

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ
УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций
Институт цифрового развития

ОТЧЁТ

по лабораторной работе №2.14

Дисциплина: «Программирование на Python»

Тема: «Виртуальные окружения»

Вариант 2

Выполнил: студент 2 курса,
группы ИВТ-б-о-21-1
Богдан Александр
Анатольевич

Ставрополь 2022

Цель работы: приобретение навыков по работе с менеджером пакетов pip и виртуальными окружениями с помощью языка программирования Python версии 3.x.

Практическая часть:

1. Создал репозиторий.

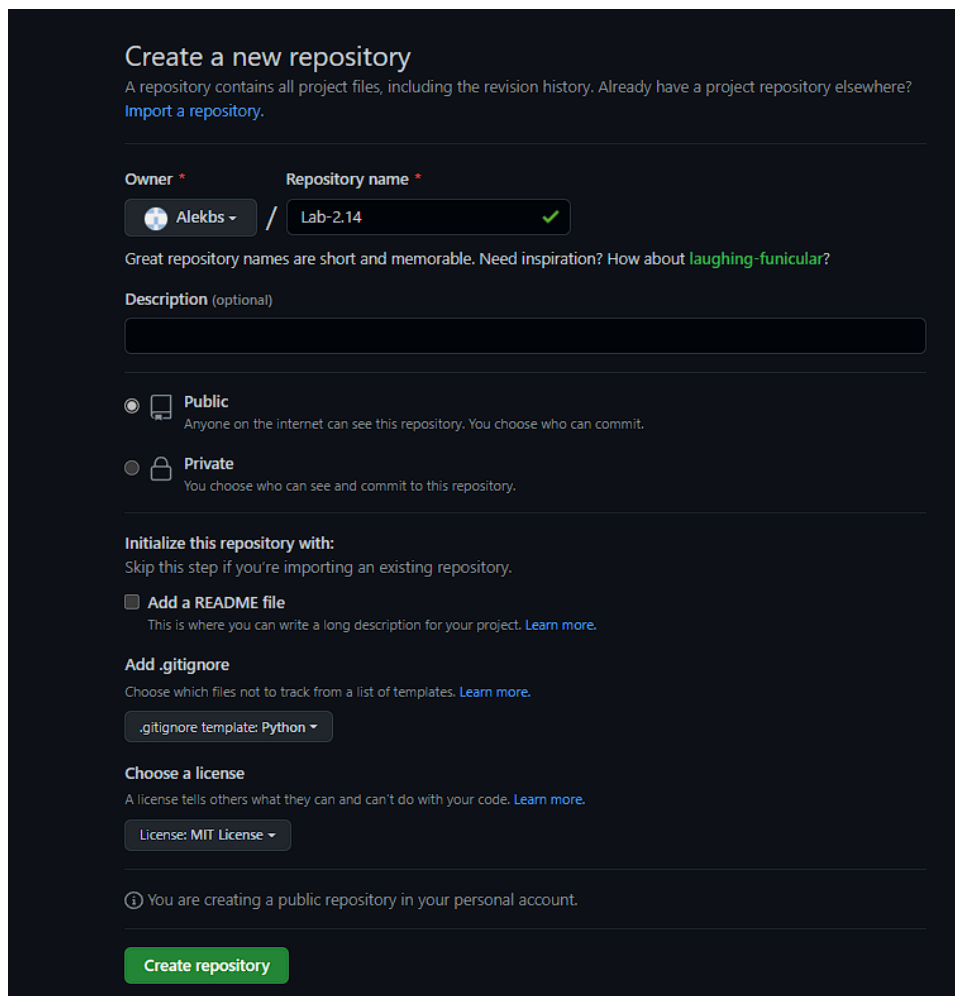
The image shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there are two input fields: 'Owner' with a dropdown menu showing 'Alekbs' and 'Repository name' with a text input showing 'Lab-2.14'. A green checkmark is visible next to the repository name. Below these fields, there is a section for 'Description (optional)' with a text area. Further down, there are two radio buttons for 'Public' (selected) and 'Private'. Below that, there is a section 'Initialize this repository with:' with options to 'Add a README file' and 'Add .gitignore'. The 'Add .gitignore' section shows a dropdown menu with '.gitignore template: Python'. At the bottom, there is a 'Choose a license' section with a dropdown menu showing 'License: MIT License'. A green button labeled 'Create repository' is at the bottom right.

Рисунок 1. Создание репозитория

2. Клонировал репозиторий.

```
D:\WAR\gi>git clone https://github.com/Alekbs/Lab-2.14.git
Cloning into 'Lab-2.14'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Рисунок 2. Клонирование репозитория

3. Организовал свой репозиторий в соответствие с моделью ветвления git-flow.

```

D:\WAR\gi\Lab-2.14>git branch develop

D:\WAR\gi\Lab-2.14>git checkout develop
Switched to branch 'develop'

D:\WAR\gi\Lab-2.14>git flow init

Which branch should be used for bringing forth production releases?
- develop
- main
Branch name for production releases: [main] main

Which branch should be used for integration of the "next release"?
- develop
Branch name for "next release" development: [develop] develop

How to name your supporting branch prefixes?
Feature branches? [feature/] feat
Bugfix branches? [bugfix/] dug
Release branches? [release/] rel
Hotfix branches? [hotfix/] hot
Support branches? [support/] sup
Version tag prefix? [] pre
Hooks and filters directory? [D:/WAR/gi/Lab-2.14/.git/hooks] hook

```

Рисунок 3. Организация репозитория в соответствии с git-flow.

1. Начало работы с виртуальным окружением.

```

D:\WAR\gi\Lab-2.14>pip --version
pip 22.3.1 from C:\Users\aleks\AppData\Local\Programs\Python\Python310\lib\site-packages\pip (python 3.10)

D:\WAR\gi\Lab-2.14>python -m venv env

D:\WAR\gi\Lab-2.14>.\env\scripts\activate

```

Рисунок 4. Создание и активация виртуального окружения

```

(env) D:\WAR\gi\Lab-2.14>pip install black
Collecting black
  Downloading black-22.12.0-cp310-cp310-win_amd64.whl (1.2 MB)
----- 1.2/1.2 MB 1.3 MB/s eta 0:00:00
Collecting click>=8.0.0
  Using cached click-8.1.3-py3-none-any.whl (96 kB)
Collecting platformdirs>=2
  Downloading platformdirs-2.6.0-py3-none-any.whl (14 kB)
Collecting toml>=1.1.0
  Using cached toml-2.0.1-py3-none-any.whl (12 kB)
Collecting mypy_extensions>=0.4.3
  Using cached mypy_extensions-0.4.3-py2.py3-none-any.whl (4.5 kB)
Collecting pathspec>=0.9.0
  Downloading pathspec-0.10.3-py3-none-any.whl (29 kB)
Collecting colorama
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: mypy_extensions, toml, platformdirs, pathspec, colorama, click, black
Successfully installed black-22.12.0 click-8.1.3 colorama-0.4.6 mypy_extensions-0.4.3 pathspec-0.10.3 platformdirs-2.6.0
tomli-2.0.1

[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

```

Рисунок 5. Установка пакета

```

(env) D:\WAR\gi\Lab-2.14>python -m pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.17.1-py3-none-any.whl (8.8 MB)
----- 8.8/8.8 MB 1.3 MB/s eta 0:00:00
Requirement already satisfied: platformdirs<3,>=2.4 in d:\war\gi\lab-2.14\env\lib\site-packages (from virtualenv) (2.6.0)
Collecting filelock<4,>=3.4.1
  Downloading filelock-3.8.2-py3-none-any.whl (10 kB)
Collecting distlib<1,>=0.3.6
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)
----- 468.5/468.5 kB 1.4 MB/s eta 0:00:00
Installing collected packages: distlib, filelock, virtualenv
Successfully installed distlib-0.3.6 filelock-3.8.2 virtualenv-20.17.1

[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

```

Рисунок 6. Установка virtualenv

```
(env) D:\WAR\gi\Lab-2.14>pip freeze
black==22.12.0
click==8.1.3
colorama==0.4.6
distlib==0.3.6
filelock==3.8.2
mypy-extensions==0.4.3
pathspec==0.10.3
platformdirs==2.6.0
tomli==2.0.1
virtualenv==20.17.1
```

Рисунок 7. Список пакетных зависимостей

```
(env) D:\WAR\gi\Lab-2.14>pip freeze > requirements.txt
(env) D:\WAR\gi\Lab-2.14>
```

Рисунок 8. Сохранение списка в файл

requirements.txt – Блокнот

Файл Правка Формат Вид Справка

```
black==22.12.0
click==8.1.3
colorama==0.4.6
distlib==0.3.6
filelock==3.8.2
mypy-extensions==0.4.3
pathspec==0.10.3
platformdirs==2.6.0
tomli==2.0.1
virtualenv==20.17.1
```

Рисунок 9. Файл requirements.txt

```
(env) D:\WAR\gi\Lab-2.14>deactivate
D:\WAR\gi\Lab-2.14>
```

Рисунок 10. Деактивация виртуального окружения

2. Управление пакетами с помощью Conda.

