3 Logging

Introduction

Leaving miscellaneous debugging related `print` statements in code is typically discouraged. Partially because printed output gets mixed in with logs written to standard output. Logs are a more structured means of producing details related to an application's events. For example: web requests, unhandled exceptions, low disk space warnings, etc.

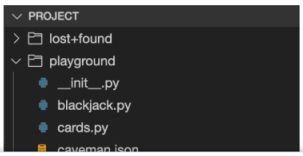
Python's standard library includes a `logging` module. The `logging` module provides a default structure for log events, allowing for customization. Log events include a **log level** which indicates the severity of the message (DEBUG, INFO, WARNING, ERROR, CRITICAL). These levels are ordered by severity (least to greatest).

Instructions

The **playground/web.py** file uses the `logging` module to log events at different log levels. Events logged at the info level include messages which demonstrate that the application is behaving as expected. These are a useful initial level to review when debugging. Unexpected patterns in info logs can be an indicator of a problem.

Events logged at the debug level may include metadata used to help debug the application. Displaying a callable's arguments or objects before and after a change can help find the exact location of a bug.

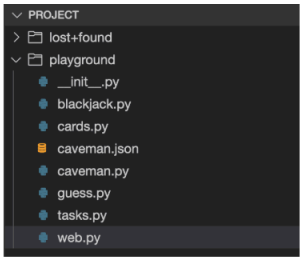
1. Locate and open the **playground/web.py** file.



Back

Start check

1. Locate and open the **playground/web.py** file.



2. Start the web application.

```
python3 playground/web.py
```

```
2022-07-18 21:25:53,289 root    INFO    44      wrap    registering route for path: /
2022-07-18 21:25:53,290 root    INFO    44      wrap    registering route for path: /reverse-it
```

The initial info log events indicate that two URL paths were registered. Each log entry is like a checkpoint, because the interpreter made it at least to the line of code that logged the event.

Once started, the WSGI application running in web.py will block the console unil the process is terminated. A new terminal is required to send HTTP requests to the web server.

3. Split the terminal pane by clicking the split button on the right hand side of the terminal.

Back

Start check

3. Split the terminal pane by clicking the split button on the right hand side of the terminal.

The newly opened (**curl**) terminal will use the `curl` application to send HTTP requests to the web server.

4. Request the index page in the **curl terminal**.

```
curl "localhost:5000/"
```

The HTTP response body displays a welcome message in the **curl terminal**.

```
Welcome! :)
```

Two info level log events are displayed in the web application's standard output. They demonstrate a successful web request. These two events are generated by the web.py application.

```
2022-07-18 21:37:42,024 root    INFO    44      wrap    registering route for path: /
2022-07-18 21:37:42,025 root    INFO    44      wrap    registering route for path: /reverse-it
2022-07-18 21:37:44,136 root    INFO    53      __call__        request made for path: /
2022-07-18 21:37:44,136 root    INFO    73      __call__        (200) response header sent
127.0.0.1 - - [18/Jul/2022 21:37:44] "GET / HTTP/1.1" 200 12
[]
```

The last event in the list follows a different event structure. The `logging` module can create different loggers for different use cases. The different types of events are the result of the web.py application using the same root logger as the `wsgiref.simple_server` module. Creating per module loggers allows for log isolation.

Logging at the error level is useful for identifying application errors.

Logging at the error level is useful for identifying application errors.

5. Observe a logged exception in the **curl terminal**.

```
curl "localhost:5000/reverse-it?text=welcome"
```

The HTTP response body displays an error summary in the **curl terminal**.

```
'list' object has no attribute 'encode'
```

The new info event shows a request was made and then an error occurred. The error event includes a traceback and a summary message.

```
2022-07-18 21:48:24,796 root    INFO    53      __call__        request made for path: /reverse-it
2022-07-18 21:48:24,797 root    ERROR   79      __call__        (500) failed running handler for path: /reverse-it
Traceback (most recent call last):
  File "/home/project/playground/web.py", line 69, in __call__
    response_body = handler(*args, **kwargs)
  File "/home/project/playground/web.py", line 115, in reverse_it
    return request['QUERY_STRING_PARSED'].get('text', '')[::-1].encode()
AttributeError: 'list' object has no attribute 'encode'
127.0.0.1 - - [18/Jul/2022 21:48:24] "GET /reverse-it?text=welcome HTTP/1.1" 500 40
```

The traceback suggests that the bug is located in the `reverse_it` function. The `reverse_it` function extracts a query string called *text* from the parsed query string. Suggesting that the bug involves the query string.

