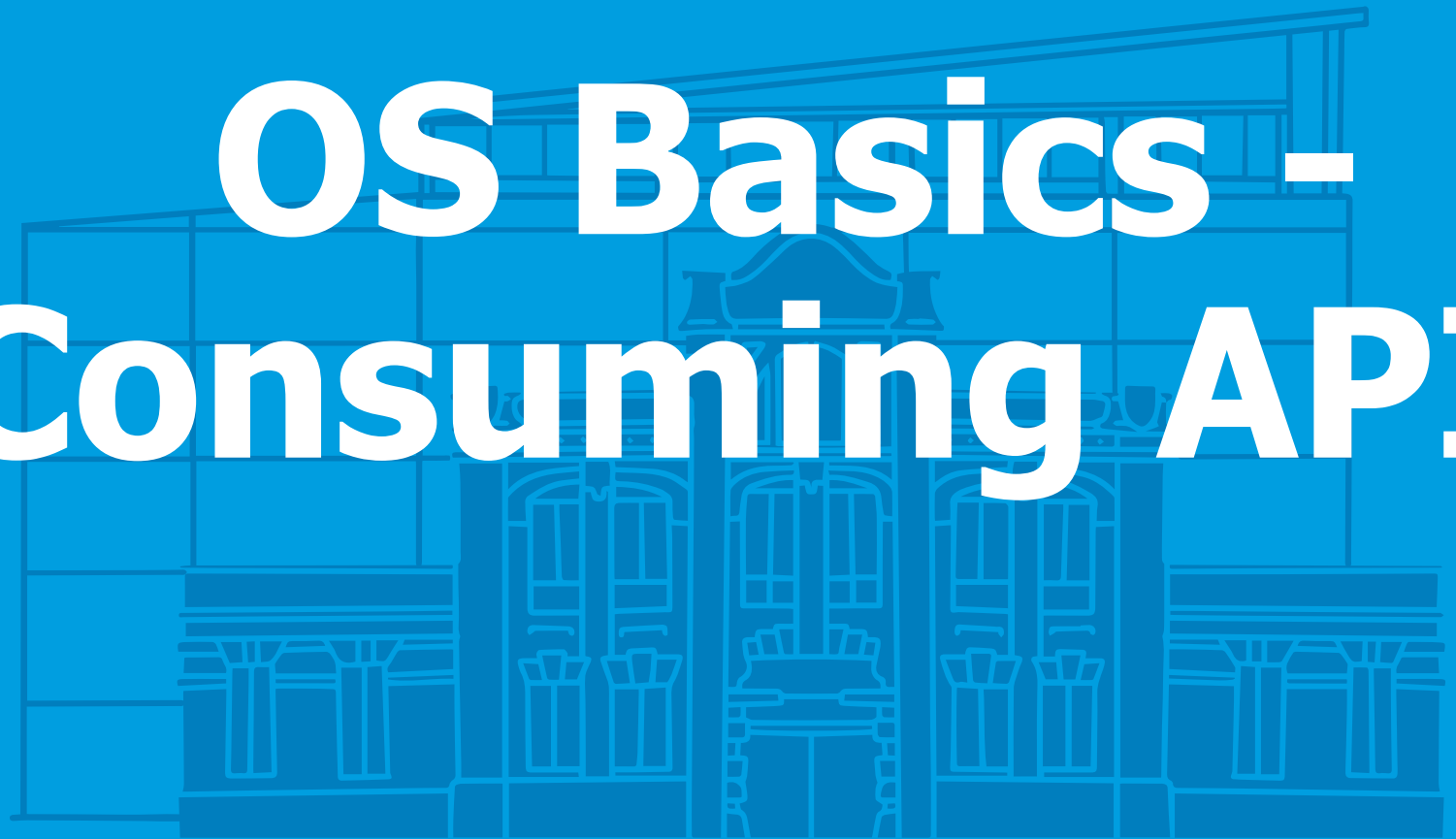
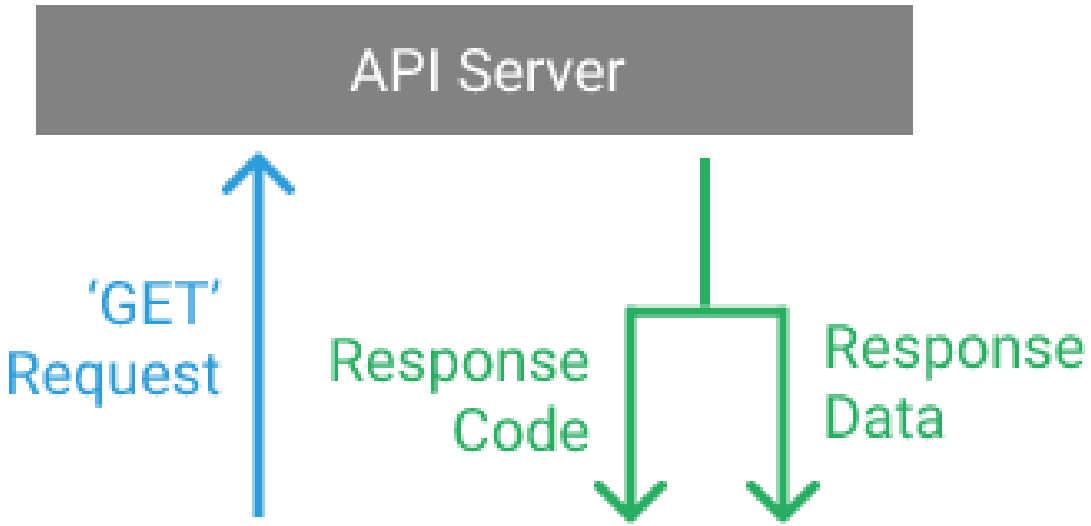
netmind
a bts company

