# Logging, & JSON

## Logging:

Logging is a means of tracking events that happen when some software runs. Logging is important for software developing, debugging, and running. If you don't have any logging record and your program crashes, there are very few chances that you detect the cause of the problem. And if you detect the cause, it will consume a lot of time. With logging, you can leave a trail of breadcrumbs so that if something goes wrong, we can determine the cause of the problem.

### Python Logging Levels

There are five built-in levels of the log message.

- **Debug:** These are used to give Detailed information, typically of interest only when diagnosing problems.
- **Info:** These are used to confirm that things are working as expected
- **Warning:** These are used as an indication that something unexpected happened, or is indicative of some problem in the near future
- **Error:** This tells that due to a more serious problem, the software has not been able to perform some function
- **Critical:** This tells serious error, indicating that the program itself may be unable to continue running

There are several logger objects offered by the base Handler itself.

- **Logger.info(msg):** This will log a message with level INFO on this logger.
- **Logger.warning(msg):** This will log a message with a level WARNING on this logger.
- **Logger.error(msg):** This will log a message with level ERROR on this logger.
- **Logger.critical(msg):** This will log a message with level CRITICAL on this logger.
- **Logger.log(lvl,msg):** This will Log a message with integer level lvl on this logger.
- **Logger.exception(msg):** This will log a message with level ERROR on this logger.
- **Logger.setLevel(lvl):** This function sets the threshold of this logger to lvl. This means that all the messages below this level will be ignored.
- **Logger.addFilter(filt):** This adds a specific filter fit into this logger.
- **Logger.removeFilter(filt):** This removes a specific filter fit into this logger.
- **Logger.filter(record):** This method applies the logger's filter to the record provided and returns True if the record is to be processed. Else, it will return False.

# Logging, & JSON

- **Logger.addHandler(hdlr):** This adds a specific handler hdlr to this logger.
- **Logger.removeHandler(hdlr) :** This removes a specific handler hdlr into this logger.
- **Logger.hasHandlers():** This checks if the logger has any handler configured or not.

### Useful Handlers

| Handler | Description |
|---|---|
| **StreamHandler** | Sends messages to streams (file-like objects). |
| **FileHandler** | Sends messages to disk files. |
| **BaseRotatingHandler** | Base class for handlers that rotate log files at a certain point. Use **RotatingFileHandler** or **TimedRotatingFileHandler** inst ead. |
| **RotatingFileHandler** | Sends messages to disk files, with support for maximum log file sizes and log file rotation. |
| **TimedRotatingFileHandl er** | Sends messages to disk files, rotating the log file at certain timed intervals. |
| **SocketHandler** | Sends messages to TCP/IP sockets. Also supports Unix domain sockets since Python 3.4. |
| **DatagramHandler** | Sends messages to UDP sockets. Also supports Unix domain sockets since Python 3.4. |
| **SMTPHandler** | Sends messages to a designated email address. |
| **SysLogHandler** | Sends messages to a Unix Syslogthe daemon, possibly on a remote machine. |

# Logging, & JSON

| Handler | Description |
|---|---|
| **NTEventLogHandler** | Sends messages to a Windows NT/2000/XP event log. |
| **MemoryHandler** | Sends messages to a buffer in memory, which is flushed whenever specific criteria are met. |
| **HTTPHandler** | Sends messages to an HTTP server using either GET or POST semantics. |
| **WatchedFileHandler** | Watches the file it is logging to. If the file changes, it is closed and reopened using the file name. |
| **QueueHandler** | Sends messages to a queue, such as those implemented in the queue or multiprocessing modules. |
| **NullHandler** | Does nothing with error messages. Used by library developers to avoid 'No handlers could be found for logger' message. |

**Try yourself:**

**Code 1:**

```
import logging
name = 'GFG'
logging.error('%s raised an error', name)
```

#Output- Displaying by associates

**Code 2:**

```
# importing module
import logging
```

# Logging, & JSON

```
# Create and configure logger
logging.basicConfig(filename="newfile.log",
                                    format='%(asctime)s %(message)s',
                                    filemode='w')

# Creating an object
logger = logging.getLogger()

# Setting the threshold of logger to DEBUG
logger.setLevel(logging.DEBUG)

# Test messages
logger.debug("Harmless debug Message")
logger.info("Just an information")
logger.warning("Its a Warning")
logger.error("Did you try to divide by zero")
logger.critical("Internet is down")
```

# Logging, & JSON

```
# importing module
import logging

# Create and configure logger
logging.basicConfig(filename="newfile.log",
                    format='%(asctime)s %(message)s',
                    filemode='w')

# Creating an object
logger = logging.getLogger()

# Setting the threshold of logger to DEBUG
logger.setLevel(logging.DEBUG)

# Test messages
logger.debug("Harmless debug Message")
logger.info("Just an information")
logger.warning("Its a Warning")
logger.error("Did you try to divide by zero")
logger.critical("Internet is down")
```

```
DEBUG:root:Harmless debug Message
INFO:root:Just an information
WARNING:root:Its a Warning
ERROR:root:Did you try to divide by zero
CRITICAL:root:Internet is down
```

## JSON:

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

import json

# Logging, & JSON

# some JSON:

x = '{ "name":"John", "age":30, "city":"New York"}'

# parse x:

y = json.loads(x)

# the result is a Python dictionary:

print(y["age"])

JSON to Python:

```
import json

# some JSON:
x = '{ "name":"John", "age":30, "city":"New York"}'

# parse x:
y = json.loads(x)

# the result is a Python dictionary:
print(y["age"])
```

```
30
```

When you convert from Python to JSON, Python objects are converted into the JSON (JavaScript) equivalent:

| Python | JSON |
|---|---|
| dict | Object |
| list | Array |
| tuple | Array |
| str | String |

# Logging, & JSON

| | |
|---|---|
| int | Number |
| float | Number |
| True | true |
| False | false |
| None | null |

Python to JSON:

```
import json

# a Python object (dict):
x = {
  "name": "John",
  "age": 30,
  "city": "New York"
}

# convert into JSON:
y = json.dumps(x)

# the result is a JSON string:
print(y)S
```

```
{"name": "John", "age": 30, "city": "New York"}
```

**Try yourself:**

json.dumps(x, indent=4, separators=(". ", " = "))

json.dumps(x, indent=4, sort_keys=True)

***Thank you***