mjooln

Release alpha 0.6.0.1

Nov 04, 2022

Contents

About mjooln

mjooln [*no: "mjøln"; en: "myawln"*] is a file handling toolbox for Microservice Developers and Data Scientists working in Python

- **Source code**: mjooln

CHAPTER 1

Getting started

1.1 What is mjooln?

mjooln [*no: "mjøln"; en: "myawln"*] is a file handling toolbox for Microservice Developers and Data Scientists working in Python

1.1.1 Background

Development was motivated by writing `os.path.join()` one time too many, the usefulness of simplified read/write of encrypted configuration files and the never ending issue of datetime and timezones

Also, despite Data Lakes, Delta Lakes, Lakehouses and Object Stores, there always seems to be some stage where a lot of csv-files have to be compressed and stored in a decent folder structure in the local filesystem

1.1.2 Overview

File and *Folder* facilitate file and folder handling, as well as read/write – with or without compression/encryption

Zulu extends `datetime.datetime` with convenience methods, and is timezone aware and *always* UTC

Dic mirrors object attributes to/from a dictionary, while *Doc* mirrors object attributes to/from JSON and YAML

1.2 Installation

Install Python 3.6 or later.

Install mjooln using pip in your Python 3 virtual environment

```
$ pip install mjooln
```

Or use pip3 to make sure you are on the right Python version

```
$ pip3 install mjooln
```

1.3 Examples

Note: This page is still work in progress, but there are some examples in the API docs

1.3.1 Quick Start

Note: To be continued...

1.3.2 Snippets

Note: To be continued...

The following import is assumed in all examples

```
import mjooln as mj
```

Read text from file

```
contents = mj.File('/path/to/file/my_file.txt').read()
```

Write text to file

```
mj.File('/path/to/file/my_file.txt').write(contents)
```

Write dictionary to JSON file

```
mj.File('/path/to/file/my_dictionary.json').write_json(di)
```

1.3.3 Advanced

Note: To be continued...

Create an encrypted access data file for S3

```
import mjooln as mj
```

CHAPTER 2

API

2.1 API

2.1.1 Core

Atom

class mjooln.**Atom**(*key*, *zulu: mjooln.Zulu = None*, *identity: mjooln.Identity = None*)

Triplet identifier intended for objects and data sets alike

Format: `<zulu>___<key>___<identity>`

Zulu represents t0 or creation time

Key defines grouping of the contents

Identity is a unique identifier for the contents

Constructor initializes a valid atom, and will raise an `AtomError` if a valid atom cannot be created based on input parameters.

The constructor must as minimum have *Key* as input, although string version (seed) of key is allowed:

```
atom = Atom('zaphod__ship_33__inventory')
atom.key()
    'zaphod__ship_33__inventory'
atom.zulu()
    Zulu(2020, 5, 22, 13, 13, 18, 179169, tzinfo=<UTC>)
atom.identity()
    '060AFBD5_D865_4974_8E37_FDD5C55E7CD8'
```

Output methods:

```
atom
    Atom('zaphod__ship_33__inventory', zulu=Zulu(2020, 5, 22, 13, 13, 18, 179169),
↪ identity=Identity('060AFBD5_D865_4974_8E37_FDD5C55E7CD8'))
```

```
str(atom)
    '20200522T131318u179169Z___zaphod__ship_33__inventory___060AFBD5_D865_4974_
↪8E37_FDD5C55E7CD8'

atom.seed()
    '20200522T131318u179169Z___zaphod__ship_33__inventory___060AFBD5_D865_4974_
↪8E37_FDD5C55E7CD8'

atom.to_dict()
    {
        'zulu': Zulu(2020, 5, 22, 13, 13, 18, 179169),
        'key': Key('zaphod__ship_33__inventory'),
        'identity': Identity('060AFBD5_D865_4974_8E37_FDD5C55E7CD8')
    }
```

Atom inherits *Doc* and therefore has a `to_doc()` method:

```
atom.to_doc()
    {
        'zulu': '2020-05-22T13:13:18.179169+00:00',
        'key': 'zaphod__ship_33__inventory',
        'identity': '060AFBD5_D865_4974_8E37_FDD5C55E7CD8'
    }
```

The `to_doc()` is used for output to the equivalent `to_json()` and `to_yaml()`, with equivalent methods for creating an instance from `dict`, doc or a JSON or YAML file.

When storing an atom as part of another dictionary, the most compact method would however be `seed`, unless readability is of importance.

date_elements(*elements=3*)

Get selected date elements

Intented usage is creating sub folders for files with atom naming

Examples:

```
atom.date_elements(None)
    ['20200522']
atom.element_count(None)
    1

atom.date_elements(0)
    []
atom.element_count(0)
    0

atom.date_elements(2)
    ['2020', '05']
atom.element_count(2)
    2

atom.date_elements(-2)
    ['202005']
atom.element_count(-2)
    1
```

Parameters **elements** – Elements

Returns Elements

Return type *list*

classmethod element_count(*elements: int = None*)
Count number of elements represented by element input

For examples, see:

Atom.key_elements() Atom.date_elements() Atom.time_elements()

Parameters **elements** – Element parameter

Return type int

classmethod elf(**args*, ***kwargs*)
Attempts to create an atom based on the input arguments

> **Warning:** Elves are fickle

Raises **AngryElf** – If input arguments cannot be converted to Atom

Return type *Atom*

classmethod from_dict(*di: dict*)
Create *Atom* from input dictionary

Parameters **di** – Input dictionary

Return type *Atom*

classmethod from_doc(*doc: dict*)
Create Atom from serializable dictionary

Parameters **doc** – Dictionary with serialized objects

Return type *Atom*

classmethod from_seed(*seed: str*)
Creates an Atom from a seed string

Parameters **seed** – A valid atom seed string

Return type *Atom*

identity()
Get Atom Identity

Return type *Identity*

key()
Get Atom Key

Return type *Key*

key_elements(*elements=None*)
Get selected key elements

Intented usage is creating sub folders for files with atom naming

Examples:

```
atom.key_elements(None)
    ['zaphod__ship_33__inventory']
atom.element_count(None)
    1

atom.key_elements(0)
    []
atom.element_count(0)
    0

atom.key_elements(2)
    ['zaphod', 'ship_33']
atom.element_count(2)
    2

atom.key_elements(-2)
    ['zaphod__ship_33']
atom.element_count(-2)
    1
```

Parameters **elements** – Elements

Returns Elements

Return type *list*

time_elements(*elements=0*)

