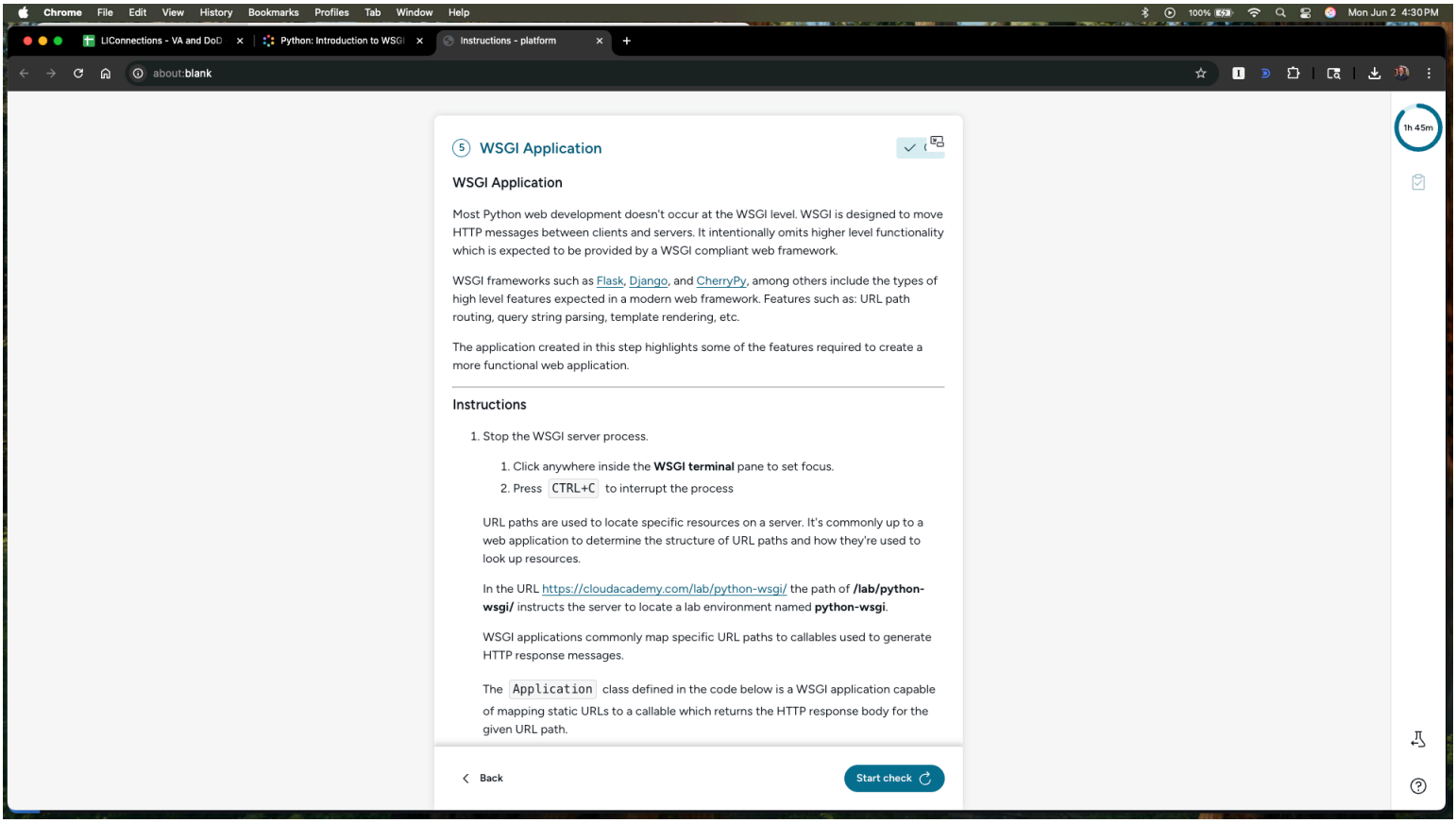
