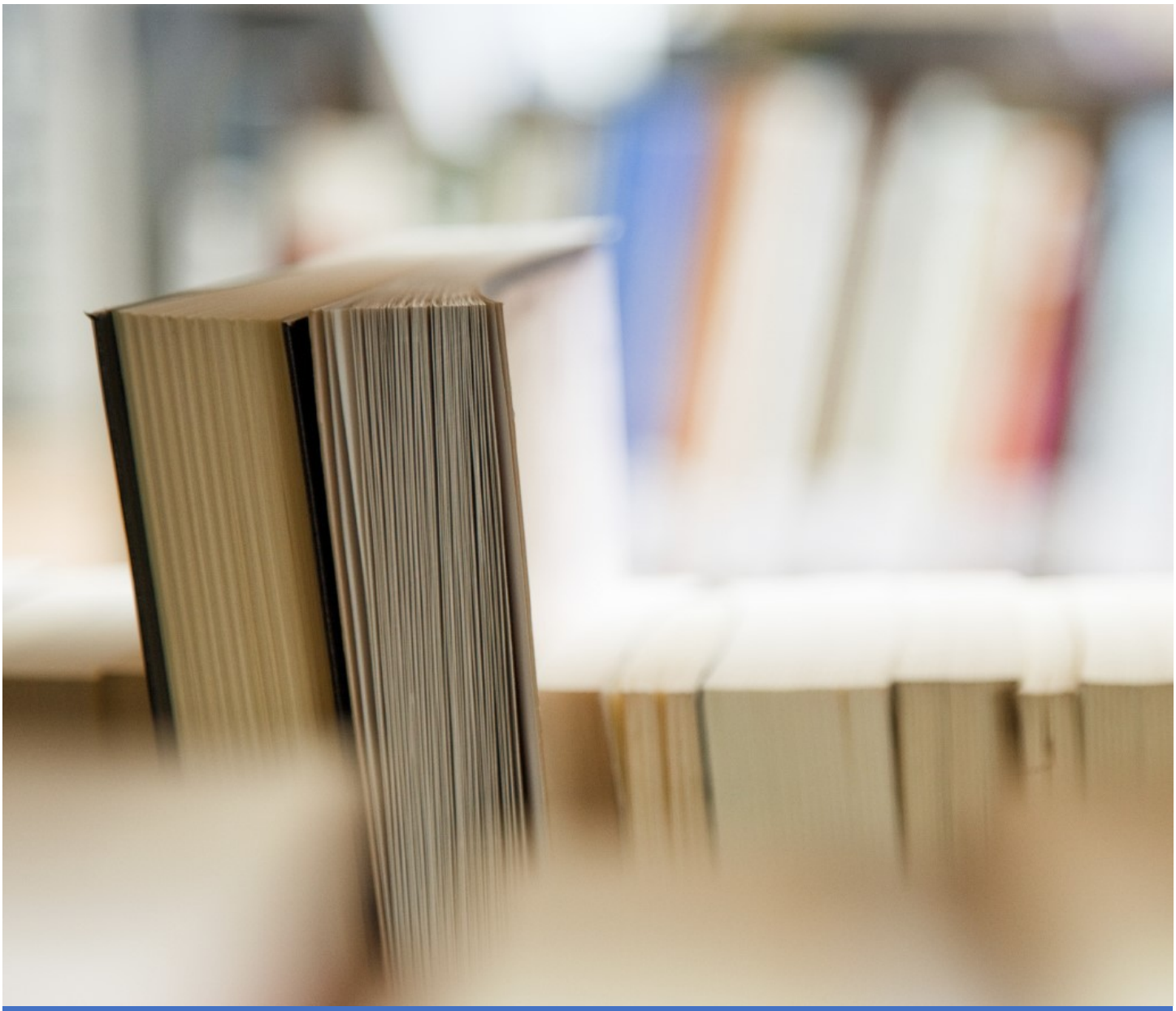

Python Logging

Debabrata Bhattacharya

dbhatta1232@gmail.com

1/14/20

Python Logging

Table of Contents

Python Logging 1
Chapter 1: About the Project 3
1.1 Aim of the Project 3
1.2 Project Files 3
1.3 Sections 3
Chapter 2: Learned Elements 4
2.1 Technologies Used 4
2.2 What I Have Learned 4
2.3 Future Scope 4
Chapter 3: Code 5
3.1 Python Script 5
3.2 CONF Config File 11
3.3 YAML Config File 12
Chapter 4: Output Logs 13
Chapter 5: Project Status 14

Chapter 1: **About the Project**

This project is an exploration of the python module ‘logging’.