Get selected time elements

Intented usage is creating sub folders for files with atom naming

Examples:

```
atom.time_elements(None)
    ['131318']
atom.element_count(None)
    1

atom.time_elements(0)
    []
atom.element_count(0)
    0

atom.time_elements(2)
    ['13', '13']
atom.element_count(2)
    2

atom.time_elements(-2)
    ['1313']
atom.element_count(-2)
    1
```

Parameters **elements** – Elements

Returns Elements

Return type *list*

to_dict(*ignore_private: bool = True*, *recursive: bool = False*)
Get Atom dict

Example from class documentantion:

```
atom.to_dict()
    {
        'zulu': Zulu(2020, 5, 22, 13, 13, 18, 179169),
        'key': Key('zaphod__ship_33__inventory'),
        'identity': Identity('060AFBD5_D865_4974_8E37_FDD5C55E7CD8')
    }
```

Parameters

- **ignore_private** – Ignore private attributes (not relevant)
- **recursive** – Recursive dicts (not relevant)

Return type dict

to_doc(*ignore_private: bool = True*)
Get Atom as a serializable dictionary

Example from class documentantion:

```
atom.to_doc()
    {
        'zulu': '2020-05-22T13:13:18.179169+00:00',
        'key': 'zaphod__ship_33__inventory',
        'identity': '060AFBD5_D865_4974_8E37_FDD5C55E7CD8'
    }
```

Parameters

- **ignore_private** – Ignore private attributes (not relevant)
- **recursive** – Recursive dicts (not relevant)

Return type dict

with_sep(*sep: str*)
Atom seed string with custom separator

Example:

```
atom.with_sep('/')
    '20200522T131318u179169Z/zaphod__ship_33__inventory/060AFBD5_D865_4974_
↪8E37_FDD5C55E7CD8'
```

Parameters **sep** – Custom separator

Return type str

zulu()
Get Atom Zulu

Return type *Zulu*

Crypt

class mjooln.**Crypt**
Wrapper for best practice key generation and AES 128 encryption

From Fernet Docs: HMAC using SHA256 for authentication, and PKCS7 padding. Uses AES in CBC mode with a 128-bit key for encryption, and PKCS7 padding.

classmethod decrypt(*data: bytes*, *key: bytes*)
Decrypts input data with the given key

Returns bytes

classmethod encrypt(*data: bytes*, *key: bytes*)
Encrypts input data with the given key

Returns bytes

classmethod generate_key()
Generates URL-safe base64-encoded random key with length 44

classmethod key_from_password(*salt: bytes*, *password: str*)
Generates URL-safe base64-encoded random string with length 44

Returns bytes

classmethod salt()
Generates URL-safe base64-encoded random string with length 24

Returns bytes

Dic

class mjooln.**Dic**(**args*, ***kwargs*)
Enables child classes to mirror attributes and dictionaries

Private variables start with underscore, and are ignored by default.

Note: Meant for inheritance and not direct use, but can be initialized with a dictionary and will then serve as a struct, where keys can be accessed using dot notation

---

Direct use example:

```
dic = Dic(a=1, b=2, c='three')
dic.to_dict()
    {'a': 1, 'b': 2, 'c': 'three'}
dic.a
    1
dic.b
    2
dic.c
    'three'

dic.c = 'four'
dic.to_dict()
    {'a': 1, 'b': 2, 'c': 'four'}
```

add(*dic: dict*, *ignore_private: bool = True*)
Add dictionary to class as attributes

Parameters

- **dic** – Dictionary to add
- **ignore_private** – Ignore private attributes flag

Returns None

add_only_existing(*dic*, *ignore_private=True*)
Add dictionary keys and items as attributes if they already exist as attributes

Parameters

- **dic** – Dictionary to add
- **ignore_private** – Ignore private attributes flag

Returns None

classmethod flatten(*di: dict*, *sep='__'*)
Flattens input dictionary with given separator :param di: Input dictionary :param sep: Separator (default is '__') :return: Flattened dictionary :rtype: dict

force_equal(*dic*, *ignore_private=True*)
Add all dictionary keys and items as attributes in object, and delete existing attributes that are not keys in the input dictionary

Parameters

- **dic** – Dictionary to add
- **ignore_private** – Ignore private attributes flag

Returns None

classmethod from_dict(*di: dict*)
Create a new object from input dictionary

print(*ignore_private=True*, *indent=' '*, *width=80*, *flatten=False*, *separator='__'*)
Pretty print object attributes in terminal

Parameters

- **ignore_private** – Ignore private variables flag
- **indent** – Spacing for sub dictionaries
- **width** – Target width of printout
- **flatten** – Print as joined keys
- **separator** – Key separator when flattening

to_dict(*ignore_private: bool = True*, *recursive: bool = False*)
Return dictionary with a copy of attributes

Parameters **ignore_private** – Ignore private attributes flag

Returns dict

to_flat(*sep='__'*, *ignore_private=True*)

Flatten dictionary to top elements only by combining keys of sub dictionaries with the given separator

Parameters

- **sep** (*str*) – Separator to use, default is double underscore (__)

- **ignore_private** – Flags whether to ignore private attributes, identified by starting with underscore

Returns Flattened dictionary

Return type dict

classmethod unflatten(*di_flat: dict*, *sep='__'*)
Unflattens input dictionary using the input separator to split into sub dictionaries :param di_flat: Input dictionary :param sep: Separator (default is '__') :return: Dictionary :rtype: dict

Doc

class mjooln.**Doc**(**args*, ***kwargs*)
Enables child classes to mirror attributes, dictionaries, JSON and YAML

Note: `to_doc` and `from_doc` are meant to be overridden in child class if attributes are not serializable. Both methods are used by JSON and YAML conversions

---

add_json(*json_string: str*, *ignore_private=True*)
Convert input JSON string to dictionary and add to current object

Parameters **json_string** – JSON string

Returns Doc

add_yaml(*yaml_string: str*, *ignore_private=True*)
Convert input YAML string to dictionary and add to current object

Parameters **yaml_string** – YAML string

Returns Doc

classmethod from_doc(*doc: dict*)
Convert input dictionary to correct types and return object

Note: Override in child class to handle custom types

Parameters **doc** – Dictionary with serializable items only

Returns New Doc object instantiated with input dictionary

Return type *Doc*

classmethod from_json(*json_string: str*)
Create *Doc* from input JSON string :param json_string: JSON string :return: Doc

classmethod from_yaml(*yaml_string: str*)
Create *Doc* from input YAML string :param yaml_string: YAML string :return: Doc

to_doc()
Converts class attributes to dictionary of serializable attributes

..note:: Override in child class to handle custom types

Parameters **ignore_private** – Ignore private flag

Returns Dictionary of serialized objects

to_json(*human: bool = False*)
Convert object to JSON string :param human: Use human readable format :return: JSON string :rtype: str

to_yaml()
Convert object to YAML string :return: YAML string :rtype: str

Identity

class mjooln.**Identity**(*identity: str = None*)
UUID string generator with convenience functions

Inherits str, and is therefore an immutable string, with a fixed format as illustrated below.