02

# OS Basics - Consuming APIs



- Knowing how to consume an API is one of those magical skills that, once mastered, will crack open a whole new world of possibilities, and consuming APIs using Python is really easy.
- A lot of apps and systems you use on a daily basis are connected to an API.
- From very simple and mundane things, like checking the weather in the morning, to more addictive and time-consuming actions, such as scrolling through your Instagram, TikTok, or Twitter feed, APIs play a central role.



# The requests library

- Import the **requests** library and then fetch (or get) data from the URL for the Random User Generator API.

```
import requests
requests.get("https://randomuser.me/api/")


# <Response [200]>
```

- To see the actual data, use **.text from the returned Response** object:

```
import requests
response = requests.get("https://randomuser.me/api/")
response.text

#'{"results":[{"gender":"female",
"name":{"title":"Ms","first":"Isobel","last":"Wang"}...'
```

# Endpoints and Resources

- The first thing you need to know for consuming an API is the API URL, typically called the **base URL**.
  - https://api.thedogapi.com

- An **endpoint** is a part of the URL that specifies what resource you want to fetch.
  - https://api.thedogapi.com/v1**/breeds**

- Well-documented APIs usually contain an **API reference**, which is extremely useful for knowing the exact endpoints and resources an API has and how to use them.

```
response = requests.get("https://api.thecatapi.com/v1/breeds")

response.text

# '[{..."id":"abys","name":"Abyssinian"}]'
```

# Request and Response

- All interactions between a client—in this case the Python code—and an API are split into a request and a response:
  - **Requests** contain relevant data regarding your API request call, such as the base URL, the endpoint, the method used, the headers, and so on.
  - **Responses** contain relevant data returned by the server, including the data or content, the status code, and the headers.

```
response =
requests.get("https://api.thedogapi.com/v1/breeds")

response

# <Response [200]>

response.request

# <PreparedRequest [GET]>


request = response.request

request.url

# 'https://api.thedogapi.com/v1/breeds'

request.path_url

# '/v1/breeds'

request.method

# 'GET'

request.headers

# {'User-Agent': 'python-requests/2.24.0', 'Accept-
Encoding': 'gzip, deflate',

'Accept': '*/*', 'Connection': 'keep-alive'}
```

# Request and Response

```
response
```

```
# <Response [200]>
```

```
response.text
```

```
#'[{"weight":{"imperial":"6 - 13","metric":"3 - 6"}, "height":{"imperial":"9 -
11.5","metric":"23 - 29"},"id":1, "name":"Affenpinscher", ...}]'
```

```
response.status_code
```

```
# 200
```

```
response.headers
```

```
# {'Cache-Control': 'post-check=0, pre-check=0', 'Content-Encoding': 'gzip',
'Content-Type': 'application/json; charset=utf-8', 'Date': 'Sat, 25 Jul 2020
17:23:53 GMT'...}
```

