool*) – If clone is true, the array will be shallow cloned.

Returns Array from the given object.

Return type list

Example

```
>>> create('abc def 109;cd')
['a', 'b', 'c', ' ', 'd', 'e', 'f', ' ', '1', '0', '9', ';', 'c', 'd']
>>> create(1234)
[1234]
>>> create([11, 22, 33, 44], copy=True)
[11, 22, 33, 44]
>>> create(True)
[True]
>>> create()
[]
```

New in version TODO.

sugar.arrays.**every**(*array*, *value*)

Returns true if search is true for all elements of the array. In other words, this method returns True if `array` contains all the same values `value`.

Parameters

- **array** (*list*) – List of values provided by the user.
- **value** (*int/float/str*) – Value that needs to be searched.

Returns A boolean value based on the `array` having all the values as `value`.

Return type bool

Example

```
>>> every([2, 2, 2], 2)
True
>>> every([2, 2, 3], 2)
False
```

New in version TODO.

sugar.arrays.**exclude**(*array*, *value*)

Returns a new array with every element that does not match `value`.

Parameters

- **array** (*list*) – List of values provided by the user.
- **value** (*int/float/str*) – A value that needs to be excluded.

Returns List excluding the give `value`.

Return type list

Example

```
>>> exclude([11, 22, 33], 22)
[11, 33]
>>> exclude([11, 22, 33], 44)
[11, 22, 33]
>>> exclude([11, 22, 33], [11, 22])
[33]
```

New in version TODO.

sugar.arrays.**filter_**(*array*, *value=None*, *callback=None*)

Returns list of elements in the `array` that match `value`. Also, returns list of elements based on the given callback method.

Parameters

- **array** (*list*) – List of values provided by the user.
- **value** (*int/float/str*) – A value that needs to be matched with.
- **callback** – A method that takes the value, filters the variable based on the given condition and returns the filtered value.

Returns List of values that match with the `value` or the given filter.

Return type list

Example

```
>>> filter_([1, 2, 2, 4], value=2)
[2, 2]
>>> filter_([1, 2, 2, 4], callback=lambda x: x > 1)
[2, 2, 4]
```

New in version TODO.

sugar.arrays.**first**(*array*, *num=1*)

Returns the first element(s) in the array. When num is passed, returns the first num elements in the array.

Parameters

- **array** (*list*) – List of values passed in by the user.
- **num** (*int*) – Number passed in by the user.

Returns Returns an array of first num elements.

Return type list

Example

```
>>> first([11, 22, 33, 44], 1)
[11]
>>> first([11, 22, 33, 44], None)
[]
>>> first([11, 22, 33, 44], -3)
[]
>>> first([11, 22, 33, 44], 9)
[11, 22, 33, 44]
```

New in version TODO.

sugar.arrays.**from_**(*array*, *index=0*)

Returns a slice of the array from index.

Parameters

- **array** (*list*) – A list of values provided by the user.
- **index** (*int*) – Start position of the array where the slice starts.

Returns Array sliced from index).

Return type list

Example

```
>>> from_([11, 22, 33], 1)
[22, 33]
>>> from_([11, 22, 33])
[11, 22, 33]
>>> from_([11, 22, 33], 2)
[33]
```

```
>>> from_([11, 22, 33], None)
[11, 22, 33]
```

New in version TODO.

sugar.arrays.**includes**(*array*, *search*, *fromindex=0*)

Returns true if search is contained within the array. Search begins at fromindex, which defaults to the beginning of the array.

Parameters

- **array** (*list*) – A list of values provided by the user.
- **search** (*mixed*) – A value that needs to be searched (provided by the user).
- **fromindex** (*int*) – Search begins at fromindex.

Returns True if search is contained within the array beginning at fromindex position else False.

Return type bool

Example

```
>>> includes([11, 22, 33], 22, 0)
True
>>> includes([11, 22, 33], 22, 1)
True
>>> includes([11, 22, 33], 22, 2)
False
>>> includes([11, 22, 33], 11, None)
True
>>> includes([11, 22, 33], 33)
True
>>> includes([11, 22, 33], 22, -1)
False
>>> includes([11, 22, 33], 22, -2)
True
```

New in version TODO.

sugar.arrays.**is_empty**(*array*)

Returns True if the array has a length of zero.

Parameters **array** (*list*) – A list of values provided by the user.

Returns True if the list is empty else False.