Examples:

```
Identity()
    'BD8E446D_3EB9_4396_8173_FA1CF146203C'

Identity.is_in('Has BD8E446D_3EB9_4396_8173_FA1CF146203C within')
    True

Identity.find_one('Has BD8E446D_3EB9_4396_8173_FA1CF146203C within')
    'BD8E446D_3EB9_4396_8173_FA1CF146203C'
```

classmethod elf(*input*)
Try to create an identity based on input

Raises **AngryElf** – If an identity cannot be created

Return type *Identity*

classmethod from_classic(*classic: str*)
Create Identity from classic format uuid

Return type *Identity*

classmethod from_compact(*compact: str*)
Create identity from compact format uuid

Return type *Identity*

classmethod from_seed(*seed: str*)
Create Identity from seed string

Return type *Identity*

classmethod is_classic(*classic: str*)
Check if string is uuid on classic format

Return type bool

classmethod is_compact(*compact: str*)
Check if string is compact format uuid

Return type bool

JSON

class mjooln.**JSON**
Dict to/from JSON string, with optional human readable

classmethod dumps(*di*, *human=True*, *sort_keys=False*, *indent=' '*)
Convert from dict to JSON string

Parameters

- **di** (*dict*) – Input dictionary
- **human** – Human readable flag
- **sort_keys** – Sort key flag (human readable only)
- **indent** – Indent to use (human readable only)

Returns JSON string

Return type str

classmethod loads(*json_string*)
Parse JSON string to dictionary

Parameters **json_string** (*str*) – JSON string

Returns Dictionary

Return type dict

Key

class mjooln.**Key**(*key: str*)
Defines key string with limitations:

- Minimum length is 2
- Allowed characters are:
  - Lower case ascii (a-z)
  - Digits (0-9)
  - Underscore (_)
  - Double underscore (__)
- Underscore and digits can not be the first character
- Underscore can not be the last character
- The double underscore act as separator for *Word* in the key
- Triple underscore is reserved for separating keys from other keys or seeds, such as in class *Atom*

Sample keys:

```
'simple'
'with_longer_name'
'digit1'
'longer_digit2'
'word_one__word_two__word_three'
'word1__word2__word3'
'word_1__word_2__word_3'
```

classmethod elf(*key*)
Attempts to create a valid key based on the input

> **Warning:** Elves are fickle

Raises **AngryElf** – If a valid key cannot be created

Parameters **key** (*str or Key*) – Input key string or key class

Returns Key

classmethod verify_key(*key: str*)
Verify that string is a valid key

Parameters **key** – String to check

Returns True if string is valid key, False if not

with_separator(*separator: str*)
Replace separator

Example:

```
key = Key('some__key_that_could_be__path')
key.with_separator('/')
    'some/key_that_could_be/path'
```

Parameters **separator** (*str*) – Separator of choice

Returns str

words()
Return list of words in key

Example:

```
key = Key('some_key__with_two__no_three_elements')
key.words()
    [Word('some_key'), Word('with_two'), Word('three_elements')]
key.words()[0]
    Word('some_key')
```

Returns [*Word*]

Seed

class mjooln.**Seed**
Convenience methods for unique string representation of an object

Object can be created with the method `from_seed()`, but the method must be overridden in child class. `find` methods use the class variable `REGEX`, which must also be overridden in child class

If the seed has a fixed length, this can be specified in the class variable `LENGTH`, and will speed up identification (or will it...)

classmethod find_seed(*str_: str*)
Looks for and returns exactly one object from text

Uses `from_seed()` to instantiate object from seed and will fail if there are none or multiple seeds. Use find_all() to return a list of identities in text, including an empty list if there are none

Raises ***BadSeed*** – If none or multiple seeds are found in string

Parameters **str** (*str*) – String to search for seed

Returns Seed object

classmethod find_seeds(*str_: str*)
Finds and returns all seeds in text

Returns List of objects

classmethod from_seed(*str_: str*)
Must be overridden in child class.

Will create an object from seed

Parameters **str** – Seed

Returns Instance of child class

classmethod is_seed(*str_: str*)
Checks if input string is an exact match for seed

Parameters **str** – Input string

Returns True if input string is seed, False if not

seed()
Get seed of current object.

Default is `str(self)`

Returns *Seed*

classmethod seed_in(*str_: str*)
Check if input string contains one or more seeds

Parameters **str** (*str*) – String to check

Returns True if input string contains one or more seeds, false if not

classmethod verify_seed(*str_: str*)
Check if string is seed

Raises ***BadSeed*** – If string is not seed

Parameters **str** – Seed to verify

Waiter

class mjooln.**Waiter**(*keyboard_interrupt=True*)
Convenience class for waiting or sleeping

come()
Abort *wait()*

classmethod sleep(*seconds*)
Simple sleep

Parameters **seconds** – Seconds to sleep

wait(*seconds*)
Sleeps for the given time, can be aborted with *come()* and exits gracefully with keyboard interrupt

Parameters **seconds** (*float*) – Seconds to wait

Returns True if interrupted, False if not

Return type bool

Word

class mjooln.**Word**(*word: str*)

Defines a short string with limitations

- Minimum length is set in Environment with default 1
- Empty word is `n_o_n_e`
- Allowed characters are
  - Lower case ascii `a-z`
  - Digits `0-9`
  - Underscore _
- Underscore and digits can not be the first character
- Underscore can not be the last character
- Can not contain double underscore since it acts as separator for words in *Key*

Sample words:

```
'simple'
'with_longer_name'
'digit1'
'longer_digit2'
```

classmethod check(*word: str*)

Check that string is a valid word

Parameters **word** (*str*) – String to check

Returns True if `word` is valid word, False if not

Return type bool

classmethod elf(*word*)

Attempts to interpret input as a valid word

Raises **AngryElf** – If input cannot be interpreted as Word

Parameters **word** (*str or Word*) – Input word string or word class

Return type *Word*

increment()

Create a new word with index incremented

Example:

```
word = Word('my_word_2')
word.increment()
    Word('my_word_3')
```