# Status Codes

- Status codes tell if your request was successful, if it's missing data, if it's missing credentials, and so on.

| Status code | Description |
|---|---|
| 200 OK | Your request was successful! |
| 201 Created | Your request was accepted and the resource was created. |
| 400 Bad Request | Your request is either wrong or missing some information. |
| 401 Unauthorized | Your request requires some additional permissions. |
| 404 Not Found | The requested resource does not exist. |
| 405 Method Not Allowed | The endpoint does not allow for that specific HTTP method. |
| 500 Internal Server Error | Your request wasn't expected and probably broke something on the server side. |

# HTTP Headers

- HTTP headers are used to define a few parameters governing requests and responses.

| HTTP Header | Description |
| --- | --- |
| Accept | What type of content the client can accept |
| Content-Type | What type of content the server will respond with |
| User-Agent | What software the client is using to communicate with the server |
| Server | What software the server is using to communicate with the client |
| Authentication | Who is calling the API and what credentials they have |

# HTTP Headers

```
response = requests.get("https://api.thedogapi.com/v1/breeds/1")

response.request.headers

# {'User-Agent': 'python-requests/2.24.0', 'Accept-Encoding': 'gzip,
deflate', 'Accept': '*/*', 'Connection': 'keep-alive'}
```

# Custom Headers

- Custom headers start with **X-**, but they're not required to.
- API developers typically use custom headers to send or request additional custom information from clients.

```
headers = {"X-Request-Id": "<my-request-id>"}

response = requests.get("https://example.org", headers=headers)

response.request.headers

# {'User-Agent': 'python-requests/2.24.0', 'Accept-Encoding': 'gzip,
deflate', 'Accept': '*/*', 'Connection': 'keep-alive', 'X-Request-Id':
'<my-request-id>'}
```

# Content-Type

- These days, most APIs use **JSON** as the default content type, but you might need to use an API that returns **XML** or other media types, such as images or video.

```
response = requests.get("https://api.thedogapi.com/v1/breeds/1")
response.headers.get("Content-Type")
# 'application/json; charset=utf-8'
```

# Response Content

- To properly read the response contents according to the different Content-Type headers, the requests package comes with a couple of different Response attributes you can use to manipulate the response data:
  - **.text** returns the response contents in Unicode format.
  - **.content** returns the response contents in bytes.

```
response = requests.get("https://api.thedogapi.com/v1/breeds/1")
response.headers.get("Content-Type")
# 'application/json; charset=utf-8'
response.content
# b'{"weight":{"imperial":"6 - 13","metric":"3 - 6"}...'
```

# Response Content

- For JSON content, the requests library includes a specific .json() method that you can use to immediately convert the API bytes response into a Python data structure:

```
response = requests.get("https://api.thedogapi.com/v1/breeds/1")
response.headers.get("Content-Type")
# 'application/json; charset=utf-8'
response.json()
# {'weight': {'imperial': '6 - 13', 'metric': '3 - 6'}, 'height': {'imperial': '9 - 11.5', 'metric': '23 - 29'} ...}
response.json()["name"]
# 'Affenpinscher'
```

# HTTP Methods

- When calling an API, there are a **few different methods**, also called **verbs**, that you can use to specify what action you want to execute.
- For example, if you wanted to fetch some data, you'd use the method GET, and if you wanted to create some data, then you'd use the method POST.

| HTTP Method | Description | Requests method |
|---|---|---|
| POST | Create a new resource. | `requests.post()` |
| GET | Read an existing resource. | `requests.get()` |
| PUT | Update an existing resource. | `requests.put()` |
| DELETE | Delete an existing resource. | `requests.delete()` |

# HTTP Methods

```
requests.post("https://api.thedogapi.com/v1/breeds/1")
requests.get("https://api.thedogapi.com/v1/breeds/1")
requests.put("https://api.thedogapi.com/v1/breeds/1")
requests.delete("https://api.thedogapi.com/v1/breeds/1")
```

# Query Parameters

- In the API world, query parameters are **used as filters** you can send with your API request to further narrow down the responses.

```
requests.get("https://randomuser.me/api/").json()
requests.get("https://randomuser.me/api/?gender=female&nat=de").json()
```

- To avoid having to rebuild the URL over and over again, use the params attribute to **send in a dictionary** of all query parameters to append to a URL:

```
query_params = {"gender": "female", "nat": "de"}
requests.get("https://randomuser.me/api/", params=query_params).json()
```

# Authentication

- API authentication is a complex topic.
- Authentication approaches **range from the simplistic and straightforward, like those using API keys or Basic Authentication, to much more complex and safer techniques, like OAuth**.
- Typically, calling an API without credentials or with the wrong ones will return a **401 Unauthorized** or **403 Forbidden** status code.

