

Министерство образования Новосибирской области
ГБПОУ НСО «Новосибирский авиационный технический колледж им. Б.С.
Галущака»

Методические указания к разработке API на Python с использованием
библиотеки Flask

Учебная дисциплина: МДК 01.04 Системное программирование

Разработал:

Содержание

1	Общее описание фреймворка Flask	3
2	Установка Flask	4
3	Необходимое ПО.....	6
4	Начало работы.....	7
5	Создание API.....	9
5.1	Подключение библиотеки и создание экземпляра класса	9
5.2	Создания первого порта Hello World	10
5.3	Подключение функции jsonify()	11
5.4	Создание ответа в виде словаря	12
5.5	Запуск сервера.....	12
5.6	Получение ответа сервера.....	12
6	Создание API с аргументами	14
6.1	Создание функции	14
6.2	Определение словаря пользователей	14
6.3	Возврат данных из словаря по ID.....	15
6.4	Самостоятельное задание	16
6.5	Запуск приложения	16
6.5	Результат	16
	Заключение.....	19

1 Общее описание фреймворка Flask

Flask — это фреймворк, написанный для языка Python для простого создания собственных API интерфейсов, поддерживающий шаблонизатор Jinja2 (опционально)

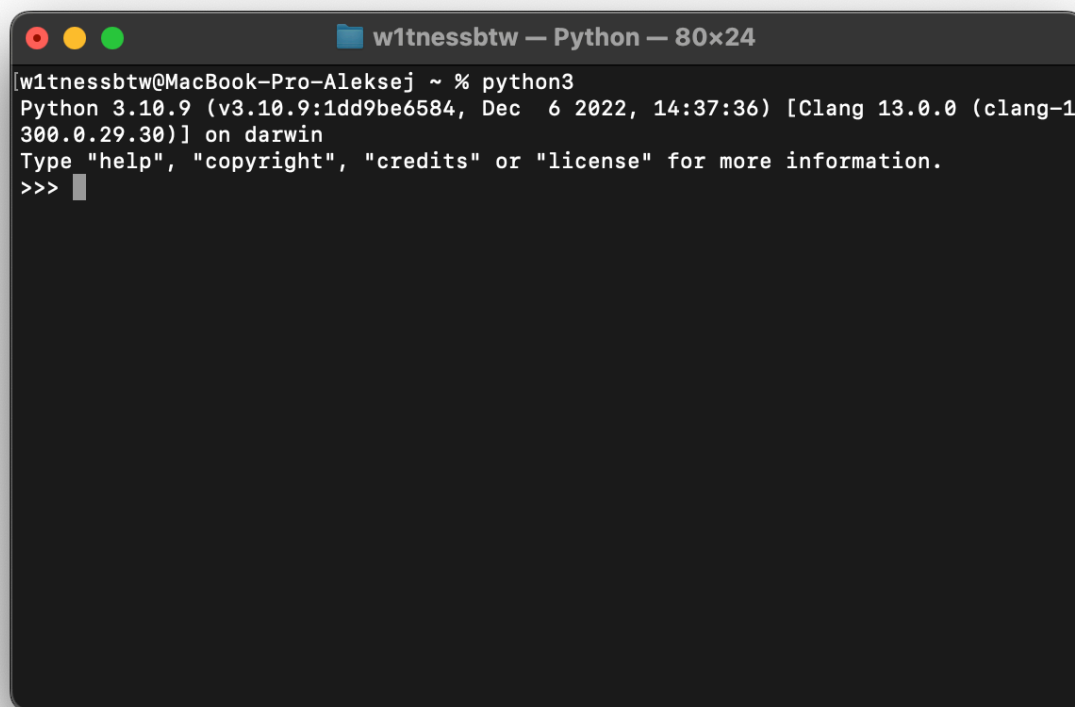
Flask включает в себя множество подпрограмм, но в данном уроке они нам не понадобятся.

Обычно, этот фреймворк используется для легких целей, например выгрузка данных из базы при помощи ответа сервера, так как на этом фреймворке очень тяжело создавать полноценные функционирующие страницы, лучше использовать Django.

2 Установка Flask

Для того, чтобы установить библиотеку Flask необходимо в терминале (в IDE или системном) ввести следующие команды:

— `python`, (или `python3`) для того чтобы определить есть ли Python на вашем ПК.

