UNIVERSIDADE CATÓLICA PORTUGUESA

BRAGA

Behavior Analysis Technologies

Session 11

Introduction to APIs

Applied Data Science

2024/2025

What is an API?

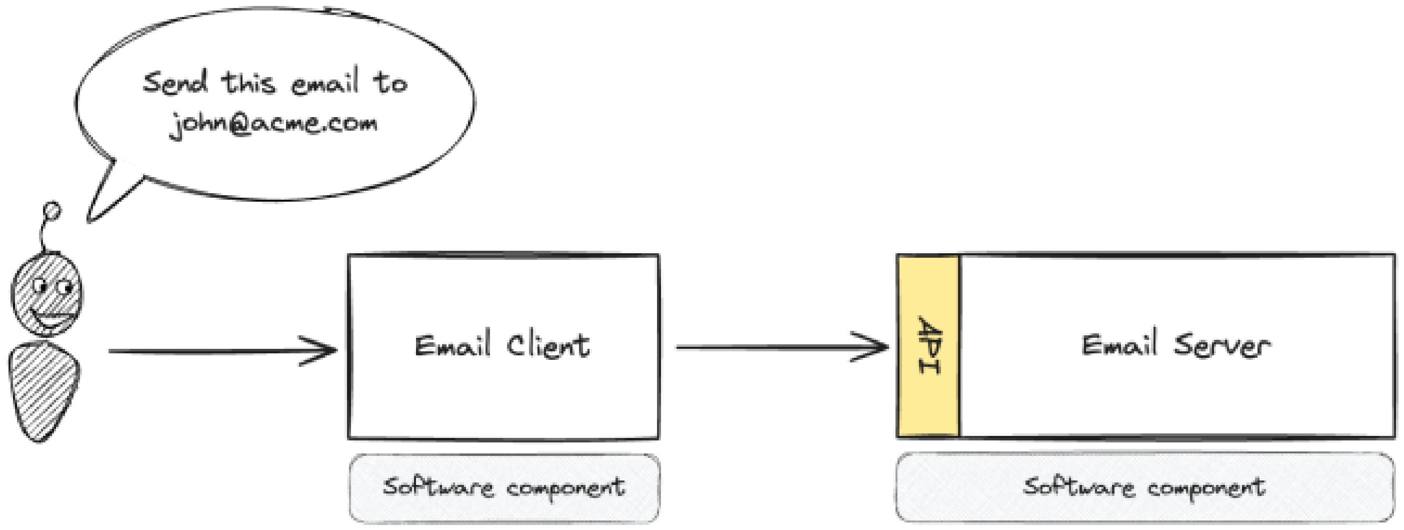
- API stands for **application programming interface**.

- **Interface**: allows a user to interact with a system.
  - **Graphical User Interface** (GUI): interact with a program using a point/click/type interface
  - **Command-Line Interface** (CLI): interact with a program via the command line

- **API**: interact with an existing program programmatically

What is an API?

- API serves as a communication layer, or interface, between systems
- Enables different systems to interact without needing to understand exactly what the others do

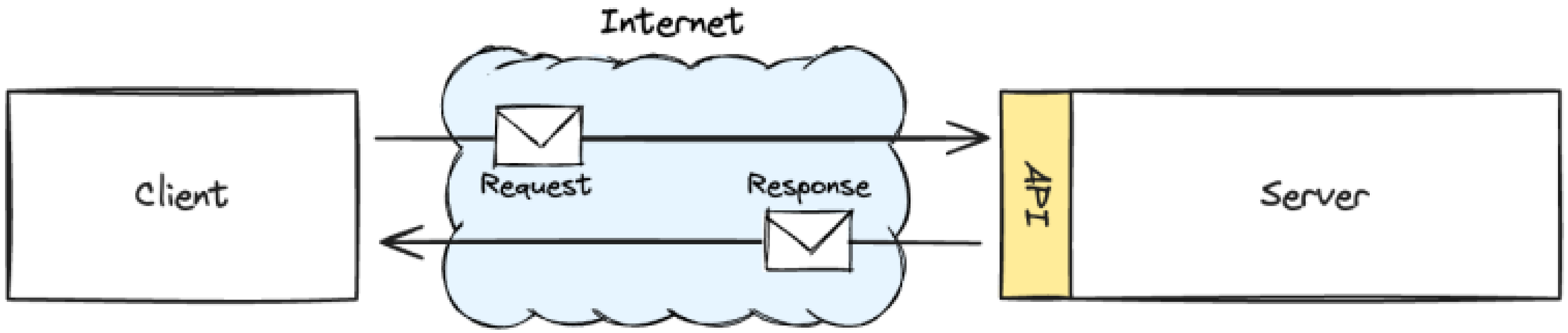


What is an API?

- APIs exist in various forms:
  - **Operating system APIs**: enables applications to interact with the underlying operating system (e.g., turning on camera/audio during a Zoom call);
  - **Web APIs:** Manage web actions like liking Instagram photos or fetching tweets;
- All APIs follow a similar process:
  - You make a **request** for data, and the API returns a **response**
  - Example: Opening Twitter or scrolling Instagram sends a request to the API, which responds with updated content
  - This process is called **calling an API**

Web APIs

- Gives you a way to ask for and receive data over the internet using **hyper-text transfer protocol** (HTTP)
- **Client** sends a **request** message to a **Server**
- **Server** returns a **response** message to the **Client**