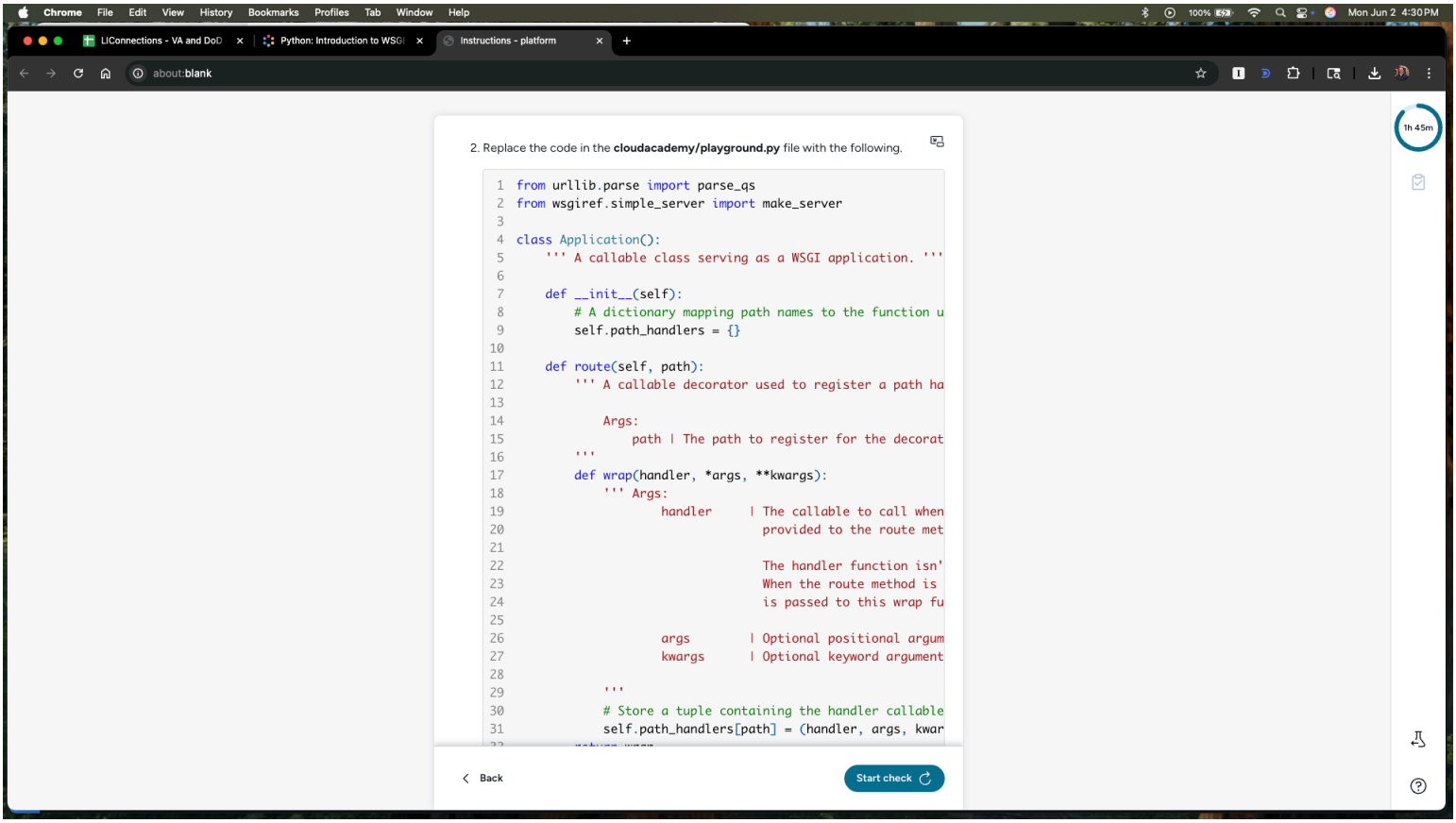
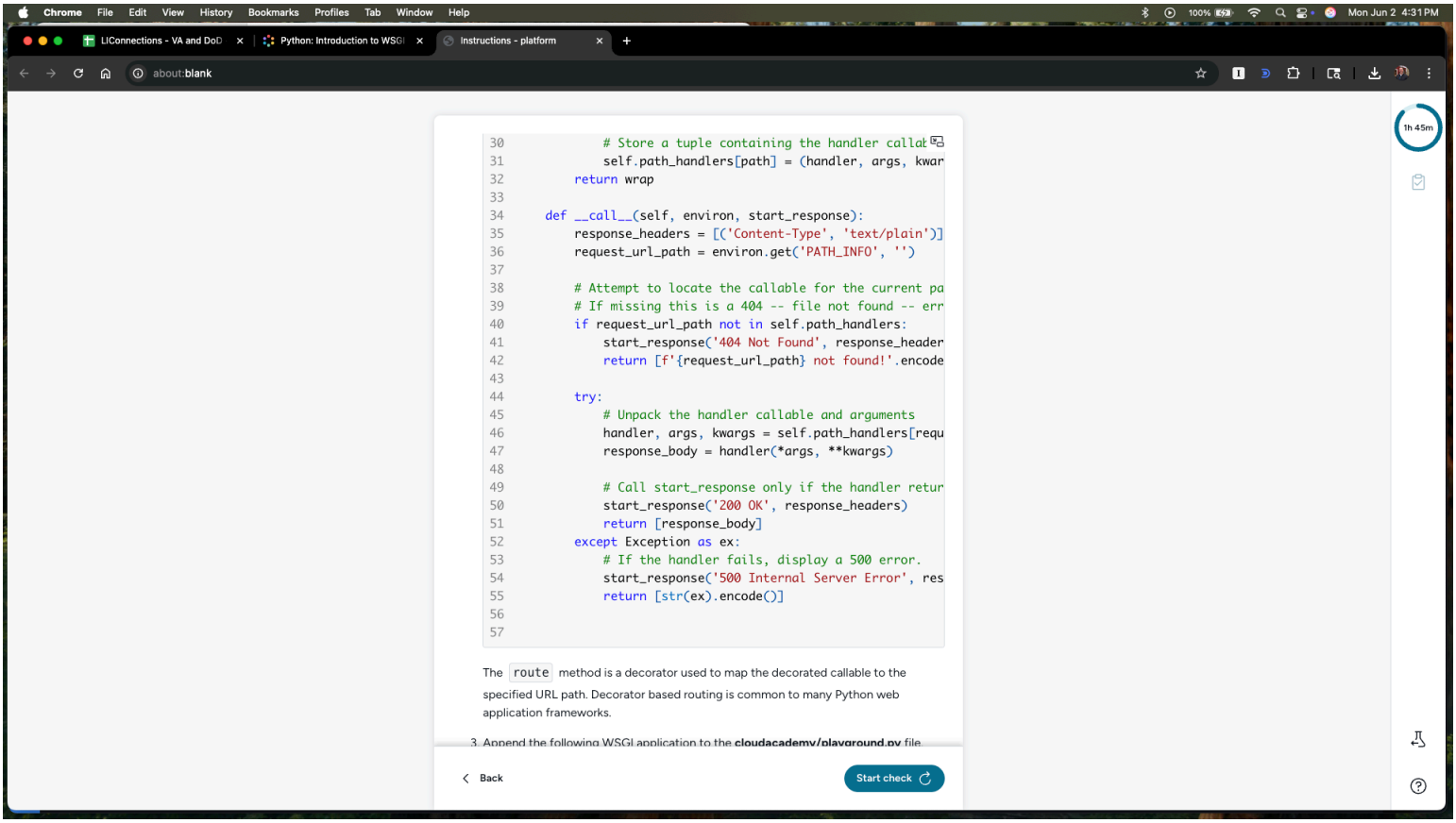