Return type bool

Example

```
>>> is_empty([])
True
>>> is_empty([None])
False
```

New in version TODO.

sugar.arrays.**is_equal**(*array_one*, *array_two*)

Returns True if array_one is equal to array_two.

Parameters

- **array_one** (*list*) – First list of values provided by the user.
- **array_two** (*list*) – Second list of values provided by the user.

Returns True if both the arrays are equal else False.

Return type bool

Example

```
>>> is_equal([1, 2], [1, 2])
True
>>> is_equal(['1'], [str(1)])
True
>>> is_equal([None], [])
False
>>> is_equal([1, 2], [2, 1])
False
>>> is_equal([], [])
True
```

New in version TODO.

sugar.arrays.**last**(*array*, *num=1*)

Returns the last element(s) in the array. When num is passed, returns the last num elements in the array.

Parameters

- **array** (*list*) – List of values passed in by the user.
- **num** (*int*) – Number passed in by the user.

Returns Returns an array of last num elements.

Return type list

Example

```
>>> last([11, 22, 33, 44], 1)
[44]
>>> last([11, 22, 33, 44], 3)
[22, 33, 44]
>>> last([11, 22, 33, 44], None)
[]
>>> last([11, 22, 33, 44], -3)
[]
>>> last([11, 22, 33, 44], 9)
[]
```

New in version TODO.

sugar.arrays.**some**(*array*, *search*, *callback=None*)

Returns true if search is true for any element in the given array.

Parameters

- **array** (*list*) – List of values passed in by the user.
- **search** (*mixed*) – A value to be searched in the given list.

- **callback** (*func*) – Function that cacn be called on each element in the given list.

Returns True if any of the elements matches the given `search` value else False.

Return type bool

Example

```
>>> some([1, 2, 3], 1)
True
>>> some([1, 2, 3], None, callback=lambda x: x == 1)
True
```

New in version TODO.

`sugar.arrays.`**`subtract`**(*array*, *item*)

Subtracts `item` from the `array` and returns the result as a new array. If `item` is also an array, all elements in it will be removed.

Parameters

- **array** (*list*) – A list of values provided by the user.
- **item** (*list/int/float/str*) – A value that needs to be removed from `array`.

Returns A new list with the `item` removed.

Return type list

Example

```
>>> subtract([1, 2, 3], 2)
[1, 3]
>>> subtract ([1, 2, 3], [1, 3])
[2]
>>> subtract([1, 2, 3], 4)
[1, 2, 3]
```

New in version 0.1.0.

`sugar.arrays.`**`union`**(*array*, *other*)

Returns a new array containing elements in both arrays with duplicates removed.

Parameters

- **array** (*list*) – List passed in by the user.
- **other** (*list*) – Other list passed in by the user to compare.

Returns List of elements without duplicate values.

Return type list

Example

```
>>> union([1, 2, 3], [2, 3, 4])
[1, 2, 3, 4]
>>> union([1, 2, 3], [4, 5, 6])
[1, 2, 3, 4, 5, 6]
```

```
>>> union([1, 2, 3], [1, 2, 3])
[1, 2, 3]
```

New in version TODO.

4.3.2 Numbers

sugar.number.**armstrongs_between**(*n1=None*, *n2=None*)

Get all the armstrong numbers between `n1` and `n2`.

Parameters

- **n1** (*int*) – Number passed in by the user.
- **n2** (*int*) – Number passed in by the user.

Returns List of all the armstrong numbers between `n1`, and `n2`.

Return type list

Example

```
>>> armstrongs_between(0, 999)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
```

New in version TODO.

Note: When you pass in different digit numbers, this method doesn't get the armstrong numbers compared to the highest digit number passed in.

---

sugar.number.**hex_**(*value*, *pad=None*)

Converts a given number to hexi-decimal.

Parameters

- **value** (*int*) – Value passed in by the user.
- **pad** (*int*) – Padding till the result can be restricted.

Returns A hex value that corresponds to the given number.

Return type hex

Example

```
>>> hex_(55)
'0x37'
>>> hex_(555)
'0x22b'
>>> hex_(555, 2)
'0x'
```

New in version TODO.

sugar.number.**is_armstrong**(*num*)

Returns True if `num` is armstrong number.

Parameters **num** (*int*) – Number passed in by the user.

Returns True if the given num is armstrong number else False.

Return type bool

Example

```
>>> is_armstrong(371)
True
>>> is_armstrong(8208)
True
>>> is_armstrong(51)
False
```

New in version TODO.

sugar.number.**is_even**(*num*)

Returns True if num is even.

