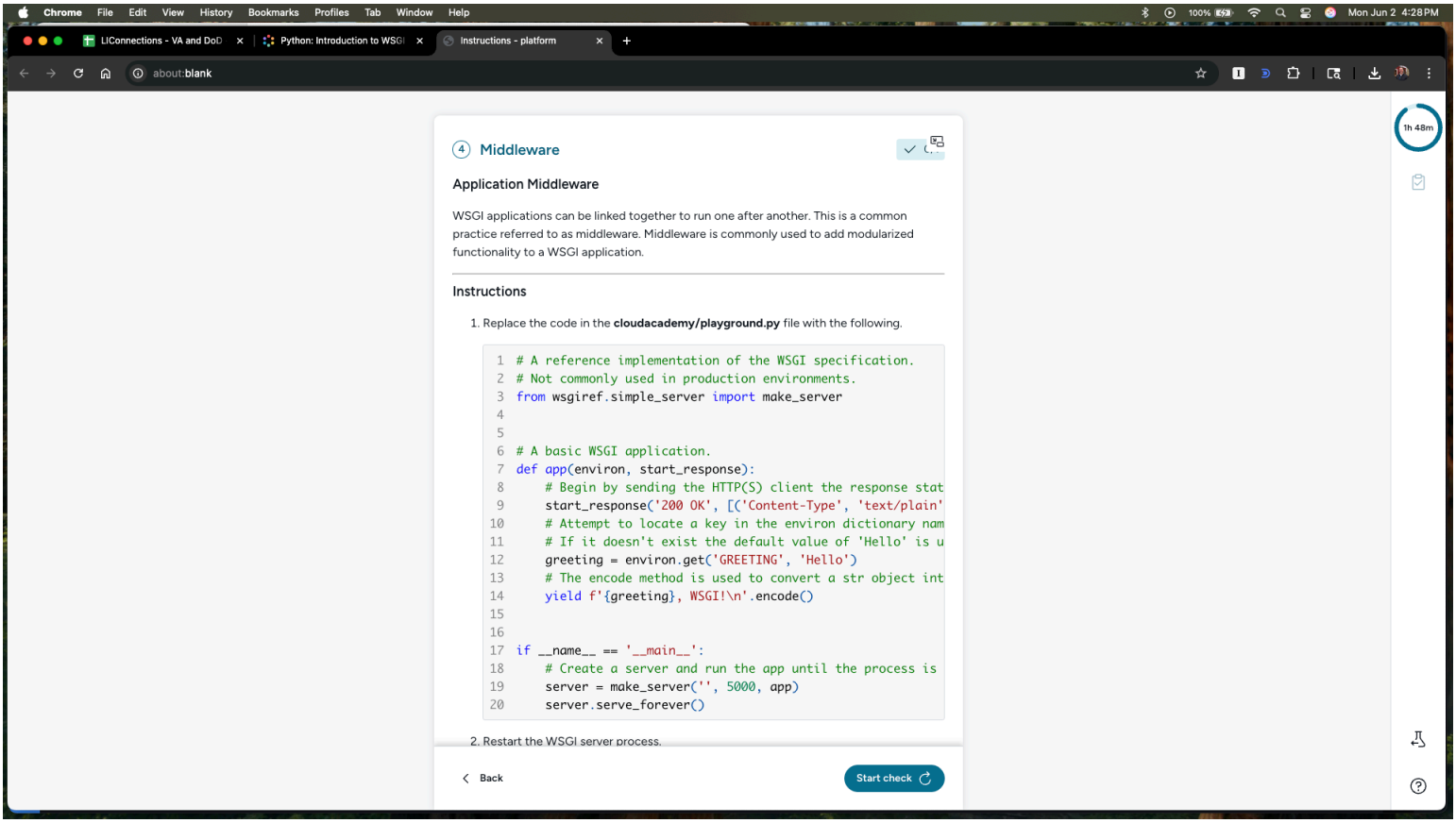
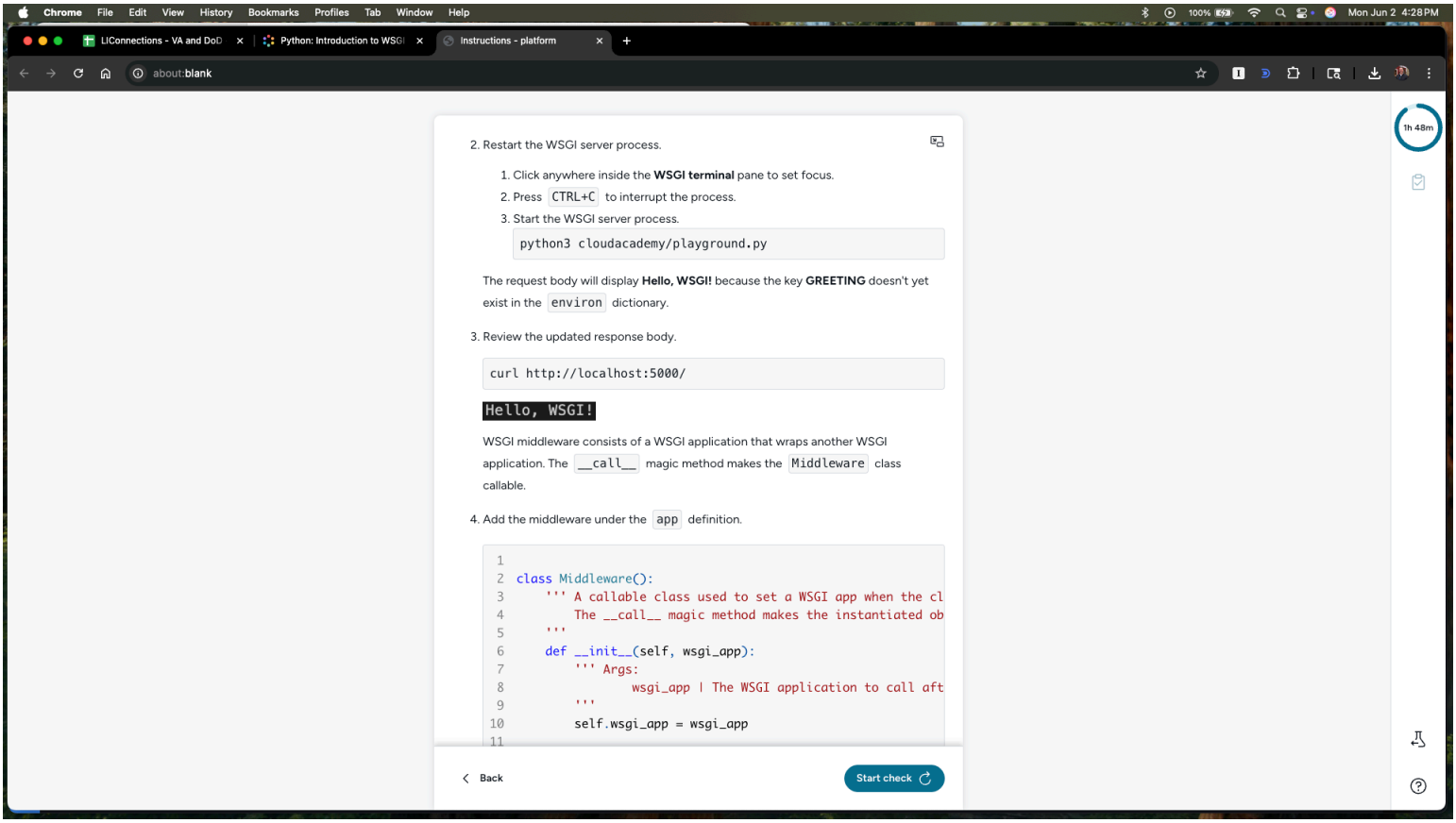
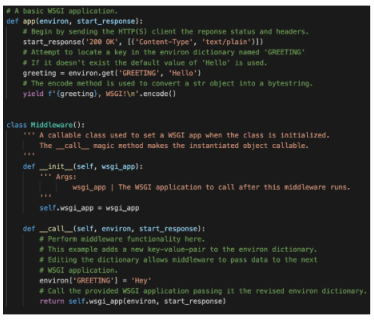

