tea Documentation

Release 0.0.0

Viktor Kerkez <alefnula@gmail.com>

Aug 22, 2020

Contents

Contents:

CHAPTER 1

API Documentation

1.1 tea api documentation

1.1.1 `tea.ctx` Module

Context library - providing usefull context managers.

`tea.ctx.`**`suppress`**(**args*, ***kwds*)

Ignore an exception or exception list.

Usage:

```
with suppress(OSError):
    os.remove('filename.txt')
```

1.1.2 `tea.decorators` Module

`tea.decorators.`**`docstring`**(*documentation*, *prepend=False*, *join=''*)

Prepend or append a string to the current documentation of the function.

This decorator should be robust even if `func.__doc__` is None (for example, if -OO was passed to the interpreter).

Usage:

```
@docstring('Appended this line')
def func():
    "This docstring will have a line below."
    pass

>>> print(func.__doc__)
This docstring will have a line below.

Appended this line
```

Parameters

- **documentation** (*str*) – Documentation string that should be added, appended or prepended to the current documentation string.
- **prepend** (*bool*) – Prepend the documentation string to the current documentation if `True` else append. default=''False''
- **join** (*str*) – String used to separate docstrings. default='n'

`tea.decorators.`**combomethod**(*method=None*, *static=False*)

Create a class method or static method.

It will be used when you call it on the class but can be overridden by an instance method of the same name that will be called when the method is called on the instance.

Usage:

```
class Foo(object):
    class_variable = 2

    def __init__(self):
        self.instance_variable = 3
        # Override class variable for test case
        self.class_variable = 4

    @combomethod(static=True)
    def static_and_instance(x):
        return x + 1

    @static_and_instance.instance
    def static_and_instance(self, x):
        return x + self.instance_variable

    @combomethod
    def class_and_instance(cls, x):
        return x + cls.class_variable

    @class_and_instance.instance
    def class_and_instance(self, x):
        return x + self.instance_variable

>>> Foo.static_and_instance(100)
101
>>> Foo.class_and_instance(100)
102
>>> f = Foo()
>>> f.static_and_instance(100)
103
>>> f.class_and_instance(100)
103
```

1.1.3 `tea.dsa` Module

Data structures and algorithms module.

class `tea.dsa.config.`**Config**(*filename=None*, *data=None*, *fmt=None*, *encoding='utf-8'*, *auto_save=True*)

Configuration class.

keys (**args*, ***kwargs*)
Return a set of top level keys in this configuration.

get (**args*, ***kwargs*)
Return a value from configuration.

Safe version which always returns a default value if the value is not found.

delete (**args*, ***kwargs*)
Delete an item from configuration.

Safe version, never, raises an error.

insert (**args*, ***kwargs*)
Insert at the index.

If the index is not provided appends to the end of the list.

class tea.dsa.config.**MultiConfig**(*filename=None*, *data=None*, *fmt=None*, *encoding='utf-8'*, *auto_save=True*)
Base class for configuration management.

keys (**args*, ***kwargs*)
Return a merged set of top level keys from all configurations.

get (**args*, ***kwargs*)
Return a value from configuration.

Safe version which always returns a default value if the value is not found.

delete (**args*, ***kwargs*)
Delete an item from configuration.

Safe version, never, raises an error.

1.1.4 tea.logger Module

Logging module.

This logging module is designed as a wrapper around the python logging module.

When the module is loaded it configure the logging object with some default parameters (log to stderr with DEBUG level). After that, user can call the L{configure_logger} function and configure logging to file and stderr.

Usage

```
>>> import logging
>>> import tempfile
>>> from tea.logger import configure_logging
>>> configure_logging(filename=tempfile.mktemp())
>>> logger = logging.getLogger('test')
>>> logger.debug('Debug level log entry')
>>> logger.info('Info level log entry')
>>> logger.warning('Warn level log entry')
WARNING     - Warn level log entry [test:1]
>>> logger.error('Error level log entry')
ERROR       - Error level log entry [test:1]
>>> logger.critical('Critical level log entry')
CRITICAL    - Critical level log entry [test:1]
>>> try:
```

(continues on next page)

(continued from previous page)

```
...     raise Exception('Test exception')
... except Exception:
...     logger.exception('Error level log entry with stack trace')
ERROR       - Error level log entry with stack trace [test:4]
Traceback (most recent call last):
  ...
Exception: Test exception
```

tea.logger.**configure_logging**(*filename=None*, *filemode='a'*, *datefmt='%Y.%m.%d %H:%M:%S'*, *fmt='%(asctime)s.%(msecs)03d %(levelname)11s: %(message)s [%(name)s:%(lineno)d]'*, *stdout_fmt='%(levelname)-11s - %(message)s [%(name)s:%(lineno)d]'*, *level=10*, *stdout_level=30*, *initial_file_message=''*, *max_size=1048576*, *rotations_number=5*, *remove_handlers=True*)

Configure logging module.