Return type *Word*

index()
Get index of word

Raises ***BadWord*** – If word is an integer and thus cannot have an index

Returns 0 if word has no index, otherwise returns index

Return type int

is_int()
Check if word is an integer

Return type bool

is_none()
Check if word is `n_o_n_e`, i.e. word representation of `None`

Return type bool

is_numeric()
Check if word is numeric, i.e. can be converted to integer

Return type bool

classmethod none()
Return Word repesentation of `None`

Returns `n_o_n_e`

Return type *Word*

to_int()
Convert word to integer

Raises ***NotAnInteger*** – If word is not an integer

Return type int

YAML

class mjooln.**YAML**

classmethod dumps(*di: dict*)
Convert dictionary to YAML string

Parameters **di** (*dict*) – Input dictionary

Returns YAML string

Return type str

classmethod loads(*yaml_str*)
Convert YAML string to dictionary

Parameters **yaml_str** (*str*) – Input YAML string

Returns Dictionary

Return type dict

Zulu

class mjooln.**Zulu**

Timezone aware datetime objects in UTC

Create using constructor:

```
Zulu() or Zulu.now()
    Zulu(2020, 5, 21, 20, 5, 31, 930343)

Zulu(2020, 5, 12)
    Zulu(2020, 5, 12)

Zulu(2020, 5, 21, 20, 5, 31)
    Zulu(2020, 5, 21, 20, 5, 31)
```

Seed.seed() is inherited from *Seed* and returns a string on the format <date>T<time>u<microseconds>Z, and is 'designed' to be file name and double click friendly, as well as easily recognizable within some string when using regular expressions. Printing a Zulu object returns seed, and Zulu can be created using *from_seed()*:

```
z = Zulu(2020, 5, 12)
print(z)
    20200512T000000u000000Z

z.seed()
    '20200512T000000u000000Z'

str(z)
    '20200512T000000u000000Z'

Zulu.from_seed('20200512T000000u000000Z')
    Zulu(2020, 5, 12)
```

For an ISO 8601 formatted string, use custom function:

```
z = Zulu('20200521T202041u590718Z')
z.iso()
    '2020-05-21T20:20:41.590718+00:00'
```

Similarly, Zulu can be created from ISO string:

```
Zulu.from_iso('2020-05-21T20:20:41.590718+00:00')
    Zulu(2020, 5, 21, 20, 20, 41, 590718)
```

Inputs or constructors may vary, but Zulu objects are *always* UTC. Hence the name Zulu.

Constructor also takes regular datetime objects, provided they have timezone info:

```
dt = datetime.datetime(2020, 5, 23, tzinfo=pytz.utc)
Zulu(dt)
    Zulu(2020, 5, 23, 0, 0, tzinfo=<UTC>)

dt = datetime.datetime(2020, 5, 23, tzinfo=dateutil.tz.tzlocal())
Zulu(dt)
    Zulu(2020, 5, 22, 22, 0, tzinfo=<UTC>)
```

Zulu has element access like datetime, in addition to string convenience attributes:

```
z = Zulu()
print(z)
    20200522T190137u055918Z
z.month
    5
z.str.month
    '05'
z.str.date
    '20200522'
z.str.time
    '190137'
```

Zulu has a method *delta()* for timedelta, as well as *add()* for adding timedeltas directly to generate a new Zulu:

```
Zulu.delta(hours=1)
    datetime.timedelta(seconds=3600)

z = Zulu(2020, 1, 1)
z.add(days=2)
    Zulu(2020, 1, 3)
```

For more flexible ways to create a Zulu object, see *Zulu.elf()*

add(*days=0*, *hours=0*, *minutes=0*, *seconds=0*, *microseconds=0*, *weeks=0*)
Adds the input to current Zulu object and returns a new one

Parameters

- **days** – Number of days
- **hours** – Number of hours
- **minutes** – Number of minutes
- **seconds** – Number of seconds
- **microseconds** – Number of microseconds
- **weeks** – Number of weeks

Returns Current object plus added delta

Return type *Zulu*

classmethod all_timezones()
Returns a list of all allowed timezone names

Timezone 'local' will return a datetime object with local timezone, but is not included in this list

Wrapper for `pytz.all_timezones()`

Returns List of timezones

Return type *list*

classmethod delta(*days=0*, *hours=0*, *minutes=0*, *seconds=0*, *microseconds=0*, *weeks=0*)
Wrapper for `datetime.timedelta()`

Parameters

- **days** – Number of days
- **hours** – Number of hours

- **minutes** – Number of minutes
- **seconds** – Number of seconds
- **microseconds** – Number of microseconds
- **weeks** – Number of weeks

Returns datetime.timedelta

classmethod elf(**args*, *tz='local'*)

General input Zulu constructor

Takes the same inputs as constructor, and also allows Zulu objects to pass through. If timeozone is missing it will assume the input timezone `tz`, which is set to local as default

It takes both seed strings and iso strings:

```
Zulu.elf('20201112T213732u993446Z')
    Zulu(2020, 11, 12, 21, 37, 32, 993446)

Zulu.elf('2020-11-12T21:37:32.993446+00:00')
    Zulu(2020, 11, 12, 21, 37, 32, 993446)
```

It takes UNIX epoch:

```
e = Zulu(2020, 1, 1).epoch()
e
    1577836800.0
Zulu.elf(e)
    Zulu(2020, 1, 1)
```

It will guess the missing values if input integers are not a full date and/or time:

```
Zulu.elf(2020)
    Zulu(2020, 1, 1)

Zulu.elf(2020, 2)
    Zulu(2020, 2, 1)

Zulu.elf(2020,1,1,10)
    Zulu(2020, 1, 1, 10, 0, 0)
```

Warning: Elves are fickle

Raises **AngryElf** – If an instance cannot be created from the given input

Parameters

- **args** – Input arguments
- **tz** – Time zone to assume if missing. 'local' will use local time zone. Use *all_timezones()* for a list of actual timezones. Default is 'local'

Returns Best guess Zulu object

Return type *Zulu*

epoch()

Get UNIX epoch (seconds since January 1st 1970)

Wrapper for `datetime.datetime.timestamp()`

Returns Seconds since January 1st 1970

Return type float

format(*pattern*)
Format Zulu to string with the given pattern

Wrapper for `datetime.datetime.strftime()`

Parameters **pattern** – Follows standard Python strftime reference

Returns str

classmethod from_epoch(*epoch*)
Create Zulu object from UNIX Epoch

Parameters **epoch** (*float*) – Unix epoch

Returns Zulu instance

Return type *Zulu*

classmethod from_iso(*str_: str*, *tz=None*)
Create Zulu object from ISO 8601 string

Parameters

- **iso** – ISO 8601 string
- **tz** – Timezone string to use if missing in ts_str

Returns Zulu

Return type *Zulu*

classmethod from_seed(*seed: str*)
Create Zulu object from seed string

Parameters **seed** – Seed string

Return type *Zulu*

classmethod from_str(*st: str*)
Converts seed or iso string to Zulu

Parameters **st** – Seed or iso string

Return type *Zulu*

classmethod from_unaware(*ts*, *tz='utc'*)
Create Zulu from timezone unaware datetime