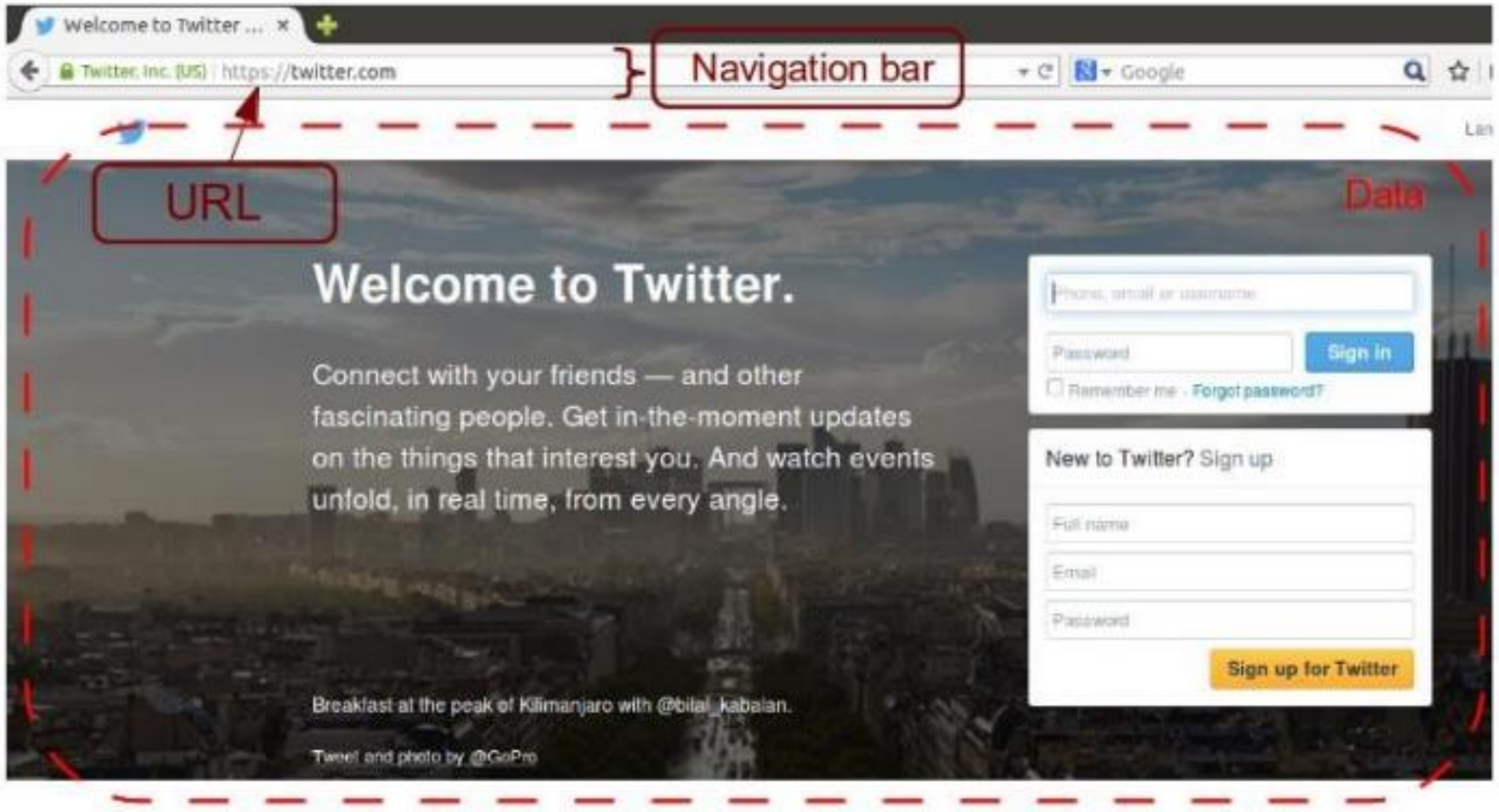
UNIVERSIDADE CATÓLICA PORTUGUESA
BRAGA

Types of Web APIs

- SOAP
  - Declined in popularity due to complexity and inflexibility compared to REST and GraphQL
  - Standardized method for software communication in the early 2000s
  - Enterprise applications
- REST
  - Most common type of API today
  - Web-based, allowing information exchange over the internet
  - When making a request, a REST API provides all available data in response
- GraphQL
  - Newer and less common but growing in popularity
  - Work similarly to REST but only return specific data requested
  - Allows you to specify the exact fields you need, reducing data load and improving speed

Asking for Data

- When you type a URL into the navigation bar of your browser, you are requesting data for that webpage


Welcome to Twitter ...
Twitter, Inc. (US) https://twitter.com
Navigation bar
Google
URL
Data
Welcome to Twitter.
Connect with your friends — and other fascinating people. Get in-the-moment updates on the things that interest you. And watch events unfold, in real time, from every angle.
Phone, email or username
Password
Sign in
Remember me · Forgot password?
New to Twitter? Sign up
Full name
Email
Password
Sign up for Twitter
Breakfast at the peak of Kilimanjaro with @bilal_kabalan.
Tweet and photo by @GoPro

Asking for Data with URLs

- We will be asking for data with URLs (Universal Resource Locator)

