# API testing in Python

using the requests library

An open source workshop by …

Originally created by Bas Dijkstra – bas@ontestautomation.com – https://www.ontestautomation.com

## What are we going to do?

_RESTful APIs

_requests

_Hands-on exercises

## Preparation

_Install Python 3

_Install PyCharm (or any other IDE)

_Import project into IDE
  _https://github.com/basdijkstra/requests-workshop

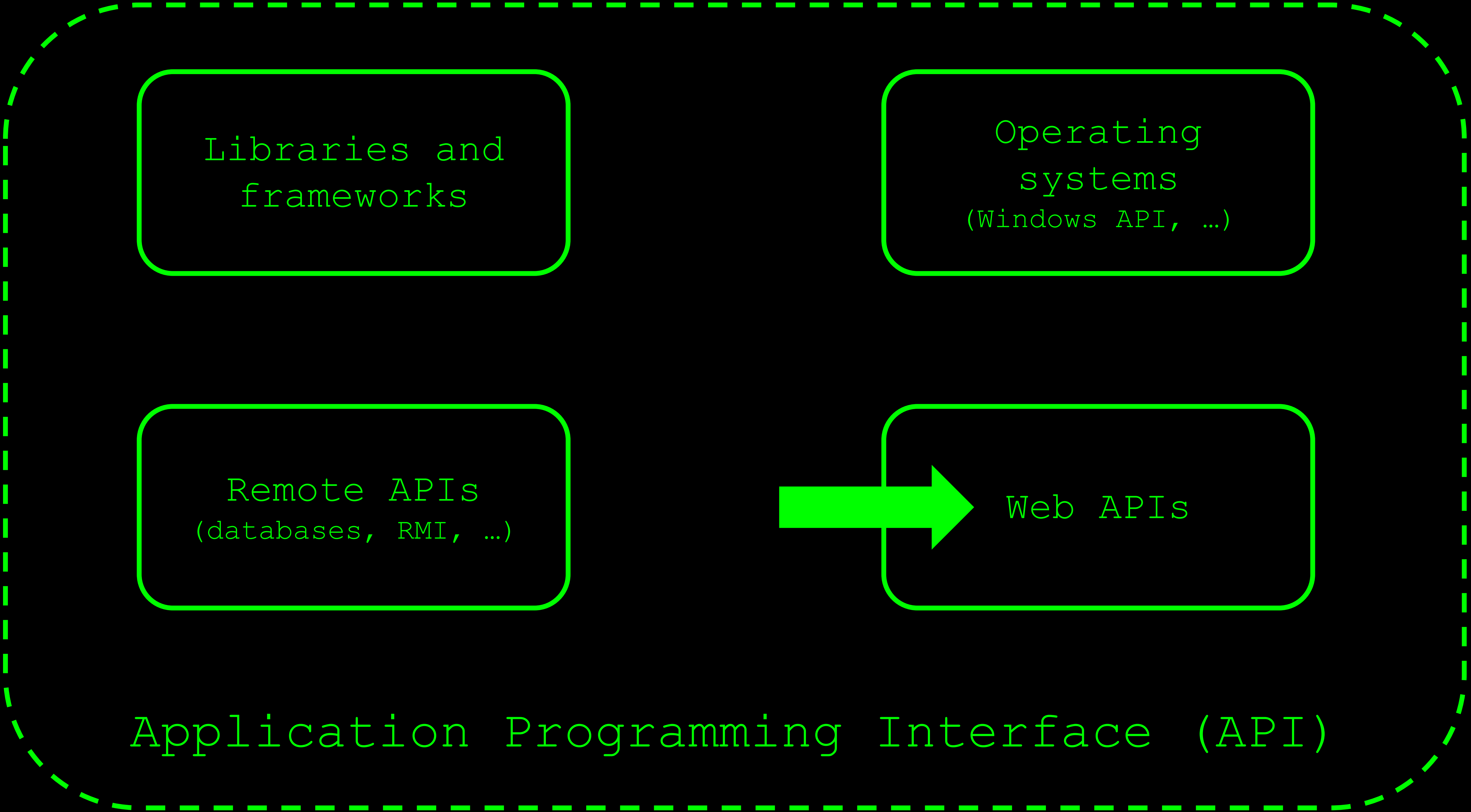
_Install dependencies, from project root:

```
pip install -r requirements.txt
```

## So, what is an API?

*“An* ***application programming interface (API)*** *is an interface or communication protocol between different parts of a computer program intended to simplify the implementation and maintenance of software”*

https://en.wikipedia.org/wiki/Application_programming_interface


Libraries and frameworks
Operating systems
(Windows API, …)
Remote APIs
(databases, RMI, …)
Web APIs
Application Programming Interface (API)

From now on, I'll refer to these Web APIs simply as 'APIs'

## Where are APIs used?

Mobile

Internet of Things

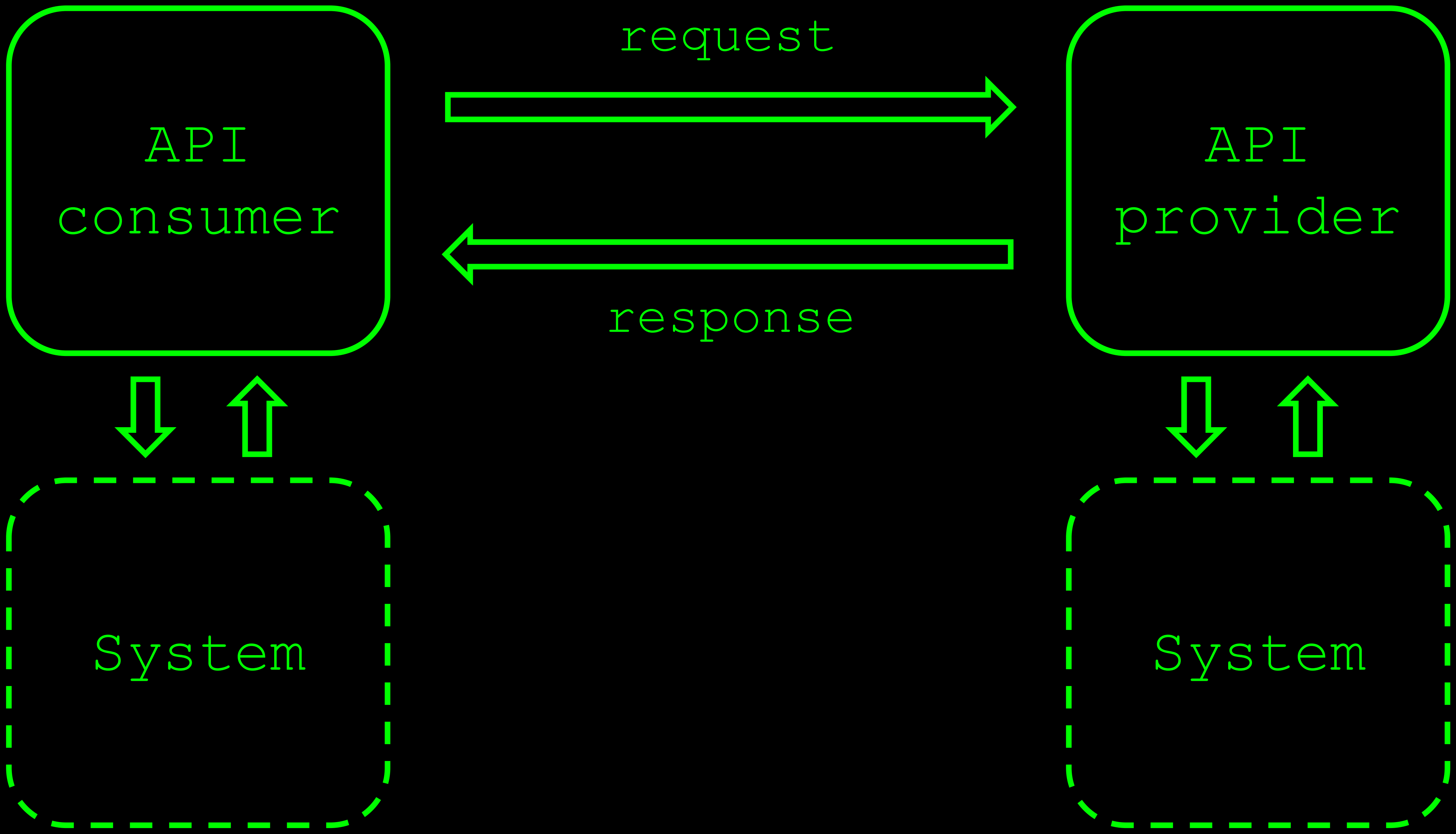
API economy

## Where are APIs used?

Web applications

Microservices architectures

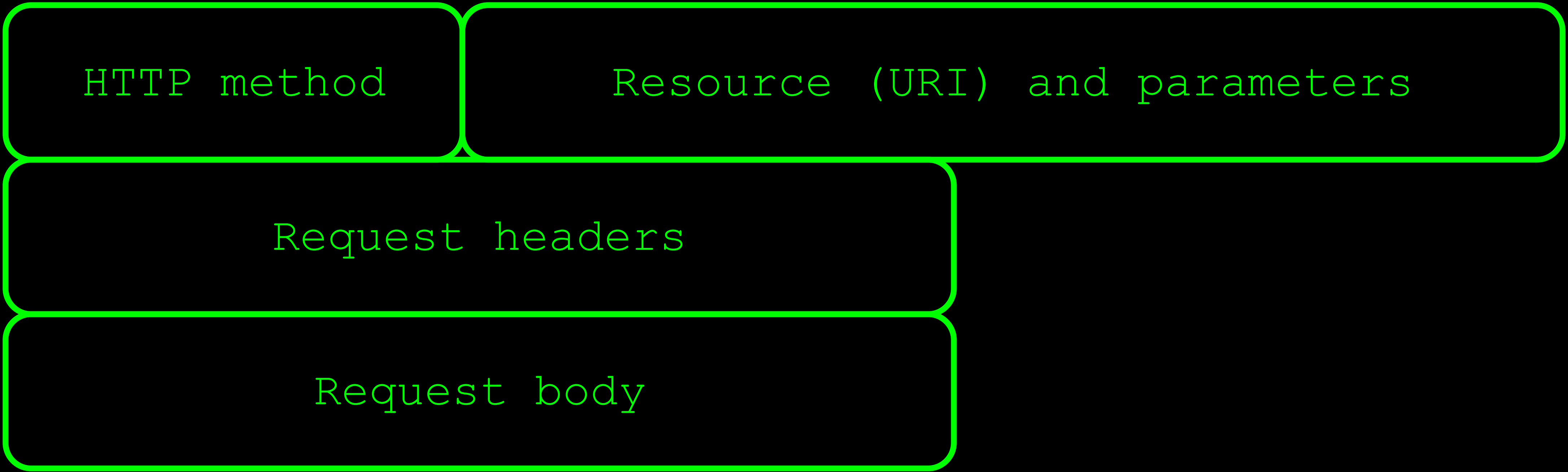
APIs are commonly used to exchange data between two parties

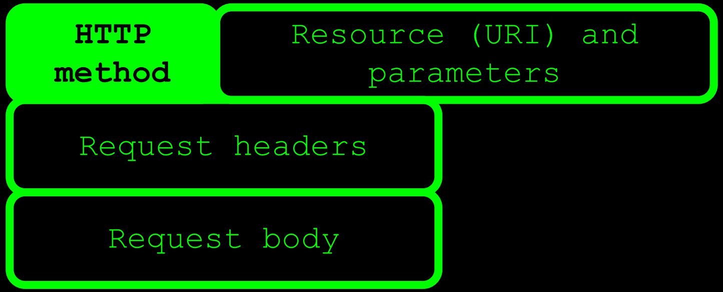

request
API consumer
API provider
response
System
System

# SOAP and REST

| | **SOAP** | **REST** |
|---|---|---|
| *Protocol* | HTTP, SMTP, … | HTTP |
| *Message format* | XML | XML, JSON, text, … |
| *Specification* | WSDL | WADL, RAML, Swagger, … |
| *Standardized?* | Yes | No |

# A REST API request





# HTTP methods

_GET, POST, PUT, PATCH, DELETE, OPTIONS, …

_CRUD operations on data

| | |
|---|---|
| POST | Create |
| GET | Read |
| PUT / PATCH | Update |
| DELETE | Delete |
| … | … |

_Conventions, not standards!

HTTP method | **Resource (URI) and parameters**

Request headers

Request body

## Resources and parameters

_Uniform Resource Identifier

_Uniquely identifies the resource to operate on

_Can contain parameters

  _Query parameters

  _Path parameters

## Resources and parameters

_Path parameters

  _http://api.zippopotam.us/us/90210

  _http://api.zippopotam.us/ca/B2A

_Query parameters

  _http://md5.jsontest.com/?text=testcaseOne

  _http://md5.jsontest.com/?text=testcaseTwo

_There is no official standard!

# Request headers

_Key-value pairs

_Can contain metadata about the request body
  _Content-Type (what data format is the request body in?)
  _Accept (what data format would I like the response body to be in?)
  _…

_Can contain session and authorization data
  _Cookies
  _Authorization tokens
  _…

## Authorization: Basic

_Username and password sent with every request

_Base64 encoded (not really secure!)

_Ex: username = aladdin and password = opensesame

*Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l*

HTTP method | Resource (URI) and parameters | **Request headers** | Request body

# Authorization: Bearer

_Token with limited validity is obtained first

_Token is then sent with all subsequent requests

_Most common mechanism is OAuth(2)

_JWT is a common token format

*Authorization: Bearer RsT5OjbzRn430zqMLgV3Ia*

# Request body

_Data to be sent to the provider

_REST does not prescribe a specific data format

_Most common:
  _JSON
  _XML
  _Plain text

_Other data formats can be sent using REST, too

# A REST API response

HTTP status code

Response headers

Response body

## HTTP status code

_Indicates result of request processing by provider

_Five different categories

| | | |
|---|---|---|
| _1XX | Informational | 100 Continue |
| _2XX | Success | 200 OK |
| _3XX | Redirection | 301 Moved Permanently |
| _4XX | Client errors | 400 Bad Request |
| _5XX | Server errors | 503 Service Unavailable |

HTTP status code

**Response headers**

Response body

# Response headers

_Key-value pairs

_Can contain metadata about the response body
- _Content-Type (what data format is the response body in?)
- _Content-Length (how many bytes in the response body?)

