
Friend Documentation

Release 0.2.1

Joseph Wright

Jan 21, 2022

Contents:

1	Introduction	1
1.1	What is Friend?	1
2	API Reference	3
2.1	friend package	3
3	Indices and tables	11
	Python Module Index	13
	Index	15

1.1 What is Friend?

Friend is all of those of utilities you keep reimplementing across your Python projects, that all got together and decided to live in harmony.

Maybe you just parsed some YAML that contained some configuration in “snake_case”, and you want to pass this configuration to a `boto3` function which takes the same values but in “PascalCase”. Then you might find `snake_to_pascal_obj` or one of its variants to come in handy.

```
with open('conf.yml') as f:
    conf = yaml.load(f)

ec2 = boto3.resource('ec2')
ec2.create_instances(
    ImageId='ami-12345678',
    BlockDeviceMappings=snake_to_pascal_obj(conf['block_device_mappings']),
    ....
)
```

Or you need to add a retry to that script that keeps breaking your Jenkins job because the corporate proxy fails about 5% of the time. Sure, you can add a `try/except` and wrap it in a `for` loop, but putting the `retryable` decorator on top of that problematic function will do that for you in one configurable line.

```
@retryable(times=5)
def flaky_function():
    status = requests.get('https://service.corp/v2/status').json()
    if 'error' in status:
        send_important_email(status['error'])
```


2.1 friend package

2.1.1 friend.collections module

`friend.collections.select_dict(coll, key, value)`

Given an iterable of dictionaries, return the dictionaries where the values at a given key match the given value. If the value is an iterable of objects, the function will consider any to be a match.

This is especially useful when calling REST APIs which return arrays of JSON objects. When such a response is converted to a Python list of dictionaries, it may be easily filtered using this function.

Parameters

- **coll** (*iter*) – An iterable containing dictionaries
- **key** (*obj*) – A key to search in each dictionary
- **value** (*obj* or *iter*) – A value or iterable of values to match

Returns A list of dictionaries matching the query

Return type `list`

Example

```
>>> dicts = [  
...     {'hi': 'bye'},  
...     {10: 2, 30: 4},  
...     {'hi': 'hello', 'bye': 'goodbye'},  
... ]  
>>> select_dict(dicts, 'hi', 'bye')  
[{'hi': 'bye'}]  
>>> select_dict(dicts, 'hi', ('bye', 'hello'))  
[{'hi': 'bye'}, {'hi': 'hello', 'bye': 'goodbye'}]
```

2.1.2 friend.net module

`friend.net.random_ipv4(cidr='10.0.0/8')`

Return a random IPv4 address from the given CIDR block.

Key str cidr CIDR block

Returns An IPv4 address from the given CIDR block

Return type `ipaddress.IPv4Address`

2.1.3 friend.strings module

Utility functions related to strings.

`friend.strings.camel_to_kebab(stringue)`

Convert a “camel case” string to a “kebab case” string.

Parameters `stringue (str)` – The string to convert

Returns A kebab case string

Return type `str`

Example

```
>>> camel_to_kebab('camelCaseString')
'camel-case-string'
```

`friend.strings.camel_to_kebab_obj(obj)`

Take a dictionary with string keys and recursively convert all keys from “camel case” to “kebab case”.

The dictionary may contain lists as values, and any nested dictionaries within those lists will also be converted.

Parameters `obj (object)` – The object to convert

Returns A new object with keys converted

Return type `object`

Example

```
>>> obj = {
...     'dictList': [
...         {'oneKey': 123, 'twoKey': 456},
...         {'threeKey': 789, 'fourKey': 456},
...     ],
...     'someOtherKey': 'someUnconvertedValue'
... }
>>> camel_to_kebab_obj(obj)
{
    'dict-list': [
        {'one-key': 123, 'two-key': 456},
        {'four-key': 456, 'three-key': 789}
    ],
    'some-other-key': 'someUnconvertedValue'
}
```

`friend.strings.camel_to_snake(stringue)`

Convert a “camel case” string to a “snake case” string.

Parameters `stringue (str)` – The string to convert

Returns A snake case string

Return type `str`

Example

```
>>> camel_to_snake('camelCaseString')
'camel_case_string'
```

`friend.strings.camel_to_snake_obj(obj)`

Take a dictionary with string keys and recursively convert all keys from “camel case” to “snake case”.

The dictionary may contain lists as values, and any nested dictionaries within those lists will also be converted.

Parameters `obj (object)` – The object to convert

Returns A new object with keys converted

Return type `object`

Example

```
>>> obj = {
...     'dictList': [
...         {'oneKey': 123, 'twoKey': 456},
...         {'threeKey': 789, 'fourKey': 456},
...     ],
...     'someOtherKey': 'someUnconvertedValue'
... }
>>> camel_to_snake_obj(obj)
{
    'dict_list': [
        {'one_key': 123, 'two_key': 456},
        {'four_key': 456, 'three_key': 789}
    ],
    'some_other_key': 'someUnconvertedValue'
}
```

`friend.strings.format_obj_keys(obj, formatter)`

Take a dictionary with string keys and recursively convert all keys from one form to another using the formatting function.

