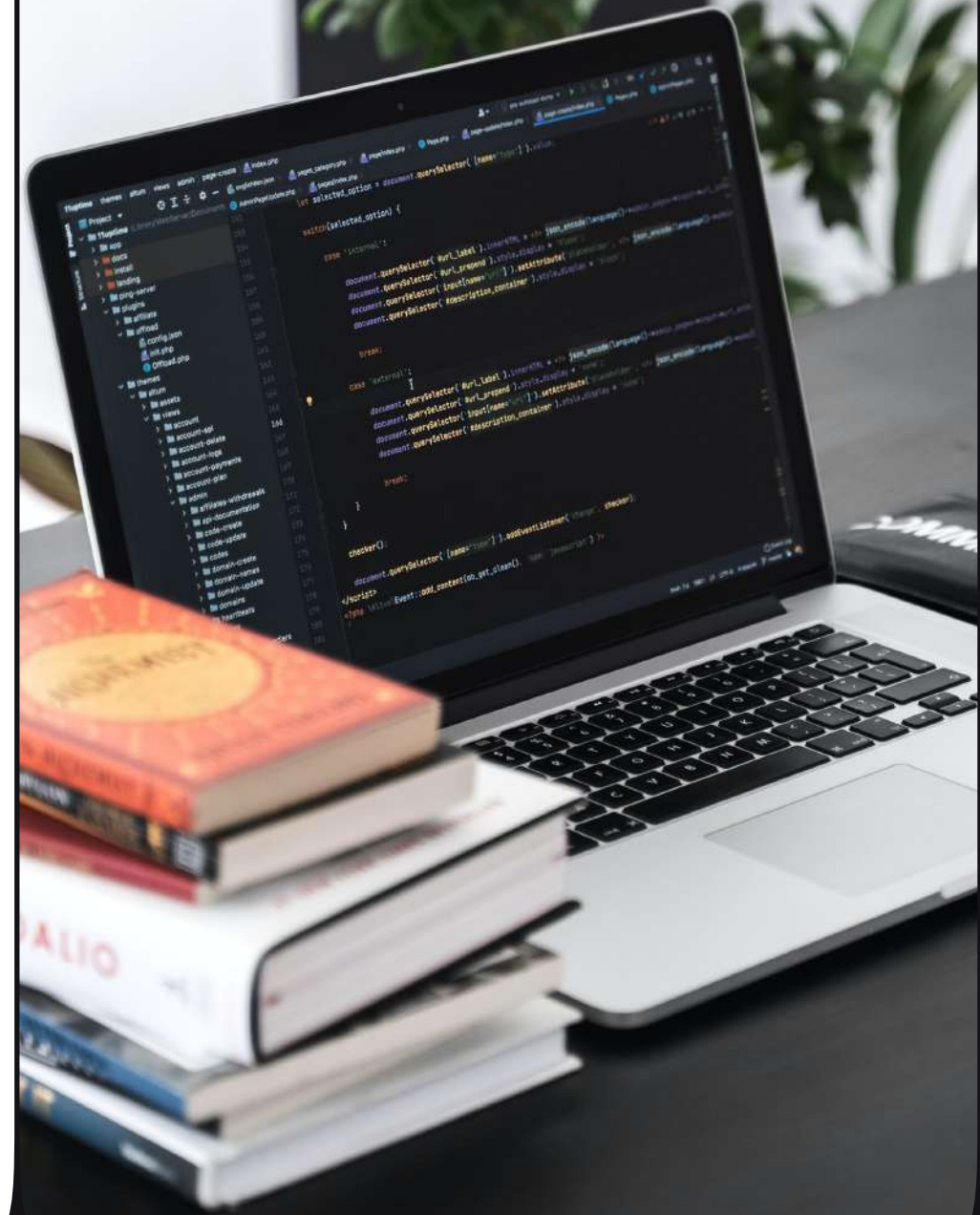


Модуль 4 Занятие 1

Фреймворк для WEB-приложений. Flask





Фреймворки

**Виртуальное
окружение**

Flask

Введение

Знакомство с Flask



Установка Flask

Изучение процесса создания
WEB-приложения

Работа с HTML и CSS

Подключение базы данных



Создание WEB-
приложения
с использованием Flask

Flask

Flask — микрофреймворк для создания веб-приложений на языке программирования Python. Создатель и автор проекта — австрийский программист Армин Ронахер, начал работу над Flask в 2010 году. Сейчас Flask используется для создания как простых приложений, так и сложных проектов.



Фреймворки

Framework (Фреймворк) — специализированное программное обеспечение, определяющее структуру программы и облегчающее разработку большого программного проекта.

Фреймворки и Микрофреймворки

Веб-фреймворки разделяют на фреймворки и микрофреймворки.

Фреймворк «из коробки» содержит весь необходимый набор модулей и библиотек для работы.

Микрофреймворки содержат в себе минимально необходимый функционал и при необходимости нужно самостоятельно установить необходимый модуль или библиотеку.



Например, Flask не умеет «из коробки» работать с базами данных, для этого нужно использовать стороннюю библиотеку.

Фреймворки и Микрофреймворки

Установка Flask использует следующие зависимости:



Jinja2 — приложение для обработки шаблонов



Werkzeug — инструмент для работы с WSGI.
WSGI — это стандарт взаимодействия между Python-программой, которая выполняется на стороне сервера, и самим веб-сервером.

Для того, чтобы установить **Flask**, создайте новый проект, разверните виртуальное окружение и затем выполните установку модуля.