A screenshot of a macOS terminal window titled "w1tnessbtw — Python — 80x24". The terminal shows the command `python3` being executed. The output displays the Python version `3.10.9` and the architecture `arm64` on a Darwin system. The prompt `>>>` is visible at the end of the line.

```
w1tnessbtw@MacBook-Pro-Aleksej ~ % python3
Python 3.10.9 (v3.10.9:1dd9be6584, Dec 6 2022, 14:37:36) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Рисунок 1 — Проверка версии Python

Если Python установлен то вы получите следующее сообщение: Python 3.10.9

— `pip install flask` (или `pip3 install flask`)

```
witnessbtw@MacBook-Pro-Aleksej ~ % pip3 install flask
Collecting flask
  Obtaining dependency information for flask from https://files.pythonhosted.org/packages/36/42/015c23096649b908c809c69388
a805a571a3bea44362fe87e33fc3afa01f/flask-3.0.0-py3-none-any.whl.metadata
  Downloading flask-3.0.0-py3-none-any.whl.metadata (3.6 kB)
Collecting Werkzeug>=3.0.0 (from flask)
  Obtaining dependency information for Werkzeug>=3.0.0 from https://files.pythonhosted.org/packages/b6/a5/54b01f663d60d533
4f6c9c87c26274e94617a4fd463d812463626423b10d/werkzeug-3.0.0-py3-none-any.whl.metadata
  Downloading werkzeug-3.0.0-py3-none-any.whl.metadata (4.1 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)
Collecting itsdangerous>=2.1.2 (from flask)
  Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.1.3 (from flask)
