# Introduction to APIs

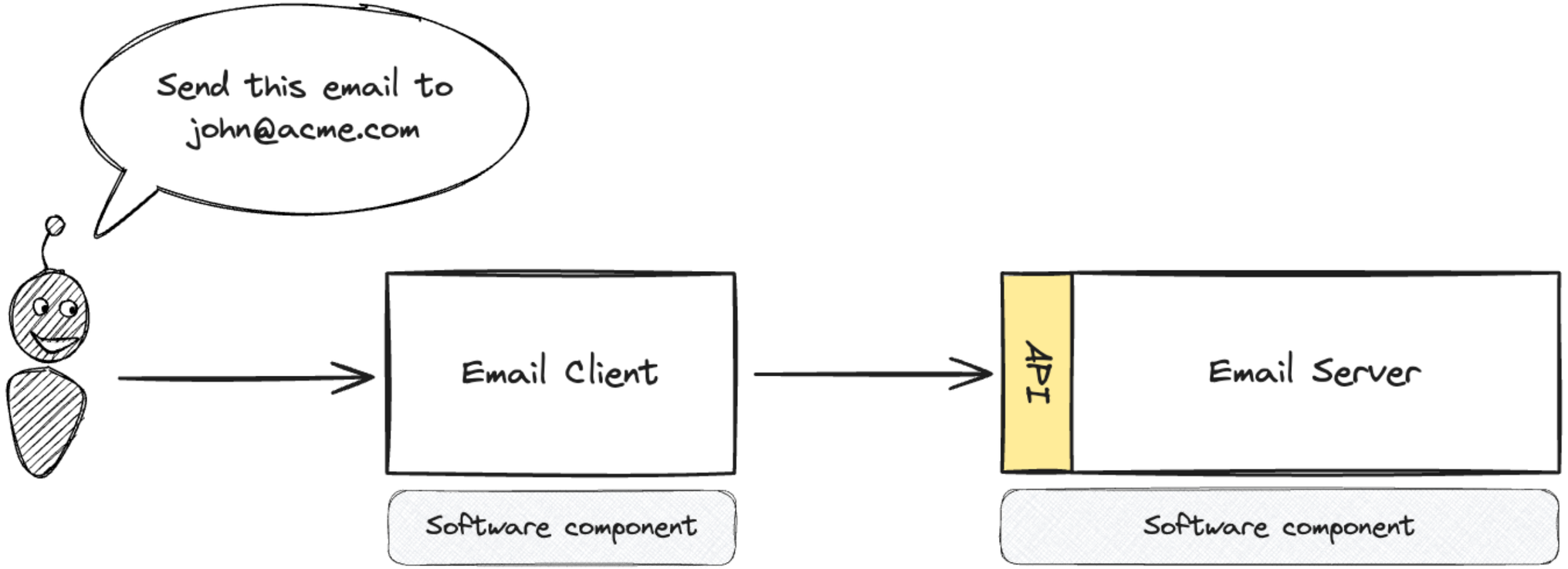
INTRODUCTION TO APIS IN PYTHON

**Chris Ramakers**
Engineering Manager

datacamp

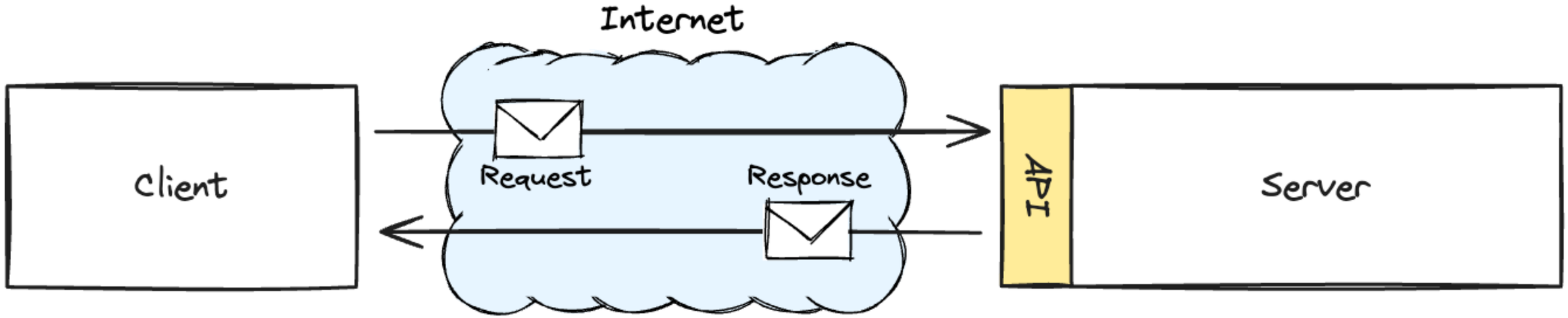
# What is an API?