- e.g., If you want to see a specific YouTube video, you ask YouTube for the video by encoding its ID in the URL:

https://www.youtube.com/watch?v=1wnE4vF9CQ4

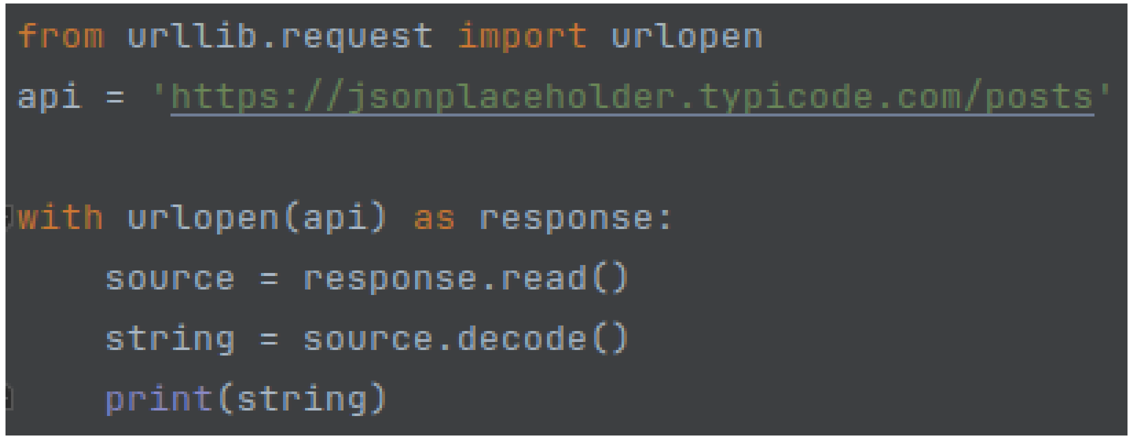
HTTP GET URL --> server returns 200 OK and data

Receiving Data

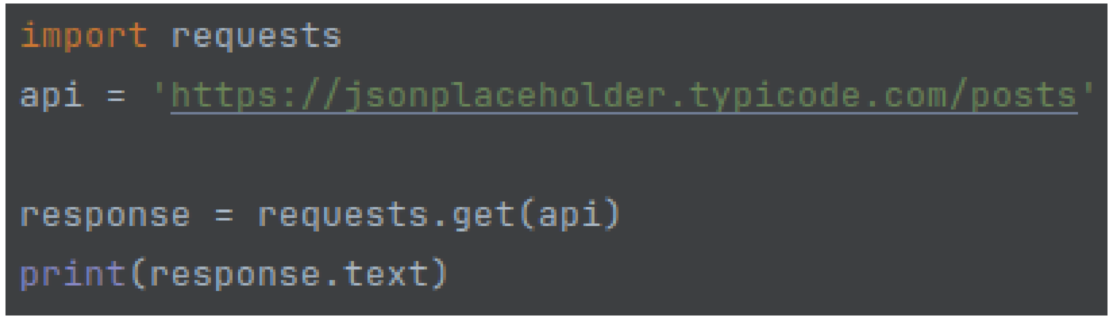
- When requesting to view a webpage in your browser, the information is sent back to you as HTML
- Your browser parses and displays the page based on the HTML it receives!
- APIs often return data in JSON format, as it is easy to parse and display information

Working with APIs in Python

- urllib
  - Bundled with Python
  - Powerful but not very developer-friendly
- requests
  - Many built-in features
  - Easier to use

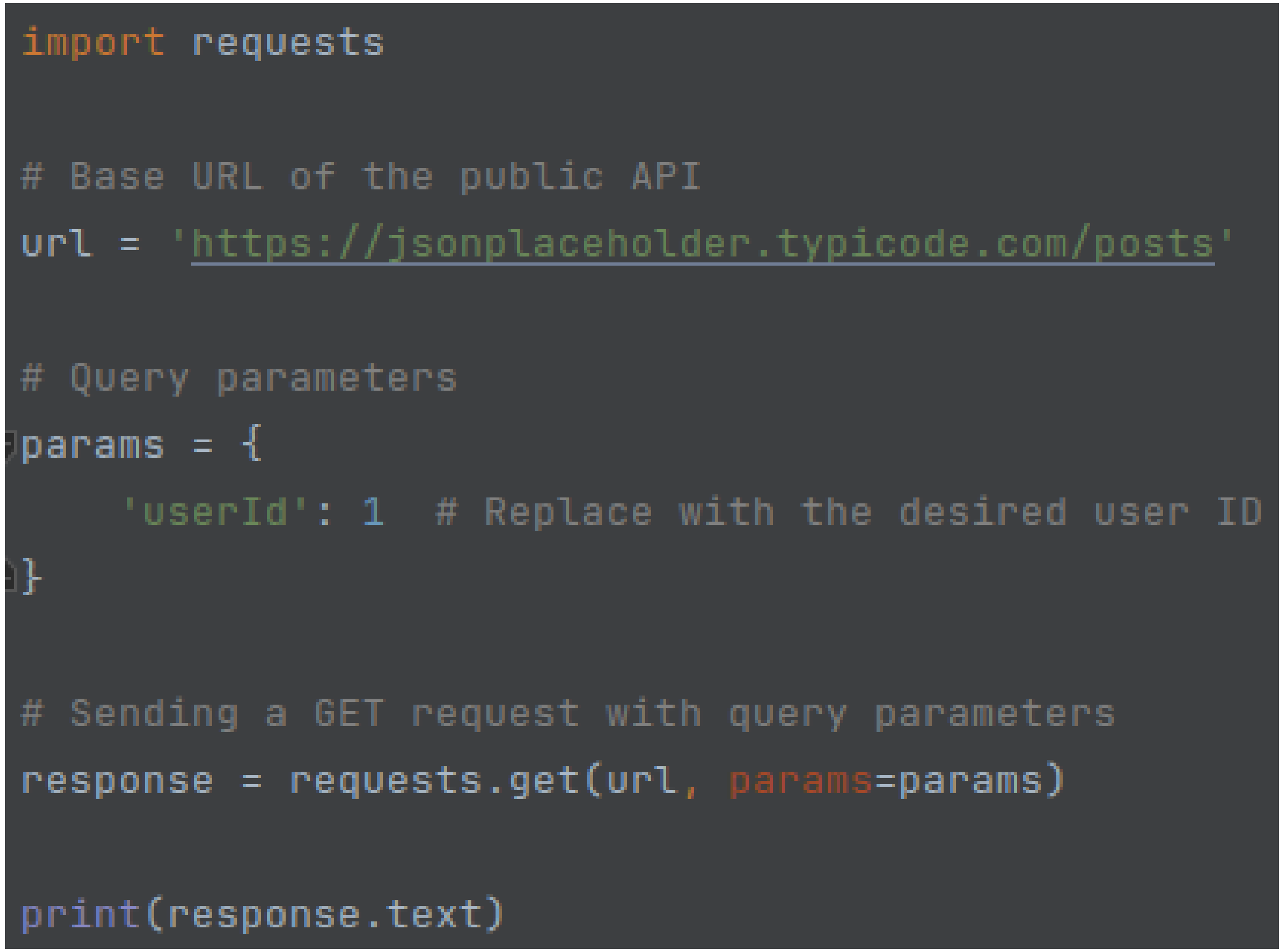


```python
from urllib.request import urlopen
api = 'https://jsonplaceholder.typicode.com/posts'

with urlopen(api) as response:
    source = response.read()
    string = source.decode()
    print(string)
```



```python
import requests
api = 'https://jsonplaceholder.typicode.com/posts'

response = requests.get(api)
print(response.text)
```

