sutools

Release 0.1

Aaron Stopher

Mar 17, 2023

CONTENTS:

CHAPTER

ONE

SUTOOLS MODULE

su (Super User) tools

Per module utilities, designed to be lightweight, easy to configure, and reduce boilerplate code.

info

This package is intended to create an lower barrier for entry for logging and module level cli with sensible defaults; sutools is not intended to replace click, argparse, logging or other utility libraries. If your project requires a more flexible configuration please use the appropriate tooling.

sutools.**cli**(*desc=None*, *logs=False*)

init cli and register to store

Parameters

- **desc** – description of the CLI
- **logs** – enable logging in CLI

sutools.**log**()

retrieve loggers namespace from store

sutools.**logger**(*name='<frozen run'*, *loggers=None*, *loglvl=20*, *filename='2023-03-17_02-45-07'*, *filepath=None*, *filefmt=<logging.Formatter object>*, *fhandler=None*, *filecap=None*, *filetimeout=None*, *file=True*, *streamfmt=<logging.Formatter object>*, *shandler=<StreamHandler <stderr> (NOTSET)>*, *stream=False*)

init logger object and register to store

Parameters

- **name** – name of the logger
- **loggers** – list of names for the logger to create
- **loglvl** – logging level to use
- **filename** – name of the log file
- **filepath** – name of folder to create logs in
- **filefmt** – format of the file logger
- **fhandler** – file handler to use for logging
- **filecap** – maximum number of log files to keep
- **filetimeout** – define a timeout period for log files to be removed
- **file** – toggle file logging
- **streamfmt** – format of the stream logger

- **shandler** – stream handler to use for logging
- **stream** – toggle stream logging

sutools.**register**(*func*)

register a function to the store

Parameters
func – the function to register

CHAPTER

TWO

SUTOOLS USAGE

2.1 register functions with sutools

Using the register decorator *@su.register* on your functions will register it with sutools *meta_handler*. Stored functions are available across tools. This registry is intended to be used by logger and cli utilities.

```
import sutools as su

@su.register
def add(x : int, y : int):
    '''add two integers'''
    return x + y
```

NOTE: Adding type hinting to your functions enforces types in the cli and adds positional arg class identifiers in the help docs for the command.

2.2 cli - initialization standard

It is suggested to define the command line interface after *if __name__ == '__main__'*. Any code before the cli will run even if a cli command is used; code after the cli definition will not run when passing a cli command.

```
import sutools as su

# registered functions...

su.logger(*args, **kwargs) # optional

# module level function calls...

if __name__ == '__main__':
    # main code (will run even when using cli commands)...
    su.cli(*args, **kwargs)
    # main code (will NOT run when using cli commands)...
```

NOTE: The CLI should be defined after the logger if you choose to use the two utilities in parallel.

2.3 cli - usage example

The logger utility should be instantiated after any registered functions but before any module level functions.

```
"""This module does random stuff."""
import sutools as su

@su.register
def meet(name : str, greeting : str = 'hello', farewell : str = 'goodbye') -> str:
    "meet a person"
    output = f'\n{greeting} {name}\n{farewell} {name}'
    su.log().meeting.info(output)
    return output

su.logger() # optional

# module level function calls...

if __name__ == '__main__':
    # main code (will run even when using cli commands)...
    su.cli(desc = __doc__)
    # main code (will NOT run when using cli commands)...
```

NOTE: Adding type hinting to your functions enforces types in the cli and adds positional arg class identifiers in the help docs for the command.

command usage:

```
python module.py meet foo
```

output:

```
hello foo
goodbye foo
```

module help output:

```
usage: module [-h] {meet} ...

This module does random stuff.

options:
-h, --help  show this help message and exit

commands:
{meet}
    meet      meet a person
```

command help output:

```
usage: dev meet [-gr <class 'str'>] [-sa <class 'str'>] [-h] name

meet(name: str, greeting: str = 'hello', farewell: str = 'goodbye') -> str
```

(continues on next page)

(continued from previous page)

```
positional arguments:
name                  <class 'str'>

options:
-gr <class 'str'>, --greeting <class 'str'>
                        default: hello
-fa <class 'str'>, --farewell <class 'str'>
                        default: goodbye
-h, --help            Show this help message and exit.
```

2.4 logger - initialization standard

The logger utility should be instantiated after any registered functions but before any module level functions.

```
import sutools as su

# registered functions...

su.logger(*args, **kwargs)

# module level function calls...

if __name__ == '__main__':
    # main code (will run even when using cli commands)...
    su.cli(*args, **kwargs) # optional
    # main code (will NOT run when using cli commands)...
```

2.5 logger - usage examples

accessing defined loggers is done with a *log()* helper function. Note the use of *su.log()* in the below functions to access a specified logger before defining the log level and message.

using registered function names

```
import sutools as su

@su.register
def add(x : int, y : int):
    '''add two integers'''
    su.log().add.info(f'{x} + {y} = {x+y}')
    return x + y

@su.register
def minus(x : int, y : int):
    '''subtract two integers'''
    su.log().minus.info(f'{x} - {y} = {x-y}')
    return x - y
```

(continues on next page)

(continued from previous page)

```
su.logger() # logger definition

# module level function calls
add(1,2)
minus(1,2)

if __name__ == '__main__':
    # main code (will run even when using cli commands)...
    su.cli() # optional
    # main code (will NOT run when using cli commands)...
```

log output

```
16:16:34, 961 add INFO 1 + 2 = 3
16:16:34, 961 minus INFO 1 - 2 = -1
```

using custom logger names

```
import sutools as su

@su.register
def add(x : int, y : int):
    '''add two integers'''
    su.log().logger1.info(f'{x} + {y} = {x+y}')
    return x + y

@su.register
def minus(x : int, y : int):
    '''subtract two integers'''
    su.log().logger2.info(f'{x} - {y} = {x-y}')
    return x - y

su.logger(loggers=['logger1','logger2']) # logger definition

# module level function calls
add(1,2)
minus(1,2)

if __name__ == '__main__':
    # main code (will run even when using cli commands)...
    su.cli() # optional
    # main code (will NOT run when using cli commands)...
```

log output

```
16:16:34, 961 add INFO 1 + 2 = 3
16:16:34, 961 minus INFO 1 - 2 = -1
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX