**Logging module**

* Used to track the activities
* Used while debugging, track no. Of requests made in a day.

**Advantages:**

1. To perform debugging and to prepare RCA (Root Cause Analysis).
2. To track information (or) to provide statistics like no. Of requests per day etc.

While implementing logging, we have to decide what type of information we are going to log. These are called “logging levels”.

**Logging levels: 6 levels**

1. **CRITICAL** : application is almost going to terminate, needs high attention: 50
2. **ERROR** : error but not much danger to the application : 40
3. **WARNING**: some alert needed : 30
4. **INFO**: represents a message with some important information: 20
5. **DEBUG**: contains debugging information: 1
6. **NOTSET**: not logging set: 0

Be default, the logging level will be **WARNING**

* Only **WARNING, ERROR, CRITICAL** are displayed by default (if level is set as WARNING, then WARNING and its higher levels are displayed i.e. WARNING, ERROR and CRITICAL
* If we set the level as INFO, INFO and its higher levels are displayed.
* If the level set is DEBUG, then all levels are displayed.
* Other wise, we can set the default level also.

**Implementing Logging**

1. We have to create a file where we can store log data.
2. We have to specify which level messages we have to store.

To do this, we need “logging” module and “basicConfig()” method of the logging module.

Ex: logging.basicConfig(filename=”log.txt”,level=logging.WARNING)

Here in “log.txt” file, we can store WARNING (or) higher level messages.

We can write messages to the log file using the following methods:

* logging.debug(message)
* logging.info(msg)
* logging.warning(msg)
* logging.error(msg)
* logging.critical(msg)

other 2 logging calls are:

logging.log()

logging.exception()

Ex: Write a program to create a log file and write warning and higher level messages

import logging

logging.basicConfig(filename="log.txt",level=logging.WARNING)

print("Python logging demo")

logging.debug("debug msg")

logging.info("info msg")

logging.warning("warning msg")

logging.error("error msg")

logging.critical("critical msg")

o/p: Python logging demo

But it creates a file “log.txt”, where warning and higher level msgs are written as:

WARNING:root:warning msg

ERROR:root:error msg

CRITICAL:root:critical msg

Note: “root” means from the main part of the program, this info is printed.

Ex: If we don’t mention the level also, by default “warning” is considered.

import logging

logging.basicConfig(filename="log.txt")

print("Python logging demo")

logging.debug("debug msg")

logging.info("info msg")

logging.warning("warning msg by default")

logging.error("error msg")

logging.critical("critical msg")

o/p:

python logging demo

In the file log.txt,

WARNING:root:warning msg

ERROR:root:error msg

CRITICAL:root:critical msg

WARNING:root:warning msg

ERROR:root:error msg

CRITICAL:root:critical msg

Note:

We have to add these lines inside exception.

Ex: If level=DEBUG,

import logging

logging.basicConfig(filename="log.txt",level=logging.DEBUG)

print("Python logging demo")

logging.debug("debug msg")

logging.info("info msg")

logging.warning("warning msg")

logging.error("error msg")

logging.critical("critical msg")

o/p:

Python logging demo

Inside the file, debug and its higher level msgs are added as:

WARNING:root:warning msg

ERROR:root:error msg

CRITICAL:root:critical msg

WARNING:root:warning msg

ERROR:root:error msg

CRITICAL:root:critical msg

WARNING:root:warning msg

ERROR:root:error msg

CRITICAL:root:critical msg

Ex: To write IP address, time and date also to the log file,

**Writing python exceptions to the log file**

import logging

logging.basicConfig(filename="log.txt",level=logging.INFO)

print("Python exceptions demo")

try:

a=int(input("first number:"))

b=int(input("second number:"))

c=a/b

print("Result f division=",c)

except ZeroDivisionError as msg:

print("Can't divide a number with zero")

logging.exception(msg)

except ValueError as msg:

print("Enter only integers")

logging.exception(msg)

logging.info("Request processing completed")

O/p:

Complete info is available in the log file , whenever there are exceptions

o/p:

ERROR:root:division by zero

Traceback (most recent call last):

File "C:\Users\hai\.spyder-py3\temp.py", line 10, in <module>

c=a/b

ZeroDivisionError: division by zero

ERROR:root:invalid literal for int() with base 10: 'hi'

Traceback (most recent call last):

File "C:\Users\hai\.spyder-py3\temp.py", line 9, in <module>

b=int(input("second number:"))

ValueError: invalid literal for int() with base 10: 'hi'

Other options:

To add time,

logging.basicConfig(filename=”log.txt”,level=logging.WARNING,format=’%(asctime)s:%(message)s’)

-\*- coding: utf-8 -\*-

import logging

import os

import datetime

import time

class SingletonType(type):

\_instances = {}

def \_\_call\_\_(cls, \*args, \*\*kwargs):

if cls not in cls.\_instances:

cls.\_instances[cls] = super(SingletonType, cls).\_\_call\_\_(\*args, \*\*kwargs)

return cls.\_instances[cls]

# python 3 style

class MyLogger(object, metaclass=SingletonType):

# \_\_metaclass\_\_ = SingletonType # python 2 Style

\_logger = None

def \_\_init\_\_(self):

self.\_logger = logging.getLogger("crumbs")

self.\_logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s \t [%(levelname)s | %(filename)s:%(lineno)s] > %(message)s')

now = datetime.datetime.now()

dirname = "./log"

if not os.path.isdir(dirname):

os.mkdir(dirname)

fileHandler = logging.FileHandler(dirname + "/log\_" + now.strftime("%Y-%m-%d")+".log")

streamHandler = logging.StreamHandler()

fileHandler.setFormatter(formatter)

streamHandler.setFormatter(formatter)

self.\_logger.addHandler(fileHandler)

self.\_logger.addHandler(streamHandler)

print("Generate new instance")

def get\_logger(self):

return self.\_logger

# a simple usecase

if \_\_name\_\_ == "\_\_main\_\_":

logger = MyLogger.\_\_call\_\_().get\_logger()

logger.info("Hello, Logger")

logger.debug("bug occured")

Singleton class