МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчёт о лабораторной работе №2 по дисциплине основы программной инженерии

Выполнила:
Нестеренко Тамара Антоновна,
2 курс, группа ПИЖ-б-о-20-1,
Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г.

ВЫПОЛНЕНИЕ

1. Практическая часть

```
(base) C:\Users\тома нестеренко>conda create -n Python_LR_2-2 python=3.8
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

Рисунок 1 – Пример создания виртуального окружения с именем репозитория

```
(base) C:\Users\тома нестеренко>conda activate Python_LR_2-2

(Python_LR_2-2) C:\Users\тома нестеренко>
```

Рисунок 2 – Пример активации виртуального окружения

```
(Python_LR_2-2) C:\Users\тома нестеренко>conda install pip
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

Рисунок 3 – Пример установки пакета pip

```
(Python_LR_2-2) C:\Users\тома нестеренко>conda install numpy
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

Рисунок 4 – Пример установки пакета NumPy

```
(Python_LR_2-2) C:\Users\тома нестеренко>conda install pandas
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

Рисунок 5 – Пример установки пакета Pandas

```
(Python_LR_2-2) C:\Users\тома нестеренко>conda install scipy
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

Рисунок 6 – Пример установки пакета SciPy

```
(Python_LR_2-2) C:\Users\тома нестеренко>conda install tensorflow
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

Рисунок 7 – Пример установки пакета TensorFlow

```
(Python_LR_2-2) C:\Users\тома нестеренко>pip freeze > requirements.txt
```

Рисунок 8 – Пример создания файла requirements.txt

```
(Python_LR_2-2) C:\Users\тома нестеренко>conda env export > enviroment_exit.yml
```

Рисунок 9 – Пример создания файла environment.yml

2. Вопросы для защиты

1. **Каким способом можно установить пакет Python, не входящий в стандартную библиотеку?**
   Существует Python Package Index (PyPI) – это репозиторий, открытый для всех разработчиков, в нём можно найти пакеты для решения практических задач.
2. **Как осуществить установку менеджера пакетов pip?**
   Pip – это консольная утилита (без графического интерфейса). После того, как вы её скачаете и установите, она пропишется в PATH и будет доступна для использования.
   Чтобы установить утилиту pip, нужно скачать скрипт get-pip.py
3. **Откуда менеджер пакетов pip по умолчанию устанавливает пакеты?**
   По умолчанию в Linux Pip устанавливает пакеты в /usr/local/lib/python2.7/dist-packages. Использование virtualenv или --user во время установки изменит это местоположение по умолчанию. Важный момент: по умолчанию pip устанавливает пакеты глобально. Это может привести к конфликтам между версиями пакетов.

4. Как установить последнюю версию пакета с помощью pip?

```
$ pip install ProjectName
```

5. Как установить заданную версию пакета с помощью pip?

```
$ pip install ProjectName==3.2
```

6. Как установить пакет из git репозитория (в том числе GitHub) с помощью pip?

```
$ pip install -e git+https://gitrepo.com/ProjectName.git
```

7. Как установить пакет из локальной директории с помощью pip?

```
$ pip install ./dist/ProjectName.tar.gz
```

8. Как удалить установленный пакет с помощью pip?

```
$ pip uninstall ProjectName
```

9. Как обновить установленный пакет с помощью pip?

```
$ pip install --upgrade ProjectName
```

10. Как отобразить список установленных пакетов с помощью pip?

```
$ pip list
```

11. Каковы причины появления виртуальных окружений в языке Python?

Если разработчик работает над проектом не один, а с командой, ему нужно передавать и получать список зависимостей, а также обновлять их на своем компьютере таким образом, чтобы не нарушалась работа других его проектов. Значит нам нужен механизм, который вместе с обменом проектами быстро устанавливал бы локально и все необходимые для них пакеты, при этом не мешая работе других проектов.

Идея виртуального окружения родилась раньше, чем была реализована стандартными средствами Python. Попыток было несколько, но в основу PEP 405 легла утилита virtualenv Яна Бикинга. Были проанализированы возникающие при работе с ней проблемы. После этого в работу интерпретатора Python версии 3.3 добавили их решения. Так был создан встроенный в Python модуль venv, а утилита virtualenv теперь дополнительно использует в своей работе и его.