- Application Programming Interface
- Set of communication rules and abilities
- Enables interactions between software applications



# Web APIs, clients and servers

- Web APIs communicate over the internet using HTTP
- Client sends a request message to a Server
- Server returns a response message to the Client




- Request/Response cycle

# Types of Web APIs

- **SOAP**
  - Focus on strict and formal API design
  - Enterprise applications
- **REST**
  - Focus on simplicity & scalability
  - Most common API architecture
- **GraphQL**
  - Focus on flexibility
  - Optimized for performance

[1] https://www.postman.com/state-of-api/api-technologies/#api-technologies

# Working with APIs in Python

## urllib

- Bundled with Python
- Powerful but not very developer-friendly

```
from urllib.request import urlopen
api = "http://api.music-catalog.com/"

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

## requests

- Many powerful built-in features
- Easier to use

```
import requests
api = "http://api.music-catalog.com/"

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

# Let's code!

INTRODUCTION TO APIS IN PYTHON

# The basic anatomy of an API request

INTRODUCTION TO APIS IN PYTHON

**Chris Ramakers**
Engineering Manager

datacamp

# What are URLs?

- URL = Uniform Resource Locator
- The structured address to an API Resource
- Customize the URL to interact with specific API Resources

```
http://350.5th-ave.com/unit/243
```

# Dissecting the URL


- **Protocol** = the means of transportation
- **Domain** = the street address of the office building
- **Port** = the gate or door to use when entering the building
- **Path** = the specific office unit inside the building
- **Query** = any additional instructions

# Adding query parameters with requests

```
# Append the query parameter to the URL string
response = requests.get('http://350.5th-ave.com/unit/243?floor=77&elevator=True')
print(response.url)
```

```
http://350.5th-ave.com/unit/243?floor=77&elevator=True
```

- Use the `params` argument to add query parameters

```
# Create dictionary
query_params = {'floor': 77, 'elevator': True}
# Pass the dictionary using the `params` argument
response = requests.get('http://350.5th-ave.com/unit/243', params=query_params)
print(response.url)
```

```
http://350.5th-ave.com/unit/243?floor=77&elevator=True
```

# HTTP Verbs

- **Destination:** Unit 243 of the 350 5th Ave office building
- **URL:** `http://350.5th-ave.com/unit/243`

## Actions

| Verb | Action | Description |
|---|---|---|
| GET | Read | Check the mailbox contents |
| POST | Create | Drop a new package in the mailbox |
| PUT | Update | Replace all packages with a new one |
| DELETE | Delete | Remove all packages from the mailbox |

[1] There are 9 HTTP verbs in total, but for simple REST APIs only these 4 are relevant

# Sending data via POST and PUT

```
# GET = Retrieve a resource
response = requests.get('http://350.5th-ave.com/unit/243')
# POST = Create a resource
response = requests.post('http://350.5th-ave.com/unit/243', data={"key": "value"})
# PUT = Update an existing resource
response = requests.put('http://350.5th-ave.com/unit/243', data={"key": "value"})
# DELETE = Remove a resource
response = requests.delete('http://350.5th-ave.com/unit/243')
```

- Each verb has it's own method in the `requests` package
- Use the `data` argument to pass data to a POST or PUT request.