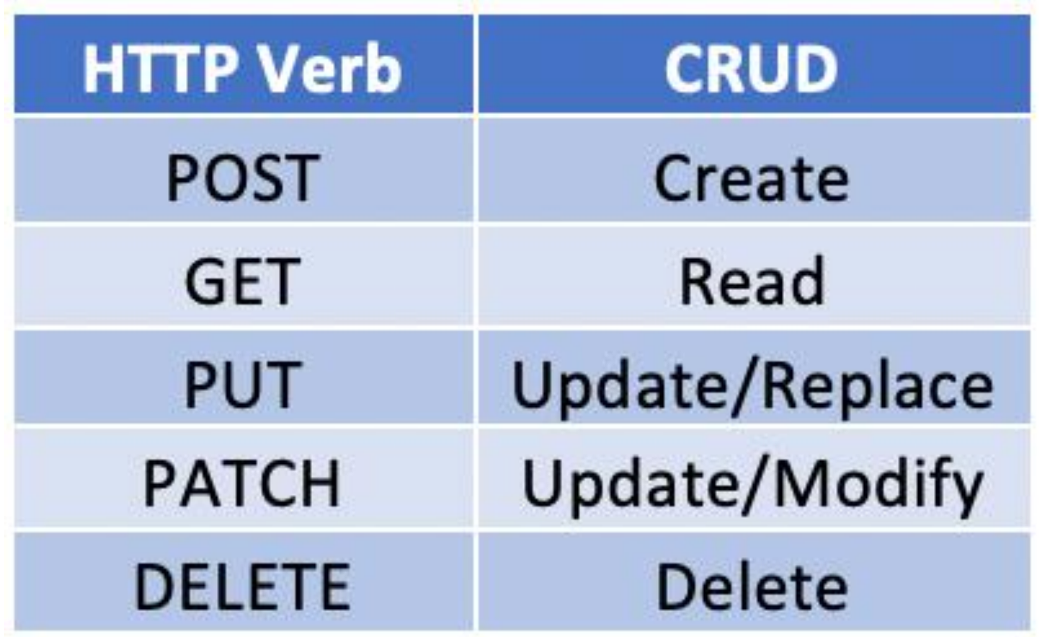
Adding Query Parameters with requests



```python
import requests

# Base URL of the public API
url = 'https://jsonplaceholder.typicode.com/posts'

# Query parameters
params = {
    'userId': 1  # Replace with the desired user ID
}

# Sending a GET request with query parameters
response = requests.get(url, params=params)

print(response.text)
```

