Creating a Simple Logger

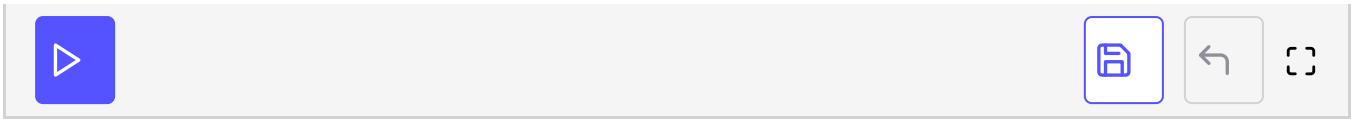
Creating a log with the logging module is easy and straight-forward. It's easiest to just look at a piece of code and then explain it, so here's some code for you to read:

```
import logging

# add filemode="w" to overwrite
logging.basicConfig(filename="sample.log", level=logging.INFO)

logging.debug("This is a debug message")
logging.info("Informational message")
logging.error("An error has happened!")

# Let's use our file reading knowledge to
# read the log file
with open("sample.log") as file_handler:
    for line in file_handler:
        print(line)
```



As you might expect, to access the logging module you have to first import it. The easiest way to create a log is to use the logging module's basicConfig function and pass it some keyword arguments. It accepts the following: filename, filemode, format, datefmt, level and stream. In our example, we pass it a file name and the logging level, which we set to INFO. There are five levels of logging (in ascending order): DEBUG, INFO, WARNING, ERROR and CRITICAL. By default, if you run this code multiple times, it will append to the log if it already exists. If you would rather have your logger overwrite the log, then pass in a **filemode="w"** as mentioned in the comment in the code. Speaking of running the code, this is what you should get if you ran it once:

```
INFO:root:Informational message
ERROR:root:An error has happened!
```

Note that the debugging message isn't in the output. That is because we set the level at INFO, so our logger will only log if it's a INFO, WARNING, ERROR or CRITICAL message. The root part just means that this logging message is coming from the root logger or the main logger. We'll look at how to change that so it's more descriptive in the next section. If you don't use **basicConfig**, then the logging module will output to the console / stdout.

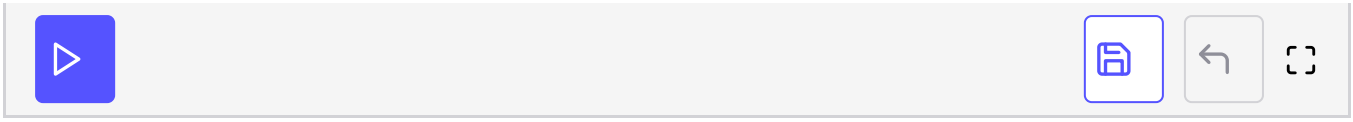
The logging module can also log some exceptions to file or wherever you have it configured to log to. Here's an example:

```
import logging

logging.basicConfig(filename="sample.log", level=logging.INFO)
log = logging.getLogger("ex")

try:
    raise RuntimeError
except RuntimeError:
    log.exception("Error!")

# Let's use our file reading knowledge to
# read the log file
with open("sample.log") as file_handler:
    for line in file_handler:
        print(line)
```



Let's break this down a bit. Here we use the **logging** module's **getLogger** method to return a logger object that has been named **ex**. This is handy when you have multiple loggers in one application as it allows you to identify which messages came from each logger. This example will force a **RuntimeError** to be raised, catch the error and log the entire traceback to file, which can be very handy when debugging.