Виртуальное окружение

Виртуальное окружение — это изолированная копия интерпретатора Python. Используя виртуальное окружение, вы можете устанавливать пакеты в конкретное виртуальное окружение, не затрагивая глобальную версию интерпретатора.

Для каждого приложения предпочтительно использовать свое виртуальное окружение.

Создание виртуального окружения

MINGW64:/c/dev/flask_app



User@DESKTOP-PVGUDRB MINGW64 /c/dev

```
$ mkdir flask_app
```

User@DESKTOP-PVGUDRB MINGW64 /c/dev

```
$ cd flask_app/
```

User@DESKTOP-PVGUDRB MINGW64 /c/dev/flask_app

```
$ python -m venv venv
```

User@DESKTOP-PVGUDRB MINGW64 /c/dev/flask_app

```
$ source venv/Scripts/activate
```

```
(venv)
```

User@DESKTOP-PVGUDRB MINGW64 /c/dev/flask_app

```
$ deactivate
```

User@DESKTOP-PVGUDRB MINGW64 /c/dev/flask_app

```
$
```

Установка зависимостей

MINGW64:/c/dev/flask_app



(venv)

User@DESKTOP-PVGUDRB MINGW64 /c/dev/flask_app

\$ pip install flask

Collecting flask

Using cached Flask-2.2.3-py3-none-any.whl (101 kB)

Collecting itsdangerous>=2.0

Using cached itsdangerous-2.1.2-py3-none-any.whl (15 kB)

Collecting click>=8.0

Using cached click-8.1.3-py3-none-any.whl (96 kB)

Collecting Werkzeug>=2.2.2

Using cached Werkzeug-2.2.3-py3-none-any.whl (233 kB)

Collecting Jinja2>=3.0

Using cached Jinja2-3.1.2-py3-none-any.whl (133 kB)

Collecting colorama

Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)

Collecting MarkupSafe>=2.0

Using cached MarkupSafe-2.1.2-cp310-cp310-win_amd64.whl (16 kB)

Installing collected packages: MarkupSafe, itsdangerous, colorama, Werkzeug, Jinja2, click, flask