HTTP Verbs

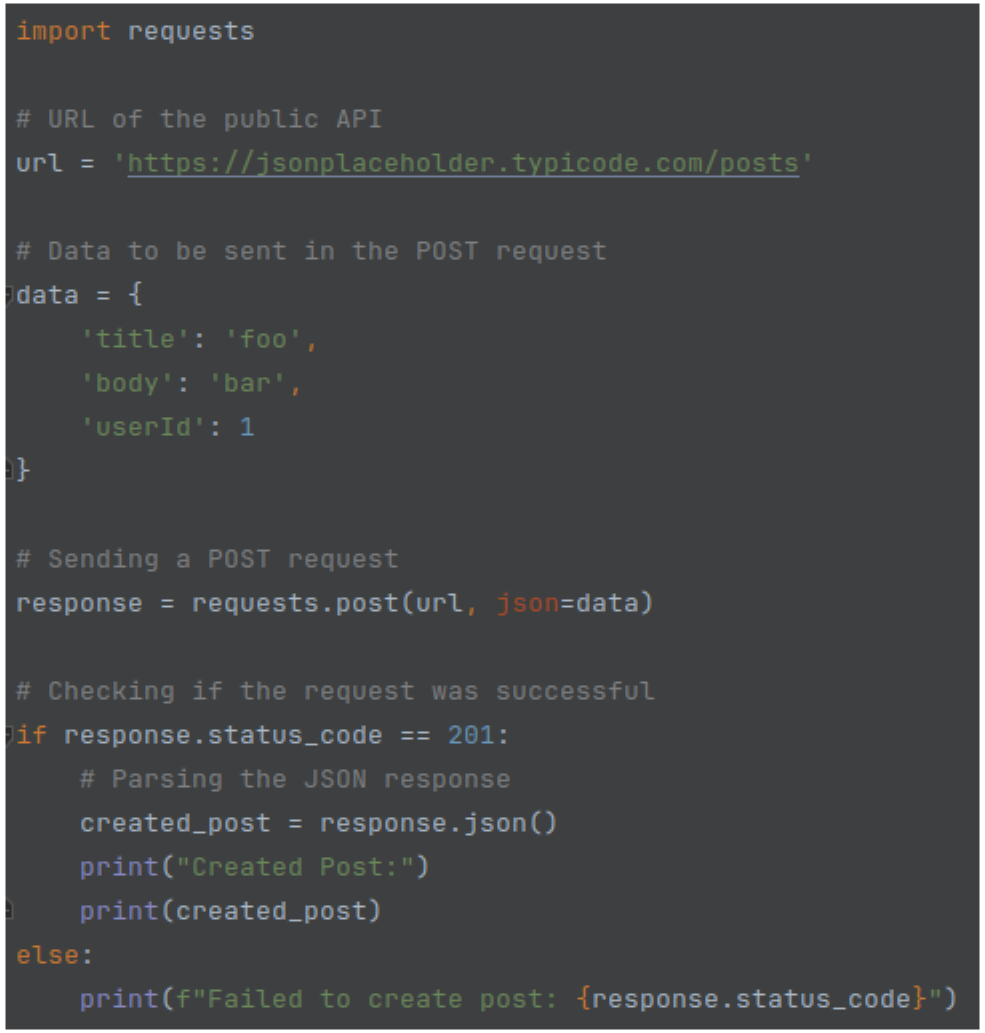
- Actions



HTTP Verb	CRUD
POST	Create
GET	Read
PUT	Update/Replace
PATCH	Update/Modify
DELETE	Delete

HTTP Verbs

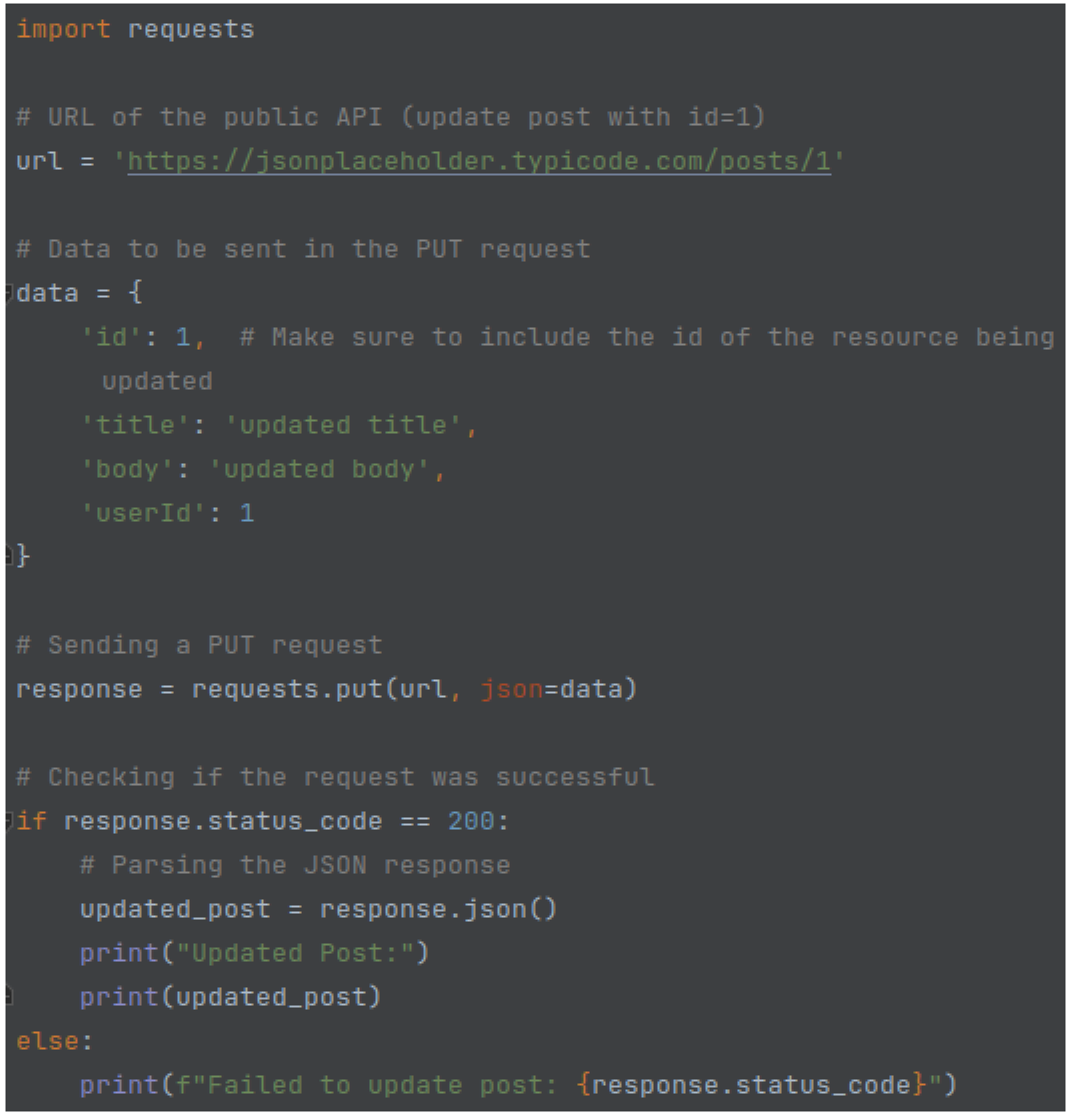
- POST



```python
import requests

# URL of the public API
url = 'https://jsonplaceholder.typicode.com/posts'

# Data to be sent in the POST request
data = {
    'title': 'foo',
    'body': 'bar',
    'userId': 1
}

# Sending a POST request
response = requests.post(url, json=data)

# Checking if the request was successful
if response.status_code == 201:
    # Parsing the JSON response
    created_post = response.json()
    print("Created Post:")
    print(created_post)
else:
    print(f"Failed to create post: {response.status_code}")
```

