Logging in Python

Introduction:

- Python provides a very dynamic and ready-to-use logging system – logging Module.
- Be default it has 5 levels indicating the severity of events. The levels in order of increasing severity:

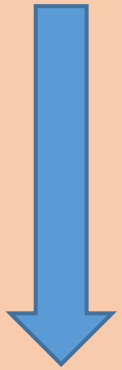
DEBUG

INFO

WARNING

ERROR

CRITICAL



Call the Logging Methods:

```
import logging

logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

- Notice that the debug() and info() messages didn't get logged. This is because, by default, the logging module logs the messages with a severity level of WARNING or above.
- This can be configured.
- We can also define our own severity levels.

Configure Logging

- Use the method basicConfig(**kwargs) to configure the logging.
- Some of commonly used parameters of this method:

  level: Set the severity level

  filename: This specifies the file.

  filemode: Default is append(a) mode.

  format: This is the format of the log message.

```
import logging

logging.basicConfig(level=logging.DEBUG)
logging.debug('This will get logged')

logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

Configure Logging

- Configure Logging to a File:

```
logging.basicConfig(filename='loggingTest.log', filemode='w',
                    format='%(name)s - %(levelname)s - %(message)s')
logging.warning('This will get logged to a file')
```

```
logging.basicConfig(level=logging.DEBUG, filename='loggingTest.log', filemode='w',
                    format =' %(asctime)s - %(name)s - %(levelname)s - %(message)s')
logging.debug('Debug is Logged')
```

Integrating Logging and Exception Stack Trace

- Python allows to capture the exception stack traces in the logging system using a parameter called exc_info.

```
import logging
try:
   print(sal)
except Exception as exp:
   logging.basicConfig(level=logging.DEBUG, filename='loggingTest.log', filemode='w',
                       format =' %(name)s - %(levelname)s - %(message)s')
   logging.error('Some Exception Occured.',exc_info=True)
```

Custom Logger

- So far we have used a default Logger – root.
- When our application has multiple modules, create custom logger for each module as an object from Logger Class.
- Classes in Logger Class:

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

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

Handler: Handlers send the LogRecord to the required output destination, like the console or a file.

Formatter : This is where you specify the format of the output

Custom Logger

- Notes:

1. Like root logger, this custom logger will not have a default output format. So we have to set it up.

2. Unlike the root logger, a custom logger can't be configured using basicConfig(). We have to configure it.

Custom Logger

```
import logging

# Create a Custom Logger
logger = logging.getLogger(__name__)

#Create Handlers
f_handler = logging.FileHandler('loggingTest.log',mode='w')
f_handler.setLevel(logging.ERROR)

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

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

# Log the output
logger.error('This is an error')
```

Logging Using Configuration File

- Create a configuration file and then import that file into each module.
- This is because we can change the configuration file with out changing the program codes.
- Also using this configuration file, we can also change the settings in a running application.

```
[loggers]
keys=root, customLogger

[handlers]
keys=sampleHandler

[formatters]
keys=sampleFormatter

[logger_root]
level=DEBUG
handlers=sampleHandler

[logger_customLogger]
level=DEBUG
handlers=sampleHandler
qualname=customLogger
propagate=0

[handler_sampleHandler]
class=FileHandler
level=DEBUG
formatter=sampleFormatter
args=('LoggingTest.log','w')

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