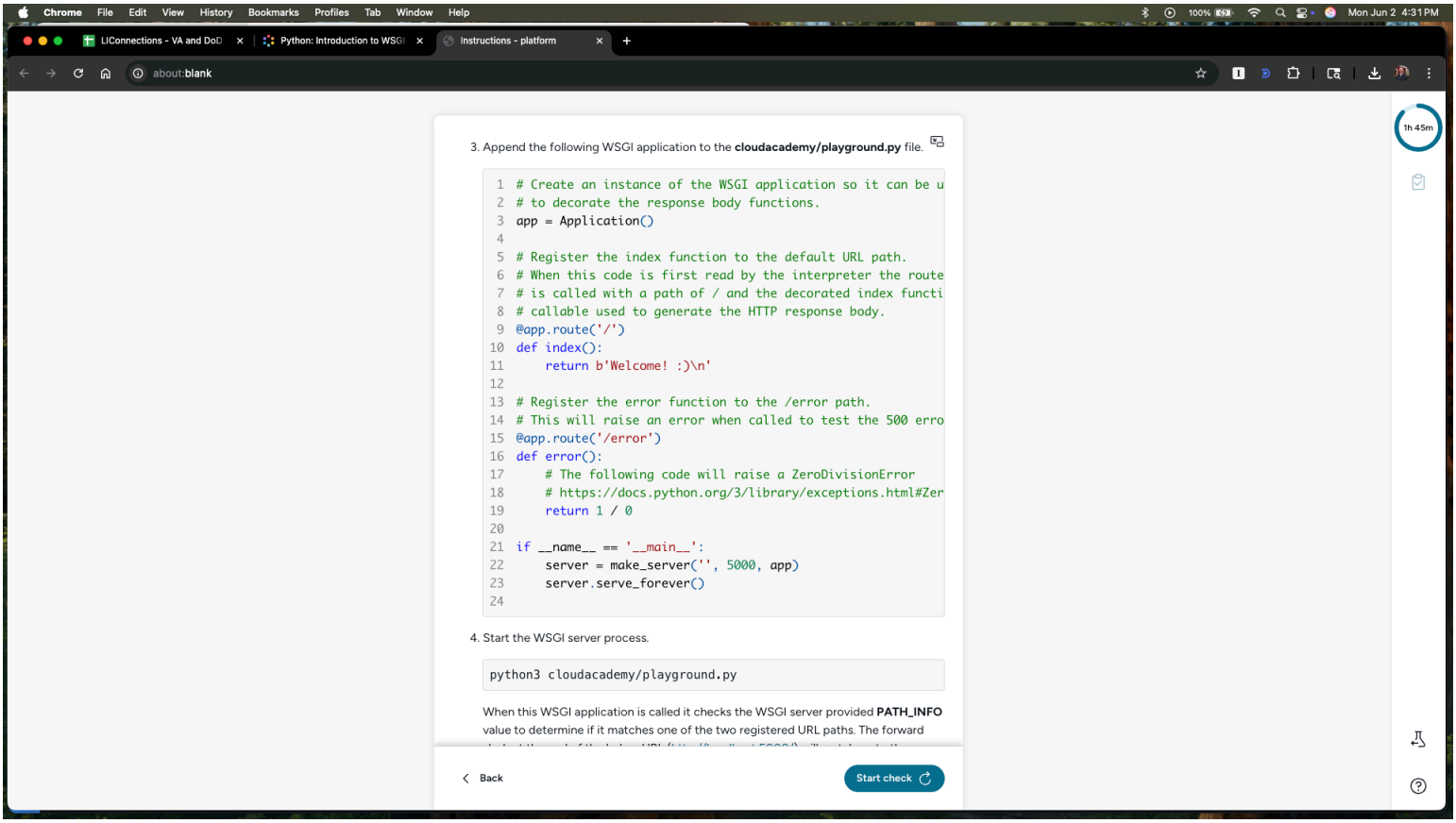