_Can contain provider-specific data
- _Caching-related headers
- _Information about the server type

## Response body

_Data returned by the provider

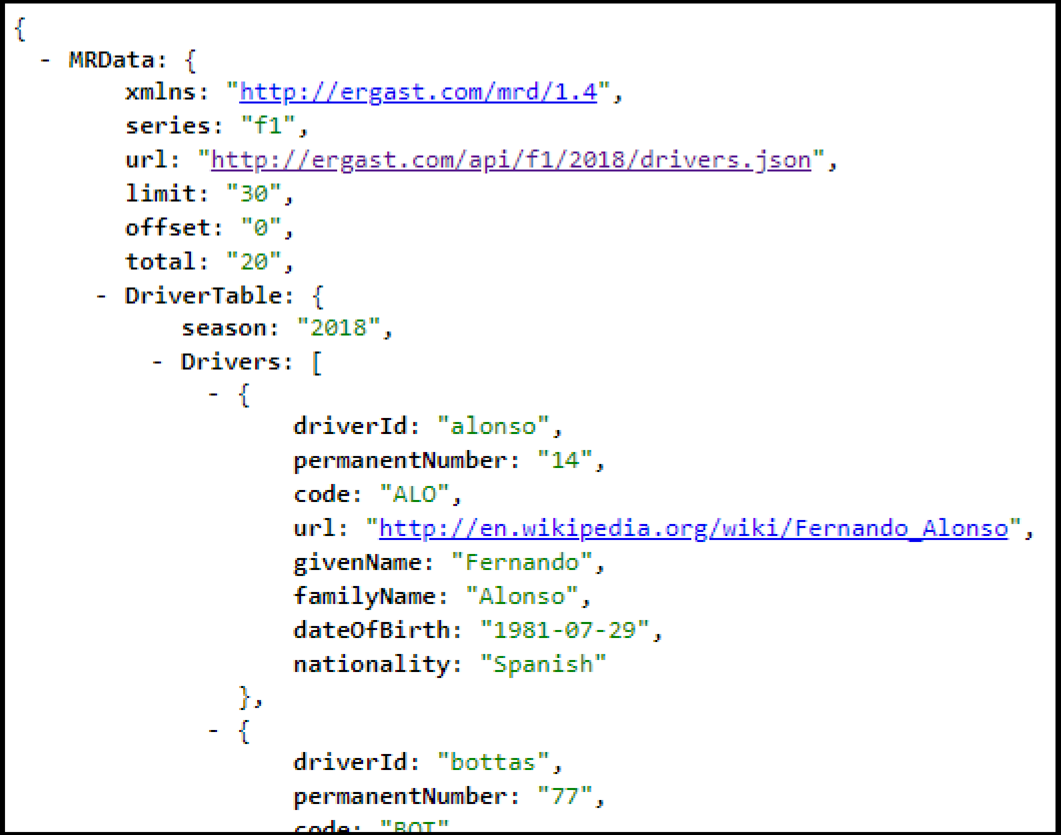
_REST does not prescribe a specific data format

_Most common:
  _JSON
  _XML
  _Plain text

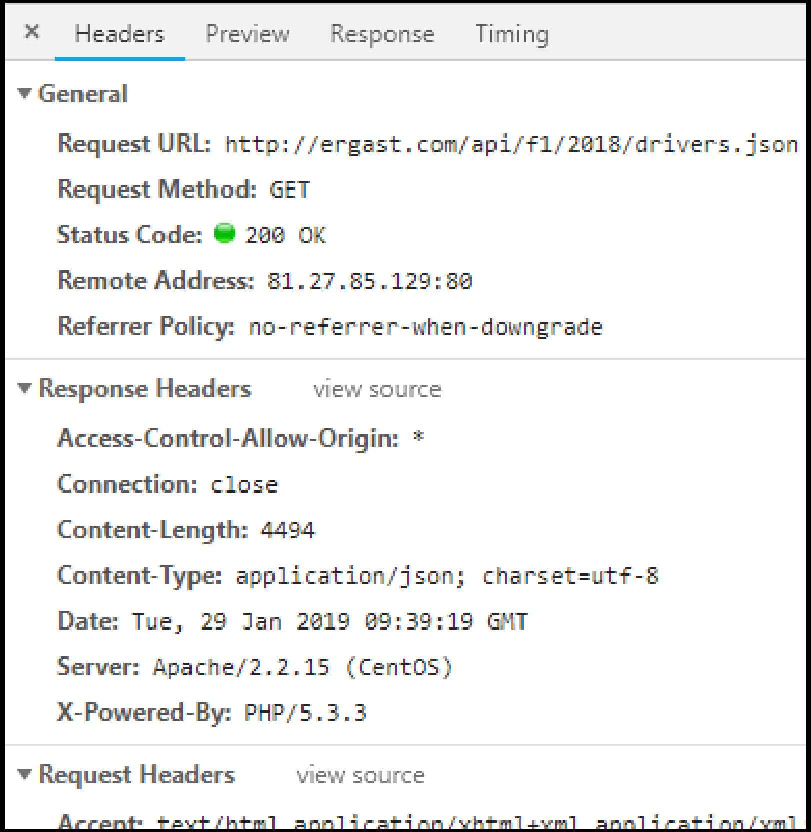
_Other data formats can be sent using REST, too

# An example

_GET http://ergast.com/api/f1/2018/drivers.json



```
{
 - MRData: {
       xmlns: "http://ergast.com/mrd/1.4",
       series: "f1",
       url: "http://ergast.com/api/f1/2018/drivers.json",
       limit: "30",
       offset: "0",
       total: "20",
     - DriverTable: {
           season: "2018",
         - Drivers: [
             - {
                   driverId: "alonso",
                   permanentNumber: "14",
                   code: "ALO",
                   url: "http://en.wikipedia.org/wiki/Fernando_Alonso",
                   givenName: "Fernando",
                   familyName: "Alonso",
                   dateOfBirth: "1981-07-29",
                   nationality: "Spanish"
               },
             - {
                   driverId: "bottas",
                   permanentNumber: "77",
                   code: "BOT"
```



```
▼ General
    Request URL: http://ergast.com/api/f1/2018/drivers.json
    Request Method: GET
    Status Code: ● 200 OK
    Remote Address: 81.27.85.129:80
    Referrer Policy: no-referrer-when-downgrade
▼ Response Headers      view source
    Access-Control-Allow-Origin: *
    Connection: close
    Content-Length: 4494
    Content-Type: application/json; charset=utf-8
    Date: Tue, 29 Jan 2019 09:39:19 GMT
    Server: Apache/2.2.15 (CentOS)
    X-Powered-By: PHP/5.3.3
▼ Request Headers      view source
    Accept: text/html,application/xhtml+xml,application/xml
```

## Why I ♥ testing at the API level

_Tests run much faster than UI-driven tests

_Tests are easier to stabilize than UI-driven tests

_Tests have a broader scope than unit tests

_Business logic is often exposed at the API level

## Tools for testing RESTful web services

_Free / open source

  _Postman, SoapUI, REST Assured, requests, …

_Commercial

  _Parasoft SOAtest, SoapUI Pro, …

_Build your own (using HTTP libraries for your language of choice)

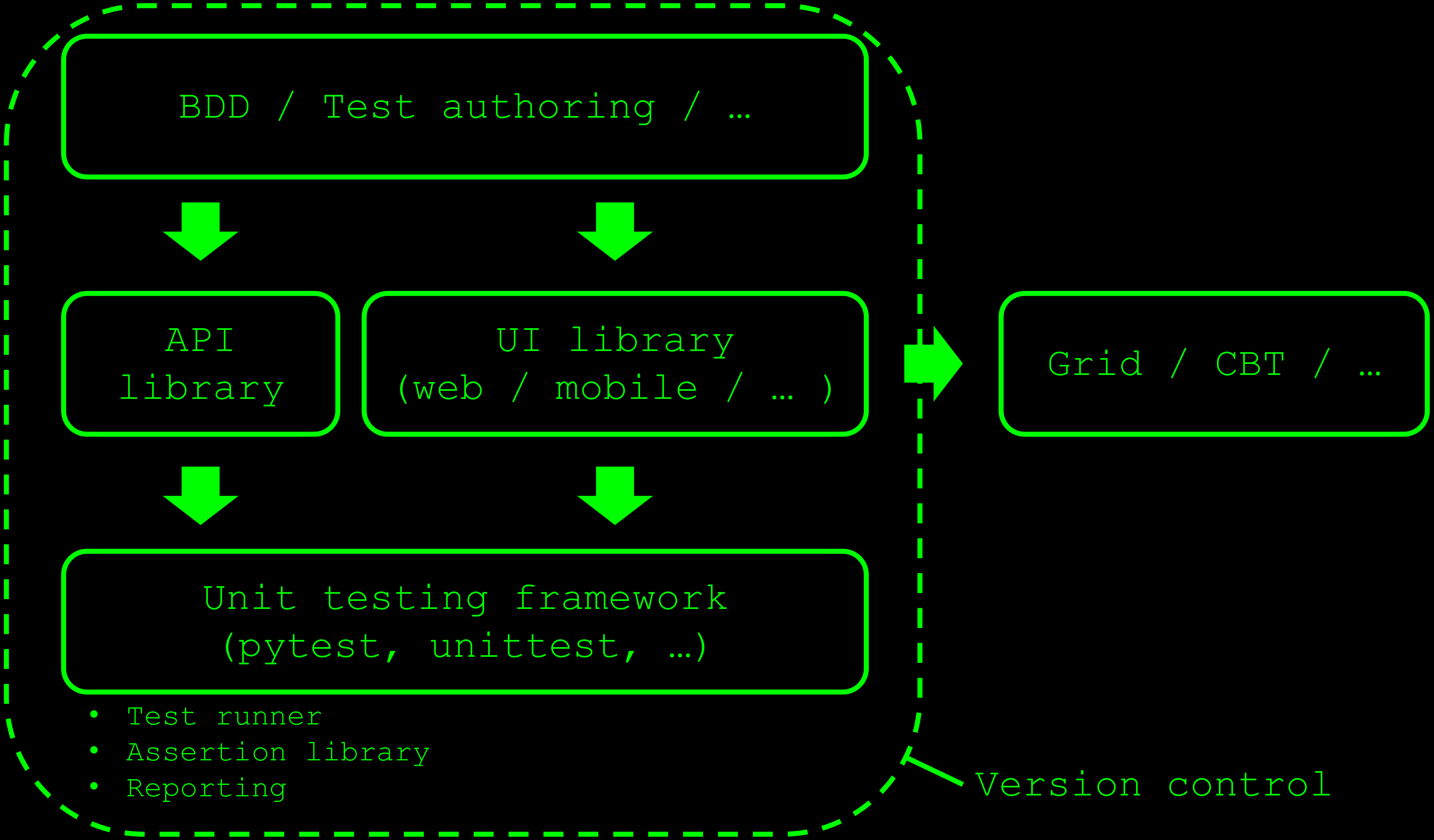
Python library for interacting with REST APIs

*“Requests is an elegant and simple HTTP library for Python, built for human beings.”*

## requests

```
pip install requests
```

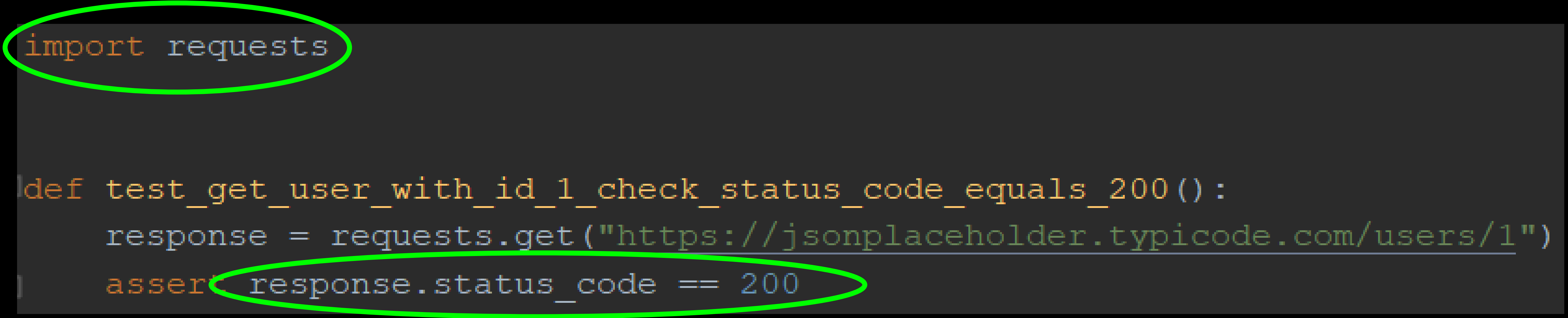
https://requests.readthedocs.io/en/master/


BDD / Test authoring / …
API library
UI library (web / mobile / … )
Unit testing framework (pytest, unittest, …)
• Test runner
• Assertion library
• Reporting
Grid / CBT / …
Version control

In this workshop, we'll use requests with pytest

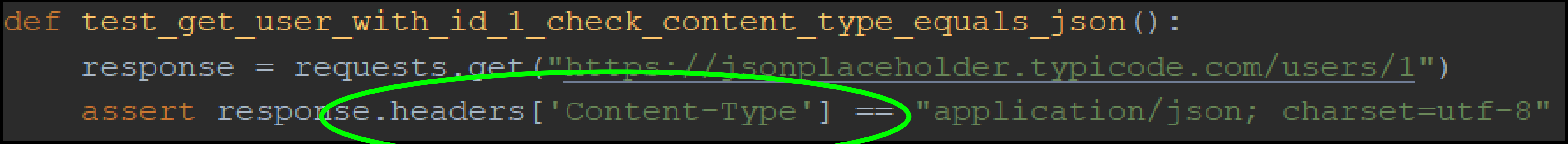
## A few example tests

## Checking response status code



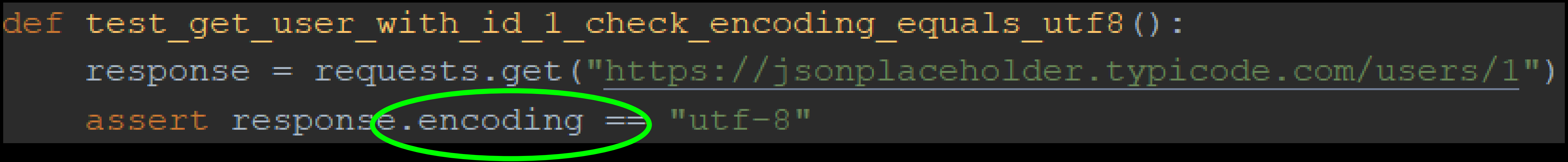
```
import requests


def test_get_user_with_id_1_check_status_code_equals_200():
    response = requests.get("https://jsonplaceholder.typicode.com/users/1")
    assert response.status_code == 200
```

# Checking response headers



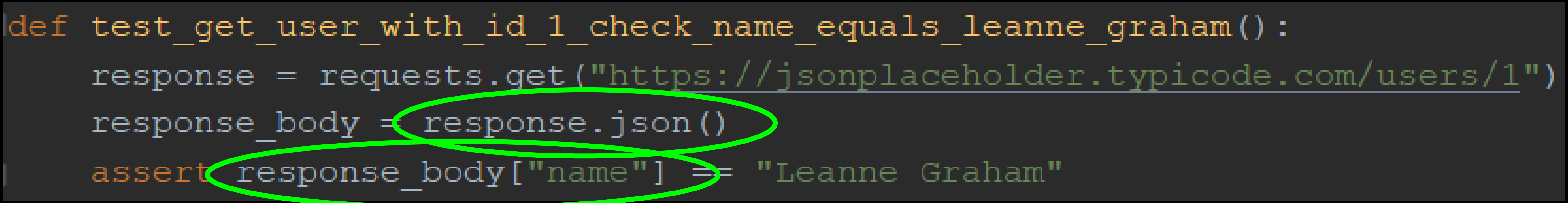
```
def test_get_user_with_id_1_check_content_type_equals_json():
    response = requests.get("https://jsonplaceholder.typicode.com/users/1")
    assert response.headers['Content-Type'] == "application/json; charset=utf-8"
```