Parameters

- **ts** (*datetime.datetime*) – Unaware time stamp
- **tz** – Time zone, with 'utc' as default. 'local' will use local time zone

Return type *Zulu*

classmethod from_unaware_local(*ts*)
Create Zulu from timezone unaware local timestamp

Parameters **ts** (*datetime.datetime*) – Timezone unaware datetime

Return type *Zulu*

classmethod from_unaware_utc(*ts*)
Create Zulu from timezone unaware UTC timestamp

Parameters **ts** (*datetime.datetime*) – Timezone unaware datetime

Return type *Zulu*

classmethod is_iso(*st: str*)
Check if input string is ISO 8601

Check is done using regex `Zulu.ISO_REGEX`

Parameters **st** – Maybe an ISO formatted string

Returns True if input string is iso, False if not

Return type bool

iso(*full=False*)
Create ISO 8601 string

Example:

```
z = Zulu(2020, 5, 21)
z.iso()
    '2020-05-21T00:00:00+00:00'

z.iso(full=True)
    '2020-05-21T00:00:00.000000+00:00'
```

Parameters **full** (*bool*) – If True, pad isostring to full length when microsecond is zero, so that all strings returned will have same length (has proved an issue with a certain document database tool, which was not able to parse varying iso string length without help)

Returns str

classmethod now(*tz=None*)
Overrides `datetime.datetime.now()`. Equivalent to `Zulu()`

Raises ***ZuluError*** – If parameter `tz` has a value. Even if value is UTC

Parameters **tz** – Do not use. Zulu is always UTC

Returns Zulu

classmethod parse(*ts_str: str*, *pattern: str*, *tz=None*)
Parse time stamp string with the given pattern

Parameters

- **ts_str** (*str*) – Timestamp string
- **pattern** – Follows standard python strftime reference
- **tz** – Timezone to use if timestamp does not have timezone info

Returns Zulu

classmethod range(*start=None*, *n=10*, *delta=datetime.timedelta(seconds=3600)*)
Generate a list of Zulu of fixed intervals

Note: Mainly for dev purposes. There are far better ways of creating a range of timestamps, such as using pandas.

Parameters

- **start** (*Zulu*) – Start time Zulu, default is *now*
- **n** (*int*) – Number of timestamps in range, with default 10
- **delta** (*datetime.timedelta*) – Time delta between items, with default one hour

Return type [*Zulu*]

to_local()
Create regular datetime with local timezone

Return type datetime.datetime

to_tz(*tz='local'*)
Create regular datetime with input timezone

For a list of timezones use *Zulu.all_timezones()*. 'local' is also allowed, although not included in the list

Parameters **tz** – Time zone to use. 'local' will return the local time zone. Default is 'local'

Return type datetime.datetime

to_unaware()
Get timezone unaware datetime object in UTC

Returns Timezone unaware datetime

Return type datetime.datetime

2.1.2 File System

File

class mjooln.**File**(*path: str*, **args*, ***kwargs*)
Convenience class for file handling

Create a file path in current folder:

```
fi = File('my_file.txt')
fi
    File('/home/zaphod/current/my_file.txt')
```

Create a file path in home folder:

```
fi = File.home('my_file.txt')
fi
    File('/home/zaphod/my_file.txt')
```

Create a file path in some folder:

```
fo = Folder.home().append('some/folder')
fo
    Folder('/home/zaphod/some/folder')
fi = fo.file('my_file.txt')
fi
    File('/home/zaphod/some/folder/my_file.txt')
```

Create and read a file:

```
fi = File('my_file.txt')
fi.write('Hello world')
fi.read()
    'Hello world'
fi.size()
    11
```

Compress and encrypt:

```
fi.compress()
fi.name()
    'my_file.txt.gz'
fi.read()
    'Hello world'

crypt_key = Crypt.generate_key()
crypt_key
    b'aLQYOIxZOLllYThEKoXTH_eqTQGEnXm9CUl2glq3a2M='
fi.encrypt(crypt_key)
fi.name()
    'my_file.txt.gz.aes'
fi.read(crypt_key=crypt_key)
    'Hello world'
```

Create an encrypted file, and write to it:

```
ff = File('my_special_file.txt.aes')
ff.write('Hello there', password='123')
ff.read(password='123')
    'Hello there'

f = open(ff)
f.read()
    'gAAAAABe0BYqPPYfzha3AKNyQCorg4TT8DcJ4XxtYhMs7ksx22GiVC03WcrMTnvJLjTLNYCz_
↪N6OCmSVwk29Q9hoQ-UkN0Sbbg=='
f.close()
```

Note: Using the `password` parameter, builds an encryption key by combining it with the builtin (i.e. hard coded) class salt. For proper security, generate your own salt with *`Crypt.salt()`*. Store this salt appropriately, then use *`Crypt.key_from_password()`* to generate a crypt_key

---

Warning: '123' is not a real password

compress (*delete_original: bool = True*)
Compress file

Parameters **delete_original** (*bool*) – If True, original file will be deleted after compression (default)

copy (*new_folder*, *new_name: str = None*, *overwrite: bool = False*)
Copy file to a new folder, and optionally give it a new name

Parameters

- **overwrite** (*bool*) – Set True to overwrite destination file if it exists

- **new_folder** (*Folder or str*) – New folder
- **new_name** (*str*) – New file name (optional). If missing, the file will keep the same name

Returns Copied file

Return type *File*

decompress(*delete_original: bool = True*, *replace_if_exists: bool = True*)
Decompress file

Parameters

- **delete_original** (*bool*) – If True, the original compressed file will be deleted after decompression
- **replace_if_exists** (*bool*) – If True, the decompressed file will replace any already existing file with the same name

decrypt(*crypt_key: bytes*, *delete_original: bool = True*)
Decrypt file

Raises ***FileError*** – If file is not encrypted or if crypt_key is missing

Parameters

- **crypt_key** (*bool*) – Encryption key
- **delete_original** (*bool*) – If True, the original encrypted file will be deleted after decryption

delete(*missing_ok: bool = False*)
Delete file

Raises ***FileError*** – If file is missing, and `missing_ok=False`

Parameters **missing_ok** (*bool*) – Indicate if an exception should be raised if the file is missing. If True, an exception will not be raised

delete_if_exists()
Delete file if exists

encrypt(*crypt_key: bytes*, *delete_original: bool = True*)
Encrypt file

Raises ***FileError*** – If file is already encrypted or if crypt_key is missing

Parameters

- **crypt_key** (*bytes*) – Encryption key
- **delete_original** (*bool*) – If True, the original unencrypted file will be deleted after encryption

extension()
Get file extension, i.e. the extension which is not reserved. A file is only supposed to have one extension that does not indicate either compression or encryption.