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 4:30 PM
LIConnections - VA and DoD
Python: Introduction to WSGI
Instructions - platform
about:blank
1h 45m
5 WSGI Application
WSGI Application
Most Python web development doesn't occur at the WSGI level. WSGI is designed to move HTTP messages between clients and servers. It intentionally omits higher level functionality which is expected to be provided by a WSGI compliant web framework.
WSGI frameworks such as Flask, Django, and CherryPy, among others include the types of high level features expected in a modern web framework. Features such as: URL path routing, query string parsing, template rendering, etc.
The application created in this step highlights some of the features required to create a more functional web application.
Instructions
1. Stop the WSGI server process.
1. Click anywhere inside the WSGI terminal pane to set focus.
2. Press CTRL+C to interrupt the process
URL paths are used to locate specific resources on a server. It's commonly up to a web application to determine the structure of URL paths and how they're used to look up resources.
In the URL https://cloudacademy.com/lab/python-wsgi/ the path of /lab/python-wsgi/ instructs the server to locate a lab environment named python-wsgi.
WSGI applications commonly map specific URL paths to callables used to generate HTTP response messages.
The Application class defined in the code below is a WSGI application capable of mapping static URLs to a callable which returns the HTTP response body for the given URL path.
Back
Start check


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 4:30 PM
LIConnections - VA and DoD
Python: Introduction to WSGI
Instructions - platform
about:blank
2. Replace the code in the cloudacademy/playground.py file with the following.
from urllib.parse import parse_qs
from wsgiref.simple_server import make_server
class Application():
''' A callable class serving as a WSGI application. '''
def __init__(self):
# A dictionary mapping path names to the function u
self.path_handlers = {}
def route(self, path):
''' A callable decorator used to register a path ha
Args:
path | The path to register for the decorat
'''
def wrap(handler, *args, **kwargs):
''' Args:
handler | The callable to call when
provided to the route met
The handler function isn'
When the route method is
is passed to this wrap fu
args | Optional positional argum
kwargs | Optional keyword argument
'''
# Store a tuple containing the handler callable
self.path_handlers[path] = (handler, args, kwar
1h 45m
Back
Start check