# Checking response encoding

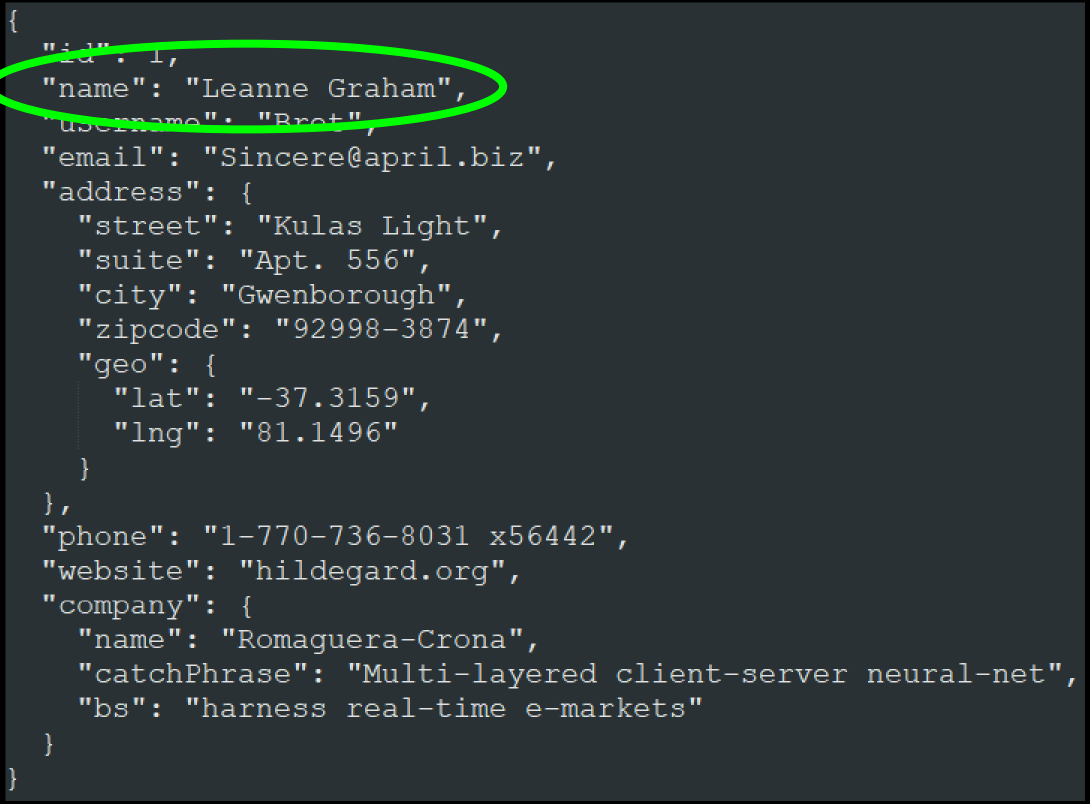


```python
def test_get_user_with_id_1_check_encoding_equals_utf8():
    response = requests.get("https://jsonplaceholder.typicode.com/users/1")
    assert response.encoding == "utf-8"
```

# Checking a JSON body element

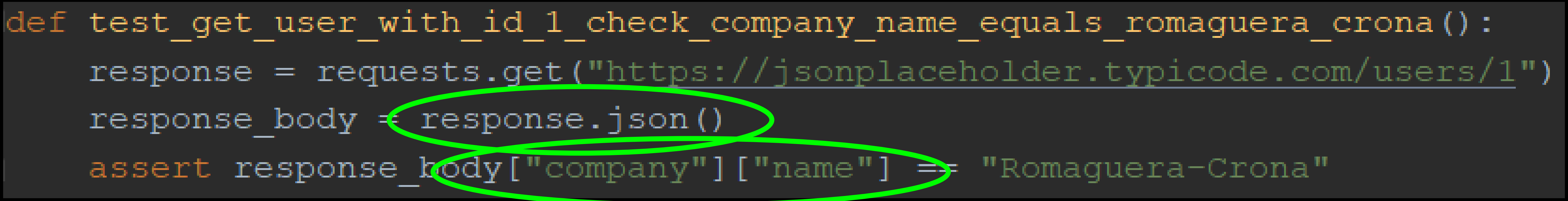


```
def test_get_user_with_id_1_check_name_equals_leanne_graham():
    response = requests.get("https://jsonplaceholder.typicode.com/users/1")
    response_body = response.json()
    assert response_body["name"] == "Leanne Graham"
```

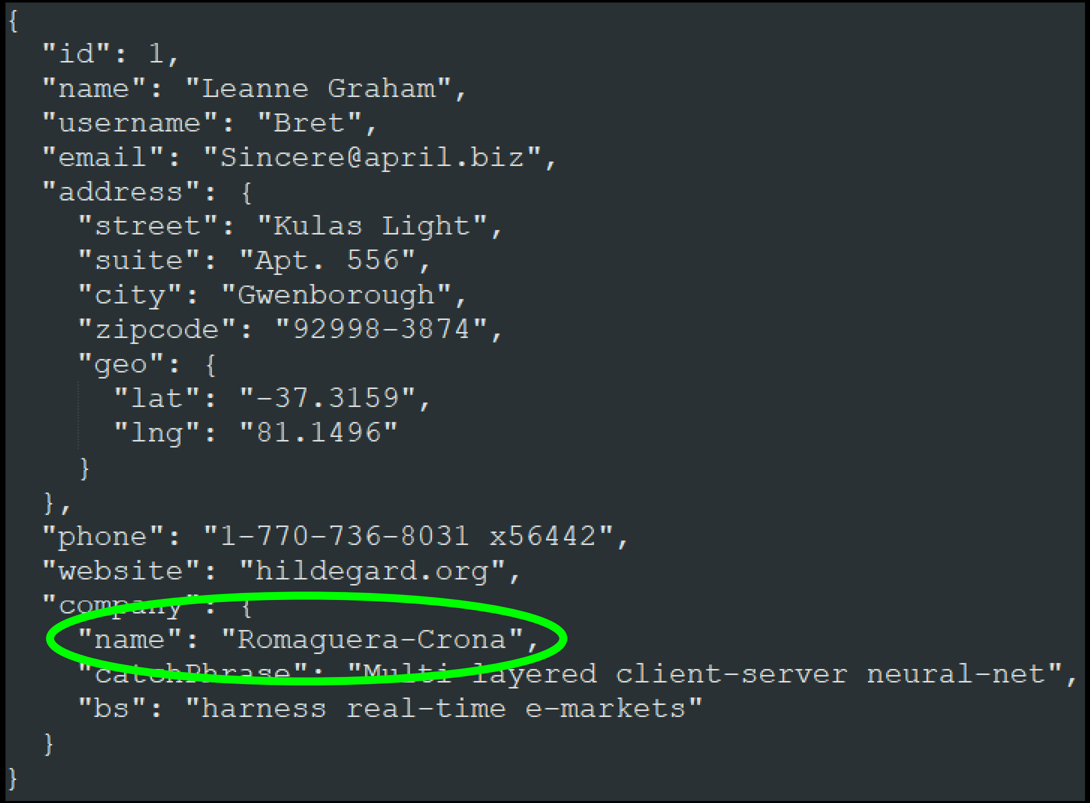


```
{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",
  "address": {
    "street": "Kulas Light",
    "suite": "Apt. 556",
    "city": "Gwenborough",
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  },
  "phone": "1-770-736-8031 x56442",
  "website": "hildegard.org",
  "company": {
    "name": "Romaguera-Crona",
    "catchPhrase": "Multi-layered client-server neural-net",
    "bs": "harness real-time e-markets"
  }
}
```

# Checking nested body elements

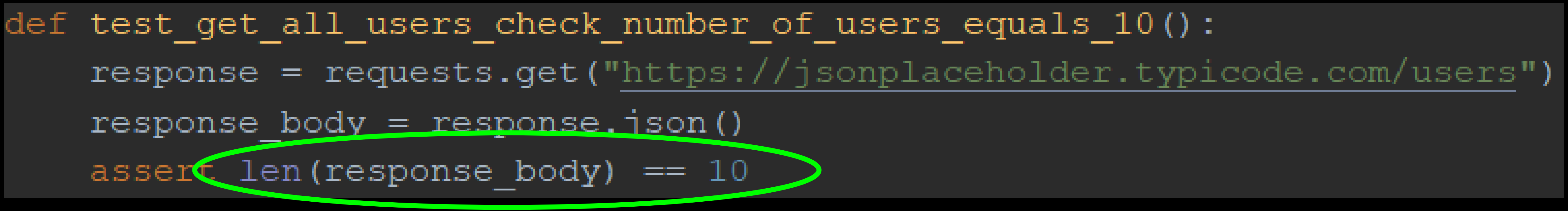


```python
def test_get_user_with_id_1_check_company_name_equals_romaguera_crona():
    response = requests.get("https://jsonplaceholder.typicode.com/users/1")
    response_body = response.json()
    assert response_body["company"]["name"] == "Romaguera-Crona"
```



```json
{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",
  "address": {
    "street": "Kulas Light",
    "suite": "Apt. 556",
    "city": "Gwenborough",
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  },
  "phone": "1-770-736-8031 x56442",
  "website": "hildegard.org",
  "company": {
    "name": "Romaguera-Crona",
    "catchPhrase": "Multi-layered client-server neural-net",
    "bs": "harness real-time e-markets"
  }
}
```

# Checking the size of an array



```python
def test_get_all_users_check_number_of_users_equals_10():
    response = requests.get("https://jsonplaceholder.typicode.com/users")
    response_body = response.json()
    assert len(response_body) == 10
```

## Our API under test

_Zippopotam.us

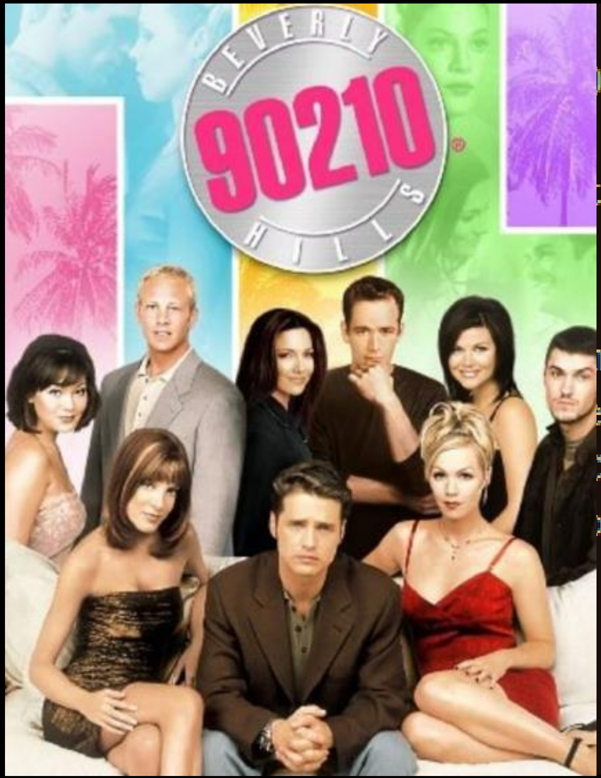
_Returns location data based on country and zip code

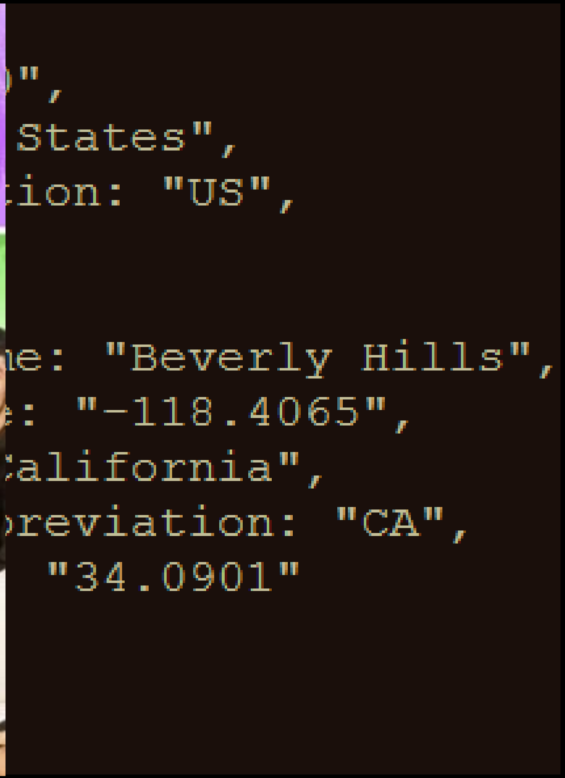
_http://api.zippopotam.us/

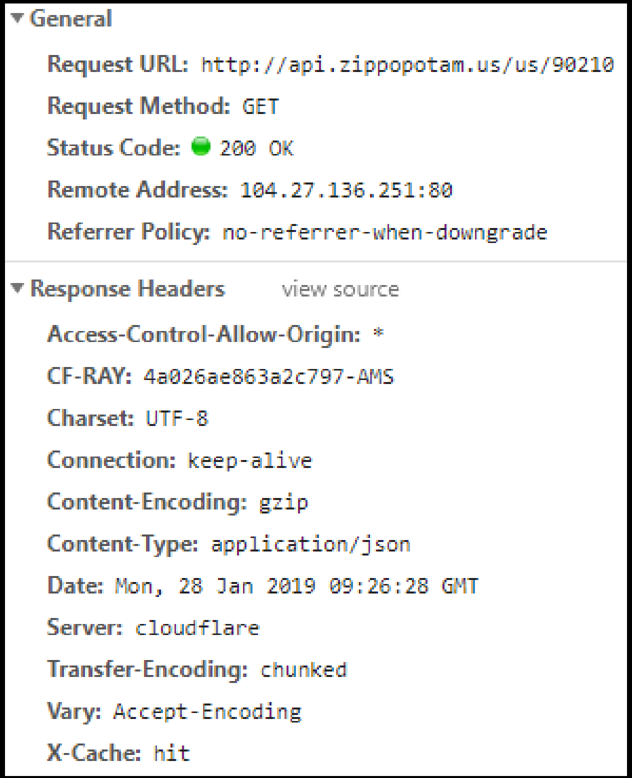
_RESTful API

## An example

_GET http://api.zippopotam.us/us/90210




```
)",
 States",
:ion: "US",


ıe: "Beverly Hills",
e: "-118.4065",
California",
oreviation: "CA",
 "34.0901"
```



▼ General

Request URL: http://api.zippopotam.us/us/90210
Request Method: GET
Status Code: ● 200 OK
Remote Address: 104.27.136.251:80
Referrer Policy: no-referrer-when-downgrade

▼ Response Headers view source

Access-Control-Allow-Origin: *
CF-RAY: 4a026ae863a2c797-AMS
Charset: UTF-8
Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/json
Date: Mon, 28 Jan 2019 09:26:28 GMT
Server: cloudflare
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Cache: hit

## Now it’s your turn!

_ exercises > exercises_01.py

_ run your answers (from the project root) using

*pytest exercises\exercises_01.py*

_ examples are in examples > examples_01.py

_ answers are in answers > answers_01.py

Exchange data between consumer and provider

GET to retrieve data from provider, POST to send data to provider, …

# APIs are all about data

Business logic and calculations often exposed through APIs

Run the same test more than once…

… for different combinations of input and expected output values

## Data driven testing

More efficient to do this at the API level…

… as compared to doing this at the UI level http://chrismcmahonsblog.blogspot.com/2017/11/ui-test-heuristic-dont-repeat-your-paths.html