Raises ***FileError*** – If file has more than one extension barring `COMPRESSED_EXTENSION` and `ENCRYPTED_EXTENSION`

Returns File extension

Return type str

extensions()
Get file extensions as a list of strings

Returns List of file extensions

Return type *list*

folder()
Get the folder containing the file

Returns Folder containing the file

Return type *Folder*

classmethod home(*file_name: str*)
Create a file path in home folder

Parameters **file_name** (*str*) – File name

Return type *File*

is_compressed()
Check if file is compressed, i.e. has COMPRESSED_EXTENSION

Returns True if compressed, False if not

Return type bool

is_encrypted()
Check if file is encrypted, i.e. has ENCRYPTED_EXTENSION

Returns True if encrypted, False if not

Return type bool

is_hidden()
Check if file is hidden, i.e. starts with HIDDEN_STARTSWITH

Returns True if hidden, False if not

Return type bool

md5_checksum()
Get MD5 Checksum for the file

Raises ***FileError*** – If file does not exist

Returns MD5 Checksum

Return type str

move(*new_folder: mjooln.Folder*, *new_name=None*, *overwrite: bool = False*)
Move file to a new folder, and with an optional new name

Parameters

- **new_folder** (Folder) – New folder
- **new_name** – New file name (optional). If missing, the file will keep the same name

Returns Moved file

Return type *File*

new(*name*)
Create a new file path in same folder as current file

Parameters **name** – New file name

Return type *File*

read(*mode='r'*, *crypt_key: bytes = None*, *password: str = None*, **args*, ***kwargs*)
Read file

If file is encrypted, use either `crypt_key` or `password`. None or both will raise an exception. Encryption requires the file name to end with `ENCRYPTED_EXTENSION`

Raises ***FileError*** – If trying to decrypt a file without `ENCRYPTED_EXTENSION`

Parameters

- **mode** – Read mode
- **crypt_key** (*bytes*) – Encryption key
- **password** (*str*) – Password (will use class salt)

Returns Data as string or bytes depending on read mode

Return type str or bytes

read_json(*crypt_key: bytes = None*, *password: str = None*, ***kwargs*)
Read json file

Extends *File.read()* with *JSON.loads()*

Parameters

- **crypt_key** (*bytes*) – Encryption key
- **password** (*str*) – Password (will use class salt)

Returns Dictionary of JSON content

Return type dict

read_yaml(*crypt_key: bytes = None*, *password: str = None*, ***kwargs*)
Read json file

Extends *File.read()* with *YAML.loads()*

Parameters

- **crypt_key** (*bytes*) – Encryption key
- **password** (*str*) – Password (will use class salt)

Returns Dictionary of YAML content

Return type dict

stub()
Get file stub, i.e. the part of the file name bar extensions and `HIDDEN_STARTSWITH`

Example:

```
fi = File('.hidden_with_extensions.json.gz')
fi.stub()
    'hidden_with_extensions'
```

Returns File stub

Return type str

touch()
Create empty file if it does not exist already

untouch (*ignore_if_not_empty=False*)
Delete file if it exists, and is empty

Parameters **ignore_if_not_empty** – If True, no exception is raised if file is not empty and thus cannot be deleted with untouch

Returns

write (*data*, *mode='w'*, *crypt_key: bytes = None*, *password: str = None*, ***kwargs*)
Write data to file

For encryption, use either `crypt_key` or `password`. None or both will raise an exception. Encryption requires the file name to end with `ENCRYPTED_EXTENSION`

Raises ***FileError*** – If using `crypt_key` or `password`, and the file does not have encrypted extension

Parameters

- **data** (*str or bytes*) – Data to write
- **mode** (*str*) – Write mode
- **crypt_key** (*bytes*) – Encryption key
- **password** (*str*) – Password (will use class salt)

write_json (*data: dict*, *human: bool = False*, *crypt_key: bytes = None*, *password: str = None*, ***kwargs*)
Write dictionary to JSON file

Extends *JSON.dumps()* with *File.write()*

For encryption, use either `crypt_key` or `password`. None or both will raise an exception. Encryption requires the file name to end with `ENCRYPTED_EXTENSION`

Raises ***FileError*** – If using `crypt_key` or `password`, and the file does not have encrypted extension

Parameters

- **data** (*str or bytes*) – Data to write
- **human** (*bool*) – If True, write JSON as human readable
- **crypt_key** (*bytes*) – Encryption key
- **password** (*str*) – Password (will use class salt)

write_yaml (*data: dict*, *crypt_key: bytes = None*, *password: str = None*, ***kwargs*)
Write dictionary to YAML file

Extends *YAML.dumps()* with *File.write()*

For encryption, use either `crypt_key` or `password`. None or both will raise an exception. Encryption requires the file name to end with `ENCRYPTED_EXTENSION`

Raises ***FileError*** – If using `crypt_key` or `password`, and the file does not have encrypted extension

Parameters

- **data** (*str or bytes*) – Data to write
- **crypt_key** (*bytes*) – Encryption key
- **password** (*str*) – Password (will use class salt)

Folder

class mjooln.**Folder**(*path*, **args*, ***kwargs*)

append(**args*)
Append strings or list of strings to current folder

Example:

```
fo = Folder.home()
print(fo)
    '/Users/zaphod'

fo.append('dev', 'code', 'donald')
    '/Users/zaphod/dev/code/donald'

parts = ['dev', 'code', 'donald']
fo.append(parts)
    '/Users/zaphod/dev/code/donald'
```

Parameters **args** – Strings or list of strings

Returns Appended folder as separate object

Return type *Folder*

create(*error_if_exists=True*)
Create new folder, including non existent parent folders

Raises ***FolderError*** – If folder already exists, *and* `error_if_exists=True`

Parameters **error_if_exists** (*bool*) – Error flag. If True, method will raise an error if the folder already exists

Returns True if it was created, False if not

Return type bool

classmethod current()
Get current folder path

Wrapper for `os.getcwd()`

Returns Path to current folder

Return type *Folder*

disk_usage(*include_folders: bool = False*, *include_files: bool = True*)
Recursively determines disk usage of all contents in folder

Parameters

- **include_folders** – If True, all folder sizes will be included in total, but this is only the folder object and hence a small number. Default is therefore False
- **include_files** – If True, all file sizes are included in total. Default is obviously True

