[ ]:
```
# **Tworzenie API we Flasku - Wprowadzenie**

W tym ćwiczeniu nauczysz się, jak stworzyć proste API w Flasku, uruchomić je, wysyłać do niego zapytania oraz wykorzystać model decyzyjny w oparciu o podstawową regułę logiczną.

## **1️⃣ Tworzenie podstawowego API**
Najpierw utworzymy podstawową aplikację Flask.

### **Zapisanie kodu API do pliku**
W Jupyter Notebooku użyj magicznej komendy `%%file`, aby zapisać kod podstawowej aplikacji flask do pliku `app.py`: Kod znajdziesz na [cw1](https://sebkaz-teaching.github.io/RTA_2025/cw1.html)
Jako tekst do wyświetlenie strony głównej użyj `Witaj w moim API!`.
```

*[36]:
```
%%file app.py
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def home():
    return jsonify({"message": "Witaj w moim API!"})

if __name__ == '__main__':
    app.run(port=5000)
```

```
Overwriting app.py
```

Teraz uruchom API w terminalu, wpisując:

```
python app.py
```

Flask uruchomi serwer lokalnie pod adresem `http://127.0.0.1:5000/`.

Sprawdzenie działania API

W Jupyter Notebooku wykonaj zapytanie GET do strony głównej. Na podstawie pola `status_code` napisz wyrażenie warunkowe które dla status_code 200 wyświetli zawartość odpowiedzi (z pola `content`).

*[37]:
```
import requests
#response = pass # TWOJ KOD

response = requests.get("http://127.0.0.1:5000/")
if response.status_code == 200:
    print(response.content.decode())
```

```
{"message":"Witaj w moim API!"}
```

Jeśli wszystko działa poprawnie, zobaczysz komunikat `Witaj w moim API!`.

2️⃣ Dodanie nowej podstrony

Dodajmy nową podstronę `mojastrona`, która zwróci komunikat `To jest moja strona!`.

*[38]:
```
%%file app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return jsonify({"message": "Witaj w moim API!"})

@app.route('/mojastrona')
def page():
    return "To jest moja strona!"

if __name__ == '__main__':
    app.run(port=5000)
```

```
Overwriting app.py
```

Ponownie uruchom API i wykonaj zapytanie do strony `"http://127.0.0.1:5000/mojastrona"`:

*[40]:
```
import requests

response = requests.get("http://127.0.0.1:5000/mojastrona")
if response.status_code == 200:
    print(response.content.decode())
```

```
To jest moja strona!
```

Powinieneś zobaczyć: `To jest moja strona!`

---

3 Automatyczne uruchamianie serwera z Jupyter Notebook

Zamknij wcześniej uruchomiony serwer (`Ctrl+C` w terminalu) i uruchom go ponownie bezpośrednio z Jupyter Notebook, korzystając z `subprocess.Popen`:

[41]:
```
import subprocess
p = subprocess.Popen(["python", "app.py"])
```

Po testach zamknij serwer wykorzystując metodę `kill`:

[42]:
```
p.kill()
```

```
---

## **🔧 Obsługa parametrów w adresie URL**
Dodajemy nową podstronę `/hello`, która będzie przyjmować parametr `name`.

Edytuj `app.py`, dodając odpowiedni  kod
```

*[75]:
```
%%file app.py
from flask import Flask, request

# Create a flask
app = Flask(__name__)

@app.route('/')
def home():
    return jsonify({"message": "Witaj w moim API!"})

@app.route('/mojastrona')
def page():
    return "To jest moja strona!"

@app.route('/hello', methods=['GET'])
def hello():
    name = request.args.get("name", "")
    if name:
        return f"Hello {name}!"
    else:
        return "Hello!"

if __name__ == '__main__':
    app.run(port=5000)
```

```
Overwriting app.py
```

*[94]:
```
res1 = requests.get("http://127.0.0.1:5000/hello")
print(res1.text)

res2 = requests.get("http://127.0.0.1:5000/hello?name=Izabela")
print(res2.text)
```

```
Hello!
Hello Izabela!
```

```
Uruchom serwer i sprawdź działanie API:
```python
res1 = requests.get("http://127.0.0.1:5000/hello")
print(res1.content)  # Powinno zwrócić "Hello!"

res2 = requests.get("http://127.0.0.1:5000/hello?name=Sebastian")
print(res2.content)  # Powinno zwrócić "Hello Sebastian!"
```

```

5 Tworzenie API z prostym modelem ML

Stworzymy nową podstronę `/api/v1.0/predict`, która przyjmuje dwie liczby i zwraca wynik reguły decyzyjnej:

- Jeśli suma dwóch liczb jest większa niż 5.8, zwraca `1`.
- W przeciwnym razie zwraca `0`.

*[90]:
```
%%file app.py
from flask import Flask, jsonify, request

# Create a flask
app = Flask(__name__)

@app.route('/')
def home():
    return jsonify({"message": "Witaj w moim API!"})

@app.route('/mojastrona')
def page():
    return "To jest moja strona!"

@app.route('/hello', methods=['GET'])
def hello():
    name = request.args.get("name", "")
    if name:
        return f"Hello {name}!"
    else:
        return "Hello!"

# Prosty model ML - reguła decyzyjna
@app.route('/api/v1.0/predict', methods=['GET'])
def predict():
    try:
        num1 = float(request.args.get("num1", 0))
        num2 = float(request.args.get("num2", 0))
        result = 1 if (num1 + num2) > 5.8 else 0

        return jsonify({
            "prediction": result,
            "features": {
                "num1": num1,
                "num2": num2
            }
        })
    except ValueError:
        return jsonify({"error": "Nieprawidłowe dane wejściowe!"}), 400



if __name__ == '__main__':
    app.run(port=5000)
```

```
Overwriting app.py
```

*[91]:
```
res = requests.get("http://127.0.0.1:5000/api/v1.0/predict?num1=3&num2=4")
print(res.json())
```

```
{'features': {'num1': 3.0, 'num2': 4.0}, 'prediction': 1}
```

Sprawdź działanie API:

```
    res = requests.get("http://127.0.0.1:5000/api/v1.0/predict?num1=3&num2=4")
    print(res.json())  # Powinno zwrócić {"prediction": 1, "features": {"num1": 3.0, "num2": 4.0}}
```

Podsumowanie

Po wykonaniu tego ćwiczenia studenci będą umieli: ✅ Tworzyć podstawowe API w Flasku.
- ✅ Dodawać podstrony i obsługiwać parametry URL.
- ✅ Wysyłać zapytania GET i analizować odpowiedzi.
- ✅ Automatycznie uruchamiać serwer z Jupyter Notebook.
- ✅ Implementować prosty model decyzyjny w API.

Gotowi na kolejne wyzwania? 🚀