Parameters

- **filename** (*str*) – Specifies a filename to log to.
- **filemode** (*str*) – Specifies the mode to open the log file. Values: `'a'`, `'w'`. *Default:* `a`.
- **datefmt** (*str*) – Use the specified date/time format.
- **fmt** (*str*) – Format string for the file handler.
- **stdout_fmt** (*str*) – Format string for the stdout handler.
- **level** (*int*) – Log level for the file handler. Log levels are the same as the log levels from the standard `logging` module. *Default:* `logging.DEBUG`
- **stdout_level** (*int*) – Log level for the stdout handler. Log levels are the same as the log levels from the standard `logging` module. *Default:* `logging.WARNING`
- **initial_file_message** (*str*) – First log entry written in file.
- **max_size** (*int*) – Maximal size of the logfile. If the size of the file exceed the maximal size it will be rotated.
- **rotations_number** (*int*) – Number of rotations to save.
- **remove_handlers** (*bool*) – Remove all existing handlers.

tea.logger.log.**configure_logging**(*filename=None*, *filemode='a'*, *datefmt='%Y.%m.%d %H:%M:%S'*, *fmt='%(asctime)s.%(msecs)03d %(levelname)11s: %(message)s [%(name)s:%(lineno)d]'*, *stdout_fmt='%(levelname)-11s - %(message)s [%(name)s:%(lineno)d]'*, *level=10*, *stdout_level=30*, *initial_file_message=''*, *max_size=1048576*, *rotations_number=5*, *remove_handlers=True*)

Configure logging module.

Parameters

- **filename** (*str*) – Specifies a filename to log to.
- **filemode** (*str*) – Specifies the mode to open the log file. Values: `'a'`, `'w'`. *Default:* `a`.
- **datefmt** (*str*) – Use the specified date/time format.
- **fmt** (*str*) – Format string for the file handler.

- **stdout_fmt** (*str*) – Format string for the stdout handler.
- **level** (*int*) – Log level for the file handler. Log levels are the same as the log levels from the standard `logging` module. *Default:* `logging.DEBUG`
- **stdout_level** (*int*) – Log level for the stdout handler. Log levels are the same as the log levels from the standard `logging` module. *Default:* `logging.WARNING`
- **initial_file_message** (*str*) – First log entry written in file.
- **max_size** (*int*) – Maximal size of the logfile. If the size of the file exceed the maximal size it will be rotated.
- **rotations_number** (*int*) – Number of rotations to save.
- **remove_handlers** (*bool*) – Remove all existing handlers.

1.1.5 `tea.msg` Module

1.1.6 `tea.process` Module

1.1.7 `tea.shell` Module

Module mimics some of the behaviors of the builtin `shutil`.

It adds logging to all operations and abstracting some other useful shell commands (functions).

`tea.shell.`**`split`**(*s*, *posix=True*)
Split the string s using shell-like syntax.

Parameters

- **s** (*str*) – String to split
- **posix** (*bool*) – Use posix split

Returns List of string parts

Return type list of str

`tea.shell.`**`search`**(*path*, *matcher='*'*, *dirs=False*, *files=True*)
Recursive search function.

Parameters

- **path** (*str*) – Path to search recursively
- **matcher** (*str or callable*) – String pattern to search for or function that returns True/False for a file argument
- **dirs** (*bool*) – if True returns directories that match the pattern
- **files** (*bool*) – if True returns files that match the patter

Yields *str* – Found files and directories

`tea.shell.`**`chdir`**(*directory*)
Change the current working directory.

Parameters **directory** (*str*) – Directory to go to.

`tea.shell.`**`goto`**(**args*, ***kwds*)
Context object for changing directory.

Parameters

- **directory** (*str*) – Directory to go to.
- **create** (*bool*) – Create directory if it doesn't exists.

Usage:

```
>>> with goto(directory) as ok:
...     if not ok:
...         print 'Error'
...     else:
...         print 'All OK'
```

tea.shell.**mkdir**(*path*, *mode=493*, *delete=False*)
Make a directory.

Create a leaf directory and all intermediate ones. Works like `mkdir`, except that any intermediate path segment (not just the rightmost) will be created if it does not exist. This is recursive.

Parameters

- **path** (*str*) – Directory to create
- **mode** (*int*) – Directory mode
- **delete** (*bool*) – Delete directory/file if exists

Returns True if succeeded else False

Return type bool

tea.shell.**copy**(*source*, *destination*)
Copy file or directory.

Parameters

- **source** (*str*) – Source file or directory
- **destination** (*str*) – Destination file or directory (where to copy).

Returns True if the operation is successful, False otherwise.

Return type bool

tea.shell.**gcopy**(*pattern*, *destination*)
Copy all file found by glob.glob(pattern) to destination directory.