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 4:28 PM
LIConnections - VA and DoD
Python: Introduction to WSGI
Instructions - platform
about:blank
1h 48m
4 Middleware
Application Middleware
WSGI applications can be linked together to run one after another. This is a common practice referred to as middleware. Middleware is commonly used to add modularized functionality to a WSGI application.
Instructions
1. Replace the code in the cloudacademy/playground.py file with the following.
# A reference implementation of the WSGI specification.
# Not commonly used in production environments.
from wsgiref.simple_server import make_server
# A basic WSGI application.
def app(environ, start_response):
# Begin by sending the HTTP(S) client the response stat
start_response('200 OK', [('Content-Type', 'text/plain'
# Attempt to locate a key in the environ dictionary nam
# If it doesn't exist the default value of 'Hello' is u
greeting = environ.get('GREETING', 'Hello')
# The encode method is used to convert a str object int
yield f'{greeting}, WSGI!\n'.encode()
if __name__ == '__main__':
# Create a server and run the app until the process is
server = make_server('', 5000, app)
server.serve_forever()
2. Restart the WSGI server process.
Back
Start check


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 4:28 PM
LIConnections - VA and DoD
Python: Introduction to WSGI
Instructions - platform
about:blank
1h 48m
2. Restart the WSGI server process.
1. Click anywhere inside the WSGI terminal pane to set focus.
2. Press CTRL+C to interrupt the process.
3. Start the WSGI server process.
python3 cloudacademy/playground.py
The request body will display Hello, WSGI! because the key GREETING doesn't yet exist in the environ dictionary.
3. Review the updated response body.
curl http://localhost:5000/
Hello, WSGI!
WSGI middleware consists of a WSGI application that wraps another WSGI application. The __call__ magic method makes the Middleware class callable.
4. Add the middleware under the app definition.
class Middleware():
''' A callable class used to set a WSGI app when the cl
The __call__ magic method makes the instantiated ob
'''
def __init__(self, wsgi_app):
''' Args:
wsgi_app | The WSGI application to call aft
'''
self.wsgi_app = wsgi_app
Back
Start check

```
    def __call__(self, environ, start_response):
        # Perform middleware functionality here.
        # This example adds a new key-value-pair to the env
        # Editing the dictionary allows middleware to pass
        # WSGI application.
        environ['GREETING'] = 'Hey'
        # Call the provided WSGI application passing it the
        return self.wsgi_app(environ, start_response)