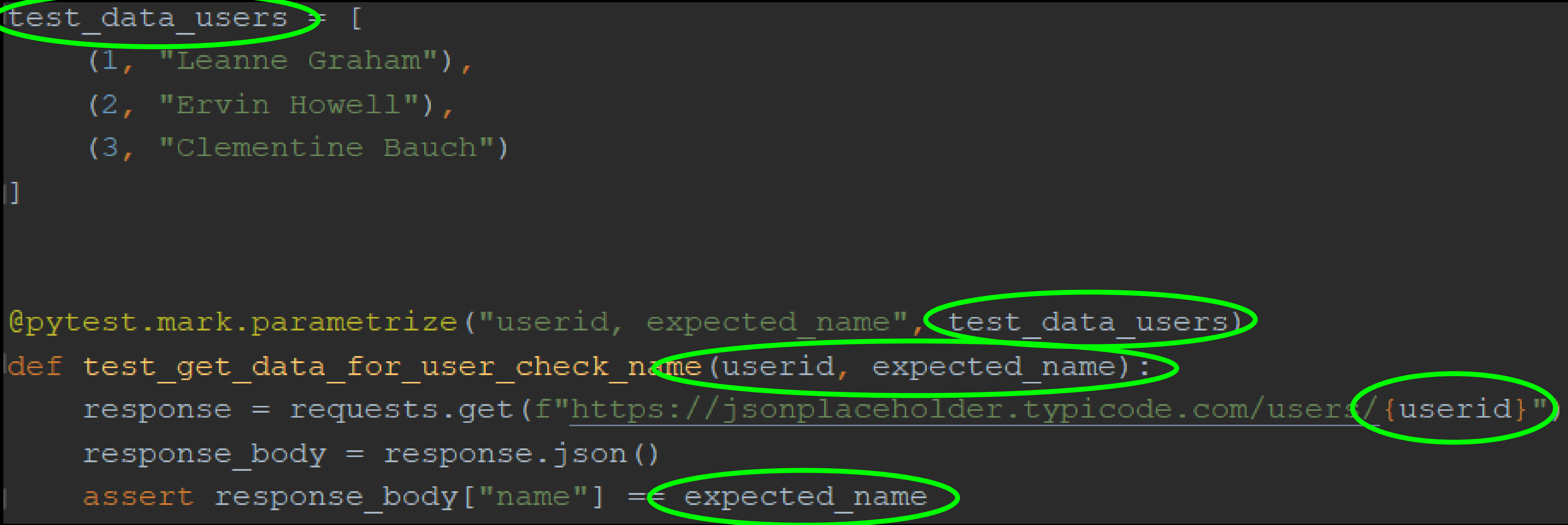
## Parameters in RESTful APIs

_Path parameters
   _http://api.zippopotam.us/us/90210
   _http://api.zippopotam.us/ca/B2A

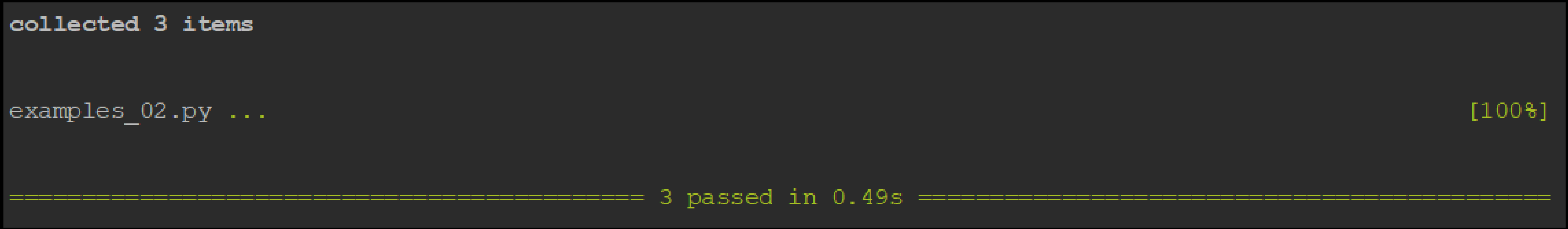
_Query parameters
   _http://md5.jsontest.com/?text=testcaseOne
   _http://md5.jsontest.com/?text=testcaseTwo

_There is no official standard!

# Data driven API testing



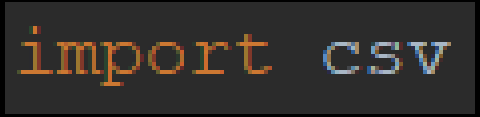
```
test_data_users = [
    (1, "Leanne Graham"),
    (2, "Ervin Howell"),
    (3, "Clementine Bauch")
]


@pytest.mark.parametrize("userid, expected_name", test_data_users)
def test_get_data_for_user_check_name(userid, expected_name):
    response = requests.get(f"https://jsonplaceholder.typicode.com/users/{userid}"
    response_body = response.json()
    assert response_body["name"] == expected_name
```



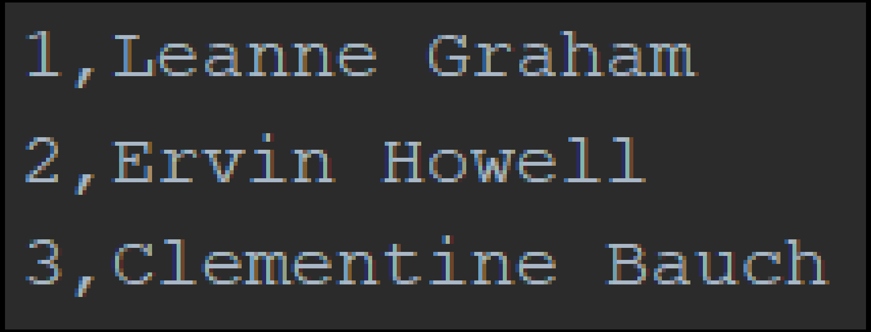
```
collected 3 items

examples_02.py ...                                                          [100%]

=============================== 3 passed in 0.49s ===============================
```

## Working with external data sources

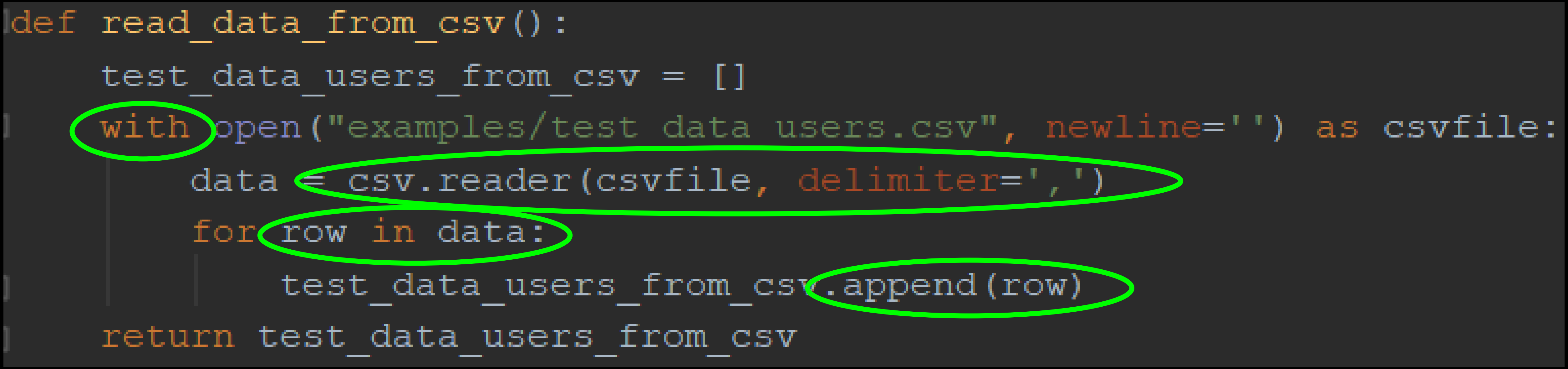
# Reading a .csv file



```python
import csv
```

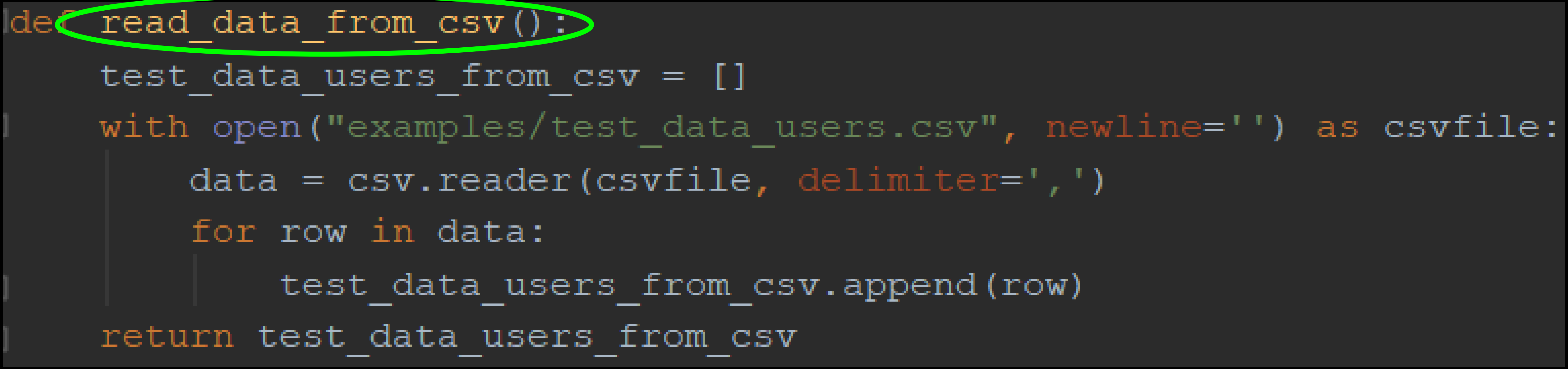


```
1,Leanne Graham
2,Ervin Howell
3,Clementine Bauch
```

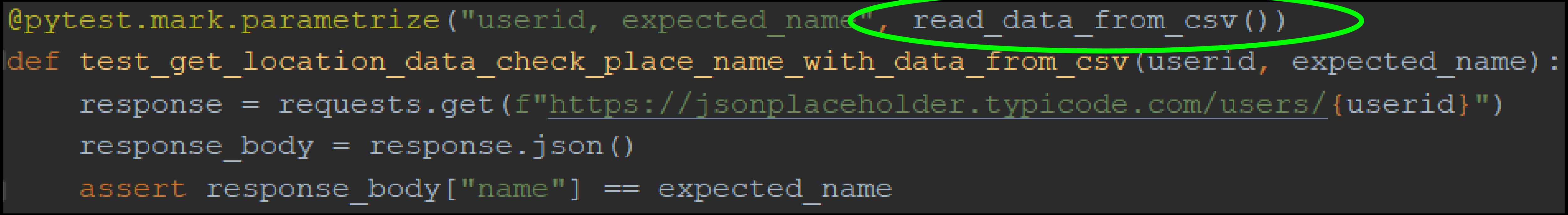


```python
def read_data_from_csv():
    test_data_users_from_csv = []
    with open("examples/test_data_users.csv", newline='') as csvfile:
        data = csv.reader(csvfile, delimiter=',')
        for row in data:
            test_data_users_from_csv.append(row)
    return test_data_users_from_csv
```

# Using .csv data to drive tests



```python
def read_data_from_csv():
    test_data_users_from_csv = []
    with open("examples/test_data_users.csv", newline='') as csvfile:
        data = csv.reader(csvfile, delimiter=',')
        for row in data:
            test_data_users_from_csv.append(row)
    return test_data_users_from_csv
```



```python
@pytest.mark.parametrize("userid, expected_name", read_data_from_csv())
def test_get_location_data_check_place_name_with_data_from_csv(userid, expected_name):
    response = requests.get(f"https://jsonplaceholder.typicode.com/users/{userid}")
    response_body = response.json()
    assert response_body["name"] == expected_name
```

## Now it’s your turn!

_ exercises > exercises_02.py

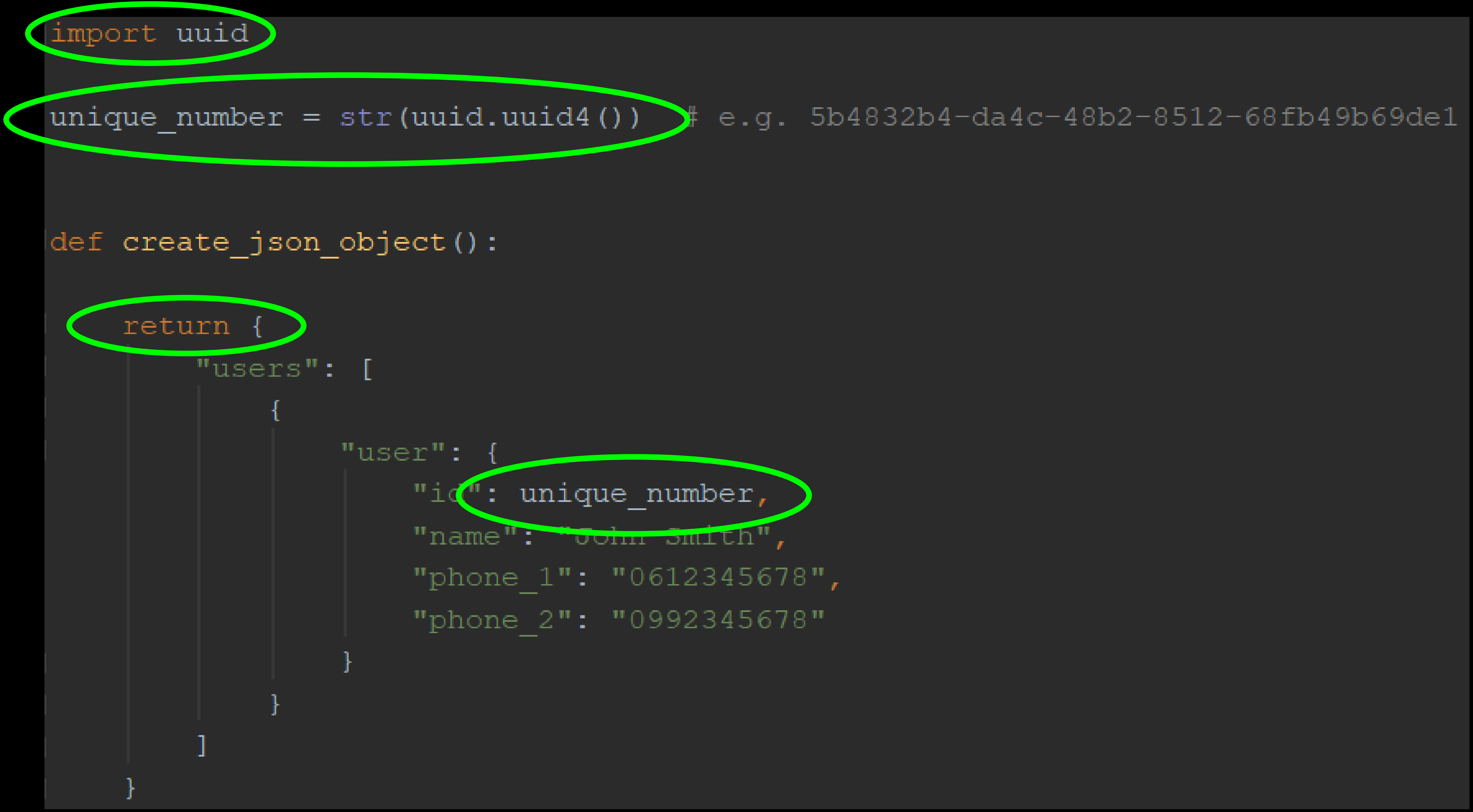
_ run your answers from the project root using