1.1 Aim of the Project

The aim here is to follow the tutorial on Real Python here. We shall be following the tutorial described therein and practicing the code and examples in the tutorial.

1.2 Project Files

1. app.log
2. config.yaml
3. file.conf
4. fileconfig.log
5. LICENSE
6. log_script.py
7. README.md

1.3 Sections

1. The Logging Module
2. Basic Configurations
3. Formatting the Output
4. Logging Variable Data
5. Capturing Stack Trace
6. Classes and Functions
7. Using Handlers
8. Other Configuration Methods
9. Keep Calm and Read the Logs

Chapter 2: **Learned Elements**

2.1 Technologies Used

1. Logging Module
2. YAML
3. CONF config file

2.2 What I Have Learned

1. How to log events
2. Configure a custom logger
3. Create a YAML file to configure a custom logger
4. Create a CONF config file to configure a custom logger

2.3 Future Scope

1. Add logging functionality to ongoing and current projects.

Chapter 3: **Code**

3.1 Python Script

```
""" The Logging Module
The logging module in Python is a ready-to-
use and powerful module that is designed to meet the needs of beginners as well a
s enterprise teams. It is used by most of the third-
party Python libraries, so you can integrate your log messages with the ones from
 those libraries to produce a homogeneous log for your application.

Adding logging to your Python program is as easy as this: """
import logging
import logging.config
from yaml import safe_load

""" With the logging module imported, you can use something called a “logger” to
log messages that you want to see. By default, there are 5 standard levels indica
ting the severity of events. Each has a corresponding method that can be used to
log events at that level of severity. The defined levels, in order of increasing
severity, are the following:

DEBUG
INFO
WARNING
ERROR
CRITICAL
The logging module provides you with a default logger that allows you to get star
ted without needing to do much configuration. The corresponding methods for each
level can be called as shown in the following example: """
def The_Logging_Module():
    logging.debug('This is a debud message')
    logging.info('This is a info message')
    logging.warning('This is a warning message')
    logging.error('This is an error message')
    logging.critical('This is a critical message')


""" output:
WARNING:root:This is a warning message
ERROR:root:This is an error message
CRITICAL:root:This is a critical message """
```

```
""" ------------------------------------------------------------------------
---------------------- """

def basic_config():
    logging.basicConfig(level='DEBUG', filename='app.log', filemode='a', format='
%(name)s - %(levelname)s - %(message)s')
    The_Logging_Module()

""" output:
root - DEBUG - This is a debud message
root - INFO - This is a info message
root - WARNING - This is a warning message
root - ERROR - This is an error message
root - CRITICAL - This is a critical message
root - DEBUG - This is a debud message
root - INFO - This is a info message
root - WARNING - This is a warning message
root - ERROR - This is an error message
root - CRITICAL - This is a critical message """


def formatting_output():
    logging.basicConfig(level='DEBUG', filename='app.log', filemode='a', format='
%(process)d-%(levelname)s-%(message)s')
    logging.warning('This is a warning with process id')

""" output:
6756-WARNING-This is a warning with process id """

def formatting_output_2():
    logging.basicConfig(level='INFO', filename='app.log', filemode='a', format='%
(asctime)s-%(message)s', datefmt='%d-%b-%y %H:%M:%S')
    logging.info('This is info with date and time')

""" output:
2019-12-19 11:59:40,336-This is info with date and time
19-Dec-19 12:02:01-This is info with date and time """

def logging_variable_data():
    name = 'John'
    logging.basicConfig(level='DEBUG', filename='app.log', filemode='a', format='
%(asctime)s-%(process)d-%(levelname)s-%(message)s')
    logging.error('%s raised an error', name)

""" output:
```

```
2019-12-19 12:14:49,628-10088-ERROR-John raised an error """

def logging_variables_with_fstrings():
    animal = 'cat'
    logging.basicConfig(level='DEBUG', filename='app.log', filemode='a', format='
%(asctime)s-%(process)d-%(levelname)s-%(message)s')
    logging.error(f'{animal} has encountered an error')

""" output:
2019-12-19 12:17:07,614-12612-ERROR-cat has encountered an error """
# pylint: disable=unused-variable
def logging_capturing_stack_traces():
    logging.basicConfig(level='DEBUG', filename='app.log', filemode='a', format='
%(asctime)s-%(process)d-%(levelname)s-%(message)s')
    a = 5
    b = 0
    try:
        c = a/b
    except Exception as e:
        logging.error("Exception occurred", exc_info=True)
# pylint: disable=anomalous-backslash-in-string
""" output:
2019-12-21 11:44:19,752-12180-ERROR-Exception occurred
Traceback (most recent call last):
  File "j:\Education\Code\Python\Python-
Logging\log_script.py", line 83, in logging_capturing_stack_traces
    c = a/b
ZeroDivisionError: division by zero """ # pylint: disable=anomalous-backslash-in-
string

def logging_capturing_stack_traces2():
    logging.basicConfig(level='DEBUG', filename='app.log', filemode='a', format='
%(asctime)s-%(process)d-%(levelname)s-%(message)s')
    a = 5
    b = 0
    try:
        c = a/b
    except Exception as e:
        logging.exception("Exception has occurred")

""" output:
2019-12-21 11:47:11,199-8548-ERROR-Exception has occurred
Traceback (most recent call last):
  File "j:\Education\Code\Python\Python-
Logging\log_script.py", line 99, in logging_capturing_stack_traces2
```

```
    c = a/b
ZeroDivisionError: division by zero
 """
# Classes and Functions
""" The most commonly used classes defined in the logging module are the followin
g:

Logger: This is the class whose objects will be used in the application code dire
ctly to call the functions.

LogRecord: Loggers automatically create LogRecord objects that have all the infor
mation related to the event being logged, like the name of the logger, the functi
on, the line number, the message, and more.

Handler: Handlers send the LogRecord to the required output destination, like the
 console or a file. Handler is a base for subclasses like StreamHandler, FileHand
ler, SMTPHandler, HTTPHandler, and more. These subclasses send the logging output
s to corresponding destinations, like sys.stdout or a disk file.

Formatter: This is where you specify the format of the output by specifying a str
ing format that lists out the attributes that the output should contain. """

def logging_example_logger():
    name = __name__
    logger = logging.getLogger(name=name)
    logger.error("This is an example logger")
""" output:
This is an example logger """

#Handlers
""" Handlers come into the picture when you want to configure your own loggers an
d send the logs to multiple places when they are generated. Handlers send the log
 messages to configured destinations like the standard output stream or a file or
 over HTTP or to your email via SMTP.

A logger that you create can have more than one handler, which means you can set
it up to be saved to a log file and also send it over email.

Like loggers, you can also set the severity level in handlers. This is useful if
you want to set multiple handlers for the same logger but want different severity
 levels for each of them. For example, you may want logs with level WARNING and a
bove to be logged to the console, but everything with level ERROR and above shoul
d also be saved to a file. """

def logging_handler_example():
```

```
    name = __name__

    # Create a custom logger
    logger = logging.getLogger(name=name)

    # Create handlers
    c_handler = logging.StreamHandler()
    f_handler = logging.FileHandler('app.log')
    c_handler.setLevel(logging.WARNING)
    f_handler.setLevel(logging.ERROR)

    # Create formatters and add it to handlers
    c_format = logging.Formatter('%(name)s - %(asctime)s-%(process)d-
%(levelname)s-%(message)s')
    f_format = logging.Formatter('%(name)s - %(asctime)s-%(process)d-
%(levelname)s-%(message)s')
    c_handler.setFormatter(c_format)
    f_handler.setFormatter(f_format)

    # Add handlers to the logger
    logger.addHandler(c_handler)
    logger.addHandler(f_handler)

    logger.warning("this is a warning")
    logger.error('this is an error')

""" output:
__main__ - 2020-01-06 13:03:53,024-5436-WARNING-this is a warning
__main__ - 2020-01-06 13:03:53,083-5436-ERROR-this is an error """

# Other Configuration Methods
""" You can configure logging as shown above using the module and class functions
 or by creating a config file or a dictionary and loading it using fileConfig() o
r dictConfig() respectively. These are useful in case you want to change your log
ging configuration in a running application. """

def logging_file_config():
    logging.config.fileConfig(fname='file.conf',disable_existing_loggers=False)

    logger = logging.getLogger('sampleLogger')

    logger.debug('This is a debug message')

def logging_yaml_config():
    with open('config.yaml', 'r') as f:
```

```
        config = safe_load(f.read())
        logging.config.dictConfig(config)
    logger = logging.getLogger('sampleLogger')
    logger.debug("This is a debug message")

The_Logging_Module()
basic_config()
formatting_output()
formatting_output_2()
logging_variable_data()
logging_variables_with_fstrings()
logging_capturing_stack_traces()
logging_capturing_stack_traces2()
logging_example_logger()
logging_handler_example()
logging_file_config()
logging_yaml_config()
```