# Let's practice!

INTRODUCTION TO APIS IN PYTHON

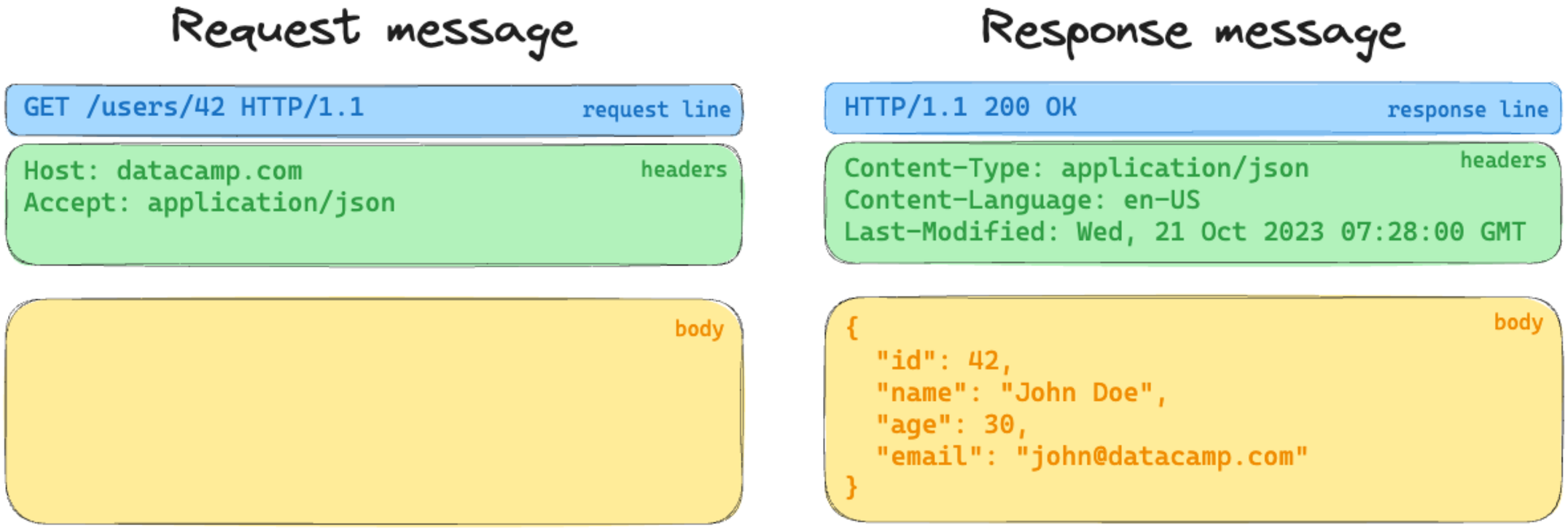
# Headers and status codes

INTRODUCTION TO APIS IN PYTHON

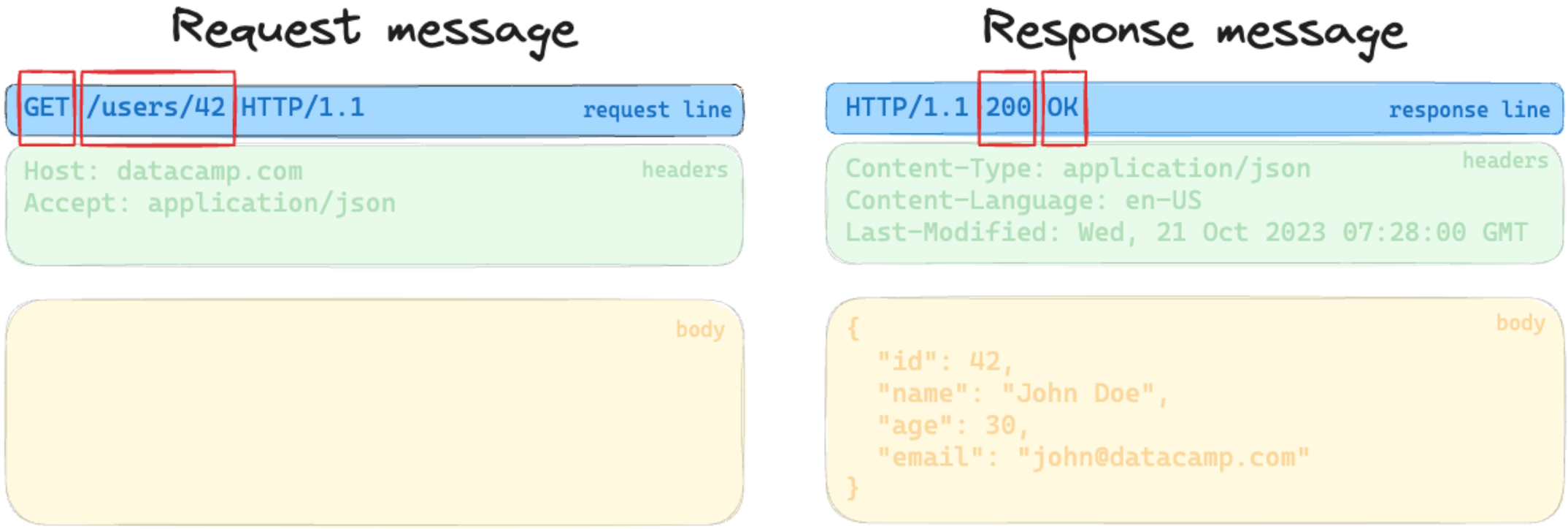
**Chris Ramakers**
Engineering Manager

datacamp

# Request and response message anatomy



# The start-line