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 4:31 PM
LIConnections - VA and DoD
Python: Introduction to WSGI
Instructions - platform
about:blank
1h 45m
# Store a tuple containing the handler callab
self.path_handlers[path] = (handler, args, kwar
return wrap
def __call__(self, environ, start_response):
response_headers = [('Content-Type', 'text/plain')]
request_url_path = environ.get('PATH_INFO', '')
# Attempt to locate the callable for the current pa
# If missing this is a 404 -- file not found -- err
if request_url_path not in self.path_handlers:
start_response('404 Not Found', response_header
return [f'{request_url_path} not found!'.encode
try:
# Unpack the handler callable and arguments
handler, args, kwargs = self.path_handlers[requ
response_body = handler(*args, **kwargs)
# Call start_response only if the handler retur
start_response('200 OK', response_headers)
return [response_body]
except Exception as ex:
# If the handler fails, display a 500 error.
start_response('500 Internal Server Error', res
return [str(ex).encode()]
The route method is a decorator used to map the decorated callable to the specified URL path. Decorator based routing is common to many Python web application frameworks.
3. Append the following WSGI application to the cloudacademy/playground.py file
Back
Start check


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 4:31 PM
LIConnections - VA and DoD
Python: Introduction to WSGI
Instructions - platform
about:blank
1h 45m
3. Append the following WSGI application to the cloudacademy/playground.py file.
# Create an instance of the WSGI application so it can be u
# to decorate the response body functions.
app = Application()
# Register the index function to the default URL path.
# When this code is first read by the interpreter the route
# is called with a path of / and the decorated index functi
# callable used to generate the HTTP response body.
@app.route('/')
def index():
return b'Welcome! :)\n'
# Register the error function to the /error path.
# This will raise an error when called to test the 500 erro
@app.route('/error')
def error():
# The following code will raise a ZeroDivisionError
# https://docs.python.org/3/library/exceptions.html#Zer
return 1 / 0
if __name__ == '__main__':
server = make_server('', 5000, app)
server.serve_forever()
4. Start the WSGI server process.
python3 cloudacademy/playground.py
When this WSGI application is called it checks the WSGI server provided PATH_INFO value to determine if it matches one of the two registered URL paths. The forward
Back
Start check

4. Start the WSGI server process.

```
python3 cloudacademy/playground.py
```

When this WSGI application is called it checks the WSGI server provided **PATH_INFO** value to determine if it matches one of the two registered URL paths. The forward slash at the end of the below URL (http://localhost:5000/) will match up to the `index` function.

5. Test the index URL path in the **curl terminal**.

```
curl http://localhost:5000/
```

```
Welcome! :)
```

If an error occurs when calling when calling the registered URL handler callable the exception handler returns a status of **500 Internal Server Error** and includes an error message.

6. Test the error URL path in the **curl terminal**.

```
curl -v http://localhost:5000/error
```

```
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 5000 (#0)
> GET /error HTTP/1.1
> Host: localhost:5000
> User-Agent: curl/7.52.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 500 Internal Server Error
< Date: Thu, 23 Jun 2022 17:55:26 GMT
< Server: WSGIServer/0.2 CPython/3.9.9
< Content-Type: text/plain
< Content-Length: 16
<
* Curl_http_done: called premature == 0
* Closing connection 0
division by zero
```

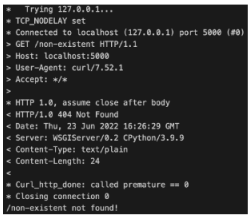
Back

Start check

If a non-existent URL path is requested a status of **404 Not Found** is returned.