# API Keys

- The most common level of authentication is the API key.
- These keys are used to identify you as an API user or customer and to trace your use of the API. API keys are typically sent as a request header or as a query parameter.

- **For example**, to fetch pictures from NASA's Mars Rover Photo API, you will need a Key.
  - You can use the DEMO_KEY API key that NASA provides by default.
  - Otherwise, you can quickly generate your own by going to NASA's main API page and clicking Get Started.

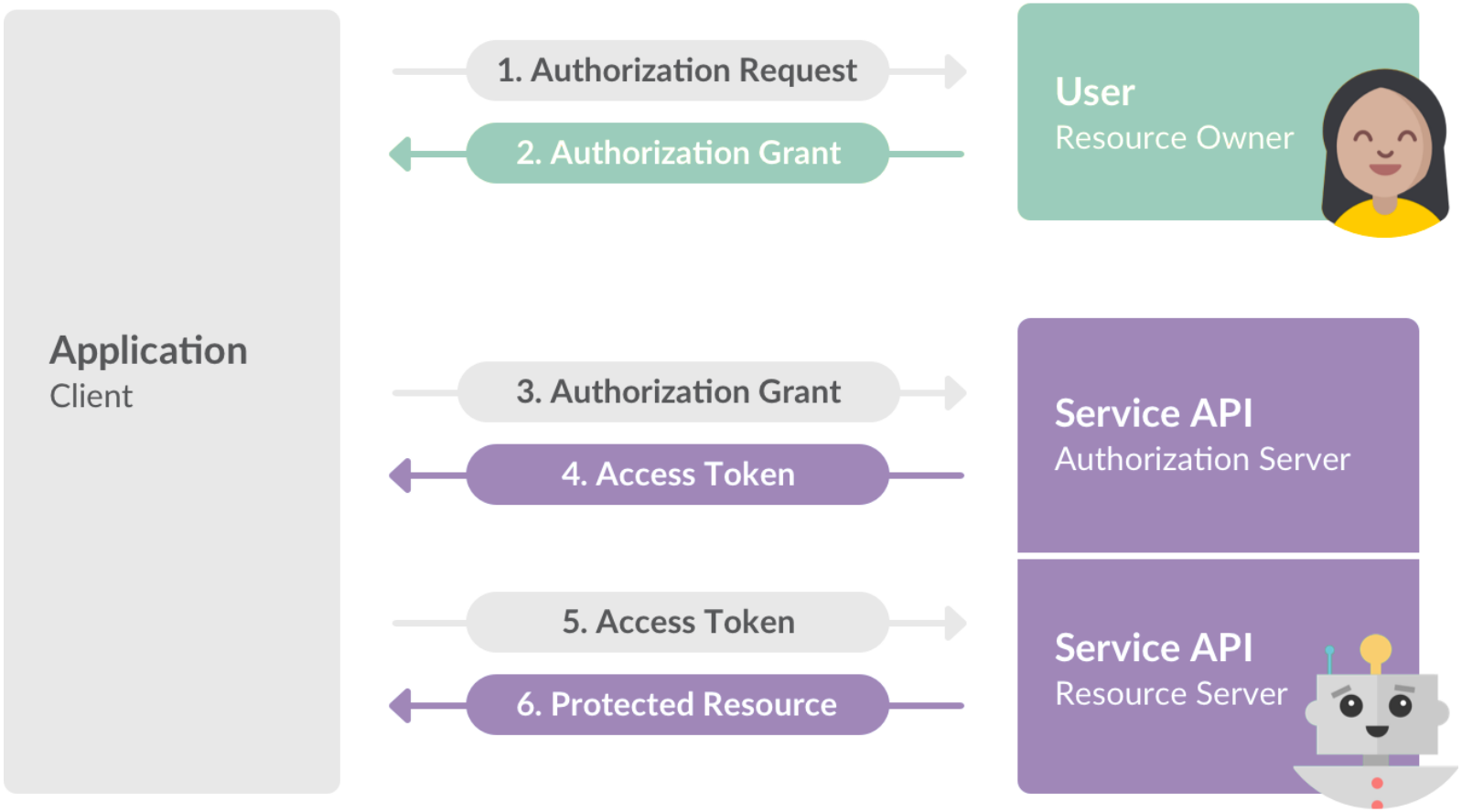
# API Keys

- You can add the API key to your request by appending the api_key= query parameter:

```
endpoint = "https://api.nasa.gov/mars-
photos/api/v1/rovers/curiosity/photos"
# Replace DEMO_KEY below with your own key if you generated one.
api_key = "DEMO_KEY"
query_params = {"api_key": api_key, "earth_date": "2020-07-01"}
response = requests.get(endpoint, params=query_params)
Response.json()
```

# OAuth

- Another very common standard in API authentication is OAuth.




Source: https://api.slack.com/legacy/oauth

# OAuth

- when consuming APIs using OAuth:
  - You need to **create an application that will have an** ID (app_id or client_id) **and a secret** (app_secret or client_secret).
  - You need to have **a redirect URL** (redirect_uri), which the API will use to send information to you.
  - You'll **get a code as the result of the authentication**, which you need to exchange for an access token.

# EXERCISE

## Consuming Github API

There’s a great step-by-step explanation on how to do this in the GitHub documentation (https://docs.github.com/en/free-pro-team@latest/developers/apps/creating-an-oauth-app) that you can follow.

The only thing to keep in mind is to use the https://httpbin.org/anything URL for the Authorization callback URL field.

Once you've created your app, copy and paste the Client_ID and Client_Secret, together with your selected redirect URL, into a Python file called github.py:

```
import requests


# REPLACE the following variables with your Client ID and Client Secret
CLIENT_ID = "<REPLACE_WITH_CLIENT_ID>"
CLIENT_SECRET = "<REPLACE_WITH_CLIENT_SECRET>"


# REPLACE the following variable with what you added in the
# "Authorization callback URL" field
REDIRECT_URI = "<REPLACE_WITH_REDIRECT_URI>"
```

