PYTHON MODULE TO FIND A LOG PATTERN

ABSTRACT

Logging is a python module in the standard library that provides the facility to work with the framework for releasing log messages from python programs. This module is widely used by the developers when they work to logging. It is very important tool which used in software development, running, and debugging. The logging is a powerful module used by the beginners as well as enterprises. This module provides a proficiency to organize different control handlers and a transfer log messages to these handlers. Now, we will call the logger to log messages that we want to see. The logging module offers the five levels that specify the severity of events. Each event contains the parallel methods that can be used to log events at the level of severity. The system involves the user creating one or more logger objects on which methods are called to log debugging notes, general information, warnings, errors, critical, can be used in the logging pattern. The logging module offers many features. It consists of several constants, classes, and methods. The constants are represented by the all caps latter; the classes are represented by capital letters. The items with lowercase represent methods are used in logging module.

INTRODUCTION

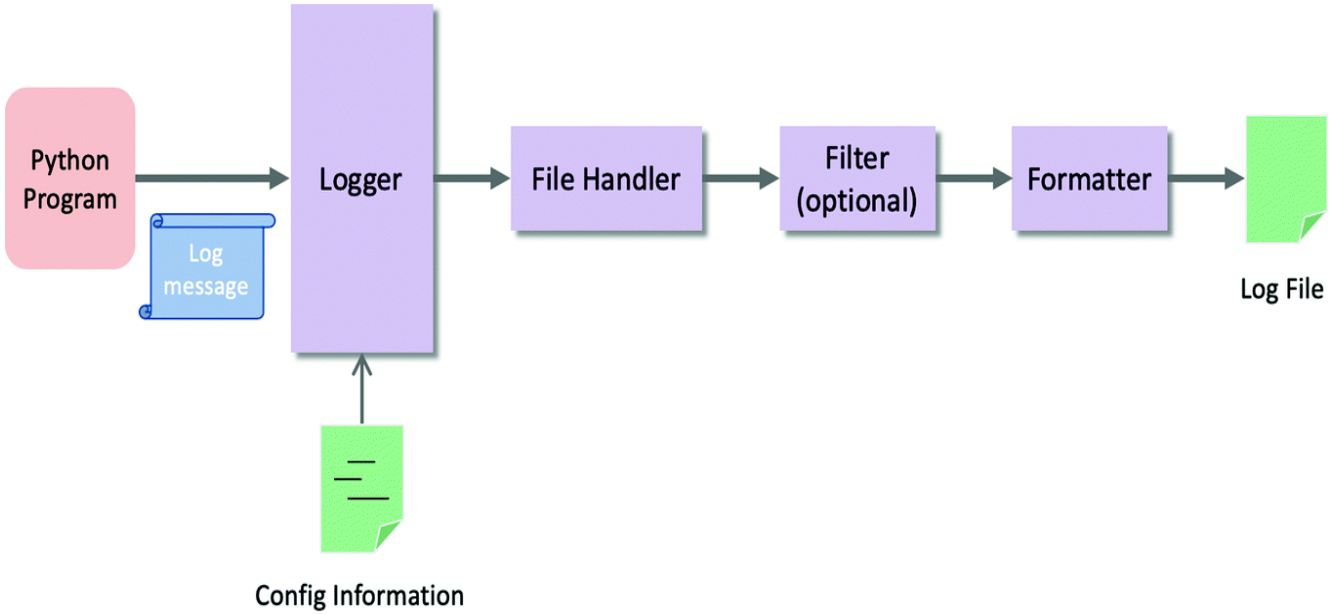
Logging is a very useful tool in a programmer's toolbox. It can help develop a better understanding of the flow of a program and discover scenarios that you might not even have thought of while developing. Logs provide developers with an extra set of eyes that are constantly looking at the flow that an application is going through. They can store information, like which user or IP accessed the application. If an error occurs, then they can provide more insights than a stack trace by telling you what the state of the program was before it arrived at the line of code where the error occurred. By logging useful data from the right places, you can not only debug errors easily but also use the data to analyze the performance of the application to plan for scaling or look at usage patterns to plan for marketing. Python provides a logging system as a part of its standard library, so you can quickly add logging to your application. In this article, you will learn why using this module is the best way to add logging to your application. The logging module in python is a ready-to-use and powerful module that is designed to meet the needs of beginners as well as 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 you application. Adding logging to

your python program is an easy as this “import logging”. With the logging module imported, you can use something called a “logger” to log messages that you want to see. By default, there are five standard levels indicting the severity of events. Each has a corresponding method that can be used to log events at the level of severity. The defined levels, in order of increasing severity, are the following are DEBUG, INFO, WARNING, ERROR and CRITICAL. The logging module provides you with a default logger that allows you to get started without needing to do much configuration. In the log module you can use the basicConfig (**args) method to configure the logging. Some of the commonly used parameters for basicConfig() are the following:

- Level: The root logger will be sent to the specified severity level.
- Filename: This specifies the file.
- Filemode: If filename is given, the file is opened in this mode. The default is a, which means append.
- Format: This is the format of the log message.

By using the level parameter, you can set what level of log messages you want to record. This can be done by passing one of the constants available in the class, and this would enable all logging calls at or above that level to be logged.

LOGGING MODULE STRUCTURE



Python has included a built-in logging modules. This module, the logging module, defines functions and classes which implement a flexible logging framework that can be used in any python application/ script or in python libraries/ modules.