7. Test a non-existent URL path in the **curl terminal**.

```
curl -v http://localhost:5000/non-existent
```




Modular WSGI applications can become middleware used to extend the functionality of a WSGI application. Self contained functionality such as parsing URL query strings is generalized enough to become middleware.

When the below `QueryStringParser` object is called it uses the `parse_qs` callable to parse the WSGI server-provided URL query string. The `parse_qs` callable returns a dictionary of key-value pairs from the query string. The `environ` argument is updated to store the parsed dictionary with the key **QUERY_STRING_PARSED**.

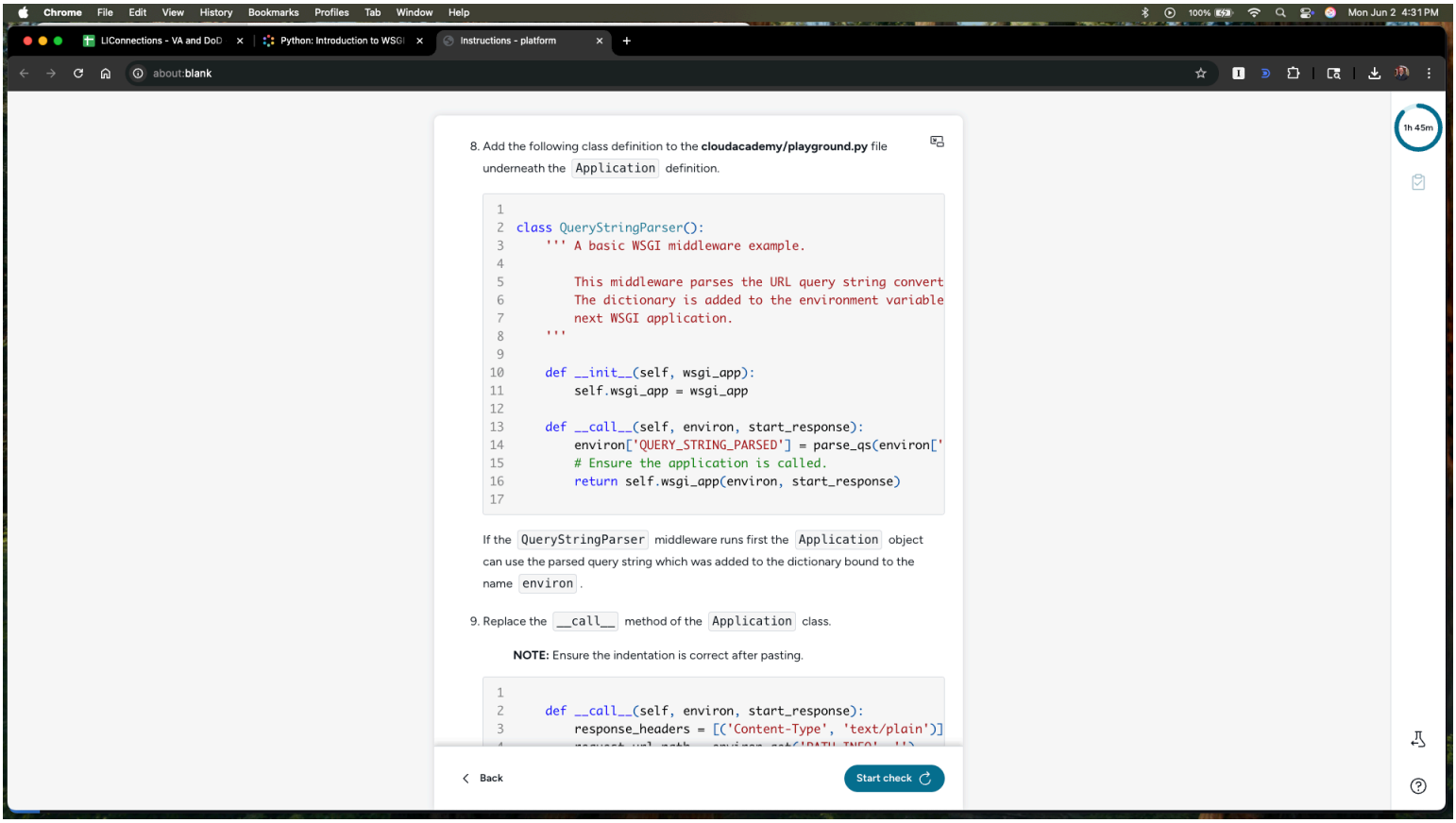
8. Add the following class definition to the **cloudacademy/playground.py** file underneath the `Application` definition.