  Obtaining dependency information for click>=8.1.3 from https://files.pythonhosted.org/packages/00/2e/d53fa4befbf2cfa7133
04affc7ca780ce4fc1fd871052771b58311a3229/click-8.1.7-py3-none-any.whl.metadata
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting blinker>=1.6.2 (from flask)
  Obtaining dependency information for blinker>=1.6.2 from https://files.pythonhosted.org/packages/bf/2b/11bcedb7dee492325
3a4a21bae3be854bcc4f06295bd827756352016d97c/blinker-1.6.3-py3-none-any.whl.metadata
  Downloading blinker-1.6.3-py3-none-any.whl.metadata (1.9 kB)
Collecting MarkupSafe>=2.0 (from Jinja2>=3.1.2->flask)
  Obtaining dependency information for MarkupSafe>=2.0 from https://files.pythonhosted.org/packages/3a/72/9f683a059bde0967
76e8acf9aa34cbbba21ddc399861fe3953790d4f2cde/MarkupSafe-2.1.3-cp312-cp312-macosx_10_9_x86_64.whl.metadata
  Downloading MarkupSafe-2.1.3-cp312-cp312-macosx_10_9_x86_64.whl.metadata (2.9 kB)
Downloading flask-3.0.0-py3-none-any.whl (99 kB)
 99.7/99.7 kB 286.3 kB/s eta 0:00:00
Downloading blinker-1.6.3-py3-none-any.whl (13 kB)
Using cached click-8.1.7-py3-none-any.whl (97 kB)
Downloading werkzeug-3.0.0-py3-none-any.whl (226 kB)
 226.6/226.6 kB 379.5 kB/s eta 0:00:00
Downloading MarkupSafe-2.1.3-cp312-cp312-macosx_10_9_x86_64.whl (13 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, flask