The debug level events in this application include the query string parameter before and after it's parsed. To display these events a configuration change is required.

The logging module can be configured to filter out log levels. Currently the logging level is set to the info level. The logging module will only display logs at or above the specified level.

Setting the log level to debug will display debug level events when the application is next restarted.

6. Change the log levels from info to debug.

```
1 logger.setLevel(logging.DEBUG)
```

Back

Start check

1h 50m

6. Change the log levels from info to debug.

```
logger.setLevel(logging.DEBUG)

handler = logging.StreamHandler(sys.stdout)
handler.setLevel(logging.DEBUG)
```

```
##########################################################################
# Configure the logger to use a more informative and structured format.
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

handler = logging.StreamHandler(sys.stdout)
handler.setLevel(logging.DEBUG)
```

7. Restart the web application.

   1. Click anywhere inside the web application terminal pane to set focus.

   2. Press `CTRL+C` to interrupt the process.

```
^CTraceback (most recent call last):
  File "/home/project/playground/web.py", line 120, in <module>
    server.serve_forever()
  File "/usr/local/lib/python3.9/socketserver.py", line 232, in serve_forever
    ready = selector.select(poll_interval)
  File "/usr/local/lib/python3.9/selectors.py", line 416, in select
    fd_event_list = self._selector.poll(timeout)
KeyboardInterrupt
```

   3. Start the web application.

```
python3 playground/web.py
```

```
2022-07-18 21:25:53,289 root    INFO    44      wrap    registering route for path: /
2022-07-18 21:25:53,290 root    INFO    44      wrap    registering route for path: /reverse-it
```

8. Retrigger the exception in the **curl terminal**.

```
curl "localhost:5000/reverse_it?text=welcome"
```

Back

Start check

1h 50m

8. Retrigger the exception in the **curl terminal**.

```
curl "localhost:5000/reverse-it?text=welcome"
```

```
2022-07-18 21:57:38,697 root    DEBUG   92      __call__        query string (before) : text=welcome
2022-07-18 21:57:38,698 root    DEBUG   98      __call__        query string (after) : {'text': ['welcome']}
```

The debug events include the query string before and after it's parsed. The parsed query string (`{'text': ['welcome']}`) hints at the problem.

The code inside the `reverse_it` function assumes that the value associated with the key *text* is a `str` type. The value is actually a `list` of `str` objects, because query strings support key reuse.

9. Replace the body of the `reverse_it` function to fix the bug.

```
1  return request['QUERY_STRING_PARSED'].get('text', [])[0][::-
```

```
@app.route('/reverse-it')
def reverse_it():
    return request['QUERY_STRING_PARSED'].get('text', [])[0][::-1].encode()
```

10. Restart the web application.

    1. Click anywhere inside the web application terminal pane to set focus.
    2. Press `CTRL+C` to interrupt the process.
    3. Start the web application.