Raises ***FolderError*** – If folder does not exist

Returns Disk usage of folder content

Return type int

empty(*name: str*)
Recursively deletes all files and subfolders

Name of folder is required to verify deleting content

> **Warning:** Be careful. Will delete all content recursively

Parameters **name** (*str*) – Folder name as given by `Folder.name()`. Required to verify deleting all contents

Raises ***FolderError*** – If folder does not exist, or if `name` is not an exact match with folder name

file(*name: str*)
Create file path in this folder

Parameters **name** (*str*) – File name

Returns File path in this folder

Return type *File*

files()
Generator listing all files in this folder recursively

Print all files larger than 1 kB in home folder and all subfolders:

```
fo = Folder.home()
for fi in fo.files():
    if fi.size() > 1000:
        print(fi)
```

Returns Generator object returning *File* for each iteration

Return type generator

folders()
Generator listing all folders in this folder recursively

Returns Generator object returning *Folder* for each iteration

Return type generator

classmethod **home**()
Get path to user home folder

Wrapper for `os.path.expanduser()`

Returns Home folder path

Return type *Folder*

is_empty()
Check if folder is empty

Raises ***FolderError*** – If folder does not exist

Returns True if empty, False if not

Return type bool

list(*pattern: str = '*', recursive: bool = False*)
List folder contents

Example patterns:

- `'*'` (default) Returns all files and folders except hidden
- `'.*` Returns all hidden files and folders
- `'*.txt'` Return all files ending with 'txt'

Note: For large amounts of files and folders, use the generator returned by *Folder.walk()* and handle them individually

---

Raises ***FolderError*** – If folder does not exist

Parameters

- **pattern** – Pattern to search for
- **recursive** – If True search will include all subfolders and files

Returns List of *File* and/or *Folder*

Return type *list*

list_files(*pattern='*', recursive=False*)
List all files in this folder matching `pattern`

Uses *Folder.list()* and then filters out all *File* objects and returns the result

---

Note: For large amounts of files, use the generator returned by *Folder.files()* and handle them individually

---

Raises ***FolderError*** – If folder does not exist

Parameters

- **pattern** – Pattern to search for
- **recursive** – If True search will include all subfolders and files

Returns List of *File* objects

Return type *list*

list_folders(*pattern='*', recursive=False*)
List all folders in this folder matching `pattern`

Uses *Folder.list()* and then filters out all *Folder* objects and returns the result

---

Note: For large amounts of folders, use the generator returned by *Folder.folders()* and handle them individually

---

Raises ***FolderError*** – If folder does not exist

Parameters

- **pattern** – Pattern to search for
- **recursive** – If True search will include all subfolders and files

Returns List of *Folder* objects

Return type *list*

parent()
Get parent folder

Returns Parent folder

Return type *Folder*

print(*count: bool = False*, *disk_usage: bool = False*)
Print folder content

Parameters

- **count** (*bool*) – Include count for each subfolder
- **disk_usage** (*bool*) – Include disk usage for each subfolder, and size for each file

remove(*error_if_not_exists: bool = True*)
Remove folder

Raises

- **OSError** – If folder exists but is not empty
- ***FolderError*** – If folder does not exist and `error_if_not_exists=True`

Parameters **error_if_not_exists** (*bool*) – If True, method will raise an error if the folder already exists

remove_empty_folders()
Recursively remove empty subfolders

touch()
Create folder if it does not exist, ignore otherwise

untouch()
Remove folder if it exists, ignore otherwise

Raises **OSError** – If folder exists but is not empty

walk(*include_files: bool = True*, *include_folders: bool = True*)
Generator listing all files and folders in this folder recursively

Returns Generator object returning *File* or *Folder* for each iteration

Return type generator

Path

class mjooln.**Path**(*path: str*)
Absolute paths as an instance with convenience functions

Intended use via subclasses *Folder* and *File*

No relative paths are allowed. Paths not starting with a valid mountpoint will be based in current folder

All backslashes are replaced with `FOLDER_SEPARATOR`

as_file()
Create *File* with same path

Return type *File*

as_folder()
Create *Folder* with same path

Return type *Folder*

as_path()
Get as `pathlib.Path` object

Returns path

Return type pathlib.Path

as_pure_path()
Get as `pathlib.PurePath` object

Returns path

Return type pathlib.PurePath

created()
Get created timestamp from operating system

Wrapper for `os.stat(<path>).st_ctime`

Note: Created timestamp tends to be unreliable, especially when files have been moved around

Returns Timestamp created (perhaps)

Return type *Zulu*

exists()
Check if path exists

Wrapper for `os.path.exists()`

Returns True if path exists, False otherwise

Return type bool

classmethod has_valid_mountpoint(*path_str*)
Flags if the path starts with a valid mountpoint

Wrapper for `os.path.isabs()`

Returns True if path has valid mountpoint, False if not

Return type bool

classmethod host()
Get host name

Wrapper for `socket.gethostname()`

Returns Host name

Return type str

is_file()
Check if path is a file

Raises ***PathError*** – If path does not exist

Returns True if path is a file, False if not

Return type bool

is_folder()
Check if path is a folder

Raises ***PathError*** – If path does not exist

Returns True if path is a folder, False if not

Return type bool

is_network_drive()
Check if path is a network drive following the same rules as in *on_network_drive()*

Note: If on Windows, a mapped network drive will not be interpreted as a network drive, since the path starts with a drive letter

---

Returns True if path is network drive, False if not

Return type bool

is_volume()
Check if path is a volume

Volume is a collective term for mountpoint, drive and network drive

Raises ***PathError*** – If path does not exist

Returns True if path is a volume, False if not

Return type bool

classmethod join(**args*)
Join strings to path

Wrapper for `os.path.join()`

Relative paths will include current folder:

```
Path.current()
    '/Users/zaphod/dev'
Path.join('code', 'donald')
    '/Users/zaphod/dev/code/donald'
```

Returns Joined path as absolute path

Return type *Path*

classmethod listdir(*path_str*)
List folder content as plain strings with relative path names

Wrapper for `os.listdir()`

Other list and walk methods in *Folder* will instantiate *File* or *Folder* objects. They are thus a bit slower

Parameters **path_str** – String with path to folder

Returns List of relative path strings

modified()
Get modified timestamp from operating system

Wrapper for `os.stat(<path>).st_mtime`

Note: Modified timestamp tends to be unreliable, especially when files have been moved around

Returns Timestamp modified (perhaps)

Return type *Zulu*

classmethod mountpoints()
List valid mountpoints/partitions or drives

Finds mountpoints/partitions on linux/osx, and drives (C:, D:) on windows.