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.3 Werkzeug-3.0.0 blinker-1.6.3 click-8.1.7 flask-3.0.0 itsdangerous-2.1
```

Рисунок 2 — Установка Flask.

После ввода команды указанной выше вы получите подобное сообщение о установке фреймворка на ваш ПК. Теперь все готово к работе.

3 Необходимое ПО

Для работы с Python и Flask необходимо следующее ПО:

- a) IDE: PyCharm или Visual Studio Code
- b) Postman: для проверки запроса программно
- c) Любой браузер: для проверки запроса в браузере

4 Начало работы

Для начала запустим выбранную вами IDE, в примере будет использоваться Visual Studio Code.

Выберем папку, в которой будем работать нажав на Файл -> Открыть папку или нажав CTRL + K или CTRL + O.

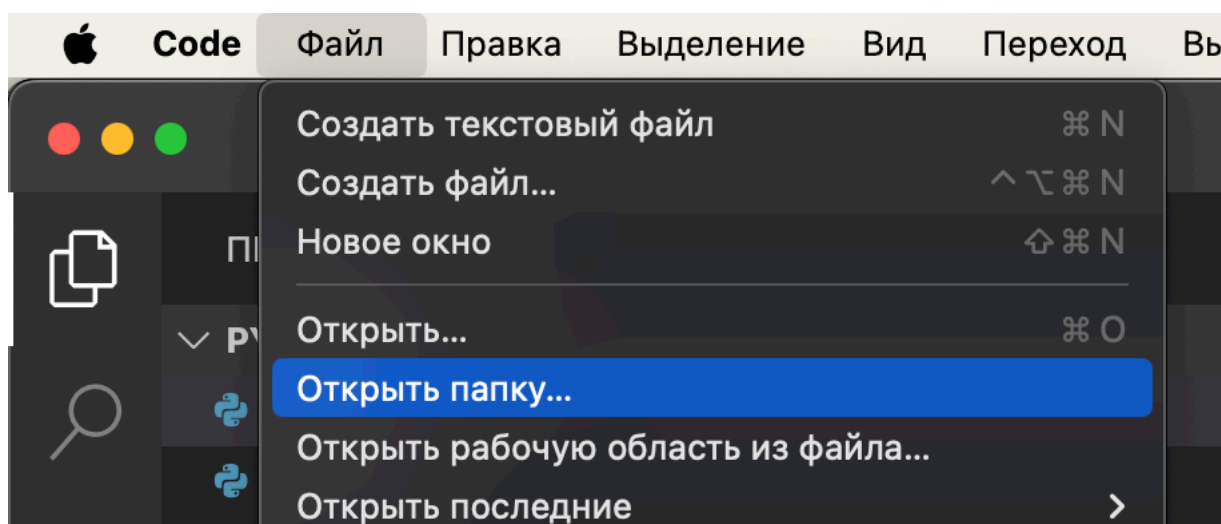


Рисунок 3 — Открытие папки

В случае примера это будет только что созданная мной папка в пути, где хранятся все мои практические работы.

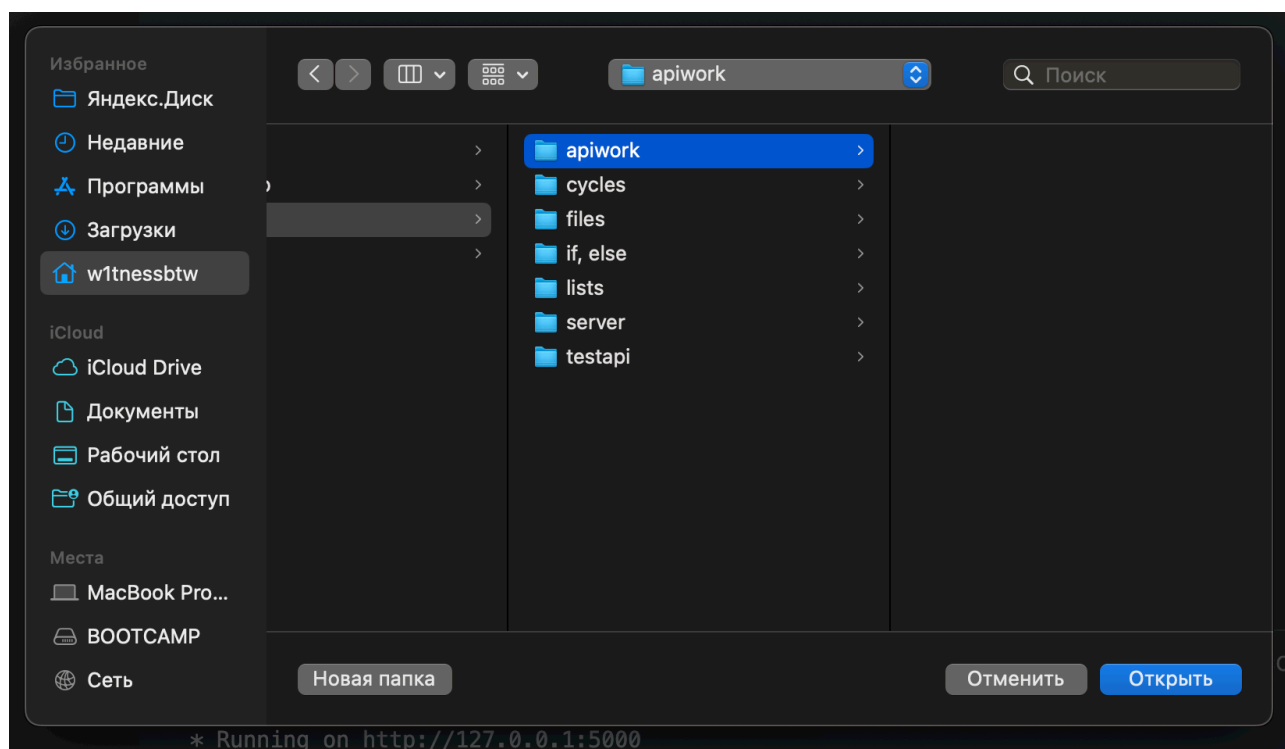


Рисунок 4 — Выбор папки

И открываем папку.

Теперь перед нами открылось пустое IDE с приветственным сообщением

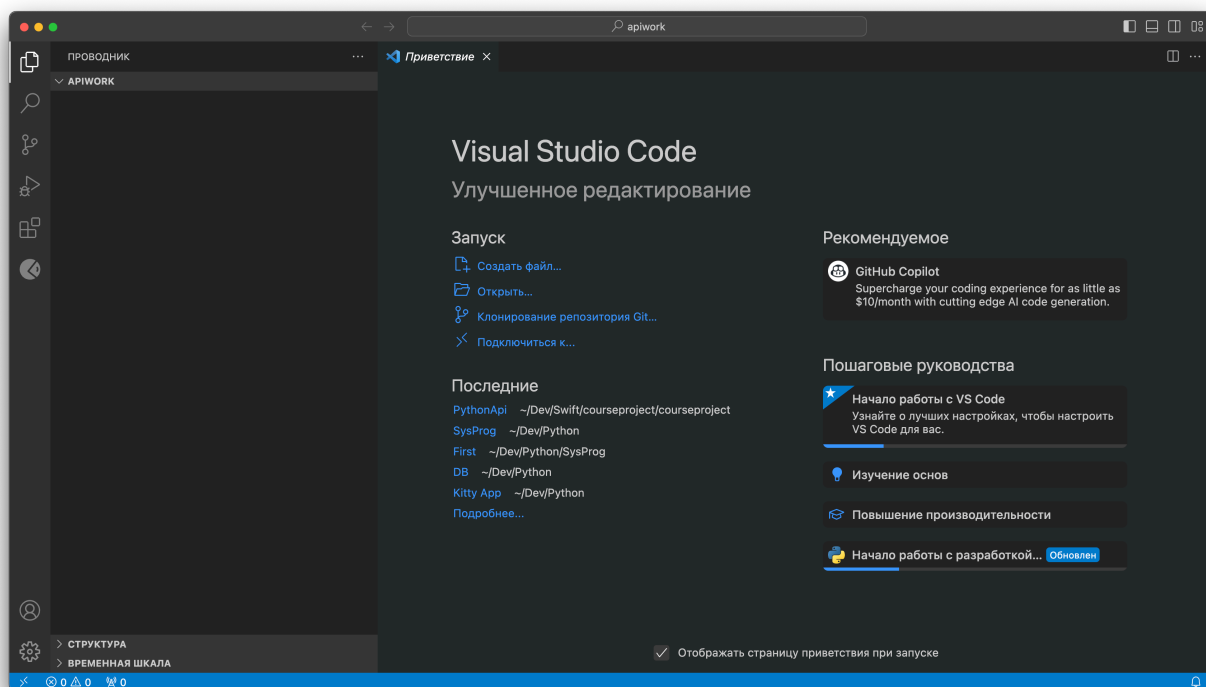


Рисунок 5 — Пустая IDE

Далее, наводим курсор на папку и нажимаем «Создать файл»

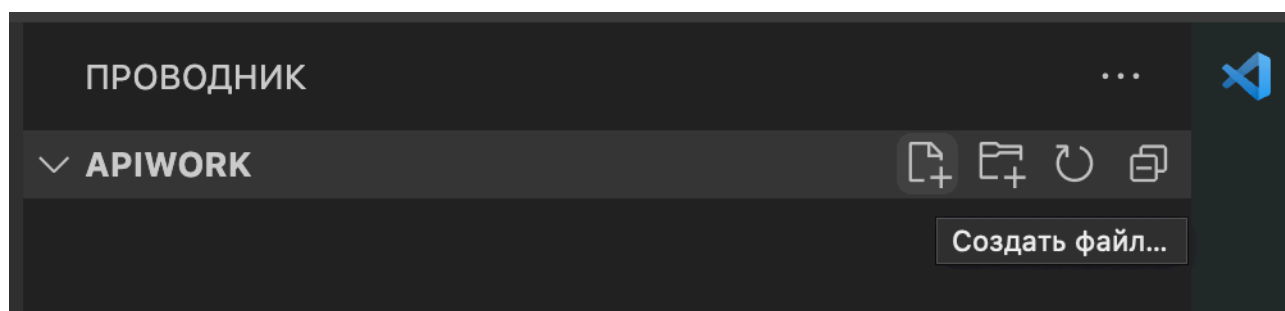


Рисунок 6 — Создание файла

И называем его произвольно. Например — api.ru

5 Создание API

5.1 Подключение библиотеки и создание экземпляра класса

Для подключения библиотеки пропишем следующие строки в начале файла:

```
from flask import Flask, request
```

Теперь библиотека подключена и готова к использованию.

Далее, создадим экземпляр класса Flask благодаря которому мы сможем использовать декораторы для создания своих портов на сайте.

Пропишем ниже следующую строку:

```
app = Flask(__name__)
```

Теперь у нас есть экземпляр класса, код выглядит следующим образом:

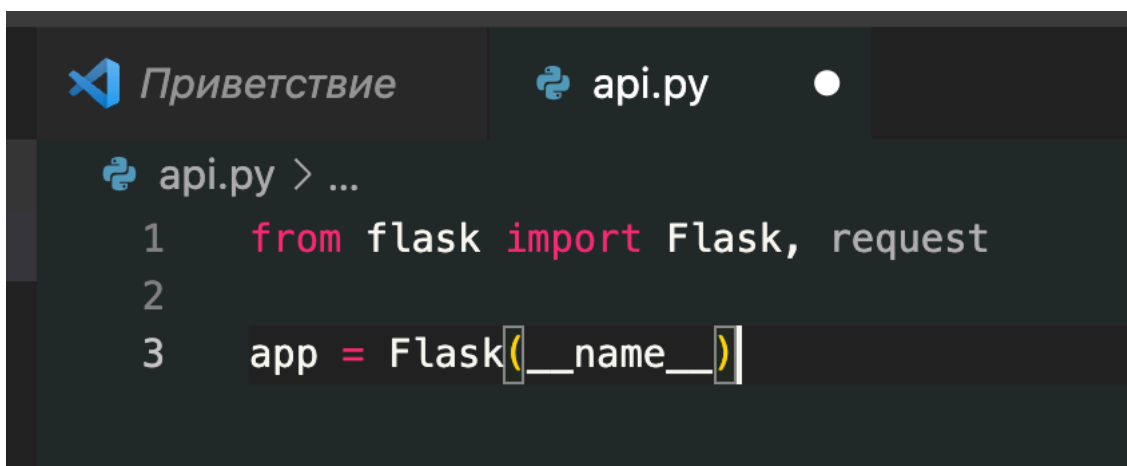


Рисунок 7 — Создание экземпляра app

Пояснение:

На 3 строчке кода мы создали переменную `app` которая будет иметь тип `<class 'flask.app.Flask'>`, именно этот экземпляр будет нам помогать создавать свои API порты.

5.2 Создания первого порта Hello World

Для создания первого нашего порта нам необходимо прописать следующие строчки:

```
@app.route('/hello_world', methods=['GET'])
def название_функции():
    pass
```

Пояснение:

`@app.route` — это декоратор (подробнее о декораторах можно найти в интернете, сейчас можно не вдаваться в подробности, особо знание этого не спасет) который помогает нам кратко описать функцию, которая определена где-то в классе библиотеки Flask, нам это знать не обязательно.

`@app.route` принимает в себя следующие аргументы:

1-ый аргумент — `'Portname'`, здесь мы задаем по какому адресу мы получим ответ. Например если мы укажем `'hello_world'` то ссылка в браузере будет выглядеть следующим образом:

https://localhost:5000/hello_world

Если мы укажем например `bye_world` то ссылка будет выглядеть следующим образом:

https://localhost:5000/bye_world

2-ой аргумент — массив с перечислением методов. Как мы знаем, есть самые базовые методы это GET и POST.

GET — позволяет нам только получить информацию

POST — позволяет создать ресурс.

```
@app.route('/hello_world', methods=['GET'])
def sayHelloWorld():

    data = {
        1: "Hello World"
    }
```

Рисунок 9 — Вид функции со словарем

В нашем примере мы будем использовать только метод GET, кстати говоря, второй аргумент опциональный, если мы не укажем массив методов то по умолчанию будет установлен метод GET.

Далее — название функции, можно называть ее как угодно, ее никто не видит, она используется только для удобства.

Далее нам стоит отредактировать наш код, задав название функции, можно описать ее так:

```
@app.route('/route_name', methods=['GET'])
def sayHelloWorld():
    pass
```

5.3 Подключение функции jsonify()

Теперь нам нужно вернуть какие то данные на странице в браузере, в формате JSON (JavaScript Object Notation), для этого в нашу самую первую строчку допишем jsonify, и наш импорт будет выглядеть так:

```
from flask import Flask, request, jsonify
```

Рисунок 8 — Подключение функции jsonify()

5.4 Создание ответа в виде словаря

Теперь чтобы получить ответ в браузере, создадим словарь, где укажем данные, которые мы будем возвращать, в нашем случае мы вернем hello world. Создадим словарь прямо в функции:

```
data = {  
    1: "Hello World"  
}
```

Теперь наша функция выглядит так:

И напоследок вернем данные в формате JSON с помощью ключевого слова `return`.