Parameters **num** (*int/float*) – Number passed in by the user.

Returns True if num is even else False

Return type bool

Example

```
>>> is_even(6)
True
>>> is_even(7)
False
```

New in version TODO.

sugar.number.**is_multiple_of**(*value*, *num*)

Returns true if the value is a multiple of num.

Parameters

- **value** (*int/float*) – Value provided by the user.
- **num** (*int/float*) – Value provided by the user.

Returns True if the value is a multiple of num.

Return type bool

Example

```
>>> is_multiple_of(6, 2)
True
>>> is_multiple_of(5, 2)
False
>>> is_multiple_of(1.5, 3)
False
>>> is_multiple_of(1.5, 0.5)
True
```

New in version TODO.

sugar.number.**is_odd**(*num*)

Returns True if num is odd.

Parameters **num** (*int/float*) – Number passed in by the user.

Returns True if num is odd else False

Return type bool

Example

```
>>> is_odd(6)
False
>>> is_odd(7)
True
```

New in version TODO.

sugar.number.**is_prime**(*num*)

Returns True if the give num is a prime number.

Parameters **num** (*int/float*) – Number passed in by the user.

Returns True if the given num is a prime number else False

Return type bool

Example

```
>>> is_prime(5)
True
>>> is_prime(7)
True
>>> is_prime(4)
False
>>> is_prime(727021)
True
```

New in version TODO.

sugar.number.**primes_between**(*n1=None*, *n2=None*)

Get all the prime numbers between n1 and n2.

Parameters

- **n1** (*int*) – Number passed in by the user.
- **n2** (*int*) – Number passed in by the user.

Returns List of all the prime numbers between n1, and n2.

Return type list

Example

```
>>> primes_between(1, 20)
[1, 2, 3, 5, 7, 11, 13, 17, 19]
>>> primes_between(21, 40)
[23, 29, 31, 37]
```

New in version TODO.

sugar.number.**random_**(*n1=None*, *n2=None*)

Returns a random integer/float from `n1` to `n2` (both inclusive)

Parameters

- **n1** (*int/float/str*) – Value given by the user.
- **n2** (*int/float/str*) – Value given by the user.

Returns Random integer/float value between `n1` and `n2`.

Return type int/float

Example

```
>>> result = random_(5, 6)
>>> assert 5 <= result <= 6
>>> result = random_(5)
>>> assert 0 <= result <= 5
```

New in version TODO.

4.3.3 Predicates

sugar.predicates.**is_array**(*obj*)

Validates whether the given value is a list or not.

Parameters **obj** (*mixed*) – Value passed in by the user.

Returns True if the given object is a list. False if the given object is not a list.

Return type bool

Example

```
>>> is_array([1, 2, 3, 4])
True
>>> is_array('abcd')
False
>>> is_array(1234)
False
```

New in version TODO.

sugar.predicates.**is_boolean**(*obj*)

Validates whether the given object is a boolean or not.

Parameters **obj** (*mixed*) – Value passed in by the user.

Returns True if the given object is a boolean. False if the given object is not a boolean.

Return type bool

Example

```
>>> is_boolean(True)
True
>>> is_boolean('abcd')
False
>>> is_boolean(1234)
False
```

New in version TODO.

sugar.predicates.**is_none**(*value*)
Returns True if the `value` is None.

Parameters **value** (*mixed*) – Value passed in by the user.

Returns True if the given value is None else False

Return type bool

Example

```
>>> is_none(None)
True
>>> is_none([])
False
```

New in version TODO.

sugar.predicates.**is_number**(*value*)
Validates whether the given value is an integer/float.

Parameters **value** (*mixed*) – Value passed in by the user.

Returns True if the given object is a number. False if the given object is not a number.

Return type bool

Example

```
>>> is_number(1234)
True
>>> is_number('abcd')
False
>>> is_number([11, 22, 33, 44])
False
```

New in version TODO.

sugar.predicates.**is_string**(*value*)
Validates whether the given value is a string or not.

Parameters **value** (*mixed*) – Value passed in by the user.

Returns True if the given object is a string. False if the given object is not a string.

Return type bool

Example

```
>>> is_string('abcd')
True
>>> is_string(1234)
False
>>> is_string([11, 22, 33, 44])
False
```

New in version TODO.

4.3.4 Strings

sugar.strings.**at**(*string*, *index=0*, *loop=False*)

Return the character(s) at a given index. When `loop` is true, overshooting the end of the string will begin counting from the other end. `index` may be negative. If `index` is an array, multiple elements will be returned.