*pytest exercises\exercises_02.py*

_ examples are in examples > examples_02.py

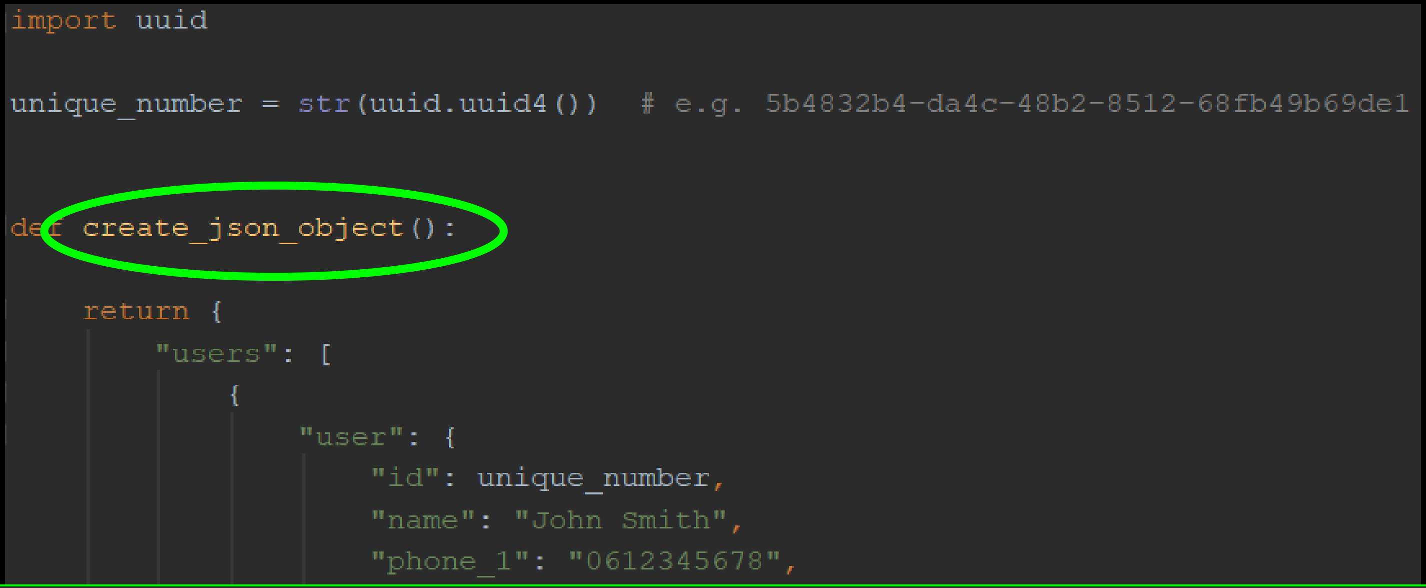
_ answers are in answers > answers_02.py

# Creating a JSON request body

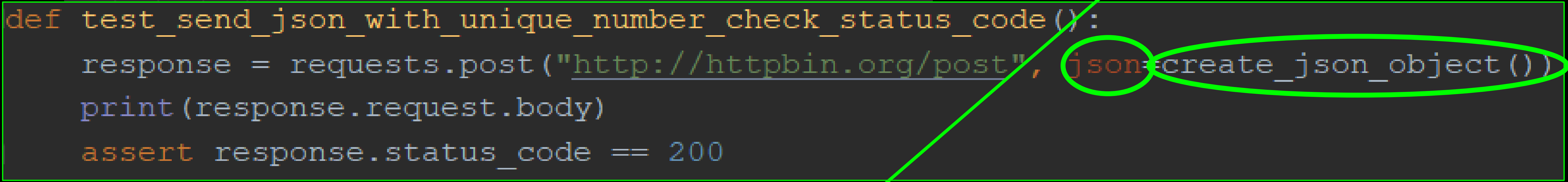


```
import uuid

unique_number = str(uuid.uuid4()) # e.g. 5b4832b4-da4c-48b2-8512-68fb49b69de1


def create_json_object():

    return {
        "users": [
            {
                "user": {
                    "id": unique_number,
                    "name": "John Smith",
                    "phone_1": "0612345678",
                    "phone_2": "0992345678"
                }
            }
        ]
    }
```

# POSTing a JSON request body

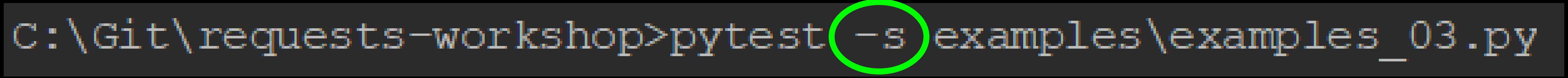


```python
import uuid

unique_number = str(uuid.uuid4())  # e.g. 5b4832b4-da4c-48b2-8512-68fb49b69de1


def create_json_object():

    return {
        "users": [
            {
                "user": {
                    "id": unique_number,
                    "name": "John Smith",
                    "phone_1": "0612345678",
```

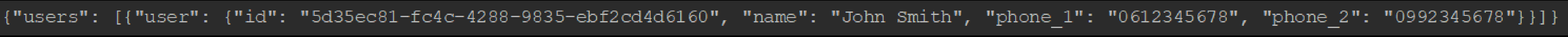


```python
def test_send_json_with_unique_number_check_status_code():
    response = requests.post("http://httpbin.org/post", json=create_json_object())
    print(response.request.body)
    assert response.status_code == 200
```

This disables output capturing by pytest, so all print() statements will be sent to the stdout / console



```
C:\Git\requests-workshop>pytest -s examples\examples_03.py
```



```
{"users": [{"user": {"id": "5d35ec81-fc4c-4288-9835-ebf2cd4d6160", "name": "John Smith", "phone_1": "0612345678", "phone_2": "0992345678"}}]}
```

## Now it’s your turn!

_ exercises > exercises_03.py

_ run your answers from the project root using

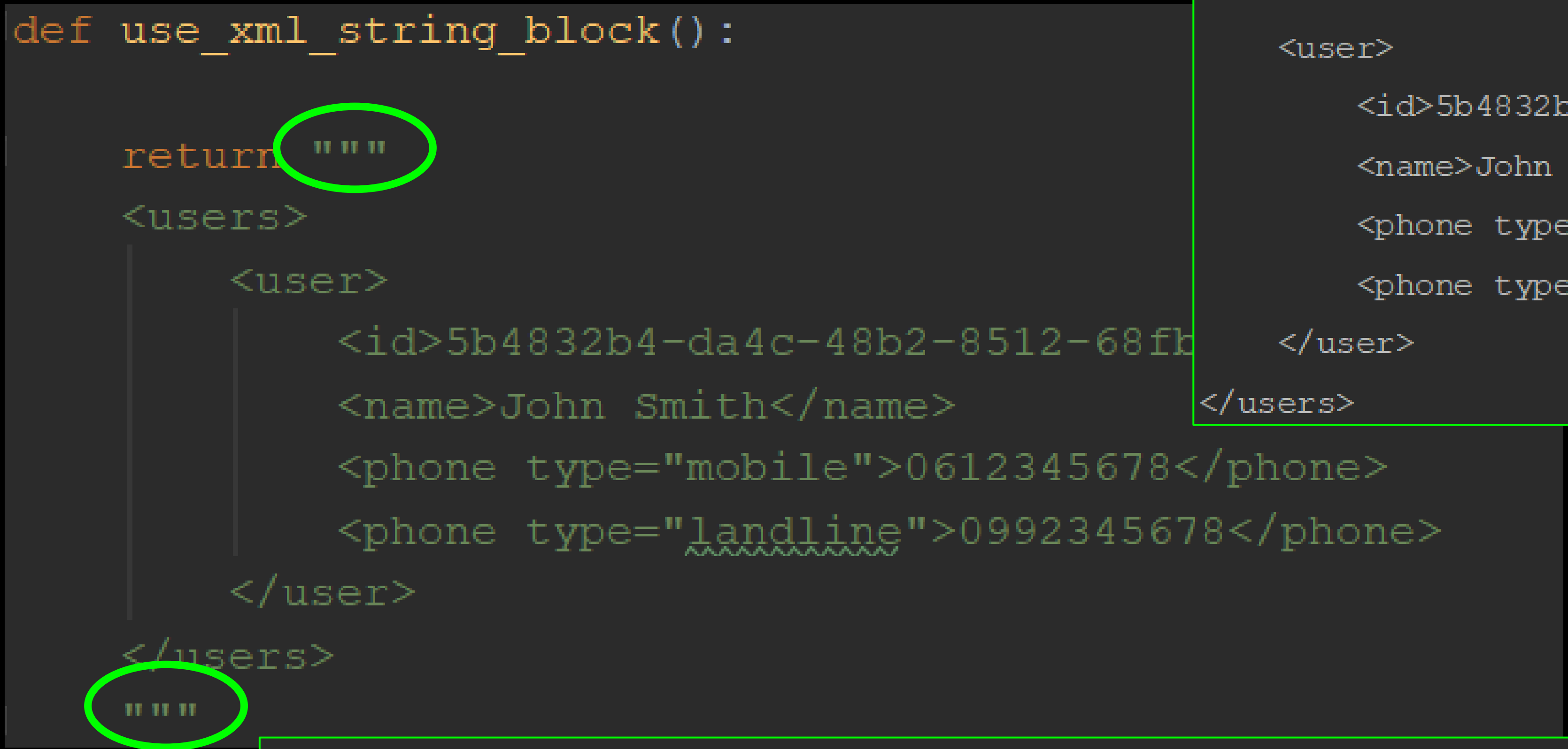
*pytest exercises\exercises_03.py*

_ examples are in examples > examples_03.py

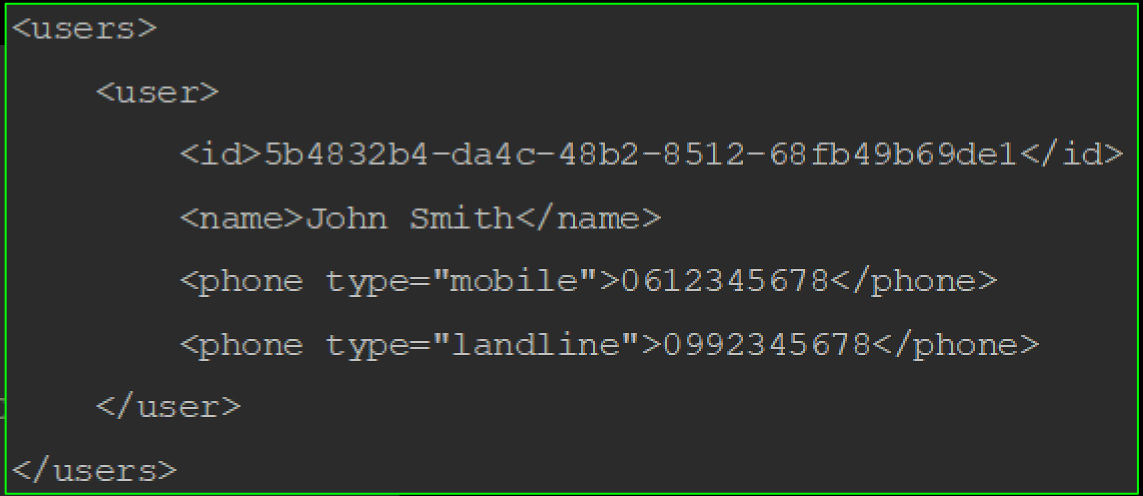
_ you will need to Google some things yourself

_ answers are in answers > answers_03.py

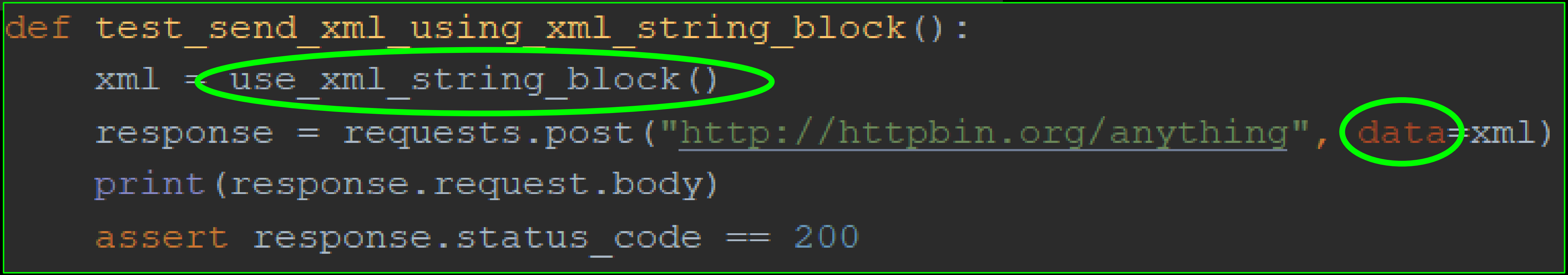
# Create XML request body using a docstring



```python
def use_xml_string_block():

    return """
    <users>
        <user>
            <id>5b4832b4-da4c-48b2-8512-68fb
            <name>John Smith</name>
            <phone type="mobile">0612345678</phone>
            <phone type="landline">0992345678</phone>
        </user>
    </users>
    """
```

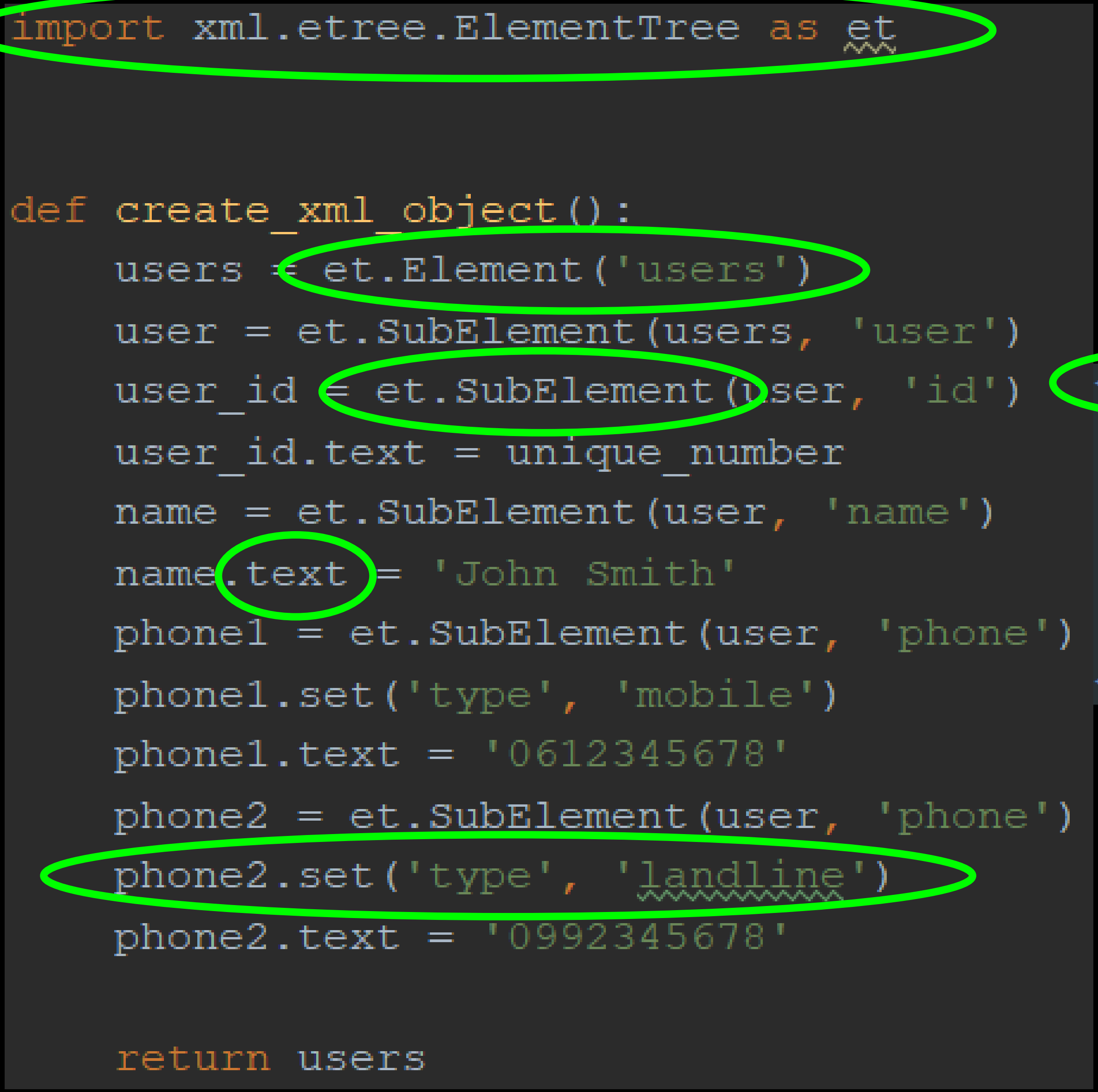


```
<users>
    <user>
        <id>5b4832b4-da4c-48b2-8512-68fb49b69de1</id>
        <name>John Smith</name>
        <phone type="mobile">0612345678</phone>
        <phone type="landline">0992345678</phone>
    </user>
</users>
```