```
(base) PS C:\Users\aleks> mkdir %virtenv9%

Каталог: C:\Users\aleks

Mode                LastWriteTime         Length Name
----                -
d-----          12.12.2022    23:45             %virtenv9%

(base) PS C:\Users\aleks> cd %virtenv9%
```

Рисунок 11. Создание директории %virtualenv9%

```
(base) PS C:\Users\aleks\%virtenv9%> conda create -n %virtualenv9% python=3.9
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
current version: 22.9.0
latest version: 22.11.1

Please update conda by running

$ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\aleks\anaconda3\envs\%virtualenv9%

added / updated specs:
- python=3.9

The following packages will be downloaded:
```

package	build	
ca-certificates-2022.10.11	haa95532_0	125 KB
certifi-2022.9.24	py39haa95532_0	154 KB

Рисунок 12. Создание виртуального окружения

```
(base) PS C:\Users\aleks\%virtenv9%> conda activate %virtualenv9%
(%virtualenv9%) PS C:\Users\aleks\%virtenv9%> _
```

Рисунок 13. Активация виртуального окружения

```
(%virtualenv9%) PS C:\Users\aleks\%virtenv9%> conda install pip, NumPy, Pandas, SciPy
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
current version: 22.9.0
latest version: 22.11.1

Please update conda by running

$ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\aleks\anaconda3\envs\%virtualenv9%

added / updated specs:
- numpy
- pandas
- pip
- scipy

The following packages will be downloaded:
```

package	build

Рисунок 14. Установка пакетов

```
(%virtualenv9%) PS C:\Users\aleks\%virtenv9%> conda install TensorFlow
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

Рисунок 15. Установка пакета TensorFlow

```
google-auth-2.6.0      83 KB | ##### 100%
flatbuffers-2.0.0     1.4 MB | ##### 100%
requests-oauthlib-1.  23 KB | ##### 100%
keras-2.9.0           1.5 MB | ##### 100%
flit-core-3.6.0       42 KB | ##### 100%
google-pasta-0.2.0    46 KB | ##### 100%
opt_einsum-3.3.0      57 KB | ##### 100%
google-auth-oauthlib  18 KB | ##### 100%
yarl-1.8.1            80 KB | ##### 100%
libcurl-7.86.0        319 KB | ##### 100%
cffi-1.15.1           238 KB | ##### 100%
termcolor-2.1.0       12 KB | ##### 100%
absl-py-1.3.0         171 KB | ##### 100%
multidict-6.0.2       46 KB | ##### 100%
tensorboard-data-ser  17 KB | ##### 100%
cachetools-4.2.2     13 KB | ##### 100%
tensorboard-plugin-w  671 KB | ##### 100%
aiosignal-1.2.0       12 KB | ##### 100%
rsa-4.7.2             28 KB | ##### 100%
libprotobuf-3.20.1    2.0 MB | ##### 100%
oauthlib-3.2.1        194 KB | ##### 100%
attrs-22.1.0          84 KB | ##### 100%
frozenlist-1.3.3      40 KB | ##### 100%
scipy-1.7.3           13.9 MB | ##### 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Retrieving notices: ...working... done
(%virtualenv9%) PS C:\Users\aleks\%virtenv9%> _
```

Рисунок 16. Установка пакета TensorFlow

Ошибки при установке данного пакета не возникает.

```
(%virtualenv9%) PS C:\Users\aleks\%virtenv9%> conda env export >environment.yml
(%virtualenv9%) PS C:\Users\aleks\%virtenv9%>
```

Рисунок 17. Формирование файла environment.yml

```
(%virtualenv9%) PS C:\Users\aleks\%virtenv9%> conda list -e > requirements.txt
(%virtualenv9%) PS C:\Users\aleks\%virtenv9%> _
```