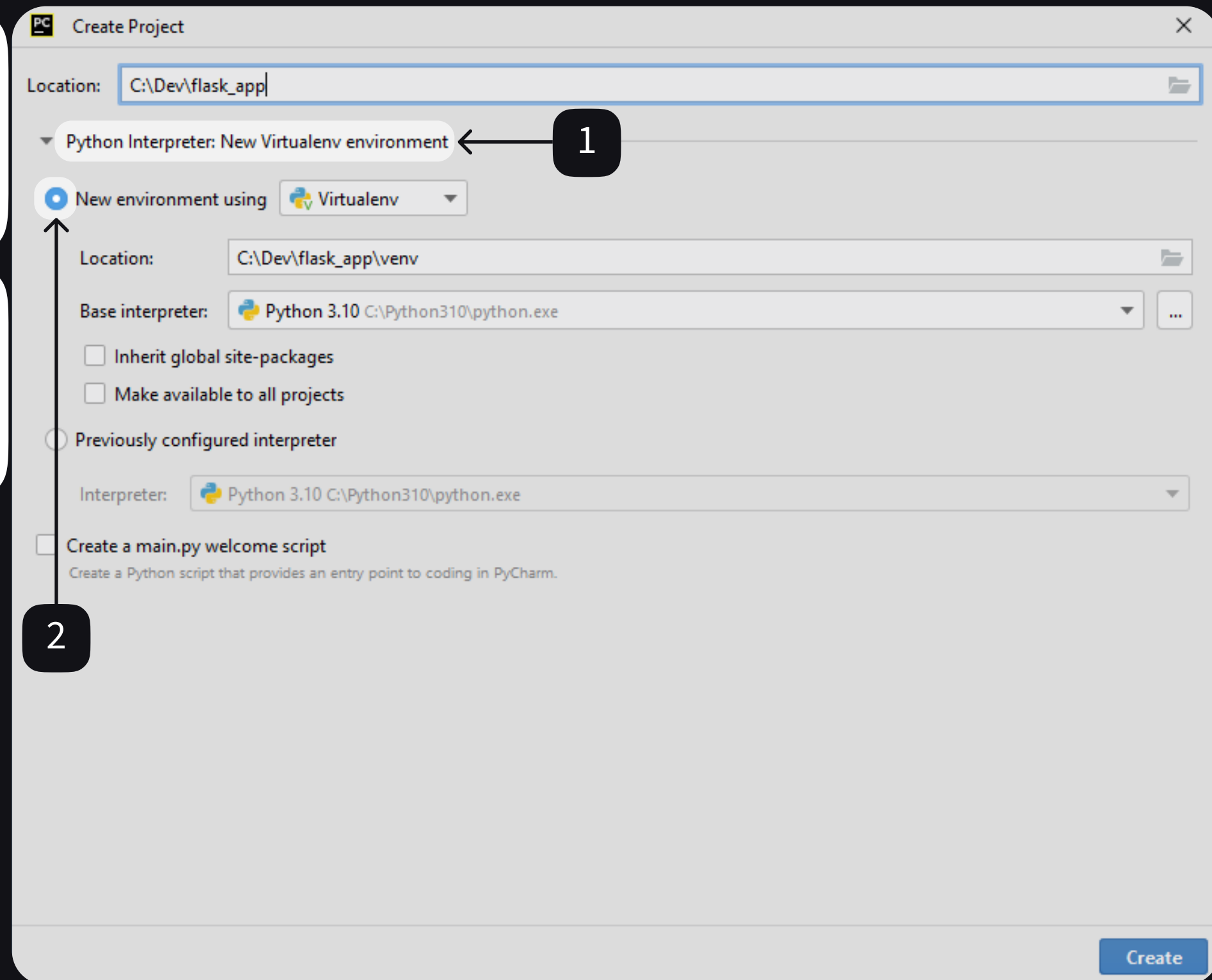
Successfully installed Jinja2-3.1.2 MarkupSafe-2.1.2 Werkzeug-2.2.3 click-8.1.3 colorama-0.4.6 flask-2.2.3 itsdangerous-2.1.2(venv)

User@DESKTOP-PVGUDRB MINGW64 /c/dev/flask_app\$

Виртуальное окружение в PyCharm

Если вы работаете в PyCharm, то создать виртуальное окружение можно сразу при создании проекта.