HTTP Verbs

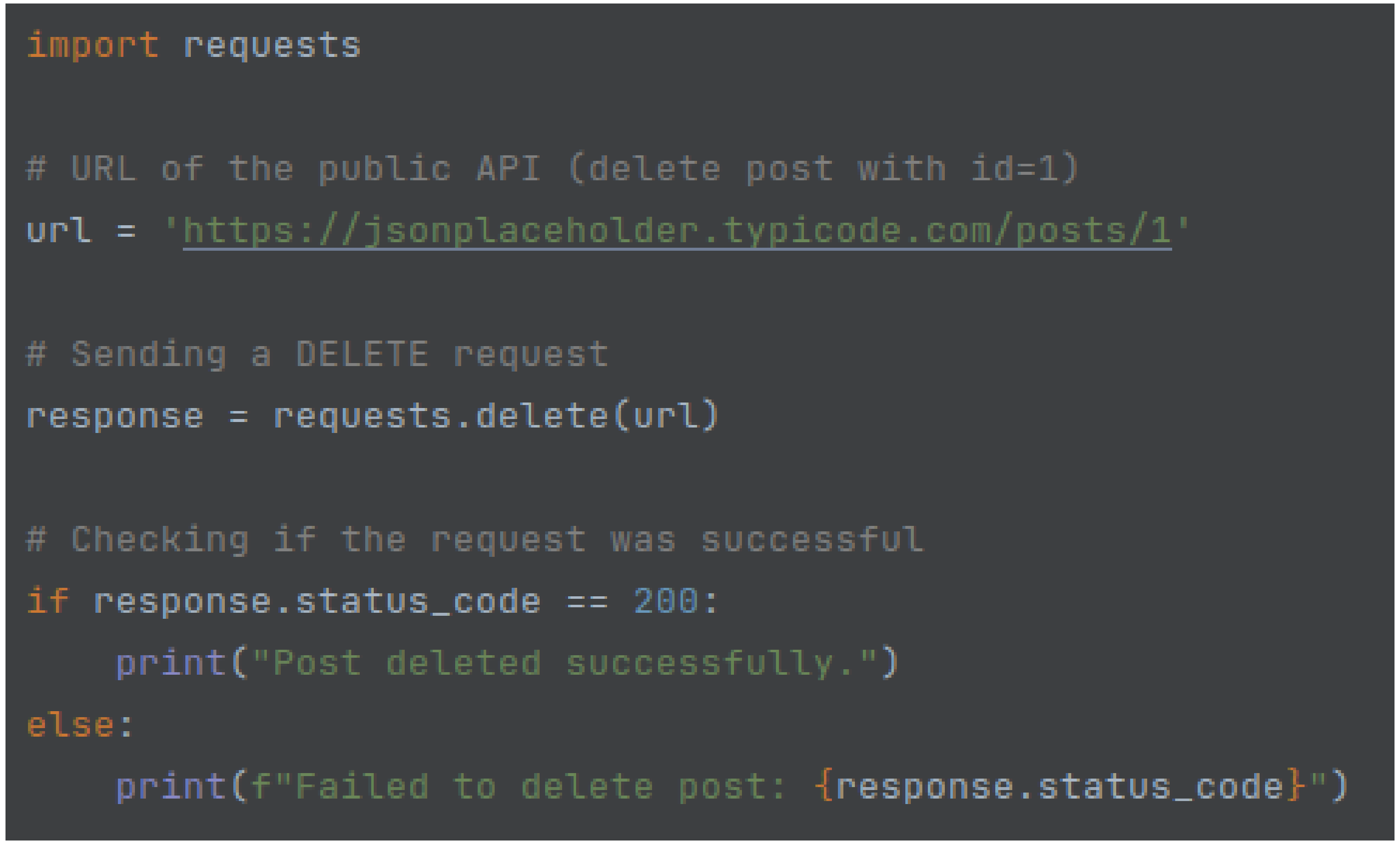
- PUT



```python
import requests

# URL of the public API (update post with id=1)
url = 'https://jsonplaceholder.typicode.com/posts/1'

# Data to be sent in the PUT request
data = {
    'id': 1,  # Make sure to include the id of the resource being
     updated
    'title': 'updated title',
    'body': 'updated body',
    'userId': 1
}

# Sending a PUT request
response = requests.put(url, json=data)

# Checking if the request was successful
if response.status_code == 200:
    # Parsing the JSON response
    updated_post = response.json()
    print("Updated Post:")
    print(updated_post)
else:
    print(f"Failed to update post: {response.status_code}")
```