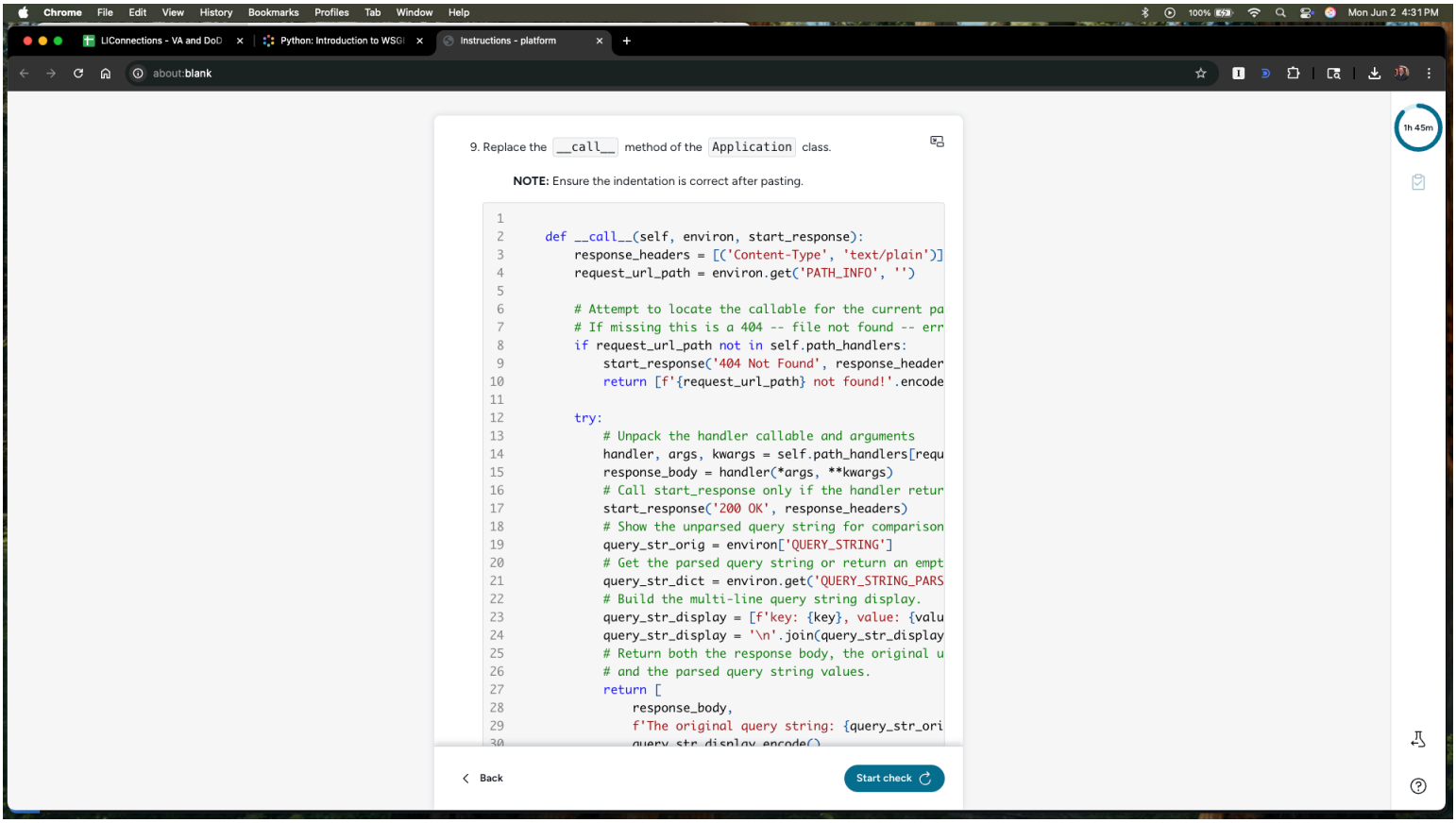
```
1
2 class QueryStringParser():
```

Back

Start check

1h 45m


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 4:31 PM
LIConnections - VA and DoD
Python: Introduction to WSGI
Instructions - platform
about:blank
1h 45m
8. Add the following class definition to the cloudacademy/playground.py file underneath the Application definition.
class QueryStringParser():
''' A basic WSGI middleware example.
This middleware parses the URL query string convert
The dictionary is added to the environment variable
next WSGI application.
'''
def __init__(self, wsgi_app):
self.wsgi_app = wsgi_app
def __call__(self, environ, start_response):
environ['QUERY_STRING_PARSED'] = parse_qs(environ['
# Ensure the application is called.
return self.wsgi_app(environ, start_response)
If the QueryStringParser middleware runs first the Application object can use the parsed query string which was added to the dictionary bound to the name environ .
9. Replace the __call__ method of the Application class.
NOTE: Ensure the indentation is correct after pasting.
def __call__(self, environ, start_response):
response_headers = [('Content-Type', 'text/plain')]
Back
Start check


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 4:31 PM
LIConnections - VA and DoD
Python: Introduction to WSGI
Instructions - platform
about:blank
1h 45m
9. Replace the __call__ method of the Application class.
NOTE: Ensure the indentation is correct after pasting.
def __call__(self, environ, start_response):
response_headers = [('Content-Type', 'text/plain')]
request_url_path = environ.get('PATH_INFO', '')
# Attempt to locate the callable for the current pa
# If missing this is a 404 -- file not found -- err
if request_url_path not in self.path_handlers:
start_response('404 Not Found', response_header
return [f'{request_url_path} not found!'.encode
try:
# Unpack the handler callable and arguments
handler, args, kwargs = self.path_handlers[requ
response_body = handler(*args, **kwargs)
# Call start_response only if the handler retur
start_response('200 OK', response_headers)