Рисунок 18. Формирование файла requirements.txt

```
(%virtualenv9%) PS C:\Users\aleks\%virtenv9%> conda deactivate
(base) PS C:\Users\aleks\%virtenv9%> _
```

Рисунок 19. Деактивация виртуального окружения

requirements.txt – Блокнот

Файл Правка Формат Вид Справка

```
# This file may be used to create an environment using:
# $ conda create --name <env> --file <this file>
# platform: win-64
_tflow_select=2.3.0=mk1
abseil-cpp=20211102.0=hd77b12b_0
abs1-py=1.3.0=py39haa95532_0
aiohttp=3.8.1=py39h2bbff1b_1
aiosignal=1.2.0=pyhd3eb1b0_0
asgiref=3.5.2=py39haa95532_0
astunparse=1.6.3=py_0
async-timeout=4.0.2=py39haa95532_0
attrs=22.1.0=py39haa95532_0
blas=1.0=mk1
blinker=1.4=py39haa95532_0
bottleneck=1.3.5=py39h080aedc_0
brotlipy=0.7.0=py39h2bbff1b_1003
ca-certificates=2022.10.11=haa95532_0
cachetools=4.2.2=pyhd3eb1b0_0
certifi=2022.9.24=py39haa95532_0
cffi=1.15.1=py39h2bbff1b_2
charset-normalizer=2.0.4=pyhd3eb1b0_0
click=8.0.4=py39haa95532_0
colorama=0.4.5=py39haa95532_0
cryptography=38.0.1=py39h21b164f_0
dataclasses=0.8=pyh6d0b6a4_7
django=4.1=py39haa95532_0
fftw=3.3.9=h2bbff1b_1
flatbuffers=2.0.0=h6c2663c_0
```

Рисунок 15. Файл requirements.txt

```
name: '%python9%'
channels:
  - defaults
dependencies:
  - _tflow_select=2.3.0=mk1
  - abseil-cpp=20211102.0=hd77b12b_0
  - abs1-py=1.3.0=py39haa95532_0
  - aiohttp=3.8.1=py39h2bbff1b_1
  - aiosignal=1.2.0=pyhd3eb1b0_0
  - asgiref=3.5.2=py39haa95532_0
  - astunparse=1.6.3=py_0
  - async-timeout=4.0.2=py39haa95532_0
  - attrs=22.1.0=py39haa95532_0
  - blas=1.0=mk1
  - blinker=1.4=py39haa95532_0
  - bottleneck=1.3.5=py39h080aedc_0
  - brotlipy=0.7.0=py39h2bbff1b_1003
  - ca-certificates=2022.10.11=haa95532_0
  - cachetools=4.2.2=pyhd3eb1b0_0
  - certifi=2022.9.24=py39haa95532_0
  - cffi=1.15.1=py39h2bbff1b_2
  - charset-normalizer=2.0.4=pyhd3eb1b0_0
  - click=8.0.4=py39haa95532_0
  - colorama=0.4.5=py39haa95532_0
  - cryptography=38.0.1=py39h21b164f_0
  - dataclasses=0.8=pyh6d0b6a4_7
  - django=4.1=py39haa95532_0
  - fftw=3.3.9=h2bbff1b_1
  - flatbuffers=2.0.0=h6c2663c_0
  - frozenlist=1.2.0=py39h2bbff1b_0
  - gast=0.5.3=pyhd3eb1b0_0
  - giflib=5.2.1=h62dcd97_0
  - google-auth=2.6.0=pyhd3eb1b0_0
  - google-auth-oauthlib=0.4.4=pyhd3eb1b0_0
```

Рисунок 16. Файл eniveronment.ym

Контрольные вопросы:

1. Каким способом можно установить пакет Python, не входящий в стандартную библиотеку?

Существует так называемый Python Package Index (PyPI) – это репозиторий, открытый для всех Python разработчиков, в нем вы можете найти пакеты для решения практически любых задач. Там также есть возможность выкладывать свои пакеты. Для скачивания и установки используется специальная утилита, которая называется `pip`.

2. Как осуществить установку менеджера пакетов `pip`?

При развертывании современной версии Python (начиная с Python 2.7.9 и Python 3.4), `pip` устанавливается автоматически. Но если, по какой-то причине, `pip` не установлен на вашем ПК, то сделать это можно вручную.

