Logging

Matt Waismann

Sep 20th, 2021

Without logging, most people check the program behaved correctly by embedding print statements at various points in the script. The better practice is to use *logging*.

There are five standard logging levels. The levels are ordered by severity:

- # 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.

Logging functionality can be accessed through the standard library 'logging'

```
import logging
```

The default level for logging is set to 'WARNING'. This means it will capture everything that is everything at level 'WARNING' and above. Therefore, it includes 'WARNING', 'ERROR', and 'CRITICAL'. The default logging behaviour is to just log results to the console. If we want to log levels below WARNING we need to modify the default behavior. Here's a logging example:

```
add_result = add(num_1, num_2)
logging.warning('Add: {} + {} = {}'.format(num_1, num_2, add_result))
```

Output in the console will look like:

```
WARNING:root:Add: 10 + 5 = 15
```

Of course, this isn't a warning so lets instead change the default logging level

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

Let's now change the logging output level to something more appropriate like 'DEBUG'

```
add_result = add(num_1, num_2)
logging.warning('Add: {} + {} = {}'.format(num_1, num_2, add_result))
```

As said earlier, default logging only outputs to the console. If you prefer to have a log file, include a file name in the logging.basicConfig mutator

```
logging.basicConfig(filename = 'test.log', level = logging.DEBUG)
```

The default log format is to include logging level is

```
LOGLEVEL:logger:message (e.g. DEBUG:root:Add: 10 + 5 = 15 )
```

To modify the log format to include things like date and time, include a 'format' argument in the logging.basicConfig mutator.

```
logging.basicConfig(filename = 'test.log', level = logging.DEBUG,
    format = '%(asctime)s:%(levelname)s:%(message)s')
```

Now the output in the log file will look like

```
2021-09-20 12:53:08,118: DEBUG: Add: 10 + 5 = 20
```

A sample program might look like this

```
import logging
logging.basicConfig(filename='employee.log', level = logging.INFO,
                    format= '%(levelname)s:%(message)s')
class Employee:
  def __init__(self, first, last):
      self.first = first
      self.last = last

      logging.info('Created Employee: {} - {}'. format(self.fullname,
          self.email))
  @property
  def email(self):
      return '{}.{}@email.com'.format(self.first, self.last)
  @property
  def fullname(self):
      return '{} {}'.format(self.first, self.last)

emp_1 = Employee('John', 'Smith')
```

```
emp_2 = Employee('Jane', 'Doe')
```

The 'employee.log' file should look like

```
INFO:Created Employee: John Smith - John.Smith@gmail.com
INFO:Created Employee: Jane Doe - Jane.Doe@gmail.com
```