

Урок 1. Создаем виртуальное окружение и шаблон Flask-проекта

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc97130791)

[СОЗДАЕМ ВИРТУАЛЬНОЕ ОКРУЖЕНИЕ 3](#_Toc97130792)

[СОЗДАЕМ ШАБЛОН FLASK-ПРОЕКТА 6](#_Toc97130793)

[немного о Flask 6](#_Toc97130794)

[cоздаем файл \_\_init\_\_.py 7](#_Toc97130795)

[выполняем тестовый запуск 8](#_Toc97130796)

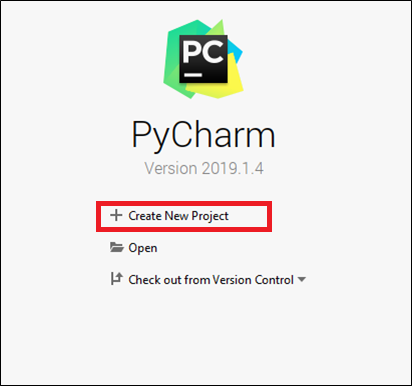
[ЗАКЛЮЧЕНИЕ 11](#_Toc97130797)



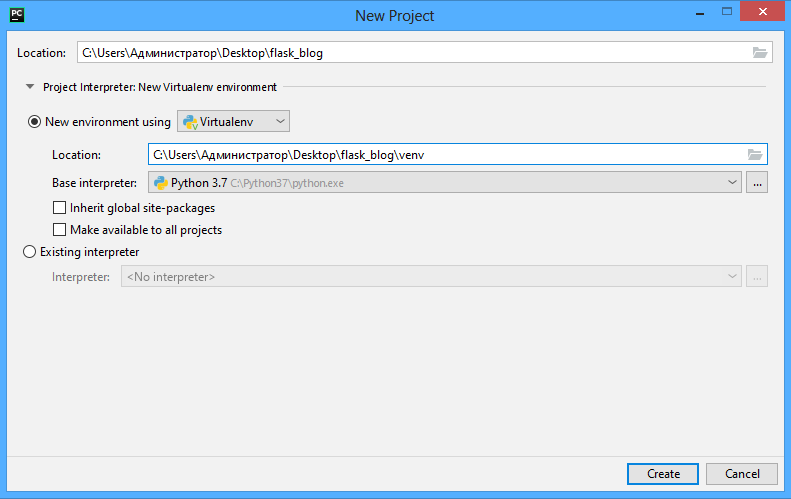
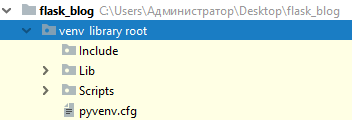
# СОЗДАЕМ ВИРТУАЛЬНОЕ ОКРУЖЕНИЕ

Пошаговый процесс создания Flask-проекта будет показан с использованием среды разработки PyCharm Community.

Запустим PyCharm Community и выберем опцию создания нового проекта. Это будет «чистый» проект на Python, не привязанный к какому-либо фреймворку.

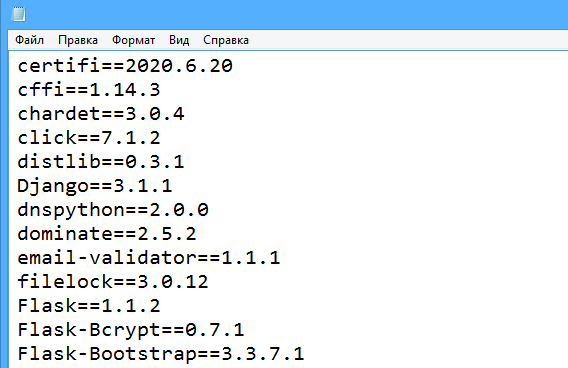


Укажем название проекта, а также отметим, что вместе с проектом будет создано новое виртуальное окружение на базе глобально установленного интерпретатора.