```



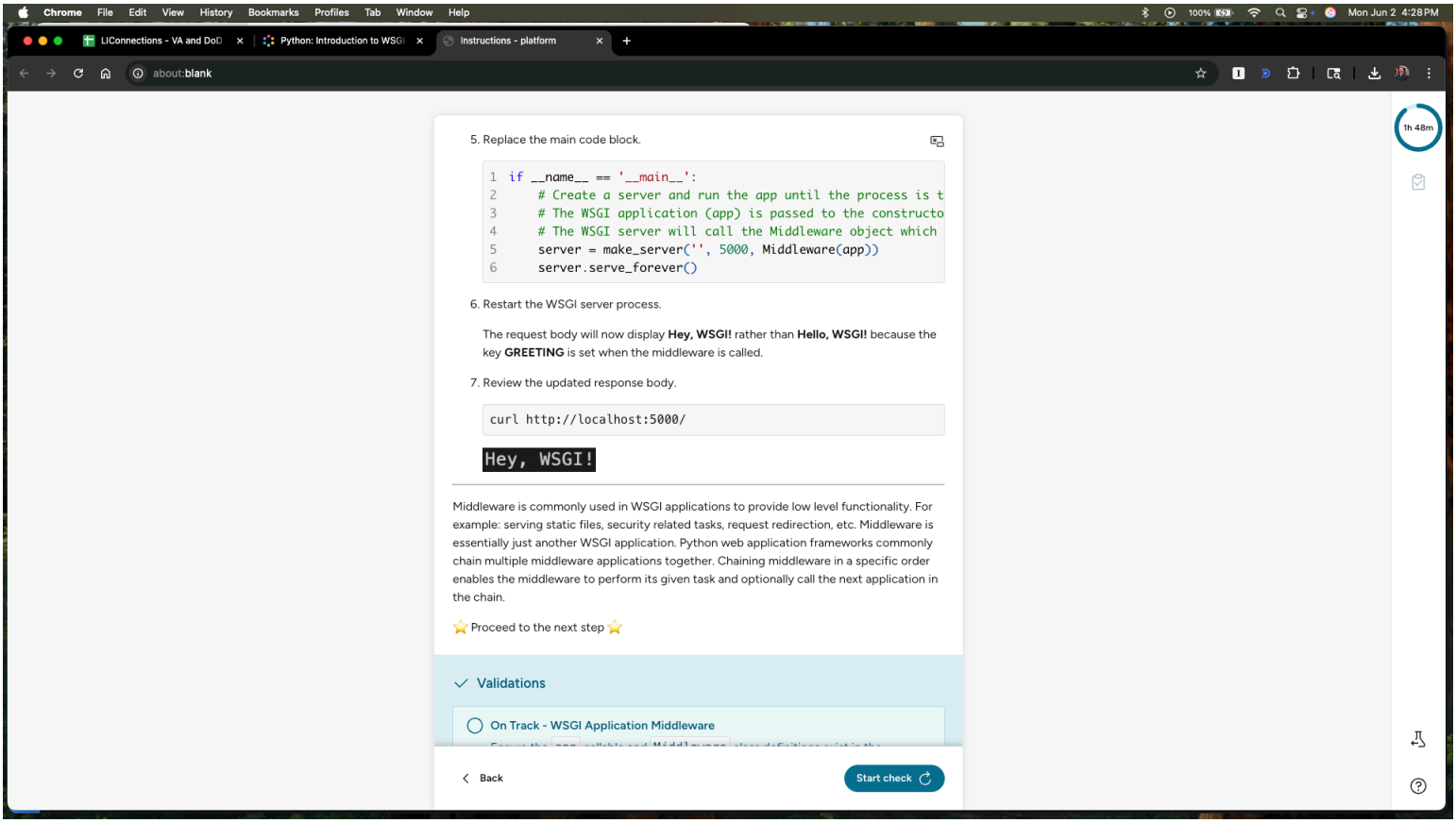
```
# A basic WSGI application.
def app(environ, start_response):
    # Begin by sending the HTTP(S) client the reponse status and headers.
    start_response('200 OK', [('Content-Type', 'text/plain')])
    # Attempt to locate a key in the environ dictionary named 'GREETING'
    # If it doesn't exist the default value of 'Hello' is used.
    greeting = environ.get('GREETING', 'Hello')
    # The encode method is used to convert a str object into a bytestring.
    yield f'{greeting}, WSGI!\n'.encode()


class Middleware():
    ''' A callable class used to set a WSGI app when the class is initialized.
        The __call__ magic method makes the instantiated object callable.
    '''
    def __init__(self, wsgi_app):
        ''' Args:
                wsgi_app | The WSGI application to call after this middleware runs.
        '''
        self.wsgi_app = wsgi_app

    def __call__(self, environ, start_response):
        # Perform middleware functionality here.
        # This example adds a new key-value-pair to the environ dictionary.
        # Editing the dictionary allows middleware to pass data to the next
        # WSGI application.
        environ['GREETING'] = 'Hey'
        # Call the provided WSGI application passing it the revised environ dictionary.
        return self.wsgi_app(environ, start_response)
```

When an instance of `Middleware` is called the interpreter calls the `__call__` magic method which matches the required signature of a WSGI application. Wrapping the `Middleware` class around the original WSGI application (`app`) results in the middleware being run first, followed by the original application.

5. Replace the main code block.

Back

Start check


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 4:28 PM
LIConnections - VA and DoD
Python: Introduction to WSGI
Instructions - platform
about:blank
1h 48m
5. Replace the main code block.
if __name__ == '__main__':
# Create a server and run the app until the process is t
# The WSGI application (app) is passed to the constructo
# The WSGI server will call the Middleware object which
server = make_server('', 5000, Middleware(app))
server.serve_forever()
6. Restart the WSGI server process.
The request body will now display Hey, WSGI! rather than Hello, WSGI! because the key GREETING is set when the middleware is called.
7. Review the updated response body.
curl http://localhost:5000/
Hey, WSGI!
Middleware is commonly used in WSGI applications to provide low level functionality. For example: serving static files, security related tasks, request redirection, etc. Middleware is essentially just another WSGI application. Python web application frameworks commonly chain multiple middleware applications together. Chaining middleware in a specific order enables the middleware to perform its given task and optionally call the next application in the chain.
Proceed to the next step
Validations
On Track - WSGI Application Middleware
Back
Start check