Parameters

- **string** (*str*) – String value passed in by the user.
- **index** (*int*) – Index value passed in by the user.
- **loop** (*bool*) – If True, this method overshoots end of string and will begin counting from the other end.

Returns A character/List of characters based on the given index positions.

Return type str/list

Example

```
>>> at('example')
'e'
>>> at('example', 4)
'p'
>>> at('example', 8, True)
'x'
>>> at('example', [4, 8], True)
['p', 'x']
>>> at('example', -4)
'm'
>>> at('example', [4, -4])
['p', 'm']
>>> at('example', [4, -10], True)
['p', 'p']
```

New in version TODO.

sugar.strings.**camelize**(*string*, *upper=True*)

Converts underscores and hyphens to camel case. If `upper` is False, the `string` will be upperCamelCase.

Parameters

- **string** (*str*) – String passed in by the user.
- **upper** (*bool*) – If True, it will return UpperCamelCase else upperCamelCase.

Returns String converted to CamelCase.

Return type str

Example

```
>>> camelize('example')
'Example'
>>> camelize('example-test')
'ExampleTest'
>>> camelize('example_test-one')
'ExampleTestOne'
>>> camelize('example_test-one', False)
'exampleTestOne'
```

New in version TODO.

sugar.strings.**chars**(*string*, *callback=None*)

Runs `callable()` against each character in the string and returns an array.

Parameters

- **string** (*str*) – String passed in by the user.
- **callback** (*func*) – Method to be run against each character in the string.

Returns List of chars after the `callback()` method is applied to all the chars in the given `string`.

Return type list

Example

```
>>> chars('example')
['e', 'x', 'a', 'm', 'p', 'l', 'e']
>>> chars('example', lambda x: 'i' if x == 'e' else x)
['i', 'x', 'a', 'm', 'p', 'l', 'i']
```

New in version TODO.

4.4 Examples

4.4.1 Arrays

```
>>> import sugar as _

>>> _.add([11, 22, 33], 88)
[11, 22, 33, 88]
>>> _.add([11, 22, 33], 88, 1)
[11, 88, 22, 33]
>>> _.add([11, 22, 33], [44, 55])
[11, 22, 33, 44, 55]
>>> _.add([11, 22, 33], [44, 55, 66, 77], 1)
[11, 44, 55, 66, 77, 22, 33]

>>> _.append([11, 22, 33], 88)
[11, 22, 33, 88]
```

```
>>> _.append([11, 22, 33], 88, 1)
[11, 88, 22, 33]
>>> _.append([11, 22, 33], [44, 55])
[11, 22, 33, 44, 55]
>>> _.append([11, 22, 33], [44, 55, 66, 77], 1)
[11, 44, 55, 66, 77, 22, 33]

>>> float(_.average([1, 2, 3]))
2.0

>>> _.clone([1, 2, 3])
[1, 2, 3]

>>> _.compact([1, None, 2, False, 3])
[1, 2, False, 3]
>>> _.compact([1, None, '', False, 2], all=True)
[1, 2]

>>> _.construct(4, lambda x: x * 2)
[0, 2, 4, 6]

>>> _.count([1, 2, 3, 3], 3)
2

>>> _.create('abc def 109;cd')
['a', 'b', 'c', ' ', 'd', 'e', 'f', ' ', '1', '0', '9', ';', 'c', 'd']
>>> _.create(1234)
[1234]
>>> _.create([11, 22, 33, 44], copy=True)
[11, 22, 33, 44]
>>> _.create(True)
[True]
>>> _.create()
[]

>>> _.every([2, 2, 2], 2)
True
>>> _.every([2, 2, 3], 2)
False

>>> _.exclude([11, 22, 33], 22)
[11, 33]
>>> _.exclude([11, 22, 33], 44)
[11, 22, 33]
>>> _.exclude([11, 22, 33], [11, 22])
[33]

>>> _.filter_([1, 2, 2, 4], value=2)
[2, 2]
>>> _.filter_([1, 2, 2, 4], callback=lambda x: x > 1)
[2, 2, 4]

>>> _.first([11, 22, 33, 44], 1)
[11]
>>> _.first([11, 22, 33, 44], None)
[]
>>> _.first([11, 22, 33, 44], -3)
[]
```

```
>>> _.first([11, 22, 33, 44], 9)
[11, 22, 33, 44]

>>> _.from_([11, 22, 33], 1)
[22, 33]
>>> _.from_([11, 22, 33])
[11, 22, 33]
>>> _.from_([11, 22, 33], 2)
[33]
>>> _.from_([11, 22, 33], None)
[11, 22, 33]

>>> _.includes([11, 22, 33], 22, 0)
True
>>> _.includes([11, 22, 33], 22, 1)
True
>>> _.includes([11, 22, 33], 22, 2)
False
>>> _.includes([11, 22, 33], 11, None)
True
>>> _.includes([11, 22, 33], 33)
True
>>> _.includes([11, 22, 33], 22, -1)
False
>>> _.includes([11, 22, 33], 22, -2)
True

>>> _.is_empty([])
True
>>> _.is_empty([None])
False

>>> _.is_equal([1, 2], [1, 2])
True
>>> _.is_equal(['1'], [str(1)])
True
>>> _.is_equal([None], [])
False
>>> _.is_equal([1, 2], [2, 1])
False
>>> _.is_equal([], [])
True

>>> _.last([11, 22, 33, 44], 1)
[44]
>>> _.last([11, 22, 33, 44], 3)
[22, 33, 44]
>>> _.last([11, 22, 33, 44], None)
[]
>>> _.last([11, 22, 33, 44], -3)
[]
>>> _.last([11, 22, 33, 44], 9)
[]

>>> _.some([1, 2, 3], 1)
True
>>> _.some([1, 2, 3], None, callback=lambda x: x == 1)
True
```

```
>>> _.subtract([1, 2, 3], 2)
[1, 3]
>>> _.subtract ([1, 2, 3], [1, 3])
[2]
>>> _.subtract([1, 2, 3], 4)
[1, 2, 3]

>>> _.union([1, 2, 3], [2, 3, 4])
[1, 2, 3, 4]
>>> _.union([1, 2, 3], [4, 5, 6])
[1, 2, 3, 4, 5, 6]
>>> _.union([1, 2, 3], [1, 2, 3])
[1, 2, 3]
```