Результат:

Итак, мы создали шаблон проекта, а также виртуальное окружение для него. Но для того, чтобы реализовать необходимый функционал для проекта, нам нужно установить зависимости, т.е. дополнительные библиотеки. Определим их в файле requirements.txt корня нашего проекта. Пока не будем обсуждать роль каждой зависимости из списка. Будем обсуждать их постепенно, в процессе работы над реализацией проекта.

Список зависимостей:

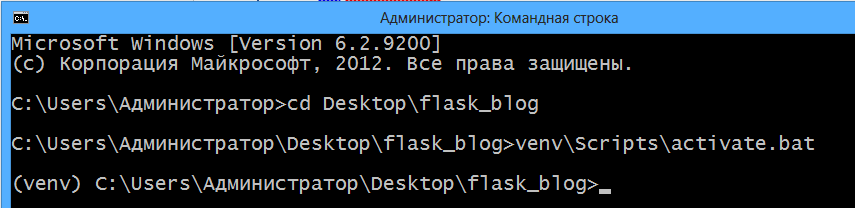


Теперь выполним установку зависимостей в наше виртуальное окружение.

Для начала активируем ранее созданное виртуальное окружение:

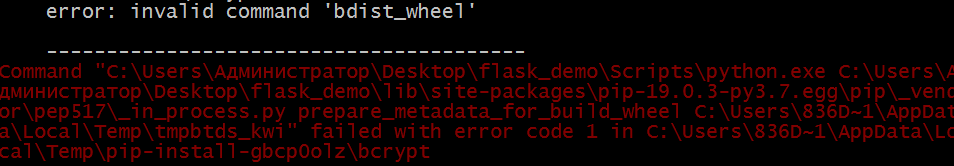
**cd Desktop\flask\_blog**

**venv\Scripts\activate.bat**

****Теперь запустим установку зависимостей и дождемся ее завершения:

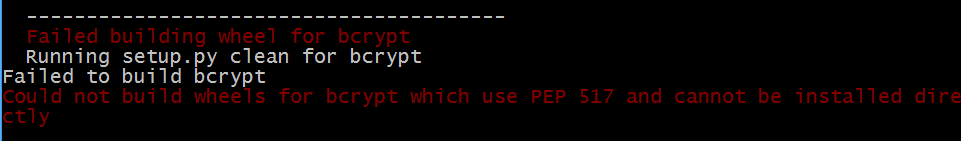
**pip install -r requirements.txt**

При появлении ошибки:

Необходимо в терминале выполнить команду:

**pip install wheel**

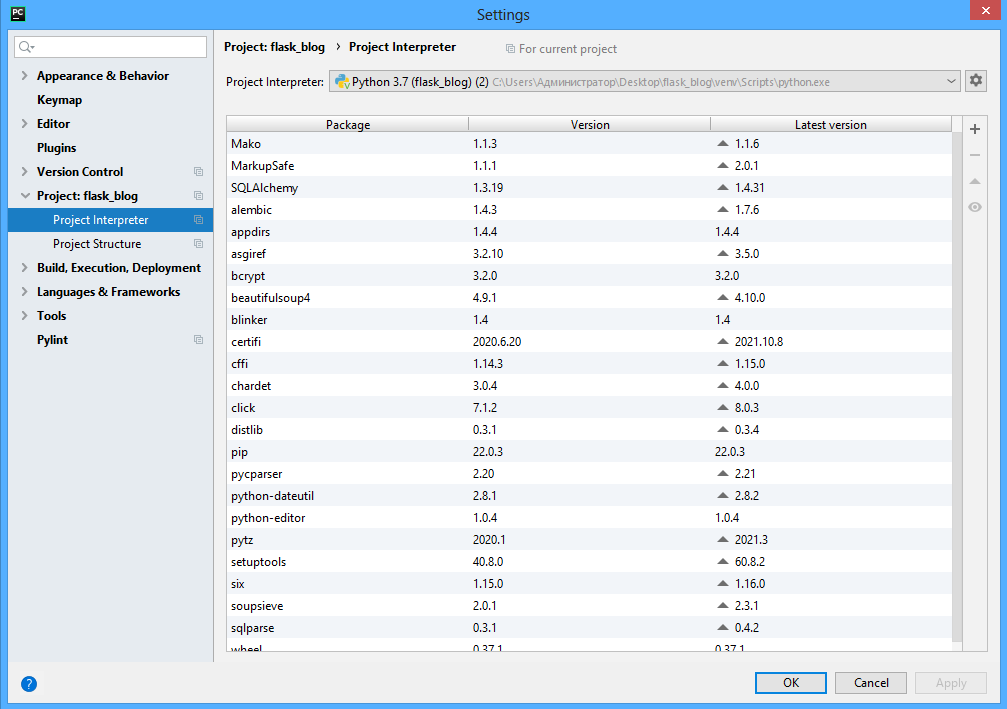
Может возникнуть и такая ошибка:

Для устранения нужно выполнить команды:

**python -m pip install --upgrade pip**

**python -m pip install --no-use-pep517 bcrypt**

После завершения выполнения команд можно проверить наше виртуальное окружение на предмет того, что зависимости действительно установились.



# СОЗДАЕМ ШАБЛОН FLASK-ПРОЕКТА

## немного о Flask

Перед тем, как приступить к созданию шаблона Flask-проекта, стоит отметить что данный фреймворк относится к категории так называемых микрофреймворков, когда разработчику предоставляются лишь некоторые базовые возможности. Этим Flask от отличается от Django, при работе с которым у разработчика существует возможность решать большее количество задач встроенными средствами. Во Flask эти задачи также можно решать, но потребуется установить дополнительные компоненты. Например, в Django есть встроенный шаблонизатор, ORM-система и т.д. При работе с Flask их придется устанавливать. Разработчику предоставляется некоторый каркас приложения, к которому затем «добавляются» необходимые компоненты.

## создаем файл \_\_init\_\_.py

В корневую директорию flask\_blog поместим вложенную директорию с таким же именем flask\_blog. Здесь будут находиться непосредственно компоненты проекта. Во вложенную директорию flask\_blog поместим специальный файл **\_\_init\_\_.py**, тем самым определим директорию в качестве пакета. Вспомним, что под пакетом в Python подразумевается директория, содержащая вложенные директории и модули, а также упомянутый выше файл **\_\_init\_\_.py**. Благодаря пакетам обеспечивается возможность работы с компонентами проекта при импортах посредством определения уровней вложенности (через точку).

Важно отметить, что, если файл **\_\_init\_\_.py** присутствует в директории, он вызывается при импорте этого пакета или входящего в него модуля. Обычно файл **\_\_init\_\_.py** оставляют пустым, но он также может содержать выражения, обеспечивающие инициализацию некоторых важных параметров на уровне пакета.

В файле **\_\_init\_\_.py** мы будем размещать некоторые служебные настройки, настройки импортов и т.д.

Разместим первые несколько выражений:

**Листинг 1. flask\_blog/flask\_blog/\_\_init\_\_.py**

**from** flask **import** Flask  
  
  
**def** create\_app():  
 print(\_\_name\_\_)  
 app = Flask(\_\_name\_\_)  
 **return** app

Важно отметить, что создание Flask-приложения рекомендуется выполнять именно в файле **\_\_init\_\_.py**.

В первой строке мы импортируем класс-конструктор Flask, экземпляр которого будет нашим веб-приложением.

app = Flask(\_\_name\_\_)

При создании экземпляра этого класса передается обязательный параметр \_\_name\_\_, который соответствует названию пакета, в котором Flask будет искать статические файлы, шаблоны и т.д.

## выполняем тестовый запуск

Теперь нам необходимо обеспечить запуск локального сервера с нашим Flask-проектом для отслеживания его работы. Создадим в контейнере проекта запускаемый модуль **run.py**.

