Yet Another Logger

Mar 13, 2020

Contents:

CHAPTER 1

Documentation for the Code

1.1 custom_logger.py

Note: Works with python3, no support for python 2.7

class custom_logger.**Logger**(*logger_prop_file_path*, *log_file_path*)

Custom Logger class. Dont actually need to get an instance of this class, but do internally create an instance to call the *__init__()* which initializes a lot of the logger variables.

Currently only provides three functions for logging

1. **Method entry logging**
2. **Method exit logging**
3. **Normal logging**

Both Method entry logging and Method exit logging are only *INFO* level logging by default

Supports *3 levels of logging* - *INFO, ERROR, DEBUG*. These are mutually exclusive (ie not hierarchial)

Supports *2 modes of logging* (simultaneously):

1. **FILE** - Writes logs to a file in the logs folder
2. **CONSOLE** - Logs to the standard output console

Log format - [<log level> <timestamp>] [Module name]-[Method name] <log text>

REQUIREMENTS

1. logger.properties file
2. logs folder

Warning: logger.properties file needs to have [logger properties] at the root

SAMPLE USAGE

```
from YALogger.custom_logger import Logger

Logger.initialize_logger(logger_prop_file_path = '.\logger.properties',
log_file_path = './logs')

Logger.perform_method_entry_logging('foo','bar')

Logger.perform_method_exit_logging('foo','bar')

Logger.log('info', 'foo','bar','this is the log text')
```

__init__ (*logger_prop_file_path*, *log_file_path*)
Initilaizing various *Logger* class properties:

1. *Logger.__level* : the logging.level
2. *Logger.__mode* : the logging mode
3. *Logger.__current_timestamp* : the current timestamp
4. *Logger.__log_file_path* : the log file path and name

Parameters

- **logger_prop_file_path** (*str*) – the path of logger.properties
- **log_file_path** (*str*) – the path of the log file

Raises NoSectionError IOError

static __new__ (*cls*, *logger_prop_file_path*, *log_file_path*)
Defining code in *__new__()* to make *Logger* a singleton class *Logger.__instance* keeps tracj of whether an instance of *Logger* exists or not. If it doesnt exist then creates it otherwise returns the existing instance of *Logger*

Parameters

- **logger_prop_file_path** (*str*) – the path of logger.properties
- **log_file_path** (*str*) – the path of the log file

Returns singleton instance of *Logger*

Return type instance of *Logger*

static _open_log_file (*log_file_path*)
Opens the log file when logging.mode = FILE Opens in append mode

Parameters **log_file_path** (*str*) – the path of the log file

Returns reference to opened file

Return type file instance

static _validate_logging_level (*logging_level*)
Checks of the logging.level specified in logger.properties and validates whether it falls under the valid values Valid values are - 'INFO', 'ERROR' , 'DEBUG'

Parameters **logging_level** (*str*) – logging level in logger.properties

Returns logging levels from logger.properties

Return type list

Raises `ValueError`

static _validate_logging_mode(*logging_mode*)
Checks of the logging.mode specified in logger.properties and validates whether it falls under the valid values Valid values are - 'FILE', 'CONSOLE'

Parameters **logging_mode** (*str*) – logging mode in logger.properties

Returns logging modes from logger.properties

Return type list

Raises `ValueError`

static initialize_logger(*logger_prop_file_path*, *log_file_path*)
Initializes the logger. Creates a new instance of *Logger* to call *__init__()*

Parameters

- **logger_prop_file_path** (*str*) – the path of logger.properties
- **log_file_path** (*str*) – the path of the log file

static log(*log_level*, *module_name*, *method_name*, *log_text*)
Call this method to log text.

Support added for log_text to be anything other than string as well (like dict, list etc)

Parameters

- **log_level** (*str*) – level of logging
- **module_name** (*str*) – module name
- **method_name** (*str*) – name of method being entered
- **log_text** (*anything*) – stuff to be logged

static perform_method_entry_logging(*module_name*, *method_name*)
Call this method before entering a method

Parameters

- **module_name** (*str*) – module name
- **method_name** (*str*) – name of method being entered

static perform_method_exit_logging(*module_name*, *method_name*)
Call this method after exiting a method

Parameters

- **module_name** (*str*) – module name
- **method_name** (*str*) – name of method being entered

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

c

y

Index

Symbols

C

I

L

P

Y