Exercise 1

First, install requests for your project using

```
pip install requests
```

Create a new Python module in your project. Use Python and requests to explore some of the different endpoints available at https://jsonplaceholder.typicode.com/

Use `print()` statements to print useful information to the console, such as

- Status code
- Response headers
- Response body

Examples:

- An HTTP GET to https://jsonplaceholder.typicode.com/users/1 retrieves data for a user with user ID 1.
- An HTTP GET to https://jsonplaceholder.typicode.com/posts/1/comments retrieves all comments associated with the post with ID 1.
- An HTTP POST to https://jsonplaceholder.typicode.com/posts creates a new post if sent the following payload:

```
{
    “userId”: 1,
    “title”: “Here goes the post title”,
    “body”: “Here goes the post body”
}
```

Here are some questions / test ideas to get you on your way:

- How do the different endpoints respond when you send invalid path parameter values?
- Are the HTTP response status codes that the API returns logical, both for ‘OK’ paths (the 2xx range) as well as consumer-side errors (the 4xx range)?
- How does the API respond if we POST invalid data?
- Does the API return the right data? Enough data? Not too much data?

Exercise 2

Time to start writing tests. If you haven't done already, install pytest using

```
pip install pytest
```

Create a new module in your project. Then, write tests that check that:

- An HTTP GET to `https://jsonplaceholder.typicode.com/users/1` returns an HTTP 200
- An HTTP GET to `https://jsonplaceholder.typicode.com/users/999` returns an HTTP 404
- An HTTP GET to `https://jsonplaceholder.typicode.com/users/1` returns a `Content-Type` header with value `application/json; charset=utf-8`
- An HTTP POST to `https://jsonplaceholder.typicode.com/posts` with the below payload returns an HTTP 201

```
{
    "userId": 1,
    "title": "Here goes the post title",
    "body": "Here goes the post body"
}
```

Run your tests from the command line using `pytest <your_module_name.py>`. Don't forget to add the `-s` flag if you have `print()` statements that you actually want to see..

Exercise 3

Time to start writing some checks for JSON response payloads.

Create a new module in your project (or append these tests to the current one).

Then, write tests that perform an HTTP GET to `https://jsonplaceholder.typicode.com/users/1` and check that:

- The value of the `name` element is equal to `Leanne Graham`
- The value of the `company` > `name` element is equal to `Romaguera-Crona`
- The value of the `street` element is equal to `Kulas Light`

Next write tests that perform an HTTP GET to `https://jsonplaceholder.typicode.com/users` and check that:

- The value of the `name` element for the second user is equal to `Ervin Howell`
- The value of the `company` > `name` element for the eighth user is equal to `Abernathy Group`

Run your tests to see if they work as expected.

Exercise 4

Let’s create a parameterized test.

First, create a new module in your project. Make sure to import requests again.

Then, create a test data structure (a list of tuples) that contains the following data records:

User id	Name	Company name
1	Leanne Graham	Romaguera-Crona
2	Ervin Howell	Deckow-Crist
3	Clementine Bauch	Romaguera-Jacobson

Using the `@pytest.mark.parametrize` marker, ‘feed’ this test data structure to the test to create a parameterized test. For all iterations (records in the test data structure):

- Use the user id as a path parameter
- Assert that the user name is equal to the specified expected value
- Assert that the user company name is equal to the specified expected value

Run your tests, check that it runs three iterations and that they all pass.

Then, add a fourth iteration (user id `9`, name `Glenna Reichert`, company name `Yost and Sons`) and run the test once more.

Exercise 5

Let's practice working with XML response payloads.

Create a new module and and the following import statements:

```
import requests

import xml.etree.ElementTree as et
```

Write tests that performs an HTTP GET to `https://parabank.parasoft.com/parabank/services/bank/customers/12212` and that check the following:

- The HTTP status code is equal to 200
- The response `Content-Type` header has value `application/xml`
- The response root element is called `customer`
- The response `phoneNumber` element has value `310-447-4131`
- **EXTRA**: The response `address` element has 4 child elements (I've not shown you how to do this yet, can you figure it out? Google is your friend…)

Exercise 6

As a final exercise, let’s have a look at GraphQL APIs.

Create a new module in your project and import requests.

Then, write a test that sends the following GraphQL query to https://api.spacex.land/graphql/

```
{
  company {
    ceo
    coo
    name
  }
}
```

and checks that:

- The response status code is 200
- The response Content-Type header is `application/json; charset=utf-8`
- The value of the `ceo` element (data > company > ceo) in the response body is `Elon Musk`
- The value of the `coo` element in the response body is `Gwynne Shotwell`
- The value of the `name` element in the response body is `SpaceX`

Next, create another test that sends the following query to the same endpoint:

```
query getRocketData($id: ID!)
{
    rocket(id: $id) {
        name
        country
    }
}
```

Pass in a query variable `id` with value `falcon1`.

Check that:

- The response status code is 200
- The value of the `name` element (data > rocket > name) is equal to `Falcon 1`
- The value of the `country` element is equal to `Republic of the Marshall Islands`