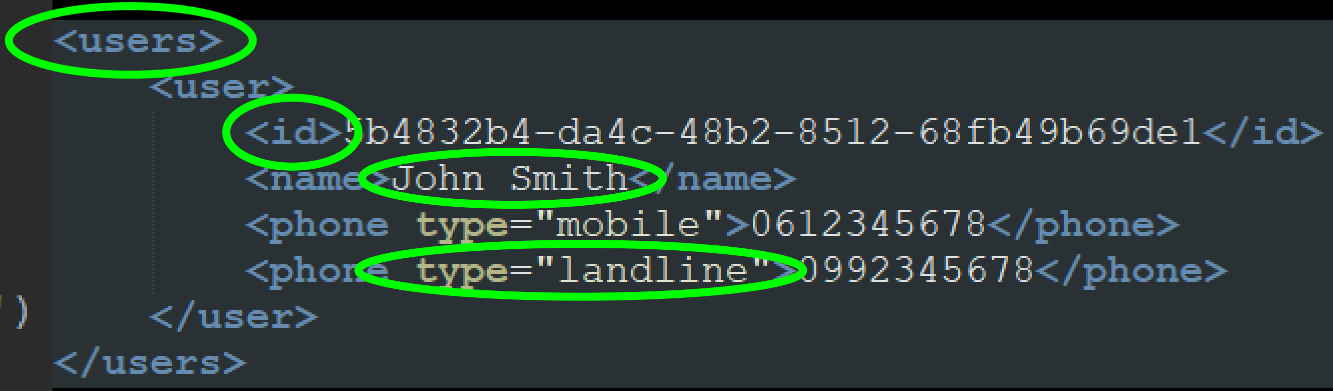


```python
def test_send_xml_using_xml_string_block():
    xml = use_xml_string_block()
    response = requests.post("http://httpbin.org/anything", data=xml)
    print(response.request.body)
    assert response.status_code == 200
```

# Create XML request body using ElementTree

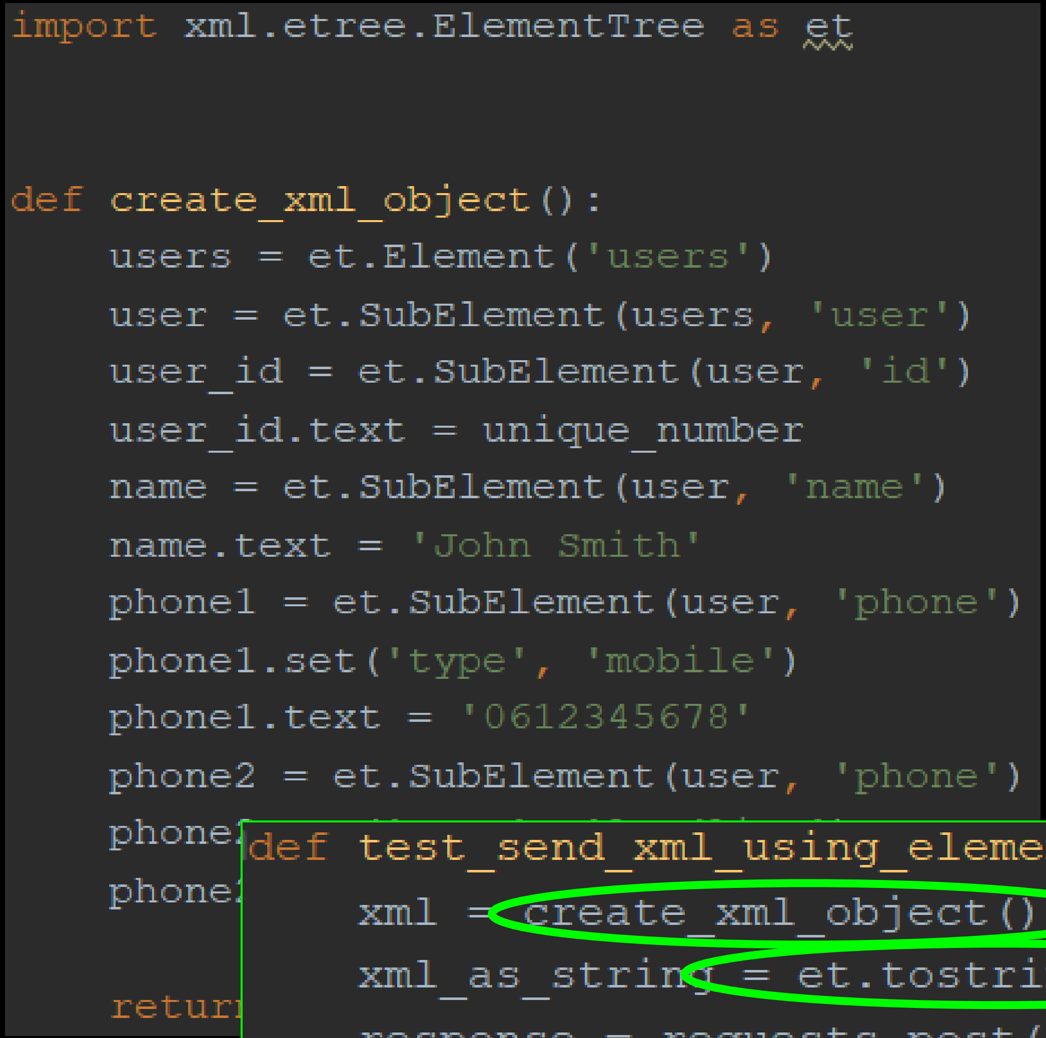


```python
import xml.etree.ElementTree as et


def create_xml_object():
    users = et.Element('users')
    user = et.SubElement(users, 'user')
    user_id = et.SubElement(user, 'id')
    user_id.text = unique_number
    name = et.SubElement(user, 'name')
    name.text = 'John Smith'
    phone1 = et.SubElement(user, 'phone')
    phone1.set('type', 'mobile')
    phone1.text = '0612345678'
    phone2 = et.SubElement(user, 'phone')
    phone2.set('type', 'landline')
    phone2.text = '0992345678'

    return users
```

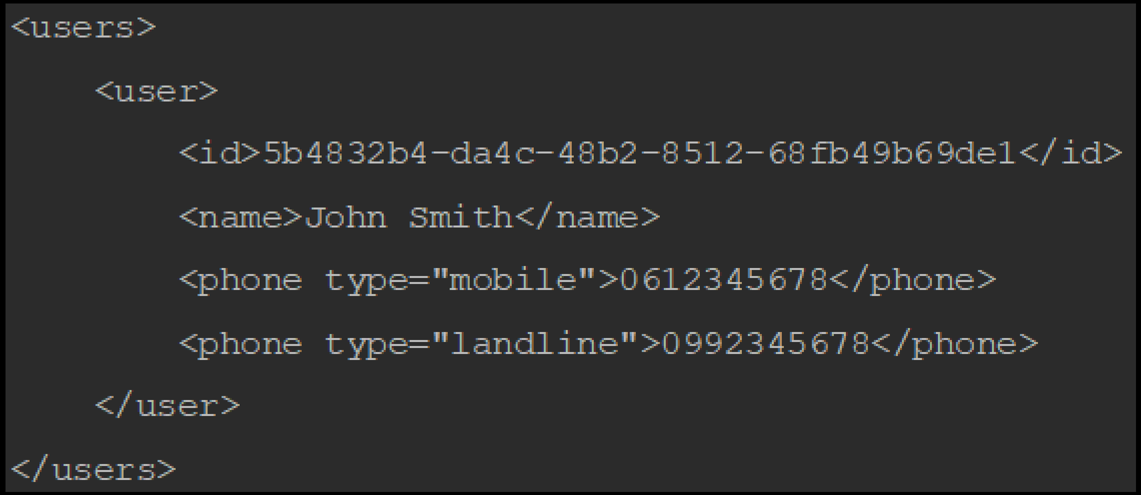


```xml
<users>
    <user>
        <id>5b4832b4-da4c-48b2-8512-68fb49b69de1</id>
        <name>John Smith</name>
        <phone type="mobile">0612345678</phone>
        <phone type="landline">0992345678</phone>
    </user>
</users>
```

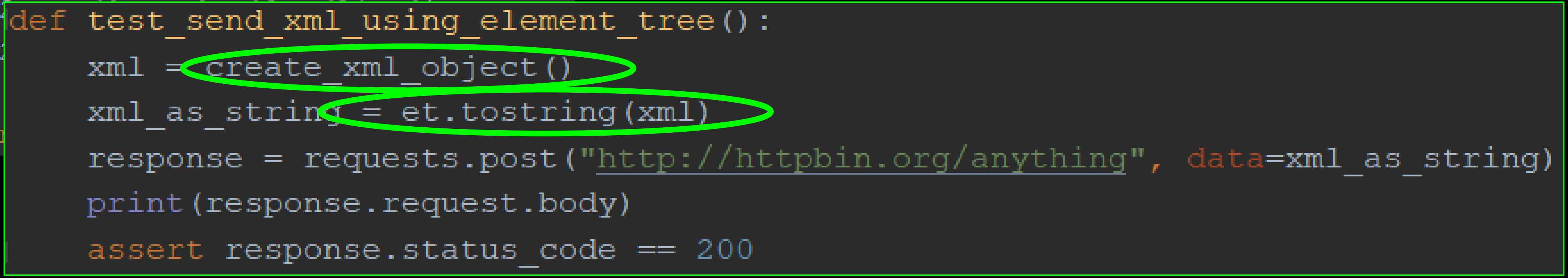
# Send XML created using ElementTree



```
import xml.etree.ElementTree as et


def create_xml_object():
    users = et.Element('users')
    user = et.SubElement(users, 'user')
    user_id = et.SubElement(user, 'id')
    user_id.text = unique_number
    name = et.SubElement(user, 'name')
    name.text = 'John Smith'
    phone1 = et.SubElement(user, 'phone')
    phone1.set('type', 'mobile')
    phone1.text = '0612345678'
    phone2 = et.SubElement(user, 'phone')
    phone
    phone

    retur
```



```
<users>
    <user>
        <id>5b4832b4-da4c-48b2-8512-68fb49b69de1</id>
        <name>John Smith</name>
        <phone type="mobile">0612345678</phone>
        <phone type="landline">0992345678</phone>
    </user>
</users>
```



```
def test_send_xml_using_element_tree():
    xml = create_xml_object()
    xml_as_string = et.tostring(xml)
    response = requests.post("http://httpbin.org/anything", data=xml_as_string)
    print(response.request.body)
    assert response.status_code == 200
```

## Now it's your turn!

_ exercises > exercises_04.py

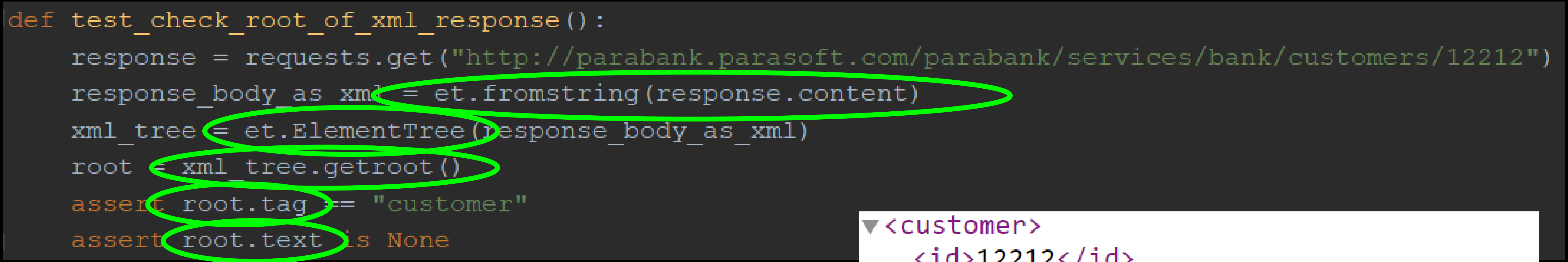
_ run your answers from the project root using

*pytest exercises\exercises_04.py*

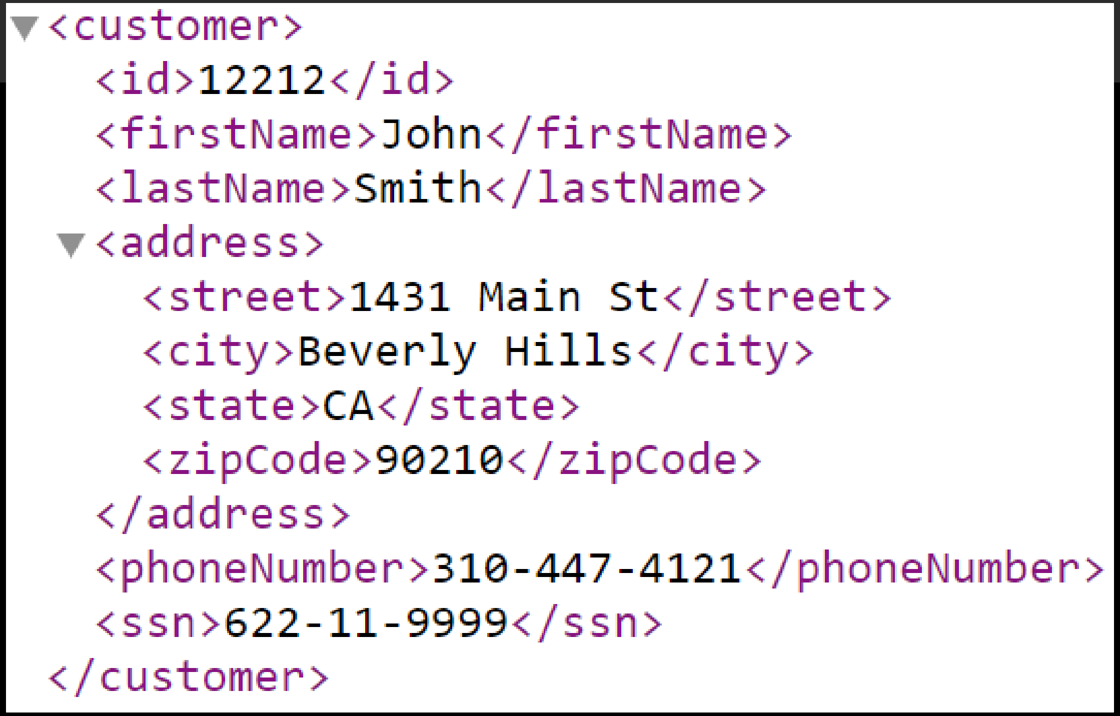
_ examples are in examples > examples_04.py

