Logging in Python

So far we've use `print` statements that go to STDOUT and the `warn` function that makes is slightly more convenient to `print` to STDERR. The trouble with this approach to writing and debugging code is that you need to remove all the `print` statements prior to releasing your code or running your tests. With the `logging` module (https://docs.python.org/3/library/logging.html), you can sprinkle messages to yourself liberally throughout your code and chose *at run time* which ones to see.

Like with `random.seed`, calls to the `logging` module affect the **global state** of how logging happens. First you need to set up how the logging will happen using the `basicConfig` (https://docs.python.org/3/library/logging.html#logging.basicConfig). Typically you will set log message to go to a `filename` with the `filemode` of "w" (write, which will overwrite existing files) at some `level` like "debug". Here is a script that does that:

```
$ cat -n basic.py
     1	#!/usr/bin/env python3
     2	
     3	import logging
     4	import os
     5	import sys
     6	
     7	prg = sys.argv[0]
     8	prg_name, _ = os.path.splitext(os.path.basename(prg))
     9	logging.basicConfig(
    10	    filename=prg_name + '.log',
    11	    filemode='w',
    12	    level=logging.DEBUG
    13	)
    14	
    15	logging.debug('DEBUG!')
    16	logging.critical('CRITICAL!')
```

Before running the program, see that there is no log file:

```
$ ls
basic.py*
```

Run it, and see that `basic.log` has been created:

```
$ ./basic.py
$ ls
basic.log  basic.py*
$ cat basic.log
DEBUG:root:DEBUG!
CRITICAL:root:CRITICAL!
```

The key is to understand the hierarchy of the levels:

1. CRITICAL
2. ERROR
3. WARNING
4. INFO
5. DEBUG
6. NOTSET

The log level includes everything above the level you set. As in the above program, we set it to `logging.DEBUG` and so a call to `critical` was included. If you change the program to `logging.CRITICAL`, then `error` through `debug` calls are not emitted:

```
$ cat -n basic.py
     1	#!/usr/bin/env python3
     2	
     3	import logging
     4	import os
     5	import sys
     6	
     7	prg = sys.argv[0]
     8	prg_name, _ = os.path.splitext(os.path.basename(prg))
     9	logging.basicConfig(
    10	    filename=prg_name + '.log',
    11	    filemode='w',
    12	    level=logging.CRITICAL
    13	)
    14	
    15	logging.debug('DEBUG!')
    16	logging.critical('CRITICAL!')
$ ./basic.py
$ cat basic.log
CRITICAL:root:CRITICAL!
```

If you find yourself repeatedly debugging some program or just need to know information about how it is proceeding. For instance, you have some functions or system calls that take a long time, and you sometimes want to monitor how they are going and other times don't (e.g., running unattended on the HPC). Here is a program that logs random levels and then sleeps for one second. To see how this could be useful, open two terminals and navigate to the `examples/long_running` directory.

Here is the program:

```
$ cat -n long.py
     1	#!/usr/bin/env python3
     2	
     3	import os
```

```python
import sys
import time
import random
import logging

prg = sys.argv[0]
prg_name, _ = os.path.splitext(os.path.basename(prg))
logging.basicConfig(
    filename=prg_name + '.log', filemode='a', level=logging.DEBUG)

logging.debug('Starting')
for i in range(1, 11):
    method = random.choice([
        logging.info, logging.warning, logging.error, logging.critical,
        logging.debug
    ])
    method('{}: Hey!'.format(i))
    time.sleep(1)

logging.debug('Done')

print('Done.')
```

Start running `long.py` in one terminal, then execute `tail -f long.log` in the other where `tail` is the program to show you the end of a file and `-f` tells `tail` to stay running and "follow" the file as it grows. (Use CTRL-C to stop following.) Here's what I see on one run:

```
$ tail -f long.log
DEBUG:root:Starting
CRITICAL:root:1: Hey!
WARNING:root:2: Hey!
DEBUG:root:3: Hey!
DEBUG:root:4: Hey!
WARNING:root:5: Hey!
ERROR:root:6: Hey!
DEBUG:root:7: Hey!
ERROR:root:8: Hey!
INFO:root:9: Hey!
DEBUG:root:10: Hey!
DEBUG:root:Done
```