**Листинг 2. flask\_blog/run.py**

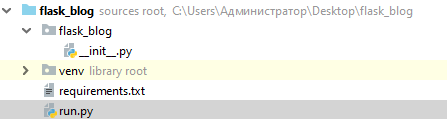
**from** flask\_blog **import** create\_app  
  
app = create\_app()  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 app.run(debug=**True**)

В представленном листинге мы создаем объект Flask-приложения на основе написанной ранее функции create\_app().

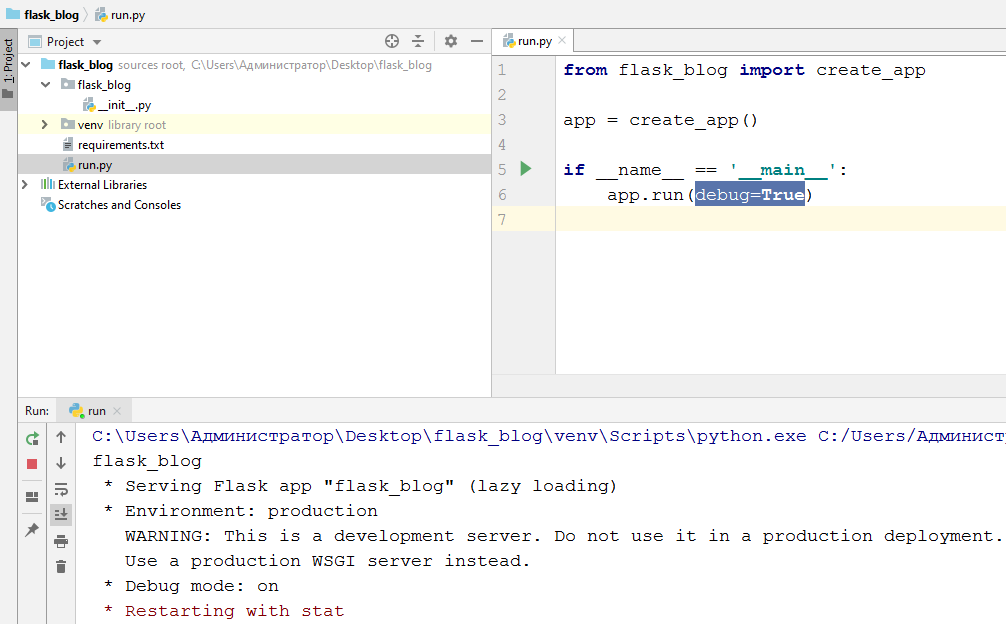
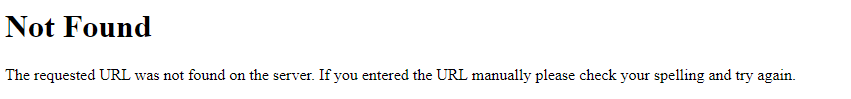
Конструкция:

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

Означает, что сервер будет запущен только при запуске текущего скрипта (**run.py**) из Python-интерпретатора, т.е. в качестве запускаемой программы, а не при импорте **run.py**. Режим отладки (debug=**True**) должен использоваться только на этапе разработки, на локальной машине. В боевых условиях режим отладки нужно отключать.

Текущая структура проекта:

Запустим в PyCharm модуль run.py:

И перейдем в браузер: 

Результат закономерный, ведь наше приложение «не знает» как ему реагировать на те или иные запросы. Нужно настраивать маршрутизацию.

# Вид сверху на рукопожатие над деловыми документамиЗАКЛЮЧЕНИЕ

Друзья, в рамках первого урока мы подготовили шаблон Flask-проекта и реализовали для него виртуальное окружение. Мы установили блок зависимостей, а также создали первые служебные модули. Мы создали объект приложения и выполнили его тестовый запуск.

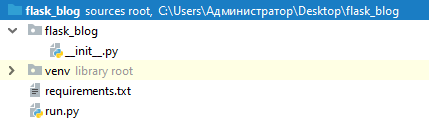
Текущая структура проекта приведена на рис. 1.

Рис. 1. Структура проекта