Part 3: The Print Statement You Can Add To

Bookmark this page

The Print Statement You Can Add To

Sometimes, it's not enough just to see the error, warning, or information message that you would put into a print statement to debug your code. Other information can be useful, such as:

- when the log message was generated;
- what Python file the log message was generated in;
- what line number the log message was generated on; or
- the name of the function that the log message was generated in.

It's easy to see how knowing the file name, line number, and function name that the log message was generated on can be useful: you might create a lot of messages and it can be easy to lose track of where all of your log statements are.

Why would you possibly want to know *when* a log message was generated? One reason is that you might want to time how long it takes for your code to get to a particular log message. But the real usefulness of knowing *when* a log message was generated will come in the next session: we'll be saving log messages to files instead of printing them at the console. When you open up a saved log file, you might not even know *what day* the message was generated on unless you include a timestamp!

Let's try it out! Make the following changes to your code:

```
import logging

log_format = "%(asctime)s %(filename)s:%(lineno)-4d %(levelname)s %(message)s"  # Add/modify these
logging.basicConfig(level=logging.WARNING, format=log_format)                   # two lines

def my_fun(n):
    for i in range(0, n):
        logging.debug(i)
        if i == 50:
            logging.warning("The value of i is 50.")
        try:
            100 / (50 - i)
        except ZeroDivisionError:
            logging.error("Tried to divide by zero. Var i was {}. Recovered
gracefully.".format(i))

if __name__ == "__main__":
    my_fun(100)
```

Let's look at these two lines:

```
log_format = "%(asctime)s %(filename)s:%(lineno)-4d %(levelname)s %(message)s"
logging.basicConfig(level=logging.WARNING, format=log_format)
```

We begin by defining a *log_format* for our log messages. All of the characters inside of the parentheses specify a different piece of information that we want to include inside of our messages. Please see the full list of these LogRecord attributes, and look for each of the attributes we included above, to get a guess for what information this formatter will include. For example, *asctime* produces a human-readable time string.

The formatting characters to the left and right of the parentheses are borrowed from *printf* formatting. For example, *%(asctime)s* means to include the time string in the log message as a string. The *-4d* in *%(lineno)-4d* means to include the line number of the log statement as a 4 character integer, padding the output on the right with spaces.

Now, what do you imagine running simple.py will produce? Here is the output:

```
$ python simple.py
2018-03-12 17:39:17,567 simple.py:10   WARNING The value of i is 50.
2018-03-12 17:39:17,567 simple.py:14   ERROR Tried to divide by zero. Var i was 50. Recovered
gracefully.
```

As expected, we see the time that the log message was produced, the file name and line number that the message was