Как работает виртуальное окружение? Ничего сверхъестественного. В отдельной папке создаётся неполная копия выбранной установки Python. Эта копия является просто набором файлов (например, интерпретатора или ссылки на него), утилит для работы с собой и нескольких пакетов (в том числе pip). Стандартные пакеты при этом не копируются.

12. Каковы основные этапы работы с виртуальными окружениями?

1) Создаём через утилиту новое виртуальное окружение в отдельной папке для выбранной версии интерпретатора Python.
2) Активируем ранее созданное виртуального окружения для работы.
3) Работаем в виртуальном окружении, а именно управляем пакетами используя pip и запускаем выполнение кода.
4) Деактивируем после окончания работы виртуальное окружение.
5) Удаляем папку с виртуальным окружением, если оно нам больше не нужно.

13. Как осуществляется работа с виртуальными окружениями с помощью venv?

Для создания виртуального окружения достаточно дать команду в формате:

```
python3 -m venv <путь к папке виртуального окружения>
```

Создадим виртуальное окружение в папке проекта. Для этого перейдём в корень любого проекта
на Python >= 3.3 и дадим команду:

```
$ python3 -m venv env
```

После её выполнения создастся папка env с виртуальным окружением.
Чтобы активировать виртуальное окружение под Windows нужно дать команду:

```
> env\\Scripts\\activate
```

После активации приглашение консоли изменится. В его начале в круглых скобках будет отображаться имя папки с виртуальным окружением.
При размещении виртуального окружения в папке проекта стоит позаботится об его исключении из репозитория системы управления версиями. Для этого, например, при использовании Git нужно добавить папку в файл .gitignore. Это делается для того, чтобы не засорять проект разными вариантами виртуального окружения.

```
$ python3 -m venv /home/user/envs/project1_env
```

Чтобы переключиться с одного окружения на другое нам нужно выполнить команду деактивации и команду активации другого виртуального окружения.

```
$ deactivate
$ source /home/user/envs/project1_env2/bin/activate
```

14. Как осуществляется работа с виртуальными окружениями с помощью virtualenv?

Для начала пакет нужно установить. Установку можно выполнить командой:

```
# Для python 3
python3 -m pip install virtualenv

# Для единственного python
python -m pip install virtualenv
```

Создание виртуального окружения с утилитой virtualenv отличается от стандартного. Например, создание в текущей папке виртуального окружения для интерпретатора доступного через команду python3 с названием папки окружения env:

```
virtualenv -p python3 env
```

Активация и деактивация такая же, как у стандартной утилиты Python.

```
> env\\Scripts\\activate

(env) > deactivate
```

15. Изучите работу с виртуальными окружениями pipenv. Как осуществляется работа с виртуальными окружениями pipenv?

Грубо говоря, pipenv можно рассматривать как симбиоз утилит pip и venv (или virtualenv), которые работают вместе, пряча многие неудобные детали от конечного пользователя.

Помимо этого pipenv ещё умеет вот такое:

· автоматически находить интерпретатор Python нужной версии (находит даже интерпретаторы, установленные через pyenv и asdf!);

· запускать вспомогательные скрипты для разработки;

· загружать переменные окружения из файла .env;

· проверять зависимости на наличие известных уязвимостей.

Стоит сразу оговориться, что если вы разрабатываете библиотеку (или что-то, что устанавливается через pip, и должно работать на нескольких версиях интерпретатора), то pipenv — не ваш путь. Этот инструмент создан в первую очередь для разработчиков конечных приложений (консольных утилит, микросервисов, веб-сервисов). Формат хранения зависимостей подразумевает работу только на одной конкретной версии интерпретатора (это имеет смысл для конечных приложений, но для библиотек это, как правило, не приемлемо). Для разработчиков библиотек существует другой прекрасный инструмент — poetry.

Установка на Windows, самый простой способ — это установка в домашнюю директорию пользователя:

$ pip install --user pipenv Теперь проверим установку:

$ pipenv --version

pipenv, version 2018.11.26

Если вы получили похожий вывод, значит, всё в порядке.

Инициализация проекта

Давайте создадим простой проект под управлением pipenv. Подготовка:

$ mkdir pipenv_demo

$ cd pipenv_demo

Создать новый проект, использующий конкретную версию Python можно вот такой командой:

$ pipenv --python 3.8

Если же вам не нужно указывать версию так конкретно, то есть шорткаты:

Создает проект с Python 3, версию выберет автоматически.