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

# Status codes

## Status code categories

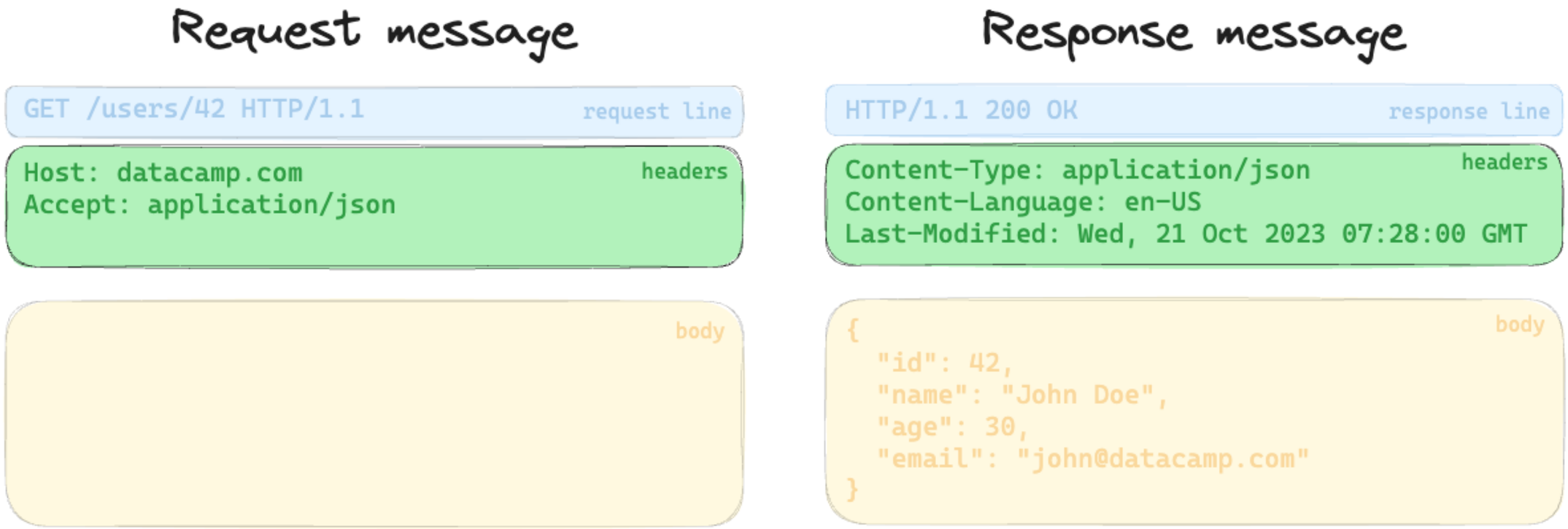
- `1XX` : Informational responses
- `2XX` : Successful responses
- `3XX` : Redirection messages
- `4XX` : Client error responses
- `5XX` : Server error responses