    ```
    python3 playground/web.py
    ```

11. Test the bug fix in the **curl terminal**.

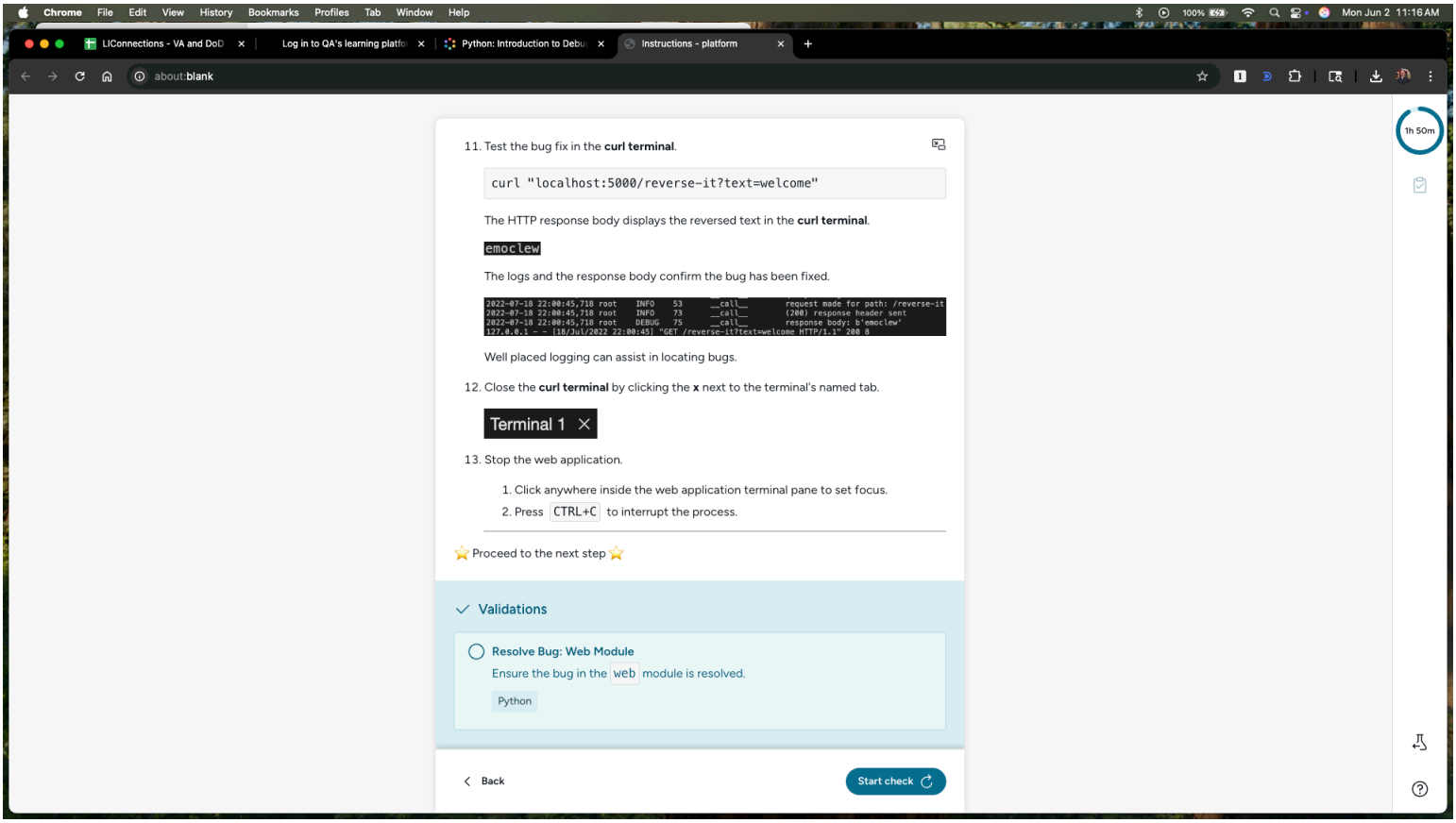
```
curl "localhost:5000/reverse-it?text=welcome"
```

The HTTP response body displays the reversed text in the **curl terminal**.

Back

Start check

1h 50m


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 11:16 AM
LIConnections - VA and DoD
Log in to QA's learning platfo
Python: Introduction to Debu
Instructions - platform
about:blank
11. Test the bug fix in the curl terminal.
curl "localhost:5000/reverse-it?text=welcome"
The HTTP response body displays the reversed text in the curl terminal.
emoclew
The logs and the response body confirm the bug has been fixed.
2022-07-18 22:00:45,710 root INFO 53 __call__ request made for path: /reverse-it
2022-07-18 22:00:45,710 root INFO 73 __call__ (200) response header sent
2022-07-18 22:00:45,710 root DEBUG 75 __call__ response body: b'emoclew'
127.0.0.1 - - [18/Jul/2022 22:00:45] "GET /reverse-it?text=welcome HTTP/1.1" 200 8
Well placed logging can assist in locating bugs.
12. Close the curl terminal by clicking the x next to the terminal's named tab.
Terminal 1
13. Stop the web application.
1. Click anywhere inside the web application terminal pane to set focus.
2. Press CTRL+C to interrupt the process.
Proceed to the next step
Validations
Resolve Bug: Web Module
Ensure the bug in the web module is resolved.
Python
Back
Start check
1h 50m