_ answers are in answers > answers_04.py

# Checking response XML - root element

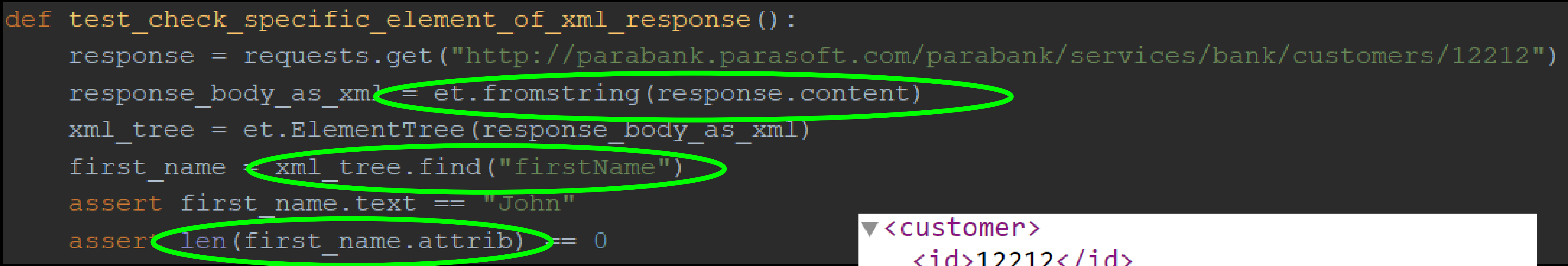


```
def test_check_root_of_xml_response():
    response = requests.get("http://parabank.parasoft.com/parabank/services/bank/customers/12212")
    response_body_as_xml = et.fromstring(response.content)
    xml_tree = et.ElementTree(response_body_as_xml)
    root = xml_tree.getroot()
    assert root.tag == "customer"
    assert root.text is None
```

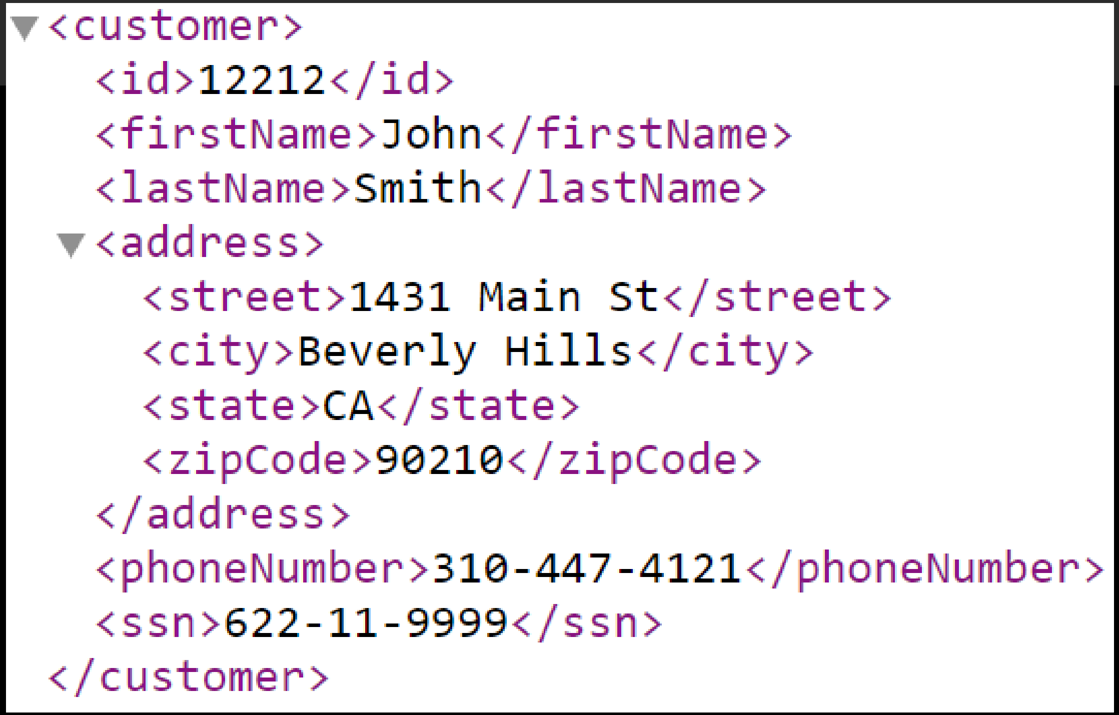


```
▼<customer>
   <id>12212</id>
   <firstName>John</firstName>
   <lastName>Smith</lastName>
 ▼ <address>
     <street>1431 Main St</street>
     <city>Beverly Hills</city>
     <state>CA</state>
     <zipCode>90210</zipCode>
   </address>
   <phoneNumber>310-447-4121</phoneNumber>
   <ssn>622-11-9999</ssn>
 </customer>
```

# Checking response XML – find an element using *find()*

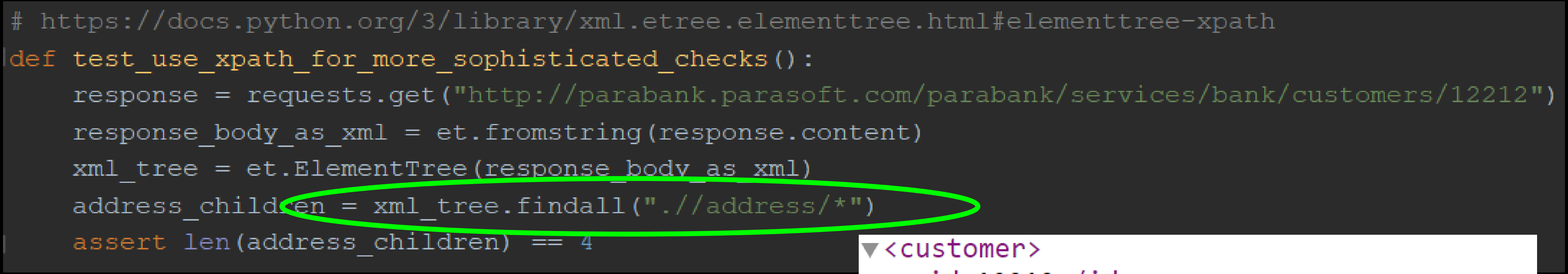


```
def test_check_specific_element_of_xml_response():
    response = requests.get("http://parabank.parasoft.com/parabank/services/bank/customers/12212")
    response_body_as_xml = et.fromstring(response.content)
    xml_tree = et.ElementTree(response_body_as_xml)
    first_name = xml_tree.find("firstName")
    assert first_name.text == "John"
    assert len(first_name.attrib) == 0
```

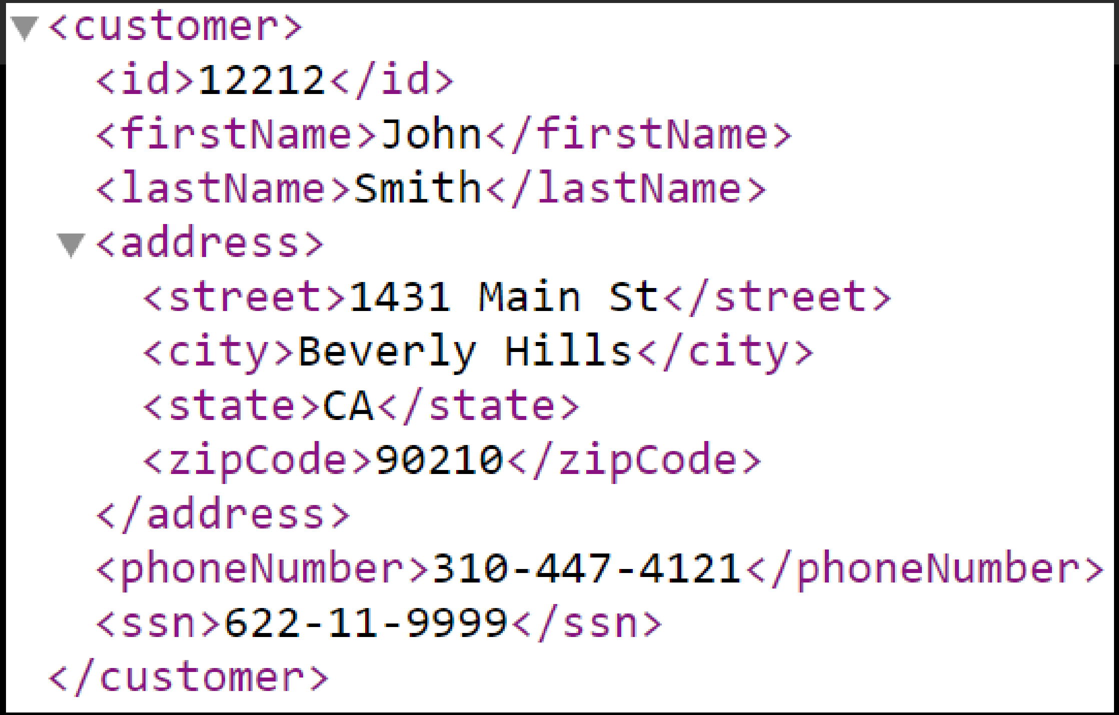


```
▼<customer>
   <id>12212</id>
   <firstName>John</firstName>
   <lastName>Smith</lastName>
 ▼<address>
     <street>1431 Main St</street>
     <city>Beverly Hills</city>
     <state>CA</state>
     <zipCode>90210</zipCode>
   </address>
   <phoneNumber>310-447-4121</phoneNumber>
   <ssn>622-11-9999</ssn>
 </customer>
```

# Checking response XML – find all elements using *findall()*



```python
# https://docs.python.org/3/library/xml.etree.elementtree.html#elementtree-xpath
def test_use_xpath_for_more_sophisticated_checks():
    response = requests.get("http://parabank.parasoft.com/parabank/services/bank/customers/12212")
    response_body_as_xml = et.fromstring(response.content)
    xml_tree = et.ElementTree(response_body_as_xml)
    address_children = xml_tree.findall(".//address/*")
    assert len(address_children) == 4
```



```xml
<customer>
  <id>12212</id>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <address>
    <street>1431 Main St</street>
    <city>Beverly Hills</city>
    <state>CA</state>
    <zipCode>90210</zipCode>
  </address>
  <phoneNumber>310-447-4121</phoneNumber>
  <ssn>622-11-9999</ssn>
</customer>
```

## Now it's your turn!

_ exercises > exercises_05.py

_ run your answers from the project root using

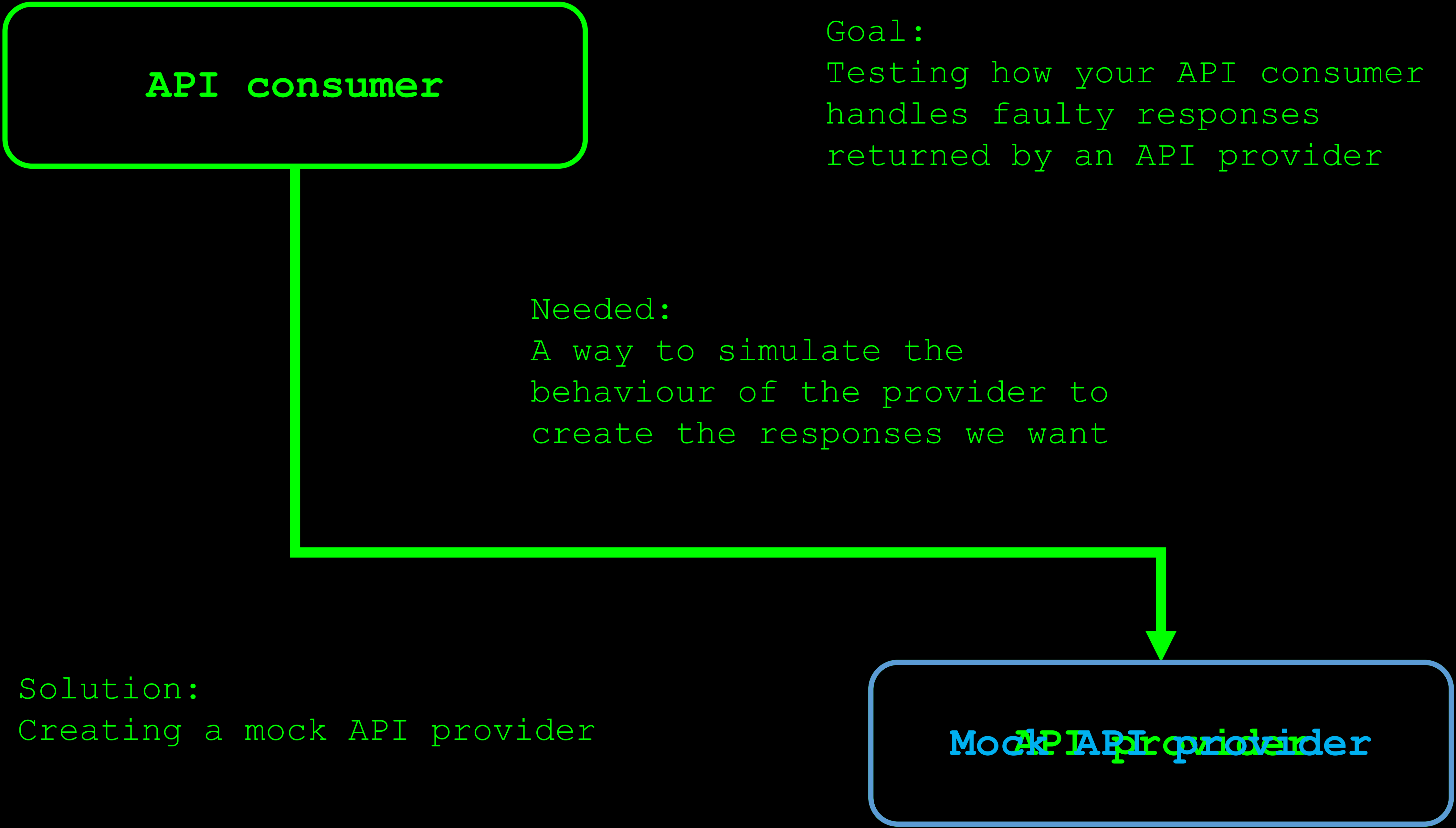
*pytest exercises\exercises_05.py*

_ examples are in examples > examples_05.py

_ you will need to Google some things yourself

_ answers are in answers > answers_05.py

## API mocking



Goal:
Testing how your API consumer handles faulty responses returned by an API provider

Needed:
A way to simulate the behaviour of the provider to create the responses we want