Warning: Windows requires installing package with an extra: `mjooln[mp]`. Alternatively, install package `psutil` manually

Warning: Network drives on windows will not be found by this method, unless they have been mapped

Note: Requires installation of Visual Studio C++ Build Tools on Windows. Go to the download page and find the Build Tools download (this is why the package `psutil` is not included by default on Windows)

Returns Valid mountpoints or drives

Return type *list*

name()
Get name of folder or file

Example:

```
p = Path('/Users/zaphod')
p
    '/Users/zaphod
p.name()
    'zaphod'

p2 = Path(p, 'dev', 'code', 'donald')
p2
    '/Users/zaphod/dev/code/donald'
p2.name()
    'donald'

p3 = Path(p, 'dev', 'code', 'donald', 'content.txt')
p3
    '/Users/zaphod/dev/code/donald/content.txt'
```

```
p3.name()
    'content.txt'
```

Returns Folder or file name

Return type str

network_drive()
Returns the first part of the path following the double slash

Example:

```
p = Path('//netwdrive/extensions/parts')
p.network_drive()
    Folder('//netwdrive')
```

Raises ***PathError*** – If path is not on a network drive (see *on_network_drive()*)

Returns Network drive part of the path

Return type *Folder*

on_network_drive()
Check if path is on a network drive

Warning: Only checks if the path starts with double slash, and may be somewhat unreliable. Make sure to test if it seems to work

Returns True if path is on network drive, False if not

Return type bool

parts()
Get list of parts in path

Example:

```
p = Path('/home/zaphod/dev/code')
p.parts()
    ['home', 'zaphod', 'dev', 'code']
```

Returns String parts of path

Return type *list*

classmethod platform()
Get platform name alias

- WINDOWS
- LINUX
- OSX

Example on a linux platform:

```
Path.platform()
    'linux'

Path.platform() == Path.LINUX
    True
```

Raises ***PathError*** – If platform is unknown

Returns Platform name alias

Return type str

raise_if_not_exists()
Raises an exception if path does not exist

Raises ***PathError*** – If path does not exist

size()
Return file or folder size

Note: If Path is a folder, `size()` will return a small number, representing the size of the folder object, not its contents. For finding actual disk usage of a folder, use *`Folder.disk_usage()`*

Raises ***PathError*** – If path does not exist

Returns File or folder size

Return type int

classmethod validate(*path_str*)
Check if path is longer than `PATH_CHARACTER_LIMIT`, which on Windows may cause problems

Parameters **path_str** (*str*) – Path to check

Raises ***PathError*** – If path is too long

volume()
Return path volume

Volume is a collective term for mountpoint, drive and network drive

Raises ***PathError*** – If volume cannot be determined

Returns Volume of path

Return type *Folder*

Archive

class mjooln.**Archive**
Zip file to gz conversion

classmethod is_zip(*file: mjooln.File*)
Check if input file is zip archive

Parameters **file** – Input file

Returns True if extension is 'zip', false if not

Return type bool

classmethod zip_to_gz(*file: mjooln.File*, *delete_source_file: bool = True*)
Convert zip file to gzip compressed file

Parameters

- **file** – Input zip archive
- **delete_source_file** – Delete source file if True

2.1.3 Experimental

Document

class mjooln.experimental.document.**Document**(*file*, *atom=None*, *crypt_key=None*, *password=None*, ***kwargs*)

Danger: Experimental class. May change without notice and suddenly disappear

Class with functionality to store attributes in file as JSON or YAML

Has an `atom` attribute as default, as well as private created and modified attributes stored in file due to unreliable file system handling of these

created()
Get Document created date

Return type *Zulu*

delete()
Delete Document file

exists()
Check if Document file exists

Return type bool

file()
Get Document file instance

Return type *File*

classmethod from_doc(*doc: dict*)
Create Document file from serialized dictionary

Parameters **doc** – Input dictionary

Return type *Document*

classmethod load(*file*, *crypt_key: bytes = None*, *password: str = None*)
Load Document file

Parameters

- **crypt_key** (*bytes*) – Encryption key
- **password** (*str*) – Password (will use builtin salt)
- **file** – Document file

Raises ***DocumentError*** – If extension is not valid. Must be 'json' or 'yaml'

Returns Document instance based on file contents

Return type *Document*

modified()
Get Document modified date

Return type *Zulu*

save(*human: bool = False*, *crypt_key: bytes = None*, *password: str = None*, ***kwargs*)
Save Document

to_doc()
Get attributes as serializable dictionary

Return type dict

App

class mjooln.experimental.app.**App**(*atom=None*, *max_workers=None*, *num_tasks=10*, *min_wait_s=0.2*, *pause_s=2.0*, *continuous=False*, *runfile=False*, ***kwargs*)

> **Danger:** Experimental class. May change without notice and suddenly disappear

Facilitates task execution by inheriting this class, then override mehtods `App.tasks()` and `App.execute()`

Store

class mjooln.experimental.store.**Store**

> **Danger:** Experimental class. May change without notice and suddenly disappear

Facilitates storage of config files and encryption keys using `Key` as a replacement for file name

System

class mjooln.experimental.system.**System**

> **Danger:** Experimental class. May change without notice and suddenly disappear

Convenience methods for system status (cores, memory, disk space)

2.1.4 Exceptions

exception mjooln.**MjoolnException**
Parent for all module specific exceptions

exception mjooln.**CryptError**
Rased by *Crypt*, mainly when password or crypt_key is invalid

exception mjooln.**BadSeed**
Raised by *Seed*

exception mjooln.**DicError**
Raised by *Dic*

exception mjooln.**DocError**
Raised by *Doc*

exception mjooln.**DocumentError**
Raised by *Document*

exception mjooln.**IdentityError**
Raised by *Identity*

exception mjooln.**BadWord**
Raised by *Word*

exception mjooln.**NotAnInteger**
Raised by *Word* when trying to get an integer from a non-integer word

exception mjooln.**ZuluError**
Raised by *Zulu*

exception mjooln.**PathError**
Raised by *Path*

exception mjooln.**FileError**
Raised by *File*

exception mjooln.**ArchiveError**
Raised by *Archive*

exception mjooln.**FolderError**
Raised by *Folder*

CHAPTER 3

References

3.1 References

Documentation strongly aided by following locust by example

Many a snippet copied from stack overflow and similar sites

The best Python reference is The Hitchhiker's Guide to Python

CHAPTER 4

Changelog

4.1 Changelog

4.1.1 0.6.3

Changed

API documentation

Fixed

Missing extra in `setup.cfg`

4.1.2 0.6.2

Remove default import of package `psutil` for Windows installations

4.1.3 0.6.1

Set PIXIE to `False` by default

4.1.4 0.6.0

Started changelog

Index

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Y

Z