Logging with Python's Basic Interface

What's Logging For?

Logging provides useful data about what your application is doing.

It can be as detailed or coarse as you want, changeable at any time.

- Gives insight into **business logic** (unlike automated tracing tools)
- Valuable **telemetry** for monitoring
- Critical for **troubleshooting** and debugging

The larger your application, the more important logging becomes.

Two Ways

You can use Python's `logging` module in two broad ways.

The Basic Interface - Simpler. Useful for scripts and some mid-size applications.

Logger Objects - More complex to set up. But FAR more powerful, and invaluable with larger apps. Scales to any size.

The Basic Interface

The easiest way:

```
import logging
logging.warning('Look out!')
```

Save this in a script and run it, and you'll see this printed to standard error:

```
WARNING:root:Look out!
```

You call `logging.warning()`, and the output line starts with `WARNING`. There's also `error()`:

```
logging.error('Look out!')
```

```
ERROR:root:Look out!
```

Log Level Spectrum

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.

error: Due to a more serious problem, the software has not been able to perform some function.

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

Thresholds

Run this as a program:

```
import logging
logging.debug("Small detail. Useful for troubleshooting.")
logging.info("This is informative.")
logging.warning("This is a warning message.")
logging.error("Uh oh. Something went wrong.")
logging.critical("We have a big problem!")
```

And here's the output:

```
WARNING:root:This is a warning message.
ERROR:root:Uh oh. Something went wrong.
CRITICAL:root:We have a big problem!
```

What's missing? Why?

Log Levels

Python loggers have a *logging threshold*. And the default threshold is `logging.WARNING`.

You can change it with `logging.basicConfig()`.

```
import logging
logging.basicConfig(level=logging.INFO)
logging.info("This is informative.")
logging.error("Uh oh. Something went wrong.")
```

Run this new program, and the `INFO` message gets printed:

```
INFO:root:This is informative.
ERROR:root:Uh oh. Something went wrong.
```

You can also pass an uppercase string:
`logging.basicConfig(level="INFO")`

Two Meanings

The phrase "log level" means two different things:

1) The **urgency** of a message.

The order is `debug()`, `info()`, `warning()`, `error()` and `critical()`, from lowest to highest urgency.

2) It can mean the **threshold** for ignoring messages.

Ignores everything less urgent than `logging.INFO`:

```
logging.basicConfig(level=logging.INFO)
```

And this means "show me everything":

```
logging.basicConfig(level=logging.DEBUG)
```

When do you call basicConfig()?

1) Call it exactly once in your program. No more than that.

AND

2) Call it BEFORE any logging statements are called... by your code, or code you reuse.

Otherwise, you'll get unpredictable results.

Safest bet: call `basicConfig()` right after you import logging, at the top of your main executable file.

```
# start file...
import logging
logging.basicConfig(...)
# Then other imports, followed by other Python code.
```

Log Destination

By default, log messages are written to stderr.

Use `filename` to write them to a file instead:

```
# Appends messages to the file, one at a time.
logging.basicConfig(filename="log.txt")
logging.error("oops")
```

You can make your program *clobber* the log file each time by setting `filemode` to "w":

```
# Wipes out previous log entries when program restarts
logging.basicConfig(filename="log.txt", filemode="w")
logging.error("oops")
```

Or set it to "a" for "append". That's the default.

Practice: `basiclog.py`

```
# Create a new file name `basiclog.py`. Type in this program:
import logging
mode = "development"
log_file = "mylog.txt"
if mode == "development":
    log_level = logging.DEBUG
    log_mode = "w"
else:
    log_level = logging.WARNING
    log_mode = "a"
logging.basicConfig(level=log_level, filename=log_file, filemode=log_mode)
logging.debug("debug message")
logging.warning("look out!")
logging.critical("we have a problem here")
```

Run to verify `mylog.txt` contains `DEBUG`, `WARNING` and `CRITICAL`. Then change `mode` to "production", and re-run several times. Verify it appends only `WARNING` and `CRITICAL` to `mylog.txt` each run.

EXTRA: Make `mode` controlled by an env variable or command-line switch.