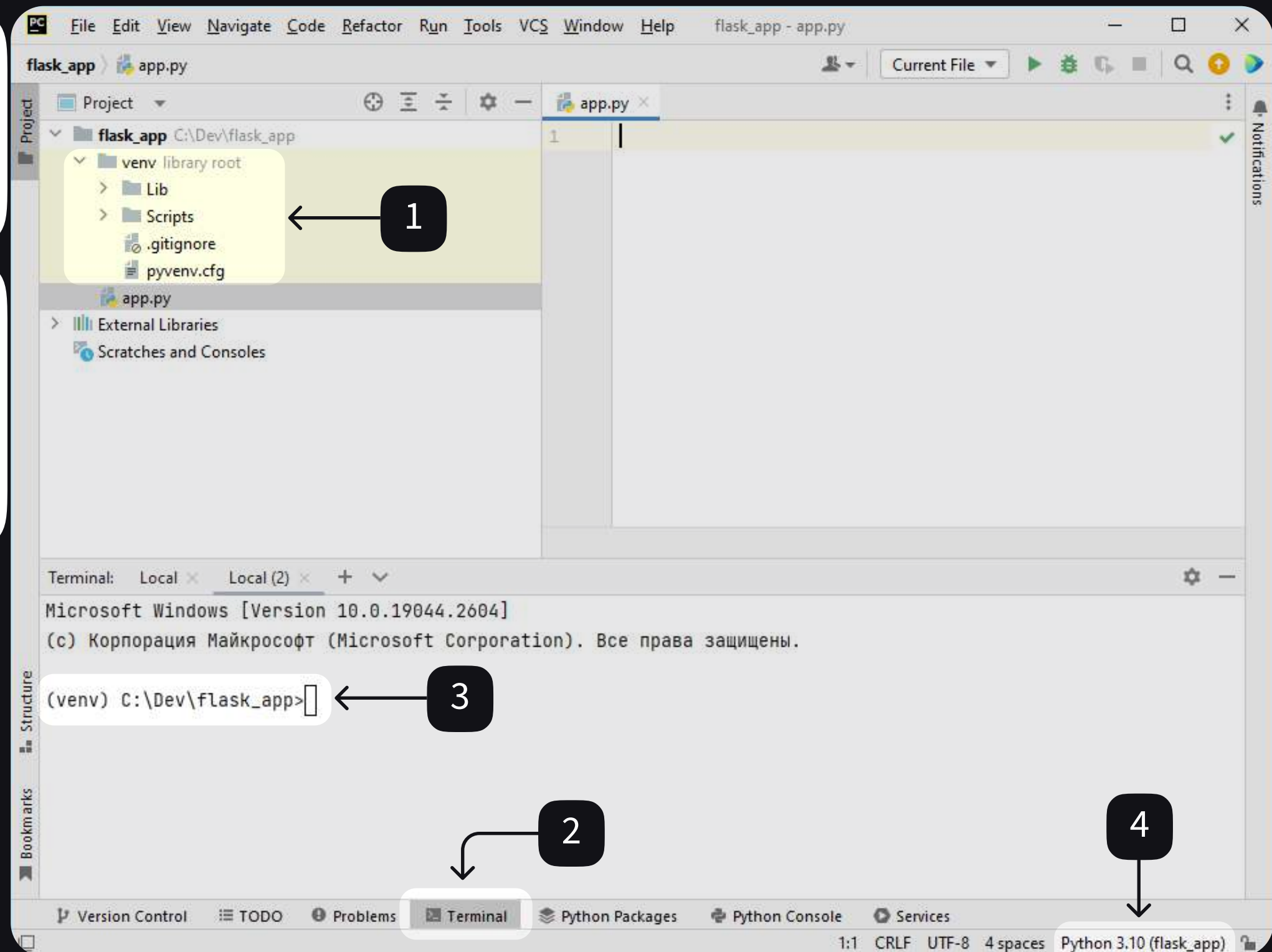
При создании нового проекта выберите пункт: New environment using.