3.2 CONF Config File

```
[loggers]
keys=root,sampleLogger

[handlers]
keys=consoleHandler,fileHandler

[formatters]
keys=sampleFormatter

[logger_root]
level=DEBUG
handlers=consoleHandler

[logger_sampleLogger]
level=DEBUG
handlers=consoleHandler,fileHandler
qualname=sampleLogger
propagate=0

[handler_fileHandler]
class=FileHandler
level=DEBUG
formatter=sampleFormatter
args=('app.log', 'a',)

[handler_consoleHandler]
class=StreamHandler
level=DEBUG
formatter=sampleFormatter
args=(sys.stdout,)

[formatter_sampleFormatter]
format=%(name)s - %(asctime)s-%(process)d-%(levelname)s-%(message)s
```

3.3 YAML Config File

```yaml
version: 1
formatters:
  simple:
    format: '%(name)s - %(asctime)s-%(process)d-%(levelname)s-%(message)s'
handlers:
  console:
    class: logging.StreamHandler
    level: DEBUG
    formatter: simple
    stream: ext://sys.stdout
  fileHandler:
    class: logging.FileHandler
    level: DEBUG
    formatter: simple
    filename: 'app.log'
    mode: 'a'
loggers:
  sampleLogger:
    level: DEBUG
    handlers: [console,fileHandler]
    propagate: no
root:
  level: DEBUG
  handlers: [console]
```

Chapter 4: **Output Logs**

```
root - DEBUG - This is a debud message
root - INFO - This is a info message
root - WARNING - This is a warning message
root - ERROR - This is an error message
root - CRITICAL - This is a critical message
root - DEBUG - This is a debud message
root - INFO - This is a info message
root - WARNING - This is a warning message
root - ERROR - This is an error message
root - CRITICAL - This is a critical message
6756-WARNING-This is a warning with process id
2019-12-19 11:59:40,336-This is info with date and time
19-Dec-19 12:02:01-This is info with date and time
2019-12-19 12:14:49,628-10088-ERROR-John raised an error
2019-12-19 12:17:07,614-12612-ERROR-cat has encountered an error
2019-12-21 11:44:19,752-12180-ERROR-Exception occurred
Traceback (most recent call last):
  File "j:\Education\Code\Python\Python-
Logging\log_script.py", line 83, in logging_capturing_stack_traces
    c = a/b
ZeroDivisionError: division by zero
2019-12-21 11:47:11,199-8548-ERROR-Exception has occurred
Traceback (most recent call last):
  File "j:\Education\Code\Python\Python-
Logging\log_script.py", line 99, in logging_capturing_stack_traces2
    c = a/b
ZeroDivisionError: division by zero
This is an example logger
__main__ - 2020-01-06 13:03:53,083-5436-ERROR-this is an error
sampleLogger - 2020-01-14 13:06:59,849-7604-DEBUG-This is a debug message
sampleLogger - 2020-01-14 13:19:53,775-13564-DEBUG-This is a debug message
```

Chapter 5: **Project Status**

Project has been successfully completed.