Parameters

- **pattern** (*str*) – Glob pattern
- **destination** (*str*) – Path to the destination directory.

Returns True if the operation is successful, False otherwise.

Return type bool

tea.shell.**move**(*source*, *destination*)
Move a file or directory (recursively) to another location.

If the destination is on our current file system, then simply use rename. Otherwise, copy source to the destination and then remove source.

Parameters

- **source** (*str*) – Source file or directory (file or directory to move).
- **destination** (*str*) – Destination file or directory (where to move).

Returns True if the operation is successful, False otherwise.

Return type bool

tea.shell.**gmove**(*pattern*, *destination*)
Move all file found by glob.glob(pattern) to destination directory.

Parameters

- **pattern** (*str*) – Glob pattern
- **destination** (*str*) – Path to the destination directory.

Returns True if the operation is successful, False otherwise.

Return type bool

tea.shell.**remove**(*path*)
Delete a file or directory.

Parameters **path** (*str*) – Path to the file or directory that needs to be deleted.

Returns True if the operation is successful, False otherwise.

Return type bool

tea.shell.**gremove**(*pattern*)
Remove all file found by glob.glob(pattern).

Parameters **pattern** (*str*) – Pattern of files to remove

Returns True if the operation is successful, False otherwise.

Return type bool

tea.shell.**read**(*path*, *encoding='utf-8'*)
Read the content of the file.

Parameters

- **path** (*str*) – Path to the file
- **encoding** (*str*) – File encoding. Default: utf-8

Returns File content or empty string if there was an error

Return type str

tea.shell.**touch**(*path*, *content=''*, *encoding='utf-8'*, *overwrite=False*)
Create a file at the given path if it does not already exists.

Parameters

- **path** (*str*) – Path to the file.
- **content** (*str*) – Optional content that will be written in the file.
- **encoding** (*str*) – Encoding in which to write the content. Default: `utf-8`
- **overwrite** (*bool*) – Overwrite the file if exists.

Returns True if the operation is successful, False otherwise.

Return type bool

1.1.8 tea.utils Module

tea.utils.**get_object**(*path=''*, *obj=None*)

Return an object from a dot path.

Path can either be a full path, in which case the *get_object* function will try to import the modules in the path and follow it to the final object. Or it can be a path relative to the object passed in as the second argument.

Parameters

- **path** (*str*) – Full or relative dot path to the desired object
- **obj** (*object*) – Starting object. Dot path is calculated relatively to this object.

Returns Object at the end of the path, or list of non hidden objects if we use the star query.

Example for full paths:

```
>>> get_object('os.path.join')
<function join at 0x1002d9ed8>
>>> get_object('tea.process')
<module 'tea.process' from 'tea/process/__init__.pyc'>
```

Example for relative paths when an object is passed in:

```
>>> import os
>>> get_object('path.join', os)
<function join at 0x1002d9ed8>
```

Example for a star query. (Star query can be used only as the last element of the path:

```
>>> get_object('tea.dsa.*')
[]
>>> get_object('tea.dsa.singleton.*')
[<class 'tea.dsa.singleton.Singleton'>,
 <class 'tea.dsa.singleton.SingletonMetaclass'>
 <module 'six' from '...'>]
>>> get_object('tea.dsa.*')
[<module 'tea.dsa.singleton' from '...'>]     # Since we imported it
```

class tea.utils.**Loader**

Module loader class loads recursively a module and all it's submodules.

Loaded modules will be stored in the `modules` attribute of the loader as a dictionary of {module_path: module} key, value pairs.

Errors accounted during the loading process will not stop the loading process. They will be stored in the `errors` attribute of the loader as a dictionary of {module_path: exception} key, value pairs.

Usage:

```
loader = Loader()
loader.load('foo')
loader.load('baz.bar', 'boo')

import baz
loader.load(baz)
```

load(**modules*)

Load one or more modules.

Parameters **modules** – Either a string full path to a module or an actual module object.

tea.utils.**load_subclasses**(*klass*, *modules=None*)
Load recursively all all subclasses from a module.

Parameters

- **klass** (*str or list of str*) – Class whose subclasses we want to load.
- **modules** – List of additional modules or module names that should be recursively imported in order to find all the subclasses of the desired class. Default: None

FIXME: This function is kept only for backward compatibility reasons, it should not be used. Deprecation warning should be raised and it should be replaces by the `Loader` class.

tea.utils.**get_exception**()
Return full formatted traceback as a string.

tea.utils.**cmp**(*x*, *y*)
Compare function from python2.

tea.utils.encoding.**smart_text**(*s*, *encoding='utf-8'*, *errors='strict'*)
Return a unicode object representing 's'.

Treats bytes using the 'encoding' codec.

tea.utils.encoding.**smart_bytes**(*s*, *encoding='utf-8'*, *errors='strict'*)
Return a bytes version of 's' encoded as specified in 'encoding'.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index