PROGRAM

```
import logging

thelogcounts= { ‘INFO’:0, ‘ERROR’:0, ‘DEBUG’:0, ‘CRITICAL’:0,
‘WARNING’:0}

class ContextFilter(logging.Filter):

    def filter(self, record):

        global thelogcounts

        record.count = thelogcounts[record.levelname]

        thelogcounts[record.levelname] +=1

        return True

logging.basicConfig(

    level =logging.DEBUG,

    format = ‘%(asctime)s: %(levelname)-8s: %(levelno)s: %(count)s:
%(message)s’)

    logger = logging.getLogger(__file__)
```

```
logger.addFilter(ContextFilter())

logger.info(“This is a INFO message!”)

logger.debug(“This is a DEBUG message!”)

logger.error(“This is a ERROR message!”)

logger.critical(“This is a CRITICAL message!”)

logger.warning(“This is a WARNING message!”)

logger.info(“This is a INFO message!”)

logger.info(“This is a INFO message!”)

logger.info(“This is a INFO message!”)

logger.info(“This is a INFO message!”)

logger.info(“This is a INFO message!”)

logger.debug(“This is a DEBUG message!”)

logger.debug(“This is a DEBUG message!”)

logger.error("This is a ERROR message!")

logger.error("This is a ERROR message!")
```

```
logger.error("This is a ERROR message!")

logger.error("This is a ERROR message!")

logger.error("This is a ERROR message!")

logger.critical("This is a CRITICAL message!")

logger.critical("This is a CRITICAL message!")

logger.critical("This is a CRITICAL message!")

logger.critical("This is a CRITICAL message!")

logger.critical("This is a CRITICAL message!")

logger.warning("This is a WARNING message!")

logger.warning("This is a WARNING message!")

logger.warning("This is a WARNING message!")

logger.warning("This is a WARNING message!")

logger.warning("This is a WARNING message!")
```

EXPLANATION

- With the logging module imported, you can use something called a “logger” to log messages that you want to see.

- Trace- Only when I would be “tracing” the code and trying to find one part of a function specifically.
- Import logging from random import choice class ContextFilter(logging.Filter): “” This is a filter which injects contextual information into the log.
- This filter assumes that we want INFO logging from all. Modules and DEBUG logging from only selected ones, but easily could be adapted for other policies.
- To add ContextFilter which brings in the global counter.
- The record count log report enables you to identify how nodes from different applications and dimensions contribute to the total record count determined by the system.
- If one returns a false value, the handler will not emit the record and levelno for the numeric value and levelname for the corresponding level name.
- Does basic configuration for the logging system by creating a StreamHandler with default formatter and adding it to root logger. The functions debug(), info(), warning(), error() and critical() will call basicConfig automatically if no handler defined for root logger.
- Loggers have the following attributes and methods.

- Logging is performed by calling methods on instances of the logger class added to both logger and handler instances (through their addFilter() method).
- If logging level is set to WARNING, INFO, DEBUG then the logger will print to or write WARNING lines to the console or log file.

OUTPUT

```
2022-03-24 15:33:49,350: INFO    : 20: 0: This is a INFO message!
2022-03-24 15:33:49,350: DEBUG   : 10: 0: This is a DEBUG message!
2022-03-24 15:33:49,350: ERROR   : 40: 0: This is a ERROR message!
2022-03-24 15:33:49,350: CRITICAL: 50: 0: This is a CRITICAL message!
2022-03-24 15:33:49,350: WARNING : 30: 0: This is a WARNING message!
2022-03-24 15:33:49,350: INFO    : 20: 1: This is a INFO message!
2022-03-24 15:33:49,350: INFO    : 20: 2: This is a INFO message!
2022-03-24 15:33:49,350: INFO    : 20: 3: This is a INFO message!
2022-03-24 15:33:49,350: INFO    : 20: 4: This is a INFO message!
2022-03-24 15:33:49,350: INFO    : 20: 5: This is a INFO message!
2022-03-24 15:33:49,350: DEBUG   : 10: 1: This is a DEBUG message!
2022-03-24 15:33:49,350: DEBUG   : 10: 2: This is a DEBUG message!
2022-03-24 15:33:49,350: ERROR   : 40: 1: This is a ERROR message!
2022-03-24 15:33:49,350: ERROR   : 40: 2: This is a ERROR message!
2022-03-24 15:33:49,350: ERROR   : 40: 3: This is a ERROR message!
2022-03-24 15:33:49,350: ERROR   : 40: 4: This is a ERROR message!
2022-03-24 15:33:49,350: ERROR   : 40: 5: This is a ERROR message!
2022-03-24 15:33:49,351: CRITICAL: 50: 1: This is a CRITICAL message!
2022-03-24 15:33:49,351: CRITICAL: 50: 2: This is a CRITICAL message!
2022-03-24 15:33:49,351: CRITICAL: 50: 3: This is a CRITICAL message!
2022-03-24 15:33:49,351: CRITICAL: 50: 4: This is a CRITICAL message!
2022-03-24 15:33:49,351: CRITICAL: 50: 5: This is a CRITICAL message!
2022-03-24 15:33:49,351: WARNING : 30: 1: This is a WARNING message!
2022-03-24 15:33:49,351: WARNING : 30: 2: This is a WARNING message!
2022-03-24 15:33:49,351: WARNING : 30: 3: This is a WARNING message!
2022-03-24 15:33:49,351: WARNING : 30: 4: This is a WARNING message!
2022-03-24 15:33:49,351: WARNING : 30: 5: This is a WARNING message!
```

CONCLUSION

The logging module lets you track events when your code runs so that when the code crashes you can check the logs and identify what caused it. Log messages have a built-in hierarchy – starting from debugging, informational, warning, error and critical message. You can include traceback information as well. It is designed for small to large python projects with multiple modules and is highly recommended for any modular python programming.