Now that you have all the important variables in place, you need to be able to create a link to redirect the user to their GitHub account, as explained in the GitHub documentation:

```
def create_oauth_link():
    params = {
        "client_id": CLIENT_ID,
        "redirect_uri": REDIRECT_URI,
        "scope": "user",
        "response_type": "code",
    }

    endpoint = "https://github.com/login/oauth/authorize"
    response = requests.get(endpoint, params=params)
    url = response.url
    return url
```

The next step in the authorization flow is to exchange the code you get for an access token:

```
def exchange_code_for_access_token(code=None):
    params = {
        "client_id": CLIENT_ID,
        "client_secret": CLIENT_SECRET,
        "redirect_uri": REDIRECT_URI,
        "code": code,
    }

    headers = {"Accept": "application/json"}
    endpoint = "https://github.com/login/oauth/access_token"
    response = requests.post(endpoint, params=params,
headers=headers).json()
    return response["access_token"]
```

Now you can add the following to your file and try running it:

```
link = create_oauth_link()

print(f"Follow the link to start the authentication with GitHub:
{link}")

code = input("GitHub code: ")

access_token = exchange_code_for_access_token(code)

print(f"Exchanged code {code} with access token: {access_token}")
```

If everything goes according to plan, then you should be rewarded with a valid access token that you can use to make calls to the GitHub API, impersonating the authenticated user.

Now fetch your user profile using the User API and to print your name, username, and number of private repositories:

```
def print_user_info(access_token=None):
    headers = {"Authorization": f"token {access_token}"}
    endpoint = "https://api.github.com/user"
    response = requests.get(endpoint, headers=headers).json()
    name = response["name"]
    username = response["login"]
    private_repos_count = response["total_private_repos"]
    print(
        f"{name} ({username}) | private repositories: {private_repos_count}"
    )
```

- Now that you have a valid access token, you need to send it on all your API requests using the Authorization header.
- The response to your request will be a Python dictionary containing all the user information.
- From that dictionary, you want to fetch the fields name, login, and total_private_repos.
- You can also print the response variable to see what other fields are available.