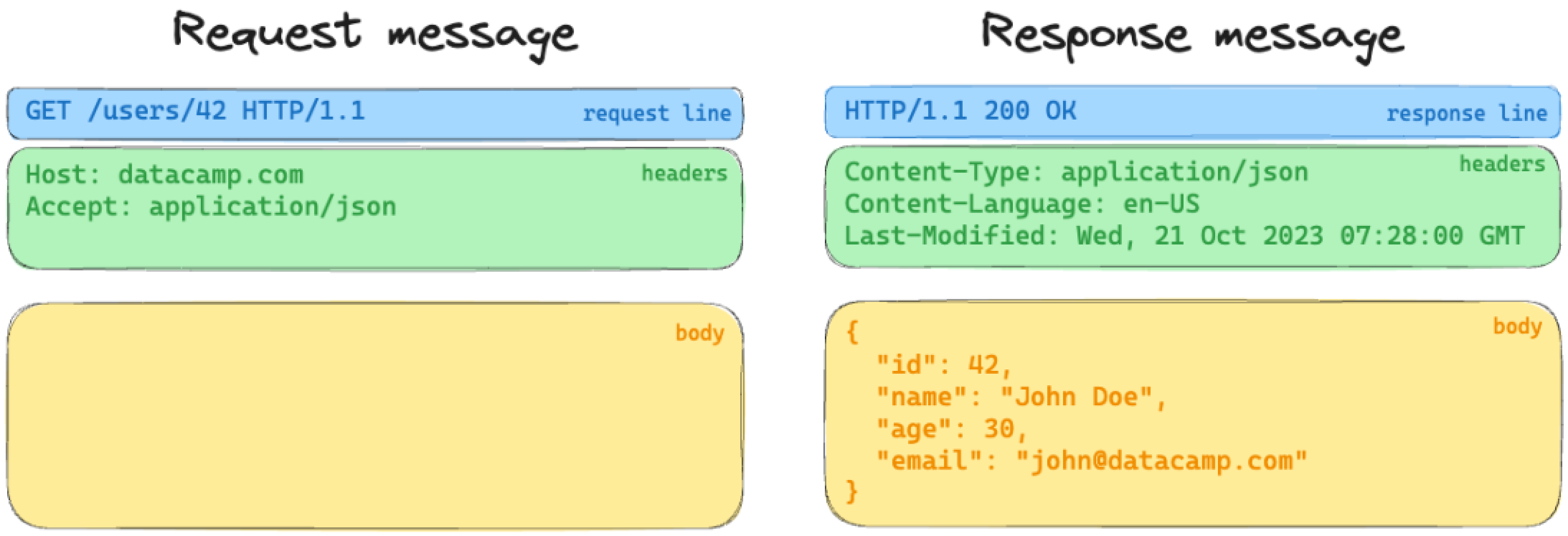
HTTP Verbs

- DELETE



```python
import requests

# URL of the public API (delete post with id=1)
url = 'https://jsonplaceholder.typicode.com/posts/1'

# Sending a DELETE request
response = requests.delete(url)

# Checking if the request was successful
if response.status_code == 200:
    print("Post deleted successfully.")
else:
    print(f"Failed to delete post: {response.status_code}")
```

Request and Response Message Anatomy



```
GET /users/42 HTTP/1.1
```

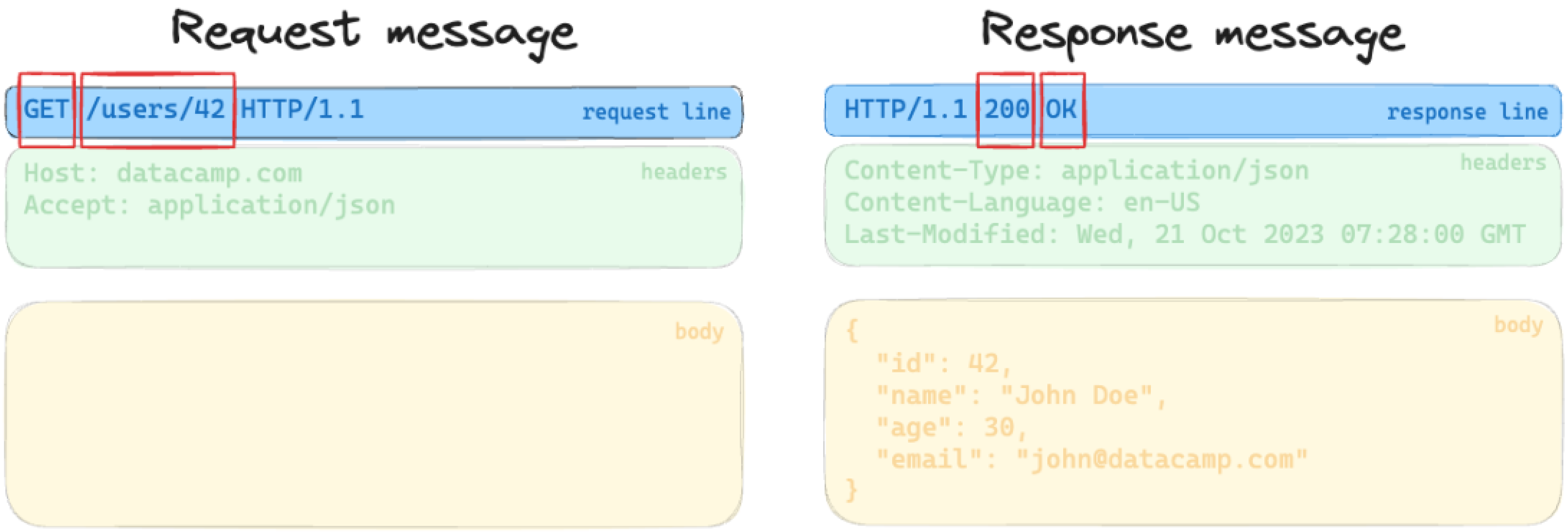
```
Host: datacamp.com
Accept: application/json
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
Content-Language: en-US
Last-Modified: Wed, 21 Oct 2023 07:28:00 GMT
```

```
{
  "id": 42,
  "name": "John Doe",
  "age": 30,
  "email": "john@datacamp.com"
}
```

