Error Tolerant Environment

A guide to making an error-tolerant Environment.

- Logging is a technique for monitoring events that take place when some software is in use. The software's creator includes logging calls in their code to denote the occurrence of specific events. A descriptive message that may or may not involve variable data describes an event (i.e. data that is potentially different for each occurrence of the event).

When do we have to do logging?

- Display console output for ordinary usage of a command-line script or program.
- Report events that occur during normal operation of a program (e.g., for status monitoring or fault investigation)
- Issue a warning regarding a particular runtime event
- Report suppression of an error without raising an exception (e.g., error handler in a long-running server process)
- Report an error regarding a particular runtime event

Types of log functions

The logging functions are named on the basics of severity or degree of the events they record.

- **DEBUG:**

  Detailed information that is usually only relevant for diagnosing issues.

- **INFO:**

  Confirmation that everything is operating as anticipated.

- **WARNING:**

  An indicator that something unexpected occurred or that a problem is imminent (for example, 'disk space low'). The software continues to function normally.

- **ERROR:**

  The software has been unable to perform several functions due to a more serious issue.

- **CRITICAL:**

  A critical error indicating that the software may not be able to continue operating.

Logging messages

With the logger object configured, the following methods create log messages:

- Logger.debug(), Logger.info(), Logger.warning(), Logger.error(), and Logger.critical() all create log records with a message and a level that corresponds to their respective method names.
- The message is actually a format string, which may contain the standard string substitution syntax of %s, %d, %f, and so on. The rest of their arguments is a list of objects that correspond with the substitution fields in the message.
- • Logger.exception() generates a log message in the same way as Logger.error() does (). The difference is that Logger.exception() also dumps a stack trace. Only call this function from an exception handler.
- • Logger.log() accepts a specified log level as an input. This approach is slightly more verbose for logging messages than the log level convenience methods given above, but it is how you log at custom log levels.

What to log in the logging file?

Following things should be logged in the log file.

- All modules must log their initialization using the string "<Module Name>: Download module started at <Datetime>" with info log level.
- All modules should log download of firmware files using the string "<Module Name>: Downloading firmware <name> <version>" with debug log level.
- All modules should log when the downloaded firmware using string<”Metadata added to database”> with info log level.
- All modules should log downloading firmware URL using the string "<URL>: Downloading firmware <File Path to Save> " with debug log level.
- All modules should log download of firmware files using the string "<Module Name>: Downloading firmware from web page <URL> " with debug log level.
- All modules should log exception when the URL is invalid or not found using string “<URL is invalid>:” with error log level.

- All modules should tell the total number of firmware files found on the web page using the digit “<count>:” with debug log level.
- All modules should log String” < Firmware name> <Firmware manufacturer name> <Firmware model name> <Firmware version> <Firmware release date>:” with debug log level.

Effects on the web crawler if html code of the web page changes

There are 2 conditions.

- If the developer is using the API method for downloading the module from web page, then there will be no effects on the web crawler
- If the developer is using the html code of the web page for web crawling and it changes at some point the develop will use the try and catch function. If the web page is not giving the response, then it will throw the exception and the developer had to fix the issue manually.

Use of try catch function

The try catch statement can handle exceptions. When you run a program, exceptions may occur. Exceptions are errors that occur during software execution. Python will not notify you of mistakes such as syntax errors (grammar faults), instead stopping suddenly. An abrupt exit is detrimental to both the end user and the developer. To correctly deal with the problem, you can use a try catch statement instead of an emergency halt.

- Try catch function will be used in every function. It will be used in every function like in the function of downloading list of files, transforming metadata into DB format and in the writing metadata into the database.
- If you are downloading the file you have to use the try catch in every function so that all the files look same.

Real time implementation

- Created a “get_logger” function in the lgs.py file. The get logger function saves and shows all the logs in the log file. The log file will show all the metadata of the file that is downloaded. Few attributes are date, time, name of the module who file is going to be downloaded, and message whether the file is downloading or not.
- Import get_logger function from the utils.logs file. **“from utils.Logs import get_logger”**

```
from utils.Logs import get_logger
```

- Indicate the module name to the logger to generate logs of each module.
  name = "Module_name" like in my case it was
  (Schneider_electric) logger = get_logger("vendors.module_name").

```
name = "schneider_electric"
logger = get_logger("vendors.schneider_electric")
```

- Use logger function in each event like info, debugging, warnings, and errors.

```
def download_single_file(url, file_path_to_save):
    logger.info(f"Downloading {url} and saving as {file_path_to_save}")
```

```
logger.error(f"{general_exception}")
```