Виртуальное окружение в PyCharm

Обратите внимание на папку venv в структуре проекта — это и есть виртуальное окружение.

При открытии терминала PyCharm автоматически активирует виртуальное окружение.



Hello, world!

Добавим файл main.py:

```
from flask import Flask

app = Flask(__name__)

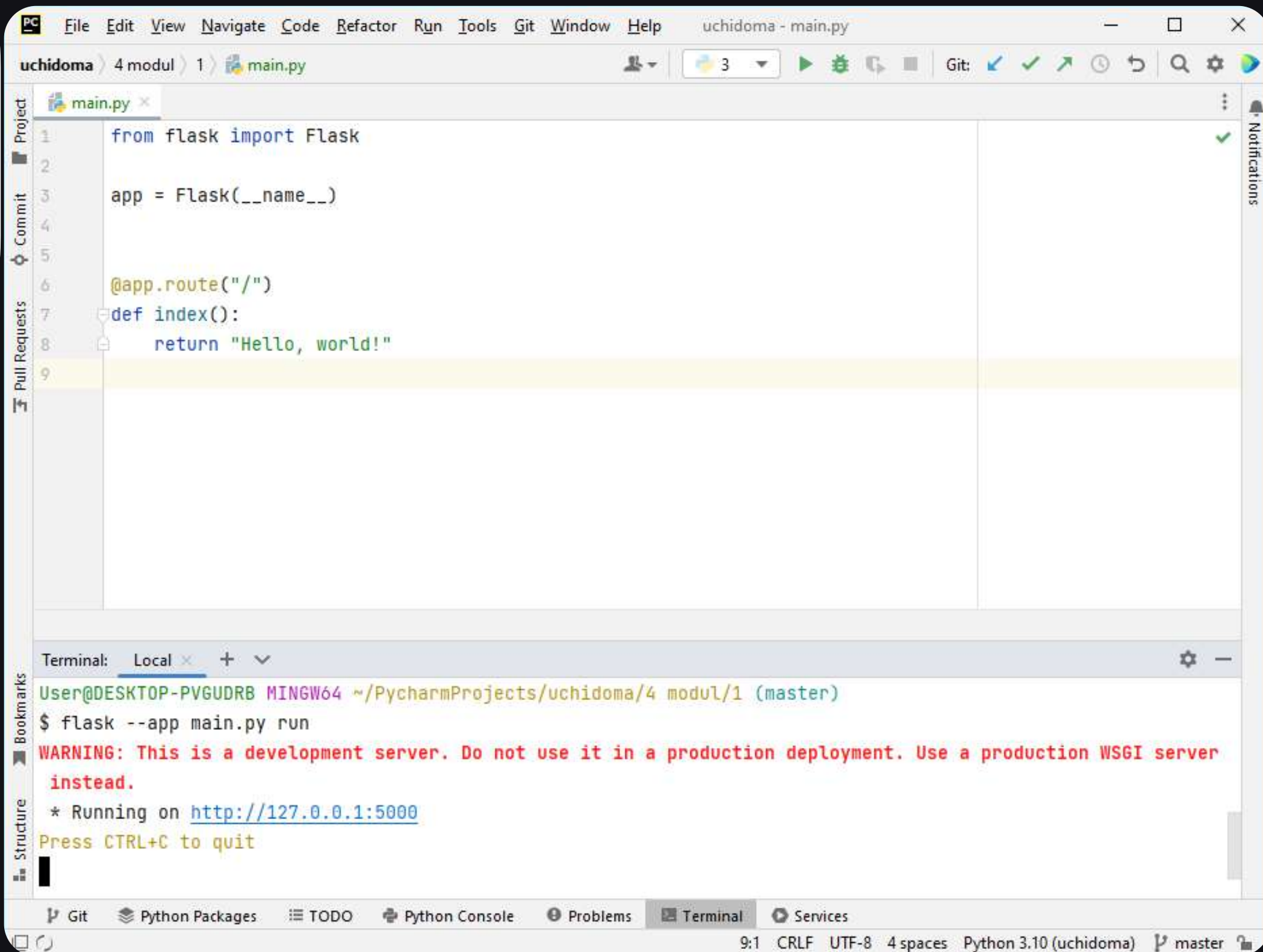
@app.route('/')
def index():
    return 'Hello, World!'
```