Solution:
Creating a mock API provider

Utility library for mocking requests

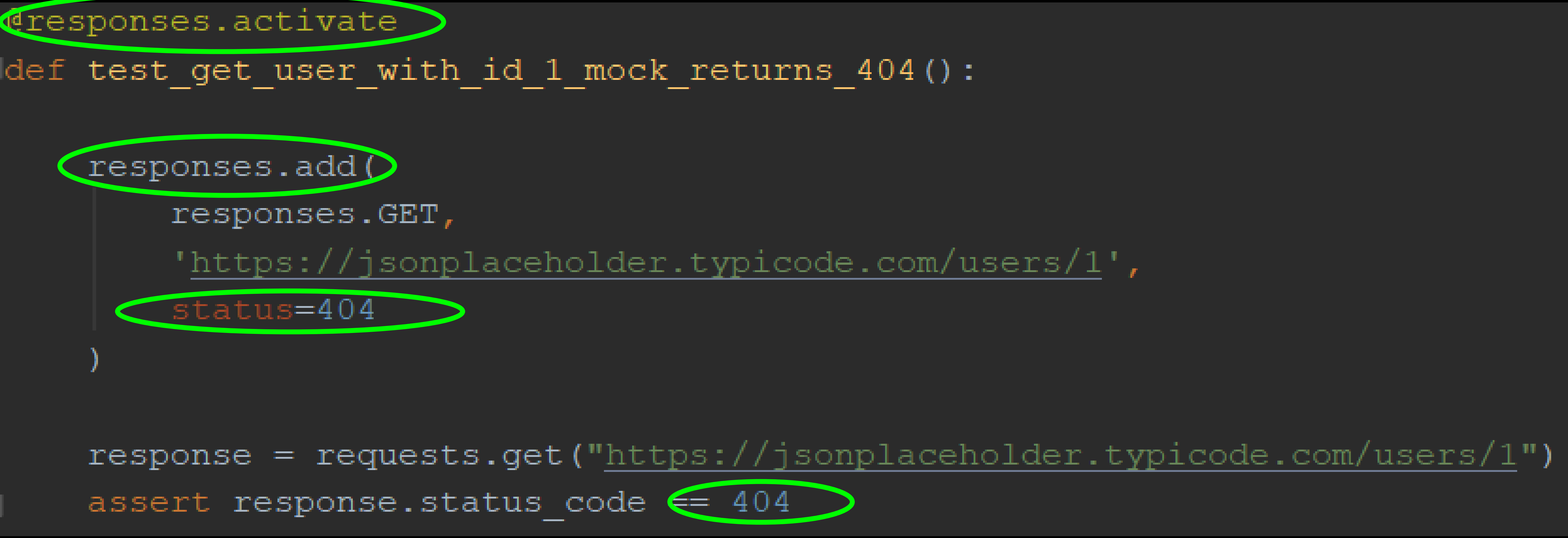
Register mock responses for HTTP calls

# responses

*pip install responses*

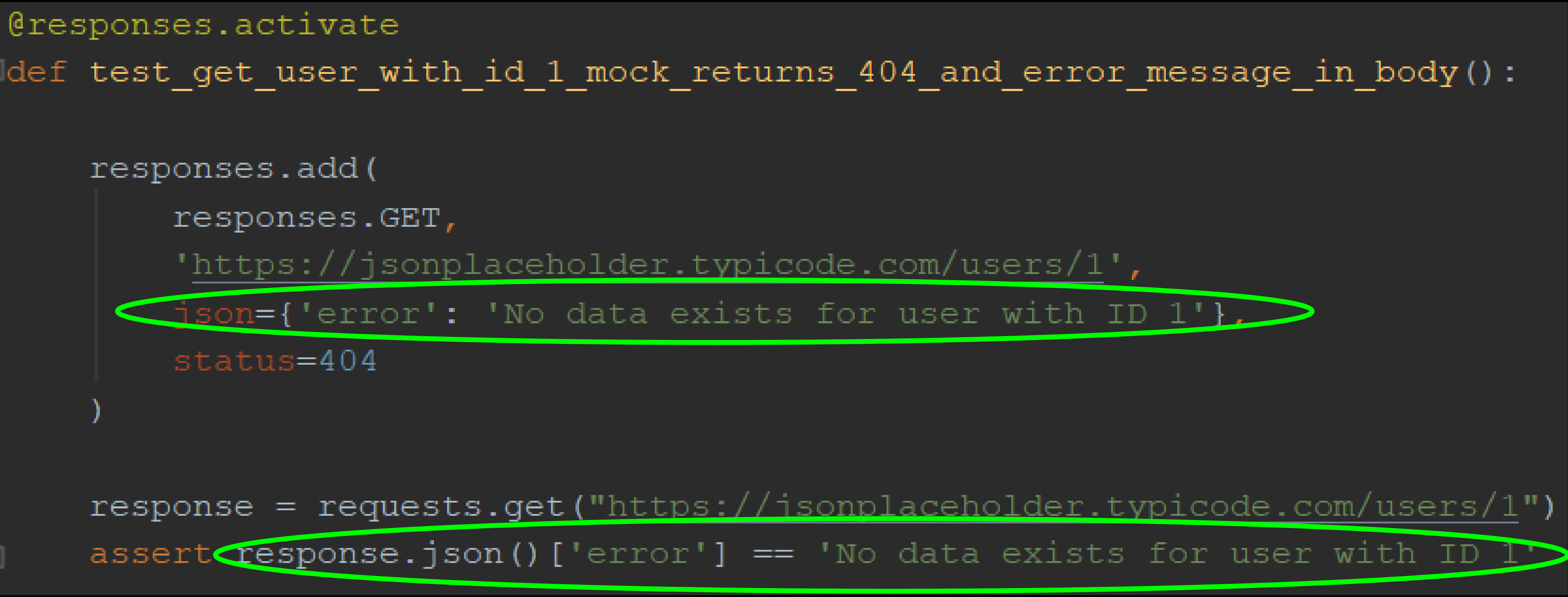
https://github.com/getsentry/responses

## Returning a different HTTP status code



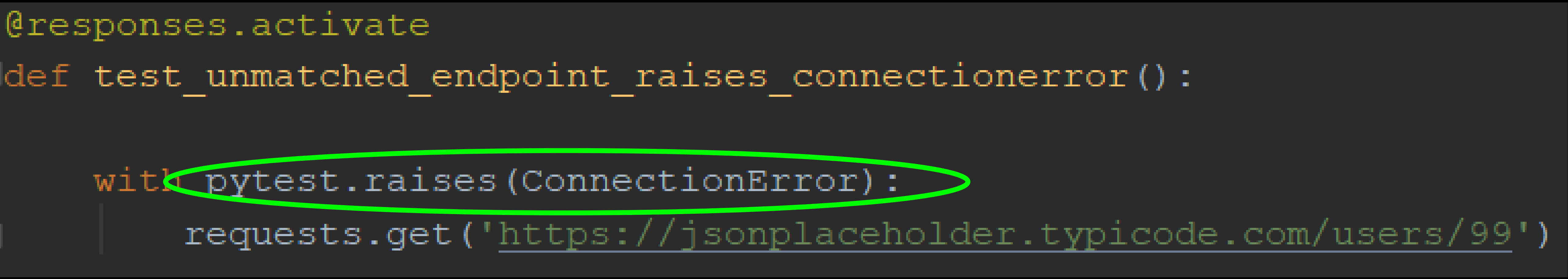
```
@responses.activate
def test_get_user_with_id_1_mock_returns_404():

    responses.add(
        responses.GET,
        'https://jsonplaceholder.typicode.com/users/1',
        status=404
    )

    response = requests.get("https://jsonplaceholder.typicode.com/users/1")
    assert response.status_code == 404
```

# Returning a specific response body



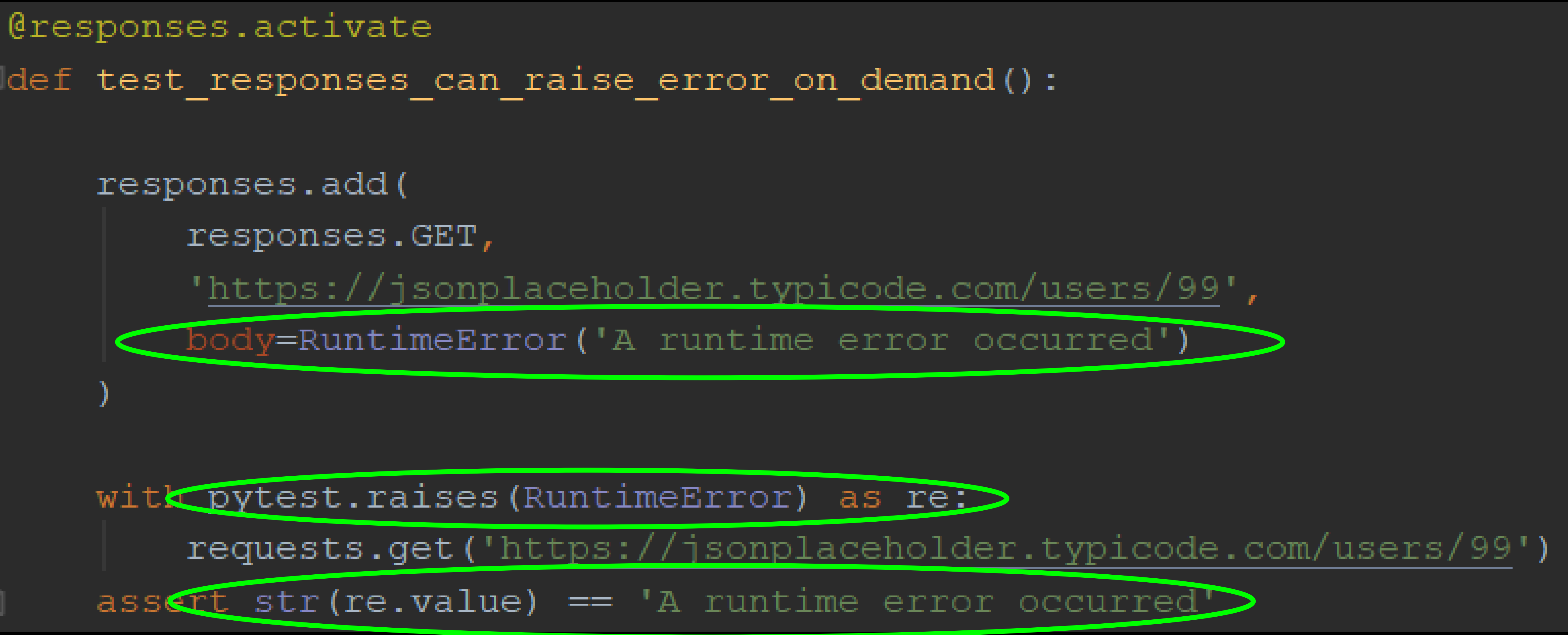
```
@responses.activate
def test_get_user_with_id_1_mock_returns_404_and_error_message_in_body():

    responses.add(
        responses.GET,
        'https://jsonplaceholder.typicode.com/users/1',
        json={'error': 'No data exists for user with ID 1'},
        status=404
    )

    response = requests.get("https://jsonplaceholder.typicode.com/users/1")
    assert response.json()['error'] == 'No data exists for user with ID 1'
```

# Unmatched requests return a ConnectionError



```
@responses.activate
def test_unmatched_endpoint_raises_connectionerror():

    with pytest.raises(ConnectionError):
        requests.get('https://jsonplaceholder.typicode.com/users/99')
```

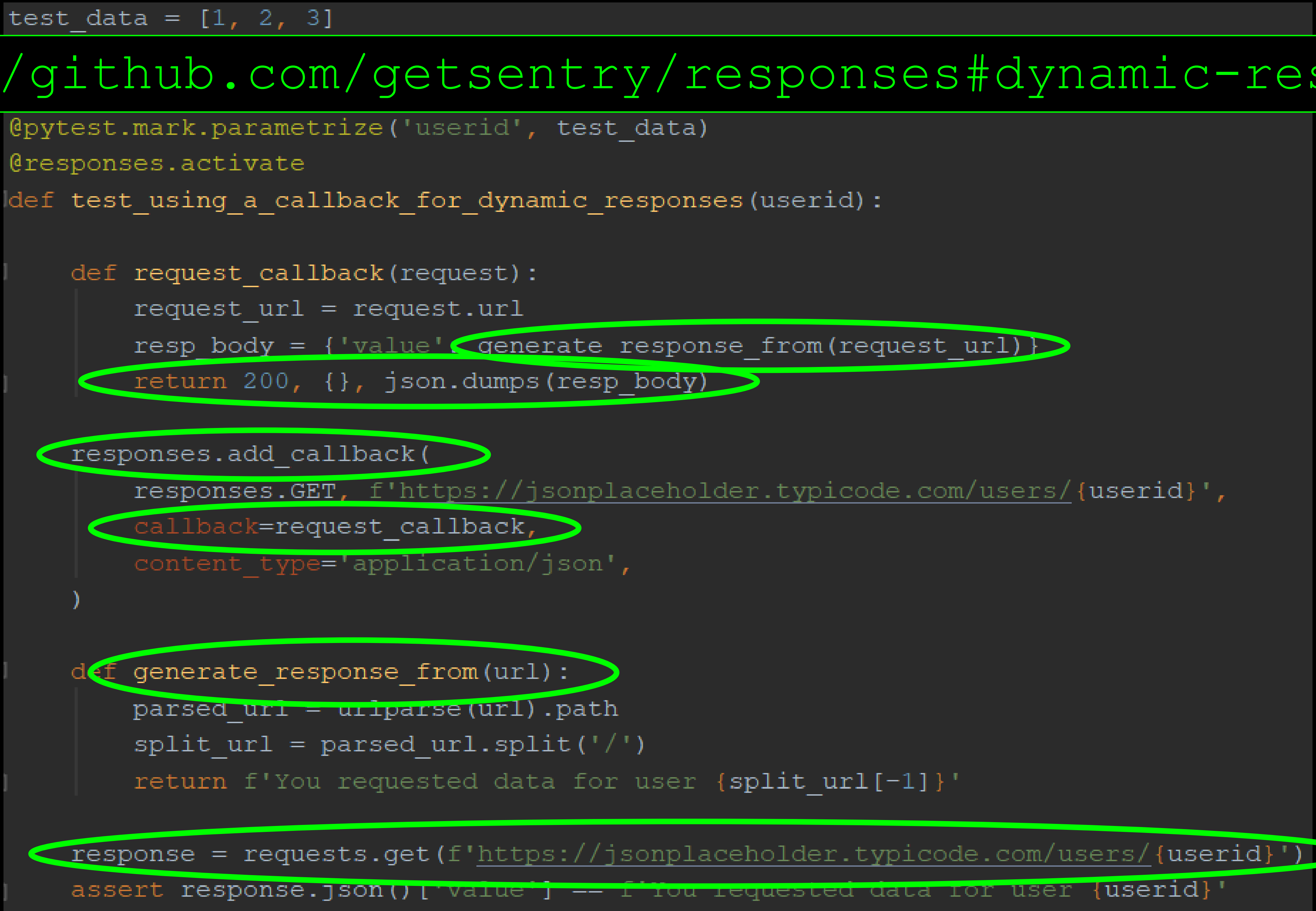
# Raise an error on an HTTP request



```python
@responses.activate
def test_responses_can_raise_error_on_demand():

    responses.add(
        responses.GET,
        'https://jsonplaceholder.typicode.com/users/99',
        body=RuntimeError('A runtime error occurred')
    )

    with pytest.raises(RuntimeError) as re:
        requests.get('https://jsonplaceholder.typicode.com/users/99')
    assert str(re.value) == 'A runtime error occurred'
```

# Create dynamic responses with callbacks

https://github.com/getsentry/responses#dynamic-responses



```python
test_data = [1, 2, 3]

@pytest.mark.parametrize('userid', test_data)
@responses.activate
def test_using_a_callback_for_dynamic_responses(userid):

    def request_callback(request):
        request_url = request.url
        resp_body = {'value': generate_response_from(request_url)}
        return 200, {}, json.dumps(resp_body)

    responses.add_callback(
        responses.GET, f'https://jsonplaceholder.typicode.com/users/{userid}',
        callback=request_callback,
        content_type='application/json',
    )

    def generate_response_from(url):
        parsed_url = urlparse(url).path
        split_url = parsed_url.split('/')
        return f'You requested data for user {split_url[-1]}'

    response = requests.get(f'https://jsonplaceholder.typicode.com/users/{userid}')
    assert response.json()['value'] == f'You requested data for user {userid}'
```

## Now it’s your turn!

_ exercises > exercises_06.py

_ run your answers from the project root using

*pytest exercises\exercises_06.py*

_ examples are in examples > examples_06.py

_ answers are in answers > answers_06.py

?