How to log From Multiple Modules (and Formatting too!)

The more you code, the more often you end up creating a set of custom modules that work together. If you want them all to log to the same place, then you've come to the right place. We'll look at the simple way and then show a more complex method that's also more customizable. Here's one easy way to do it:

```
import logging
import otherMod

def main():
    """
    The main entry point of the application
    """
    logging.basicConfig(filename="mySnake.log", level=logging.INFO)
    logging.info("Program started")
    result = otherMod.add(7, 8)
    logging.info("Done!")

if __name__ == "__main__":
    main()
```

Here we import logging and a module of our own creation ("otherMod"). Then we create our log file as before. The other module looks like this:

```
# otherMod.py
import logging

def add(x, y):
    """"""
    logging.info("added %s and %s to get %s" % (x, y, x+y))
    return x+y
```

If you run the main code, you should end up with a log that has the following contents:

```
INFO:root:Program started
INFO:root:added 7 and 8 to get 15
INFO:root:Done!
```

Do you see the problem with doing it this way? You can't really tell very easily where the log messages are coming from. This will only get more confusing the more modules there are that write to this log. So we need to fix that. That brings us to the complex way of creating a logger. Let's take a look at a different implementation:

```
import logging
import otherMod2

def main():
    """
    The main entry point of the application
    """
    logger = logging.getLogger("exampleApp")
    logger.setLevel(logging.INFO)

    # create the logging file handler
    fh = logging.FileHandler("new_snake.log")

    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    fh.setFormatter(formatter)

    # add handler to logger object
    logger.addHandler(fh)

    logger.info("Program started")
    result = otherMod2.add(7, 8)
    logger.info("Done!")

if __name__ == "__main__":
    main()
```

Here we create a logger instance named "exampleApp". We set its logging level, create a logging file handler object and a logging Formatter object. The file handler has to set the formatter object as its formatter and then the file handler has to be added to the logger instance. The rest of the code in main is mostly the same. Just note that instead of "logging.info", it's "logger.info" or whatever you call your logger variable. Here's the updated **otherMod2** code:

```
# otherMod2.py
import logging

module_logger = logging.getLogger("exampleApp.otherMod2")

def add(x, y):
    """"""
    logger = logging.getLogger("exampleApp.otherMod2.add")
    logger.info("added %s and %s to get %s" % (x, y, x+y))
```

```
    return x+y
```

Note that here we have two loggers defined. We don't do anything with the module_logger in this case, but we do use the other one. If you run the main script, you should see the following output in your file:

```
2012-08-02 15:37:40,592 - exampleApp - INFO - Program started
2012-08-02 15:37:40,592 - exampleApp.otherMod2.add - INFO - added 7 and 8 to get 15
2012-08-02 15:37:40,592 - exampleApp - INFO - Done!
```

You will notice that all references to root have been removed. Instead it uses our **Formatter** object which says that we should get a human readable time, the logger name, the logging level and then the message. These are actually known as **LogRecord** attributes. For a full list of **LogRecord** attributes, see the documentation as there are too many to list here.