4.4.2 Numbers

```
>>> import sugar as _

>>> _.armstrongs_between(0, 999)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]

>>> _.hex_(55)
'0x37'
>>> _.hex_(555)
'0x22b'
>>> _.hex_(555, 2)
'0x'

>>> _.is_armstrong(371)
True
>>> _.is_armstrong(8208)
True
>>> _.is_armstrong(51)
False

>>> _.is_even(6)
True
>>> _.is_even(7)
False

>>> _.is_multiple_of(6, 2)
True
>>> _.is_multiple_of(5, 2)
False
>>> _.is_multiple_of(1.5, 3)
False
>>> _.is_multiple_of(1.5, 0.5)
True

>>> _.is_odd(6)
False
>>> _.is_odd(7)
True

>>> _.is_prime(5)
True
```

```
>>> _.is_prime(7)
True
>>> _.is_prime(4)
False
>>> _.is_prime(727021)
True

>>> _.primes_between(1, 20)
[1, 2, 3, 5, 7, 11, 13, 17, 19]
>>> _.primes_between(21, 40)
[23, 29, 31, 37]

>>> result = _.random_(5, 6)
>>> assert 5 <= result <= 6
>>> result = _.random_(5)
>>> assert 0 <= result <= 5
```

4.4.3 Predicates

```
>>> import sugar as _

>>> _.is_none(None)
True
>>> _.is_none([])
False
```

4.4.4 Strings

```
>>> import sugar as _

>>> _.at('example')
'e'
>>> _.at('example', 4)
'p'
>>> _.at('example', 8, True)
'x'
>>> _.at('example', [4, 8], True)
['p', 'x']
>>> _.at('example', -4)
'm'
>>> _.at('example', [4, -4])
['p', 'm']
>>> _.at('example', [4, -10], True)
['p', 'p']

>>> _.camelize('example')
'Example'
>>> _.camelize('example-test')
'ExampleTest'
>>> _.camelize('example_test-one')
'ExampleTestOne',
>>> _.camelize('example_test-one', False)
'exampleTestOne'

>>> _.chars('example')
```

```
['e', 'x', 'a', 'm', 'p', 'l', 'e']
>>> _.chars('example', lambda x: 'i' if x == 'e' else x)
['i', 'x', 'a', 'm', 'p', 'l', 'i']
```

