Python Programming

by Narendra Allam

Chapter 15

Logging

Topics Covering

- **Basic Logging**
- **File Logging**
- **Rotating Filer handler**
- **Log formatting**

Python Logging

Logging is one for most important part in the programming. It is used to track how the program in running.
Prints can also be used but only for smaller programs and cannot be used for complex, large programs and for
programs running in the background hence we require the logging functionality.

In simple terms logging is nothing but adding messages into the file during program running to know the state and status of it.

We can log the message with different types of criticality levels hence based on this we can segregate them so that can
identify critical issues.

Below are different types of levels for logging message, based on the requirement we can set the log levels

CRITICAL:
    This level having the highest priority. This can be used for memory exceptions or index error.
    Numeric value for this level is 50

ERROR:
    We can use this level if there are any logical errors in code and environment related errors while setup.
    for eg., try to open file and failed, value error when converting string to int or vice versa

Numeric value for this level is 40

WARNING:

These are soft messages and no blockage in executing the code. Can be used for wrong password.

Numeric value for this level is 30

INFO:

This is just to know the information or progress of a task or program. In following cases it can be used.

for eg., service started successfully, successful login

Numeric value for this level is 20

DEBUG:

This can be used to know intermediate results of some operation or task and verify or evaluate the results.

for eg., results of adding two lists or to know contents of dict.

Numeric value for this level is 10

*Numeric value is just to define its priority based on the number

Based on the level set the specific level messages are logged. If the level set to DEBUG all types of messages are logged into file.

Let us see the basic logging functionality

Basic Logging:

```
import logging

# return the logger instance with the given name
logger = logging.getLogger(__name__) # can provide module name for the logger

# logs message on the screen
logger.warning("Warning message")
logger.error("Error message")
logger.critical("Critical message")
-------------------------------------------------------------------
```

Below is the method to log messages into the file

FileHandler:

It is the basic handler used to write the logging messages in to the file which can be used to debug and analyze the code.

Also used to maintain the state and status of the program execution at particular time. Used to analyze and resolve the issues.

```
-----------------------------------------------------------------
import logging
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

# return the file handler to write messages into file
handler = logging.FileHandler("test_log", mode="a")

# add handler to the logger
logger.addHandler(handler)
logger.error("error message in to file")
logger.warning("warning message in to file")
logger.info("Dictionary contents %s" %({'key': 'value'}))
-----------------------------------------------------------------
```

Open file and see the contents

```
>>> cat test_log
ERROR:__main__:error message in to file
ERROR:__main__:warning message in to file
INFO:__main__:Dictionary contents {'key': 'value'}
>>>
```

RotatingFileHandler:

If we use the file handler at some point of time it will fill the whole disk because there is no limit in file size.

In order to avoid this situation RotatingFileHandler can be used which is having the limit size (can set the file size).

We can provide the size of file in maxBytes and backupCount is used to take the backup of the log file once it reached to maxBytes.

Based on backupCount it will maintain those number of backup files.

for eg., backupCount is 2 then backup files created test_log.1, test_log.2. Current logs are written to test_log once limit is full it renames

```
-----------------------------------------------------------------
import logging
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

# return the rotating file handler to write messages into file
handler = logging.RotatingFileHandler("test_log", mode="a", maxBytes=1024,
backupCount=5)

# add handler to the logger
logger.addHandler(handler)
logger.error("error message in to file")
-----------------------------------------------------------------
```

List the directory to view the files

```
>>> ls | grep test_log
test_log  test_log.1 test_log.2 test_log.3 test_log.4 test_log.5
>>>
```

Logs Formatting:

For better human readability and to search the content in file in better way we can format and log the message into file.

Below are the attributes can be used in formatting the message.

```
        %(name)s          Name of the logger (logging channel)
  %(levelno)s        Numeric logging level for the message (DEBUG, INFO,
                WARNING, ERROR, CRITICAL)
  %(levelname)s      Text logging level for the message ("DEBUG", "INFO",
                "WARNING", "ERROR", "CRITICAL")
  %(pathname)s       Full pathname of the source file where the logging
                call was issued (if available)
  %(filename)s       Filename portion of pathname
  %(module)s         Module (name portion of filename)
  %(lineno)d         Source line number where the logging call was issued
                (if available)
  %(funcName)s       Function name
  %(created)f        Time when the LogRecord was created (time.time()
                return value)
  %(asctime)s        Textual time when the LogRecord was created
  %(msecs)d          Millisecond portion of the creation time
  %(relativeCreated)d Time in milliseconds when the LogRecord was created,
                relative to the time the logging module was loaded
                (typically at application startup time)
  %(thread)d         Thread ID (if available)
  %(threadName)s     Thread name (if available)
  %(process)d        Process ID (if available)
  %(message)s        The result of record.getMessage(), computed just as
                the record is emitted
```

Example program to format and log the message by using the above attributes

```
-------------------------------------------------------------------
import logging
logger = logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

formatter = logging.Formatter("%(asctime)s %(levelname)s %(lineno)d %(message)s")

# return the file handler to write messages into file
```

```
handler = logging.FileHandler("test_log", mode="a")
handler.setFormatter(formatter)

# add handler to the logger
logger.addHandler(handler)
logger.error("error message in to file")
logger.warning("warning message in to file")
-------------------------------------------------------------------
```

Contents of the log file
There are four parts in below message first part is time, second part is log level ERROR and INFO, third part is the line number where the
logger message is placed in code and fourth part is the actual message.

```
>>>cat test_log
2018-01-10 23:44:12,391 ERROR 48 error message in to file
2018-01-10 23:44:12,392 INFO 49 Dictionary contents {'key': 'value'}
>>>
```

Following is how to set the basic configuration of logging system.
This is to ensure that at least one handler is available. If the logger is added any handler this function does nothing.

Attributes used for the basicConfig:

```
        filename  Specifies file name to create FileHandler
  filemode  Specifies the mode to open the file (defaults to append mode).
  format    Use the specified format string for the handler.
  datefmt   Use the specified date/time format.
  level     Set the root logger level to the specified level.
  stream    Use the specified stream to initialize the StreamHandler. (file stream can be
used: open(filename, mode))
```

```
--------------------------------------------------------------------
import logging

logging.basicConfig(filename="test_log", filemode="a", level=logging.DEBUG,
format="%(asctime)s %(levelname)s %(lineno)d %(message)s")
logger = logging.getLogger(__name__)

logger.info("info message in to file")

--------------------------------------------------------------------
```