$ pipenv --three

Аналогично с Python 2.

В 2020 году эта опция противопоказана.

$ pipenv --two

После выполнения одной из этих команд, pipenv создал файл Pipfile и виртуальное окружение где-то в заранее определенной директории (по умолчанию вне директории проекта).

$ cat Pipfile [[source]] name = "pypi"

url = "https://pypi.org/simple" verify_ssl = true

[dev-packages]

[packages] [requires]

python_version = "3.8"

Это минимальный образец Pipfile. В секции [[source]] перечисляются индексы пакетов — сейчас тут только PyPI, но может быть и ваш собственный индекс пакетов. В секциях [packages] и [dev-packages] перечисляются зависимости приложения — те, которые нужны для непосредственной работы приложения (минимум), и те, которые нужны для разработки (запуск тестов, линтеры и прочее). В секции [requires] указана версия интерпретатора, на которой данное приложение может работать.

Если вам нужно узнать, где именно pipenv создал виртуальное окружение (например, для настройки IDE), то сделать это можно вот так:

$ pipenv --py

/Users/and-semakin/.local/share/virtualenvs/pipenv_demo- 1dgGUSFy/bin/python

Управление зависимостями через pipenv

Теперь давайте установим в проект первую зависимость. Делается это при помощи команды pipenv install:

$ pipenv install requests

Давайте посмотрим, что поменялось в Pipfile (здесь и дальше я буду сокращать вывод команд или содержимое файлов при помощи ...):

$ cat Pipfile

...

[packages] requests = "*"

...

В секцию [packages] добавилась зависимость requests с версией * (версия не фиксирована).

А теперь давайте установим зависимость, которая нужна для разработки, например, восхитительный линтер flake8, передав флаг --dev в ту же команду install:

$ pipenv install --dev flake8

$ cat Pipfile

...

[dev-packages] flake8 = "*"

...

Теперь можно увидеть всё дерево зависимостей проекта при помощи команды pipenv graph:

$ pipenv graph flake8==3.7.9

- entrypoints [required: >=0.3.0,<0.4.0, installed: 0.3]

- mccabe [required: >=0.6.0,<0.7.0, installed: 0.6.1]

- pycodestyle [required: >=2.5.0,<2.6.0, installed: 2.5.0]

- pyflakes [required: >=2.1.0,<2.2.0, installed: 2.1.1] requests==2.23.0

- certifi [required: >=2017.4.17, installed: 2020.4.5.1]

- chardet [required: >=3.0.2,<4, installed: 3.0.4]

- idna [required: >=2.5,<3, installed: 2.9]

- urllib3 [required: >=1.21.1,<1.26,!=1.25.1,!=1.25.0, installed: 1.25.9]

Это бывает полезно, чтобы узнать, что от чего зависит, или почему в вашем виртуальном окружении есть определённый пакет.

Также, пока мы устанавливали пакеты, pipenv создал Pipfile.lock, но этот файл длинный и не интересный, поэтому показывать содержимое я не буду.

Удаление и обновление зависимостей происходит при помощи команд pipenv uninstall и pipenv update соответственно. Работают они довольно интуитивно, но если возникают вопросы, то вы всегда можете получить справку при помощи флага --help:

$ pipenv uninstall --help

$ pipenv update --help

Управление виртуальными окружениями

Давайте удалим созданное виртуальное окружение:

$ pipenv --rm

И представим себя в роли другого разработчика, который только присоединился к вашему проекту. Чтобы создать виртуальное окружение и установить в него зависимости нужно выполнить следующую команду:

$ pipenv sync --dev

Эта команда на основе Pipfile.lock воссоздаст точно то же самое виртуальное окружение, что и у других разработчиков проекта.

Если же вам не нужны dev-зависимости (например, вы разворачиваете ваш проект на продакшн), то можно не передавать флаг --dev:

$ pipenv sync

Чтобы "войти" внутрь виртуального окружения, нужно выполнить:

$ pipenv shell (pipenv_demo) $

В этом режиме будут доступны все установленные пакеты, а имена python и pip будут указывать на соответствующие программы внутри виртуального окружения.

Есть и другой способ запускать что-то внутри виртуального окружения без создания нового шелла:

это запустит REPL внутри виртуального окружения

$ pipenv run python

а вот так можно запустить какой-нибудь файл

$ pipenv run python script.py

а так можно получить список пакетов внутри виртуального окружения

$ pipenv run pip freeze

16. Каково назначение файла requirements.txt ? Как создать этот файл? Какой он имеет формат?