Для запуска наберите команду в терминале:

```
flask --app main.py run
```

Результат

После запуска Flask предупреждает, что был запущен development сервер по адресу <http://127.0.0.1:5000/>.



The screenshot shows the PyCharm IDE interface. The main editor window displays the code for `main.py` in the `uchidoma` project. The code is as follows:

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5
6 @app.route("/")
7 def index():
8     return "Hello, world!"
9
```

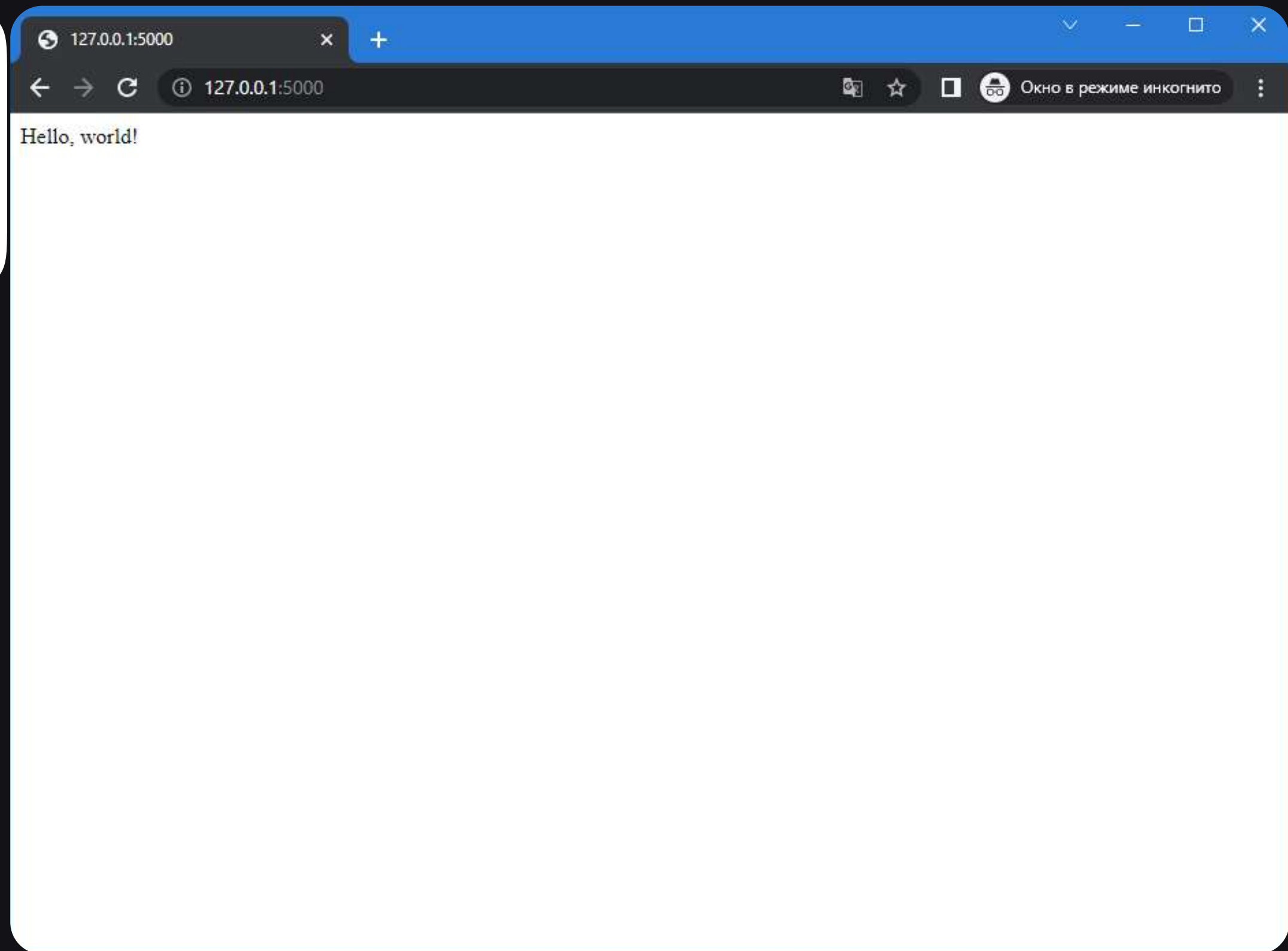
The terminal window at the bottom shows the command `$ flask --app main.py run` and the output:

```
User@DESKTOP-PVGUDRB MINGW64 ~/PycharmProjects/uchidoma/4 modul/1 (master)
$ flask --app main.py run
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
```

The status bar at the bottom indicates the file encoding is UTF-8, 4 spaces, Python 3.10 (uchidoma), and the current branch is master.

Результат

Если открыть браузер и перейти по адресу: <https://127.0.0.1:5000/>, то мы увидим текст Hello, world!



Разберем код примера

```
from flask import Flask
```

Импортируем класс-инициализатор Flask. Экземпляр этого класса будет нашим WSGI-приложением.

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

Создаем экземпляр этого класса, в качестве аргумента передаем ему имя модуля, используя служебную переменную `__name__`. Это необходимо, чтобы Flask знал, где искать шаблоны, файлы и так далее.

```
@app.route('/')  
@app.route('/index')  
def index():  
    return 'Hello, World!'
```

Декоратор `@app.route()` служит для связи функции `index()` с конкретным URL-адресом сайта. Это значит, что, когда браузер будет запрашивать этот адрес (корневой URL или URL `/index`), Flask будет вызывать эту функцию и передавать возвращаемое значение обратно в браузер в качестве ответа. Можно указать несколько декораторов.

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

Запустить development сервер можно, вызвав метод `run()` у экземпляра WSGI приложения.

Режим отладки

```
from flask import Flask

app = Flask(__name__)

counter = 0

@app.route('/')
def hello():
    counter += 1
    return counter

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

Если в приложении случится ошибка, то мы увидим примерно такое сообщение: «**500 Internal Sever Error**» и больше ничего. Но хочется также видеть ошибки и для этого нужно включить режим отладки, запуская сервер с параметром **debug=True**.