The dictionary may contain lists as values, and any nested dictionaries within those lists will also be converted.

Parameters

- `obj (object)` – The object to convert
- `formatter (function)` – The formatting function for keys, which takes and returns a string

Returns A new object with keys converted

Return type `object`

Example

```
>>> obj = {
...     'dict-list': [
...         {'one-key': 123, 'two-key': 456},
...         {'threeKey': 789, 'four-key': 456},
...     ],
...     'some-other-key': 'some-unconverted-value'
... }
```

```
... }
>>> format_obj_keys(obj, lambda s: s.upper())
{
  'DICT-LIST': [
    {'ONE-KEY': 123, 'TWO-KEY': 456},
    {'FOUR-KEY': 456, 'THREE-KEY': 789}
  ],
  'SOME-OTHER-KEY': 'some-unconverted-value'
}
```

`friend.strings.kebab_to_camel(stringue)`

Convert a “kebab case” string to a “camel case” string.

Parameters `stringue` (*str*) – The string to convert

Returns A camel case string

Return type *str*

Example

```
>>> kebab_to_camel('kebab-to-camel')
'kebabToCamel'
```

`friend.strings.kebab_to_camel_obj(obj)`

Take a dictionary with string keys and recursively convert all keys from “kebab case” to “camel case”.

The dictionary may contain lists as values, and any nested dictionaries within those lists will also be converted.

Parameters `obj` (*object*) – The object to convert

Returns A new object with keys converted

Return type *object*

Example

```
>>> obj = {
...     'dict-list': [
...         {'one-key': 123, 'two-key': 456},
...         {'three-key': 789, 'four-key': 456},
...     ],
...     'some-other-key': 'some-unconverted-value',
... }
>>> kebab_to_camel_obj(obj)
{
  'dictList': [
    {'oneKey': 123, 'twoKey': 456},
    {'fourKey': 456, 'threeKey': 789}
  ],
  'someOtherKey': 'some-unconverted-value'
}
```

`friend.strings.kebab_to_pascal(stringue)`

Convert a “kebab case” string to a “pascal case” string.

Parameters `stringue` (*str*) – The string to convert

Returns A pascal case string

Return type *str*

Example

```
>>> kebab_to_pascal('kebab-to-pascal')
'KebabToPascal'
```

`friend.strings.kebab_to_pascal_obj(obj)`

Take a dictionary with string keys and recursively convert all keys from “kebab case” to “pascal case”.

The dictionary may contain lists as values, and any nested dictionaries within those lists will also be converted.

Parameters `obj` (*object*) – The object to convert

Returns A new object with keys converted

Return type `object`

Example

```
>>> obj = {
...     'dict-list': [
...         {'one-key': 123, 'two-key': 456},
...         {'three-key': 789, 'four-key': 456},
...     ],
...     'some-other-key': 'some-unconverted-value',
... }
>>> kebab_to_pascal_obj(obj)
{
    'DictList': [
        {'OneKey': 123, 'TwoKey': 456},
        {'FourKey': 456, 'ThreeKey': 789}
    ],
    'SomeOtherKey': 'some-unconverted-value'
}
```

`friend.strings.random_alphanum(length)`

Return a random string of ASCII letters and digits.

Parameters `length` (*int*) – The length of string to return

Returns A random string

Return type `str`

`friend.strings.random_hex(length)`

Return a random hex string.

Parameters `length` (*int*) – The length of string to return

Returns A random string

Return type `str`

`friend.strings.random_string(length, charset)`

Return a random string of the given length from the given character set.

Parameters

- `length` (*int*) – The length of string to return
- `charset` (*str*) – A string of characters to choose from

Returns A random string

Return type `str`

`friend.strings.snake_to_camel(stringue)`

Convert a “snake case” string to a “camel case” string.

Parameters `stringue` (*str*) – The string to convert

Returns A camel case string

Return type `str`

Example

```
>>> snake_to_camel('snake_to_camel')
'snakeToCamel'
```

`friend.strings.snake_to_camel_obj(obj)`

Take a dictionary with string keys and recursively convert all keys from “snake case” to “camel case”.

The dictionary may contain lists as values, and any nested dictionaries within those lists will also be converted.

Parameters `obj` (*object*) – The object to convert

Returns A new object with keys converted

Return type `object`

Example

```
>>> obj = {
...     'dict_list': [
...         {'one_key': 123, 'two_key': 456},
...         {'three_key': 789, 'four_key': 456},
...     ],
...     'some_other_key': 'some_unconverted_value',
... }
>>> snake_to_camel_obj(obj)
{
    'dictList': [
        {'onekey': 123, 'twoKey': 456},
        {'fourKey': 456, 'threeKey': 789}
    ],
    'someOtherKey': 'some_unconverted_value'
}
```

`friend.strings.snake_to_pascal(stringue)`

Convert a “snake case” string to a “pascal case” string.