4.5 Upgrading

4.5.1 From v0.1.0

- 16 array methods
  - *sugar.arrays.add()*.
  - *sugar.arrays.append()*.
  - *sugar.arrays.clone()*.
  - *sugar.arrays.compact()*.
  - *sugar.arrays.create()*.
  - *sugar.arrays.every()*.
  - *sugar.arrays.exclude()*.
  - *sugar.arrays.filter_()*.
  - *sugar.arrays.first()*.
  - *sugar.arrays.from_()*.
  - *sugar.arrays.includes()*.
  - *sugar.arrays.is_empty()*.
  - *sugar.arrays.is_equal()*.
  - *sugar.arrays.last()*.
  - *sugar.arrays.some()*.
  - *sugar.arrays.union()*.
- 9 number methods
  - *sugar.number.armstrongs_between()*.
  - *sugar.number.hex_()*.
  - *sugar.number.is_armstrong()*.
  - *sugar.number.is_even()*.
  - *sugar.number.is_multiple_of()*.
  - *sugar.number.is_odd()*.
  - *sugar.number.is_prime()*.
  - *sugar.number.primes_between()*.
  - *sugar.number.random_()*
- 5 predicate methods
  - *sugar.predicates.is_array()*.

  - *sugar.predicates.is_boolean()*.
  - *sugar.predicates.is_none()*.
  - *sugar.predicates.is_number()*.
  - *sugar.predicates.is_string()*.
- 3 string methods
  - *sugar.strings.at()*.
  - *sugar.strings.camelize()*.
  - *sugar.strings.chars()*.

Improvements

The below methods are new apart from Sugar JS utility library:

- *sugar.number.armstrongs_between()*.
- *sugar.number.is_armstrong()*.
- *sugar.number.is_prime()*.
- *sugar.number.primes_between()*.

4.5.2 v0.1.0

- 4 new array methods
  - *sugar.arrays.average()*.
  - *sugar.arrays.construct()*.
  - *sugar.arrays.count()*.
  - *sugar.arrays.subtract()*.

CHAPTER 5

Project Info

5.1 License

MIT License

Copyright (c) 2016, Bharadwaj Yarlagadda

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

5.2 Versioning

This project follows Semantic Versioning.

It is recommended to only use or import objects from the main module, sugar.py.

5.3 Changelog

- Add `add`.
- Add `append`.
- Add `armstrongs_between`.
- Add `at`.
- Add `camelize`.
- Add `chars`.

- Add `clone`.
- Add `compact`.
- Add `create`.
- Add `every`.
- Add `exclude`.
- Add `filter_`.
- Add `first`.
- Add `from_`.
- Add `hex_`.
- Add `includes`.
- Add `is_armstrong`.
- Add `is_array`.
- Add `is_boolean`.
- Add `is_empty`.
- Add `is_equal`.
- Add `is_even`.
- Add `is_none`.
- Add `is_multiple_of`.
- Add `is_number`.
- Add `is_odd`.
- Add `is_prime`.
- Add `is_string`.
- Add `primes_between`.
- Add `last`.
- Add `random_`.
- Add `some`.
- Add `union`.

5.3.1 v0.1.1 (2016-10-10)

- `FIX`: Added description for the package in `setup.py`.
- `FIX`: Added keywords in `setup.py`.

5.3.2 v0.1.0 (2016-10-10)

- First release.
- Add `average`.
- Add `construct`.

- Add `count`.
- Add `subtract`.

5.4 Authors

5.4.1 Lead

- Bharadwaj Yarlagadda, yarlagaddabharadwaj@gmail.com, bharadwajyarlagadda@github

5.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/bharadwajyarlagadda/sugar.py.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" or "help wanted" is open to whoever wants to implement it.

Write Documentation

sugar.py could always use more documentation, whether as part of the official sugar.py docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/bharadwajyarlagadda/sugar.py.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.5.2 Get Started!

Ready to contribute? Here's how to set up `sugar.py` for local development.

1. Fork the `sugar.py` repo on GitHub.
2. Pull your fork locally:

```
$ git clone git@github.com:<username>/sugar.py.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenv installed, this is how you set up your fork for local development:

```
$ cd sugar.py/
$ pip install -r requirements-dev.txt
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass linting and all unit tests by testing with tox across all supported Python versions:

```
$ invoke tox
```

6. Add yourself to `AUTHORS.rst`.
7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

5.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the README.rst.
3. The pull request should work for Python 2.7, 3.4, and 3.5. Check https://travis-ci.org/bharadwajyarlagadda/sugar.py/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

S