5 levels:

# DEBUG: Detailed information, typically of interest only when diagnosing problems

# INFO: Confirmation that things are working as expected.

# WARNING: An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected. (Default level)

# ERROR: Due to a more series problem, the software has been not able to perform some function.

# CRITICAL: A serious error, indicating that the program itself may be unable to continue running.

How default logging level work is that it will only log the default level or higher information into the console. If the default is warning, it will not log **debug** or **info** level information.

To change basic configuration,

logging.basicConfig(level=logging.DEBUG) Debug has to be capitalization. This means the constant of that level.

Then, the debug level can be logged.

logging.basicConfig(filename=’test.log’,level=logging.DEBUG) To create a .txt file for the logging result in current working directory.

To create a .txt file for the logging result and change the logging output format:

See <https://docs.python.org/3/library/logging.html>

logging.basicConfig(filename=’test.log’, level=logging.DEBUG, fortmat = ‘%(asctime)s:%(levelname)s:%(message)s)’)

root means the root logger. Root logger is good for smaller applications, but it can be inefficient if the application gets bigger.

When you import a module, it actually runs the code in that module you imported. The logger used if applicable will be based on the module you imported as well.

If you have logging.basicConfig in the imported module, and you have another one in the working module(where you import into) then the working module’s basicConfig will be based on the imported, and it does not overwrite imported’s.

To solve this, create different logger for each module.

import logging

logger = logging.getLogger(\_\_name\_\_) create logger using the working module name

and you use logger.info(…)

logger.setLevel(logging.INFO)

formatter = logging.Formatter(‘%(asctime)s:%(levelname)s:%(name)s:%(message)s)’)

file\_handler = logging.FileHandler(‘employee.log’) Basically a variable that configures employee.log instead of root.

file\_handler.setFormatter(formatter) set formatting

logger.addHandler(file\_handler) Add Handler

or stream\_handler = logging.StreamHandler()

logger.exception(‘…’) this can enable traceback

~~logging.basicConfig(filename=’test.log’, level=logging.DEBUG, fortmat = ‘%(asctime)s:%(levelname)s:%(message)s)’)~~