Parameters `stringue` (*str*) – The string to convert

Returns A pascal case string

Return type `str`

Example

```
>>> snake_to_pascal('snake_to_pascal')
'SnakeToPascal'
```

`friend.strings.snake_to_pascal_obj(obj)`

Take a dictionary with string keys and recursively convert all keys from “snake case” to “pascal case”.

The dictionary may contain lists as values, and any nested dictionaries within those lists will also be converted.

Parameters `obj` (*object*) – The object to convert

Returns A new object with keys converted

Return type `object`

Example

```
>>> obj = {
...     'dict_list': [
...         {'one_key': 123, 'two_key': 456},
...         {'three_key': 789, 'four_key': 456},
...     ],
...     'some_other_key': 'some_value'
... }
>>> snake_to_pascal_obj(obj)
{
    'DictList': [
        {'OneKey': 123, 'TwoKey': 456},
        {'FourKey': 456, 'ThreeKey': 789}
    ],
    'SomeOtherKey': 'some_value'
}
```

2.1.4 friend.utils module

exception `friend.utils.IncompleteEnvironment` (*message, variables*)

Bases: `exceptions.RuntimeError`

An exception which is used to indicate when environment variables are unset.

`friend.utils.cached` (*func*)

A decorator function to cache values. It uses the decorated function's arguments as the keys to determine if the function has been called previously.

`friend.utils.ensure_environment` (*variables*)

Check `os.environ` to ensure that a given collection of variables has been set.

Parameters **variables** – A collection of environment variable names

Returns `os.environ`

Raises `IncompleteEnvironment` – if any variables are not set, with the exception's `variables` attribute populated with the missing variables

`friend.utils.retry_bool` (*callback, times=3, cap=120000*)

Retry a callback function if it returns False.

Parameters

- **callback** (*function*) – The function to call
- **times** (*int*) – Number of times to retry on initial failure
- **cap** (*int*) – Maximum wait time in milliseconds

Returns The return value of the callback

Return type `bool`

`friend.utils.retry_ex` (*callback, times=3, cap=120000*)

Retry a callback function if any exception is raised.

Parameters

- **callback** (*function*) – The function to call
- **times** (*int*) – Number of times to retry on initial failure
- **cap** (*int*) – Maximum wait time in milliseconds

Returns The return value of the callback

Raises **Exception** – If the callback raises an exception after exhausting all retries

`friend.utils.retry_wait_time(attempt, cap)`

Determine a retry wait time based on the number of the retry attempt and a cap on the wait time. The wait time uses an exponential backoff with a random jitter. The algorithm used is explained at <https://www.awsarchitectureblog.com/2015/03/backoff.html>.

Parameters

- **attempt** (*int*) – The number of the attempt
- **cap** (*int*) – A cap on the wait time in milliseconds

Returns The number of milliseconds to wait

Return type `int`

`friend.utils.retryable(retryer=<function retry_ex>, times=3, cap=120000)`

A decorator to make a function retry. By default the retry occurs when an exception is thrown, but this may be changed by modifying the `retryer` argument.

See also `retry_ex()` and `retry_bool()`. By default `retry_ex()` is used as the retry function.

Note that the decorator must be called even if not given keyword arguments.

Parameters

- **retryer** (*function*) – A function to handle retries
- **times** (*int*) – Number of times to retry on initial failure
- **cap** (*int*) – Maximum wait time in milliseconds

Example

```
@retryable()
def can_fail():
    ....

@retryable(retryer=retry_bool, times=10)
def can_fail_bool():
    ....
```

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

f

`friend.collections`, 3

`friend.net`, 4

`friend.strings`, 4

`friend.utils`, 9

C

cached() (in module friend.utils), 9
camel_to_kebab() (in module friend.strings), 4
camel_to_kebab_obj() (in module friend.strings), 4
camel_to_snake() (in module friend.strings), 4
camel_to_snake_obj() (in module friend.strings), 5

E

ensure_environment() (in module friend.utils), 9

F

format_obj_keys() (in module friend.strings), 5
friend.collections (module), 3
friend.net (module), 4
friend.strings (module), 4
friend.utils (module), 9

I

IncompleteEnvironment, 9

K

kebab_to_camel() (in module friend.strings), 6
kebab_to_camel_obj() (in module friend.strings), 6
kebab_to_pascal() (in module friend.strings), 6
kebab_to_pascal_obj() (in module friend.strings), 7

R

random_alphanum() (in module friend.strings), 7
random_hex() (in module friend.strings), 7
random_ipv4() (in module friend.net), 4
random_string() (in module friend.strings), 7
retry_bool() (in module friend.utils), 9
retry_ex() (in module friend.utils), 9
retry_wait_time() (in module friend.utils), 10
retryable() (in module friend.utils), 10

S

select_dict() (in module friend.collections), 3
snake_to_camel() (in module friend.strings), 7

snake_to_camel_obj() (in module friend.strings), 8
snake_to_pascal() (in module friend.strings), 8
snake_to_pascal_obj() (in module friend.strings), 8