Request and Response Message Anatomy

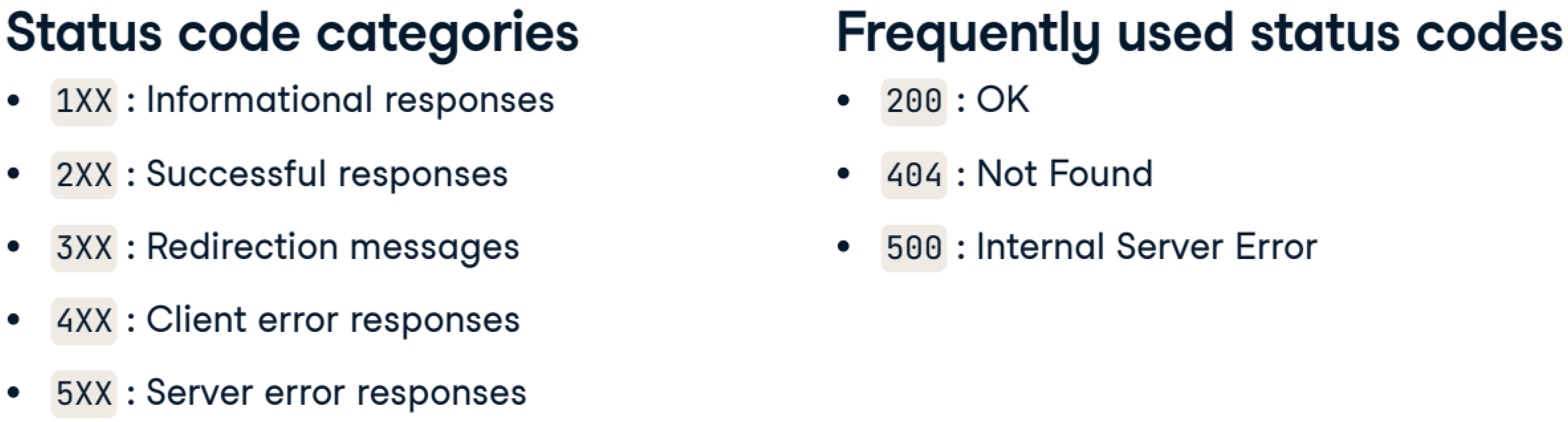


```
GET /users/42 HTTP/1.1
Host: datacamp.com
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Language: en-US
Last-Modified: Wed, 21 Oct 2023 07:28:00 GMT

{
  "id": 42,
  "name": "John Doe",
  "age": 30,
  "email": "john@datacamp.com"
}
```

- A server will always include a numeric status code in the response message

Status Codes



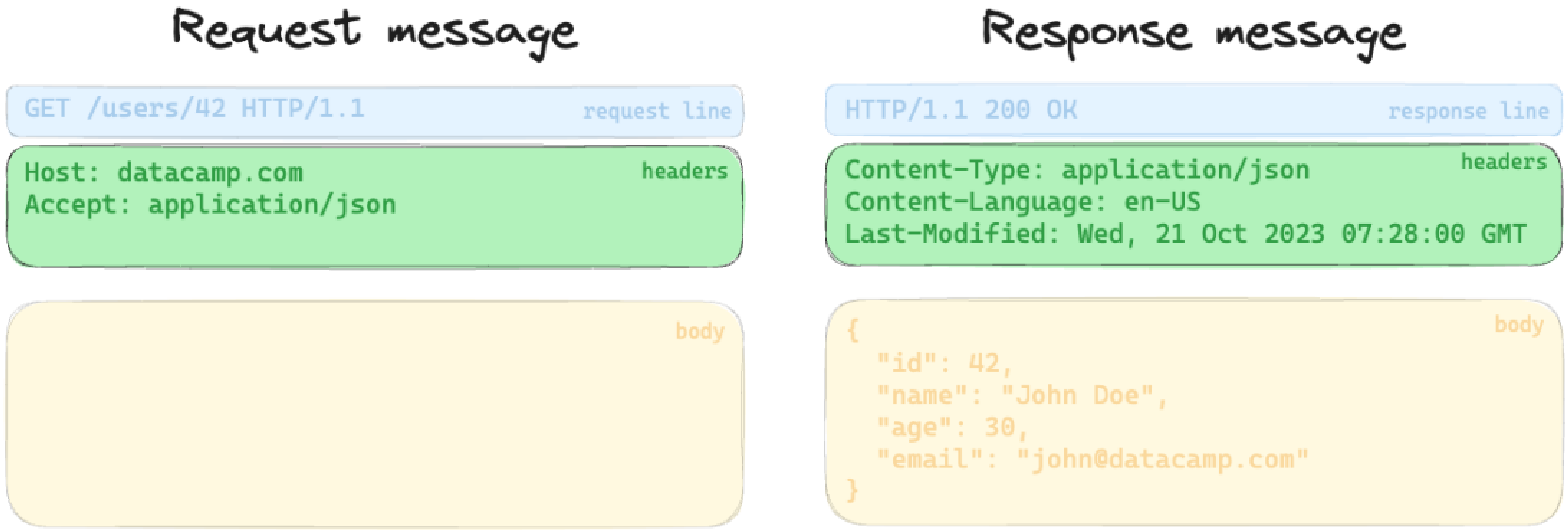
Status code categories

- `1XX` : Informational responses
- `2XX` : Successful responses
- `3XX` : Redirection messages
- `4XX` : Client error responses
- `5XX` : Server error responses

Frequently used status codes

- `200` : OK
- `404` : Not Found
- `500` : Internal Server Error

Headers



```
GET /users/42 HTTP/1.1
```

```
Host: datacamp.com
Accept: application/json
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
Content-Language: en-US
Last-Modified: Wed, 21 Oct 2023 07:28:00 GMT
```

```
{
  "id": 42,
  "name": "John Doe",
  "age": 30,
  "email": "john@datacamp.com"
}
```

```
key1: Value 1
key2: Value 2
```

Exercises

- Let's practice with the Reddit API using PRAW (Python Reddit API Wrapper).

- Follow the instructions on the notebook: "*reddit_api.ipynb*".