Part 4: The Print Statement You Can Send Elsewhere

Bookmark this page

The Print Statement You Can Send Elsewhere

Every print statement you include in your code writes its message to the console, but what if it could be sent somewhere else?

The simplest place that you can send log messages to is a file. Edit the *logging.basicConfig* statement in your *simple.py*.

```
logging.basicConfig(level=logging.WARNING, format=log_format, filename='mylog.log')
```

Now run simple.py:

```
$ python simple.py

$
```

There should now be no output sent to the console. Instead, the logging messages have been sent to a new file: mylog.log. Open this newly created file to take a look at the contents.

What happens when you run the script again? Will the contents of mylog.log be appended to, or will they be overwritten? Try it out and find the answer. What's in the log file after running simply.py two or three times?

We're really starting to show off the power of logging. Now you no longer have to wait patiently at the console for your print statements to be displayed: you can just send them to a file and read them later.

Logging is even more powerful than that. We're about to learn how to send our logging messages to multiple places. In preparation for that, I want you to make the following changes to your code:

```
import logging

log_format = "%(asctime)s %(filename)s:%(lineno)-3d %(levelname)s %(message)s"

# BEGIN NEW STUFF
formatter = logging.Formatter(log_format)

file_handler = logging.FileHandler('mylog.log')
file_handler.setFormatter(formatter)

logger = logging.getLogger()
logger.addHandler(file_handler)
# END NEW STUFF

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:
            i / (50 - i)
        except ZeroDivisionError:
            logging.error("Tried to divide by zero. Var i was {}. Recovered
gracefully.".format(i))

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

Python, and the logging library, are so easy to read that you can probably guess at the meaning of all of these new lines. The first thing to notice is that we've eliminated that *logging.basicConfig* line! We're manually building a logging configuration, consisting of a *formatter* and a *handler*.

Let me add a bit of explanation to each new line in the following comments:

```
# Create a "formatter" using our format string
formatter = logging.Formatter(log_format)

# Create a log message handler that sends output to the file 'mylog.log'
file_handler = logging.FileHandler('mylog.log')
# Set the formatter for this log message handler to the formatter we created above.
file_handler.setFormatter(formatter)

# Get the "root" logger. More on that below.
logger = logging.getLogger()
# Add our file_handler to the "root" logger's handlers.
logger.addHandler(file_handler)
```

What does this new configuration do? Well, it does exactly what our code did before: it sends warning messages and above to a file named ‘mylog.log’.

Log message handlers answer the question, “What should the system do with log messages?” Here are a few possible things that we can do with them:

- We could print them to the console.
- We could send them to a file.
- We could send them to a remote server.
- We could send them in an email.
- We could just ignore them.

Take a brief look at each of the handler classes available in the logging library. Each of the above ways to handle log messages, and more, is represented by a handler class in the logging library.

In the newest iteration of our code, we create a logging.FileHandler log message handler to send our log messages to a file. Unlike the *logging.basicConfig* command, we can’t provide the log message format to our file handler as a string. We have to create an instance of the logging.Formatter class and use *file_handler.setFormatter* to instruct our handler to use this formatter.

Next, we have to tell the logger to use this handler that we’ve created. We first get a reference to the “root” or global logger using *logging.getLogger()*. It turns out that you can have multiple loggers running in a system, although we’re not going to explore that in this lesson. Instead, we’re going to use a single logger and add multiple log message handlers to that logger. But if you’re curious, you can look at the documentation for logging.getLogger()

Now that we have a reference to the “root” or global logger, we can add our message handler to it using *logger.addHandler*. Now, our root logger will send all of its messages to the file_handler log message handler, and these messages get written to the file ‘mylog.log’.

Run the script and confirm!

Now, let’s add another handler! Imagine that you wanted to see ALL logging messages at the console while you were running your program, but only log the most important messages (WARNING and above) to your log file. You could accomplish that with this code:

```
import logging

log_format = "%(asctime)s %(filename)s:%(lineno)-3d %(levelname)s %(message)s"

formatter = logging.Formatter(log_format)

file_handler = logging.FileHandler('mylog.log')
file_handler.setLevel(logging.WARNING)           # Add this line
file_handler.setFormatter(formatter)

console_handler = logging.StreamHandler()        # Add this line
console_handler.setLevel(logging.DEBUG)          # Add this line
console_handler.setFormatter(formatter)          # Add this line

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)                   # Add this line
logger.addHandler(file_handler)
logger.addHandler(console_handler)               # Add this line

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:
            i / (50 - i)
        except ZeroDivisionError:
            logging.error("Tried to divide by zero. Var i was {}. Recovered
gracefully.".format(i))

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

You might have a few questions about this code:

- What is a StreamHandler?
- Why do we set the log level on both of the log message handlers **and also** set the log level on the root logger?

A rigorous definition of a *stream* is outside the scope of this assignment; but in rough terms, a stream is a very general concept in computer science of a store or source of information. The StreamHandler constructor will accept a stream as its first argument; but if we don’t provide an argument, then it will use its default: the sys.stderr stream. That’s one of two system streams that get printed directly to the console. So by default, the StreamHandler will send log messages to the console.

As for the second question, loggers and handlers maintain separate settings for their minimum log level. By default, a