# Show the unparsed query string for comparison
query_str_orig = environ['QUERY_STRING']
# Get the parsed query string or return an empt
query_str_dict = environ.get('QUERY_STRING_PARS
# Build the multi-line query string display.
query_str_display = [f'key: {key}, value: {valu
query_str_display = '\n'.join(query_str_display
# Return both the response body, the original u
# and the parsed query string values.
return [
response_body,
f'The original query string: {query_str_ori
query_str_display.encode()
Back
Start check

```
            # Return both the response body, the original
            # and the parsed query string values.
            return [
                response_body,
                f'The original query string: {query_str_ori
                query_str_display.encode()
            ]
        except Exception as ex:
            # If the handler fails, display a 500 error.
            start_response('500 Internal Server Error', res
            return [str(ex).encode()]

```

The `QueryStringParser` middleware parses the query string from `environ` and saves the parsed results as a dictionary. When the middleware runs first the parsed results will be available to the WSGI application. Wrapping the application with the middleware enables it to run first.

10. Replace the main code block.

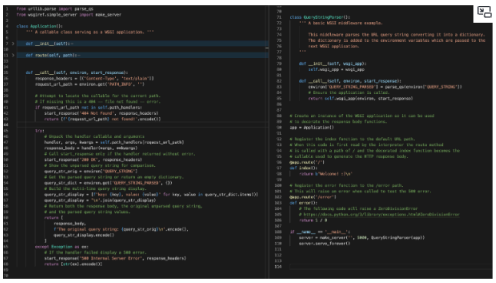
```
if __name__ == '__main__':
    server = make_server('', 5000, QueryStringParser(app))
    server.serve_forever()

```

Back

Start check

1h 44m



11. Restart the WSGI server process.

    Query string parameters appear at the end of a URL separated by a question mark ( `?` ). They're specified as key-value pairs of data separated by an equal sign ( `=` ).

    Multiple parameters are separated by an ampersand ( `&` ).

12. Test a URL query string in the **curl terminal**.

```
curl "http://localhost:5000/?greeting=Hello&target=World"
```

⭐ Proceed to the next step ⭐

## Validations

On Track - Building a WSGI application

Ensure the `Application` and `QueryStringParser` class definitions exist in the **cloudacademy/playground.py** file.

Back

Start check

1h 44m