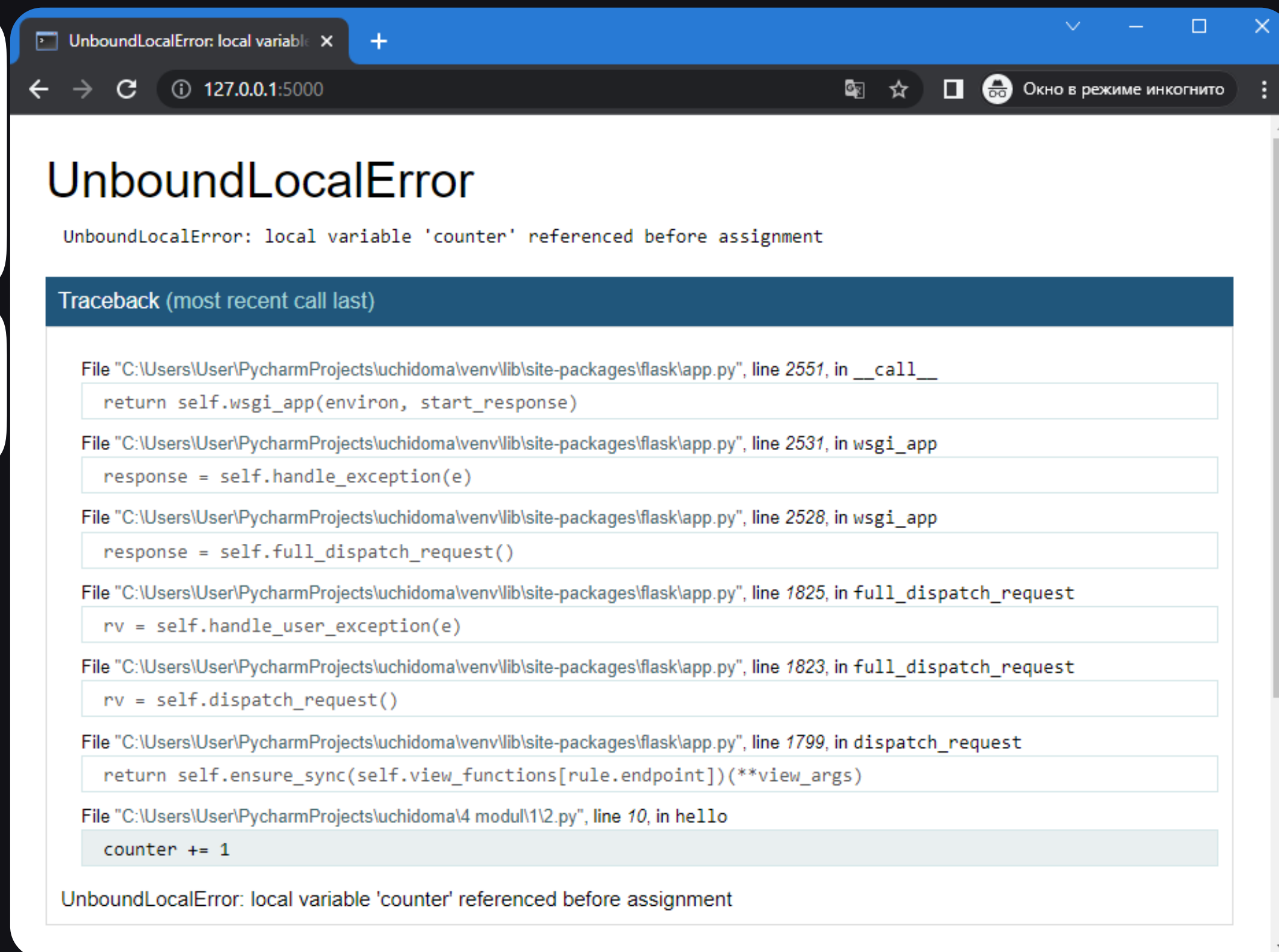
Или выполнив команду в терминале:

```
flask --app main.py --debug run
```

Результат

С включенным отладчиком все ошибки будут отображаться прямо в браузере в удобном для просмотра виде.

Найдите и исправьте ошибки в программе!



Итоги



Flask — микрофреймворк для создания веб-приложений на языке программирования Python.



Микрофреймворк, в отличие от фреймворка, содержит в себе минимально необходимый функционал и при необходимости нужно самостоятельно установить необходимый модуль или библиотеку.



Перед созданием нового flask проекта, создайте и активируйте виртуальное окружение.



Создать виртуальное окружение можно с помощью стандартного модуля `venv`, выполнив команду `python -m venv <имя виртуального окружения>`



При работе в PyCharm создать виртуальное окружение можно сразу при создании проекта.