## Frequently used status codes

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

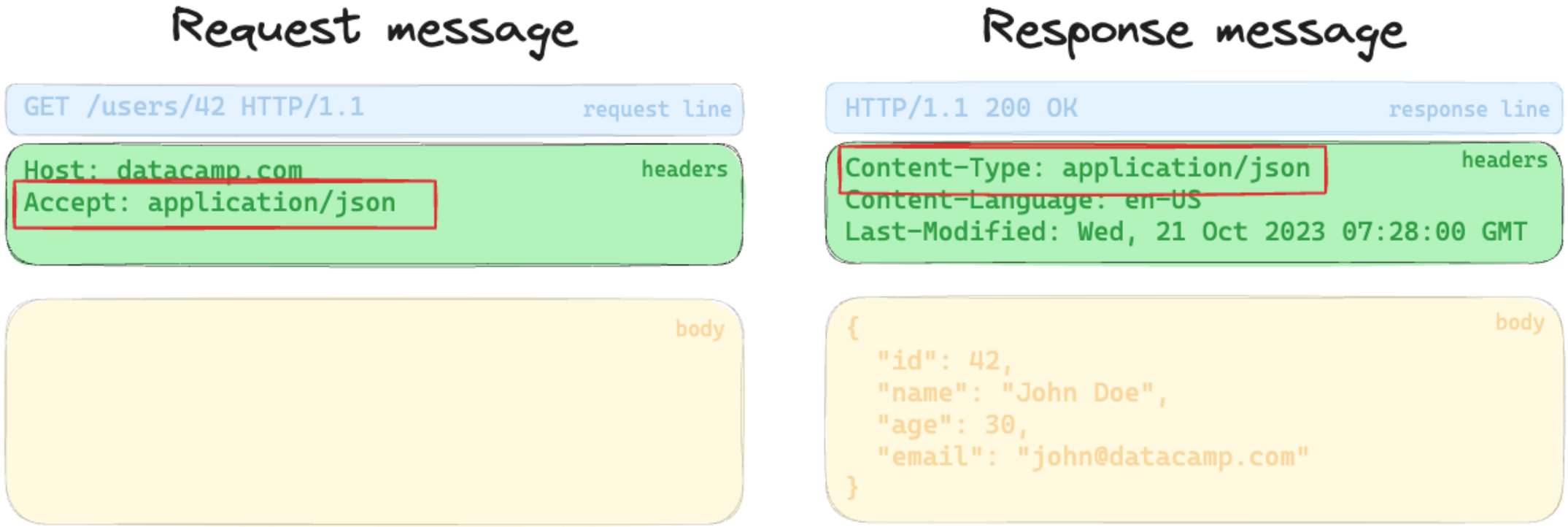
[1] For a full list of all response codes you can refer to the MDN page on status-codes via https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

# Headers




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

# Example: Content negotiation with headers




- Client adds an `accept: application/json` header to the request
- Server responds with a `content-type: application/json` header

# Headers with requests

```
# Adding headers to a request
response = requests.get(
  'https://api.datacamp.com',
  headers={'accept':'application/json'}
)
```

```
# Reading response headers
response.headers['content-type']
```

```
'application/json'
```

```
response.headers.get('content-type')
```

```
'application/json'
```

# Status codes with requests

```
# Accessing the status code
response = requests.get('https://api.datacamp.com/users/12')
response.status_code == 200
```

```
True
```

```
# Looking up status codes using requests.codes
response = requests.get('https://api.datacamp.com/this/is/the/wrong/path')
response.status_code == requests.codes.not_found
```

```
True
```

# Let's practice!

INTRODUCTION TO APIS IN PYTHON