```
return jsonify(data)
```

Здесь мы вызываем встроенную функцию `jsonify`, которую мы импортировали ранее, дописывая строчку в наш импорт.

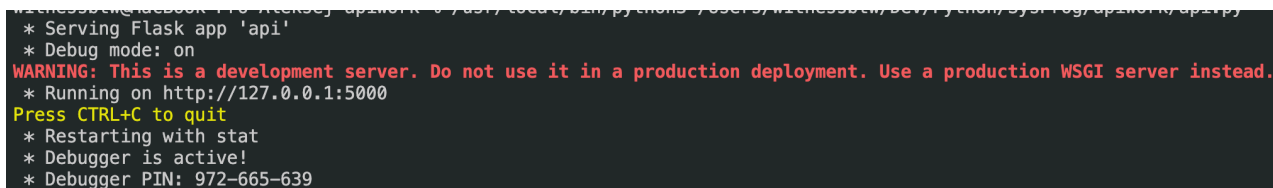
5.5 Запуск сервера

Осталось дописать следующий код вне функции:

```
if __name__ == "__main__":  
    app.run(debug=True)
```

Этот код позволит нам запустить локальный сервер. Здесь мы обращаемся к нашему экземпляру `app` и его методу `run`, который как раз и запускает сервер.

Запускаем наш Python файл и в консоли видим следующие строчки:



```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on http://127.0.0.1:5000  
Press CTRL+C to quit  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 972-665-639
```

Рисунок 10 — терминал с IP сервера

5.6 Получение ответа сервера

Нас интересует строка Running on <https://digits:port>. Копируем эту ссылку, это адрес нашего сервера, к которому мы обратимся когда сделаем запрос. Вставляем его в адресную строку но пока не переходим, мы не дописали порт.

Как мы помним, выше мы задали порт 'hello_world', соответственно наша ссылка примет вид https://digits:port/hello_world, именно по этой ссылке мы получим ответ от сервера.

Переходим по ссылке и видим ответ:

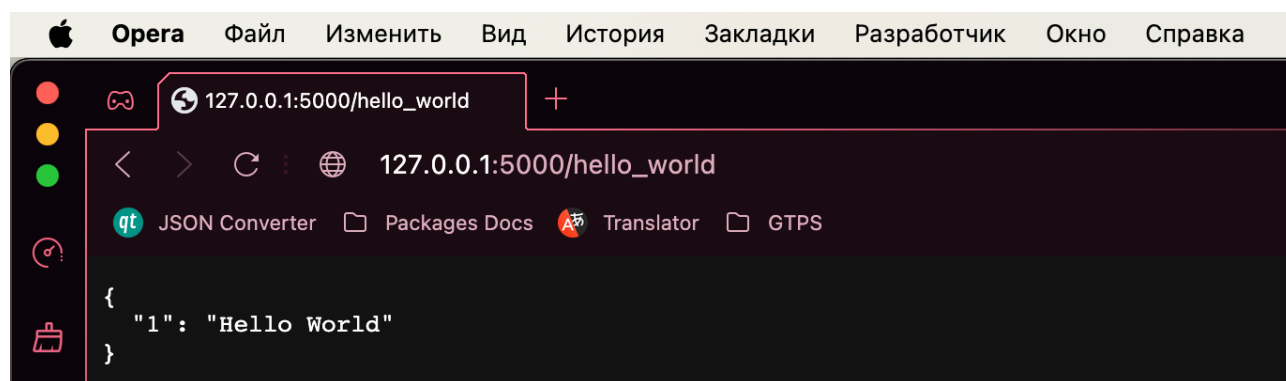


Рисунок 11 — Ответ от сервера по порту

Простейшее API создано.

6 Создание API с аргументами

Для того, чтобы создать порт, в который необходимо что то передавать, предлагаю написать новую функцию по тому же принципу, что мы и делали в прошлой главе, только теперь в порт добавим аргумент который нам нужен. Давайте будем возвращать информацию о пользователе.

6.1 Создание функции

Опишем ту же функцию, только с другими именем и портом.

```
@app.route('/users/', methods=['GET'])  
def returnUserInfo():  
    pass
```

Здесь мы создали функцию returnUserInfo() которая будет в дальнейшем нам что то возвращать. Пока оставим ее так и перейдем к созданию словаря с нашими пользователями.

Определим словарь users в котором мы будем хранить словари с данными о пользователях. Пусть ключом будет число, а значением словарь.

6.2 Определение словаря пользователей

Например, я определю пользователей следующим образом:

```
users = {  
  1: {  
    "name": "Alex",  
    "age": 25  
  },  
  2: {  
    "name": "Max",  
    "age": 28  
  },  
  3: {  
    "name": "Egor",  
    "age": 15  
  }  
}
```

Рисунок 12 — Словарь
пользователей

Здесь мы определили словарь `users` в котором хранятся данные в виде:

Id: словарь.

То есть, по какому то айдишнику у нас хранится какой то словарь. Теперь для примера вы можете создать новый файл, засунуть туда этот же словарь и написать следующую строчку:

`print(users[1])`. Эта строчка вам выведет словарь, который хранится под айдишником 1.

6.3 Возврат данных из словаря по ID

Перейдем к написанию нашей функции. Внутри функции нам необходимо сделать так, чтобы при обращении в браузере или приложении Postman по ссылке пользователю было необходимо вводить аргумент. Аргумент будет выглядеть так:

<https://localhost:5000/users>(это если что порт который мы указали сами)?
`id=числу`.

Для того, чтобы запросить аргумент, в теле функции необходимо прописать переменную, которая будет запоминать введенный в адрес айдишник с помощью метода класса request (не путать с библиотекой requests) args.get

То есть `id = request.args.get(«id»)`.

В кавычках указан аргумент, который будет вводится. Например сейчас мы указали «id», то в ссылке будет `users?id=`, если же мы укажем например number то будет `users?number=`.

Теперь наша функция выглядит так:

```
@app.route('/users/', methods=['GET'])
def returnUserInfo():
    id = request.args.get("id")
    pass
```

Рисунок 13 — Функция, принимающая параметр

Теперь, все что нам осталось, это снова вернуть данные в формате JSON с помощью функции jsonify() по полученному айдишнику.

6.4 Самостоятельное задание

Самостоятельно верните данные с помощью ключевого слова return.

Небольшая подсказка: вам нужно преобразовать переменную ID в тип Int. Это можно сделать как в return'е так и до него. На свое усмотрение.

6.5 Запуск приложения

В самом низу файла необходимо прописать следующие строчки:

```
if __name__ == "__main__":
    app.run(debug=True)
```

6.5 Результат

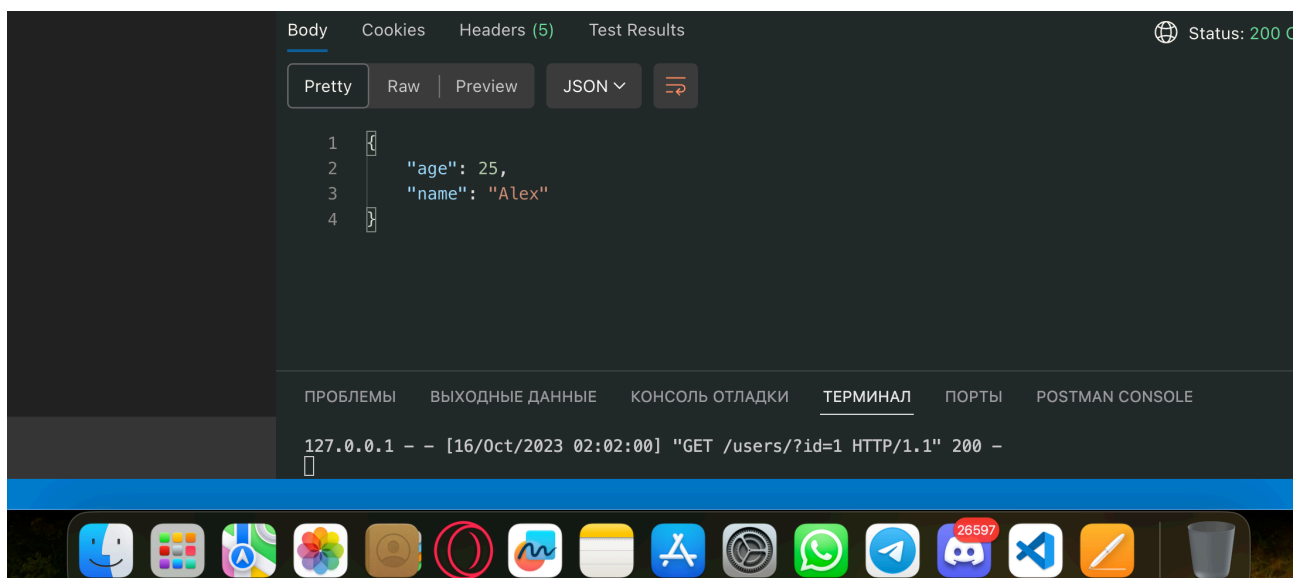


Рисунок 14 — Результат работы функции с аргументом 1

После того, как вы закончили работать с функцией, вы должны получить следующие результаты:

— по аргументу 1:

— по аргументу 2:

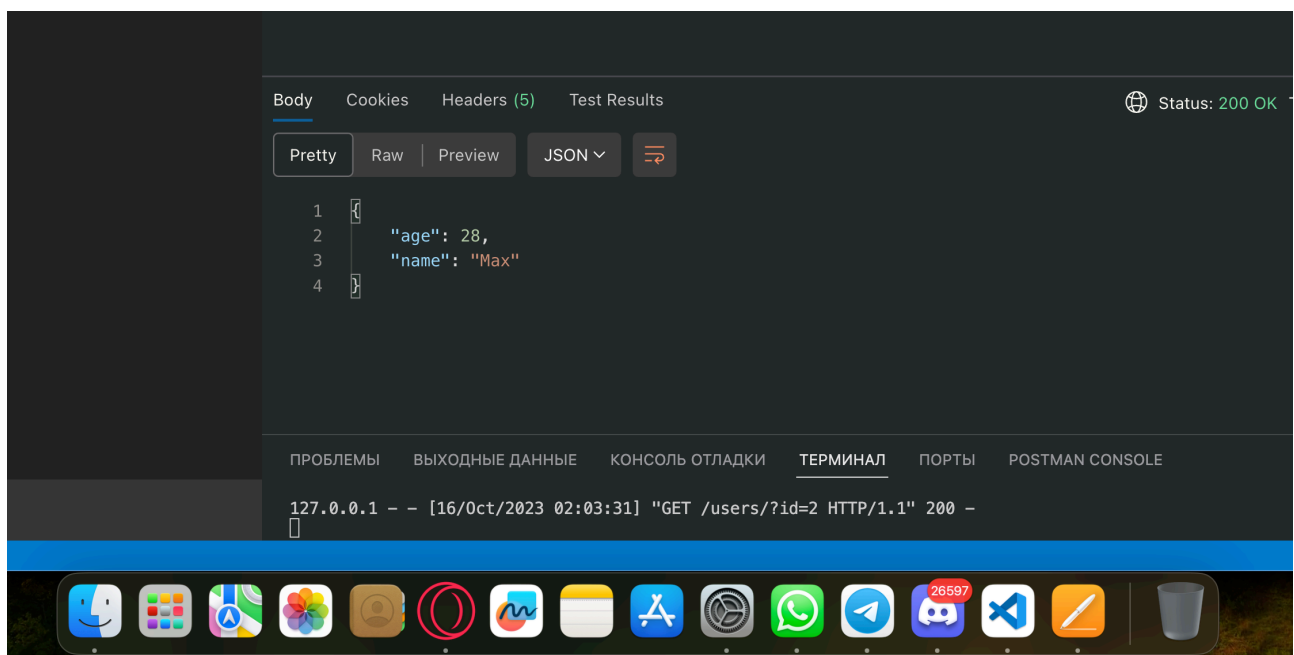


Рисунок 15 — Результат работы функции с аргументом 2

— по аргументу 3:

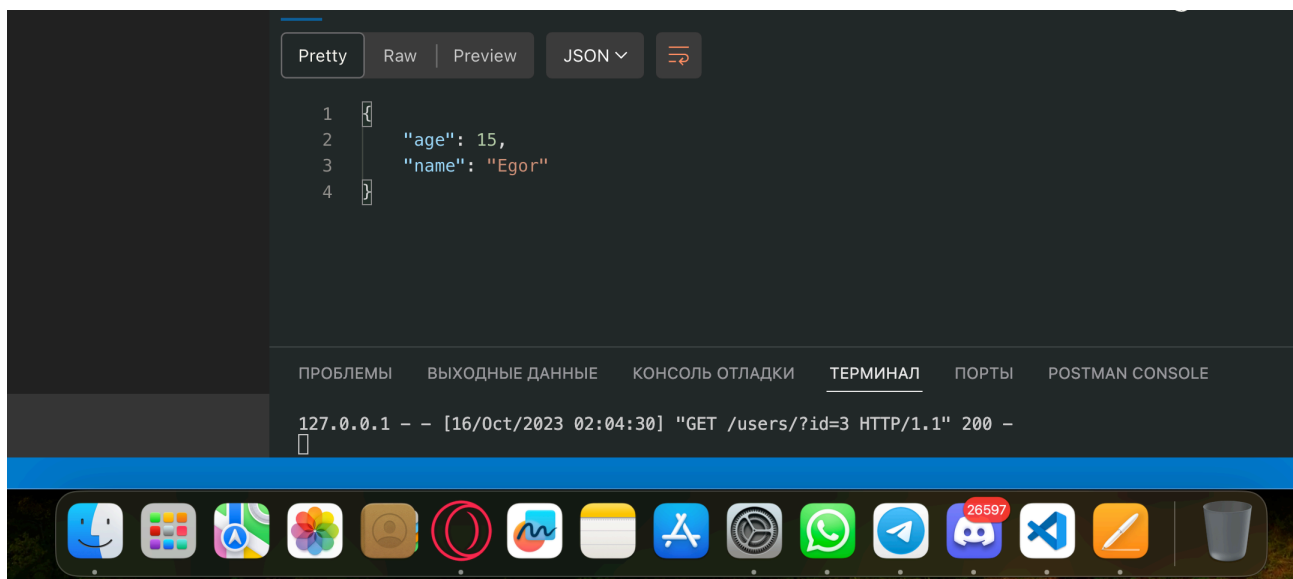


Рисунок 16 — Результат работы функции с аргументом 3

Заключение

Действуя по шагам, которые прописаны в данных методических указаниях студент должен был приобрести базовые знания по созданию API на языке Python с помощью фреймворка Flask.