Просмотреть список зависимостей мы можем командой: pip freeze > requirements.txt

Имя файла хранения зависимостей requirements.txt выбрано не зря. Оно является стандартной договоренностью и используется некоторыми утилитами автоматически.

Установка пакетов из файла зависимостей в новом виртуальном окружении так же выполняется одной командой:

pip install -r requirements.txt

Все пакеты, которые вы установили перед выполнением команды и предположительно использовали в каком-либо проекте, будут перечислены в файле с именем «requirements.txt». Кроме того, будут указаны их точные версии. Расширение: .txt

16. Каково назначение файла requirements.txt ? Как создать этот файл? Какой он имеет формат?

Можно вручную создать этот файл и наполнить его названиями и версиями нужных пакетов, а также можно использовать команду pip freeze > requirements.txt. Которая создаст requirements.txt наполнив его названиями и версиями тех пакетов, что используются в текущем окружении.

17. В чем преимущества пакетного менеджера conda по сравнению с пакетным менеджером pip?

Основная проблема заключается в том, что pip , easy_install и virtualenv ориентированы на Python. Эти инструменты игнорируют библиотеки зависимостей, реализованные с использованием других языков. Например, XSLT , HDF5 , MKL и другие, которые не имеют setup.py в исходном коде и не устанавливают файлы в директорию site-packages.

Conda же способна управлять пакетами как для Python, так и для C/ C++, R, Ruby, Lua, Scala и других. Conda устанавливает двоичные файлы, поэтому работу по компиляции пакета самостоятельно выполнять не требуется (по сравнению с pip).

Существуют также некоторые различия, если вы заинтересованы в создании собственных пакетов. Например, pip создан на основе setuptools, тогда как conda использует свой собственный формат, который имеет некоторые преимущества (например, статическая компиляция пакета).

18. В какие дистрибутивы Python входит пакетный менеджер conda?

Anaconda, miniconda и PyCharm.

19. Как создать виртуальное окружение conda?

1. Начиная проект, создайте чистую директорию и дайте ей понятное короткое имя.

Для Windows, если использьзуется дистрибутив Anaconda, то необходимо вначале запустить консоль Anaconda Powershell Prompt. Делается

это из системного меню, посредством выбора следующих пунктов: Пуск Anaconda3 (64-bit) Anaconda Powershell Prompt (Anaconda3). В результате будет отображено окно консоли, показанное на рисунке.

Обратите на имя виртуального окружения по умолчанию, которым в данном случае является base. В этом окне необходимо ввести следующую последовательность команд:

mkdir %PROJ_NAME% cd %PROJ_NAME%

copy NUL > main.py

Здесь PROJ_NAME - это переменная окружения, в которую записано имя проекта. Допускается не использовать переменные окружения, а использовать имя проекта вместо $PROJ_NAME или

%PROJ_NAME% .

20. Как активировать и установить пакеты в виртуальное окружение conda?

conda create -n %PROJ_NAME% python=3.7 conda activate %PROJ_NAME%

Установите пакеты, необходимые для реализации проекта. conda install django, pandas

21. Как деактивировать и удалить виртуальное окружение conda?

Для Windows необходимо использовать следующую команду:

```
conda deactivate
```

Если вы хотите удалить только что созданное окружение, выполните:

```
conda remove -n $PROJ_NAME
```

22. Каково назначение файла environment.yml ? Как создать этот файл?

6. Файл `environment.yml` позволит воссоздать окружение в любой нужный момент. Достаточно набрать:

```
conda env create -f environment.yml
```

23. Как создать виртуальное окружение conda с помощью файла environment.yml?

```
conda env export > enviromant.yml
```

24. Самостоятельно изучите средства IDE PyCharm для работы с виртуальными окружениями conda. Опишите порядок работы с виртуальными окружениями conda в IDE PyCharm.

Создавайте отдельное окружение Conda и инсталлируйте только нужные библиотеки для каждого проекта. PyCharm позволяет легко создавать и выбирать правильное окружение.

25. Почему файлы requirements.txt и environment.yml должны храниться в репозитории git?

Предоставляет доступ другим пользователям к файлам.

Ссылка на репозиторий: https://github.com/tamaranesterenko/Python_LR_2-2