Будем считать, что Python у вас уже установлен, теперь необходимо установить `pip`. Для того, чтобы это сделать, скачайте скрипт `get-pip.py`

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

и выполните его.

```
$ python get-pip.py
```

При этом, вместе с `pip` будут установлены `setuptools` и `wheels`. `Setuptools` – это набор инструментов для построения пакетов Python. `Wheels` – это формат дистрибутива для пакета Python. Обсуждение этих составляющих выходит за рамки урока, поэтому мы не будем на них

останавливаться.

3. Откуда менеджер пакетов `pip` по умолчанию устанавливает пакеты?

По умолчанию менеджер пакетов `pip` скачивает пакеты из Python Package Index (PyPI).

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?

В системе для интерпретатора Python может быть установлена глобально. Если вы уже сталкивались с этой проблемой, то уже задумались, что для каждого проекта нужна своя "песочница", которая изолирует зависимости. Такая "песочница" придумана и называется "виртуальным окружением" или "виртуальной средой". Только одна версия пакета. Это порождает ряд проблем.

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 виртуальное окружение 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>
```

Обычно папку для виртуального окружения называют env или venv. В описании команды выше явно указан интерпретатор версии 3.x. Под Windows и некоторыми другими операционными системами это будет просто python.

Чтобы активировать виртуальное окружение под нужно:

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

Просто под Windows мы вызываем скрипт активации напрямую.

Чтобы переключиться с одного окружения на другое нам нужно выполнить команду деактивации и команду активации другого виртуального окружения, например, так:

```
$ deactivate
```

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

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

Зачем нам нужно уметь работать с утилитой virtualenv? Ведь мы уже научились работать со стандартным модулем Python venv. Просто он очень распространён и поддерживает большее число вариантов и версий интерпретатора Python, например, PyPy и CPython.

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`.

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

Для формирования и развертывания пакетных зависимостей используется утилита `pip`.

Основные возможности `pipenv`:

- Создание и управление виртуальным окружением
- Синхронизация пакетов в `Pipfile` при установке и удалении пакетов
- Автоматическая погрузка переменных окружения из `.env` файла

После установки `pipenv` начнется работа с окружением. Его можно создать в любой папке. Достаточно установить любой пакет внутри папки. Используем `requests`, он автоматически установит окружение и создаст `Pipfile` и `Pipfile.lock`.

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

Просмотреть список зависимостей мы можем командой:

```
pip freeze
```

Что бы его сохранить, нужно перенаправить вывод команды в файл:

```
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
```

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 и Miniconda.

19. Как создать виртуальное окружение conda?

1. Начиная проект, создайте чистую директорию и дайте ей понятное короткое имя. Для Linux это будет соответствовать набору команд:

```
mkdir $PROJ_NAME
```

```
cd $PROJ_NAME
```

```
touch README.md main.py
```

Для Windows, если используется 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).

Создайте чистое conda-окружение с таким же именем:

```
conda create -n $PROJ_NAME python=3.7
```

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

```
source activate $PROJ_NAME
```

21. Как деактивировать и удалить виртуальное окружение conda?

```
conda deactivate
```

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

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

Файл environment.yml позволит воссоздать окружение в любой нужный момент.

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

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

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

Работа с виртуальными окружениями в PyCharm зависит от способа взаимодействия с виртуальным окружением: Создаём проект со своим собственным виртуальным окружением, куда затем будут устанавливаться необходимые библиотеки. Предварительно создаём виртуальное окружение, куда установим нужные библиотеки. И затем при создании проекта в PyCharm можно будет его выбирать, т.е. использовать для нескольких проектов. Для первого способа ход работы следующий: запускаем PyCharm и в окне приветствия выбираем Create New Project. В мастере создания проекта, указываем в поле Location путь расположения создаваемого проекта. Имя конечной директории также является именем проекта. Далее разворачиваем параметры окружения, щелкая по Project Interpreter. И выбираем New environment using Virtualenv. Путь расположения окружения генерируется автоматически. И нажимаем на Create. Теперь установим библиотеки, которые будем использовать в программе. С помощью главного меню переходим в настройки File → Settings. Где переходим в Project: project_name → Project Interpreter. Выходим из настроек. Для запуска программы, необходимо создать профиль с конфигурацией. Для этого в верхнем правом углу нажимаем на кнопку Add Configuration. Откроется окно Run/Debug

Configurations, где нажимаем на кнопку с плюсом (Add New Configuration) в правом верхнем углу и выбираем Python. Далее указываем в поле Name имя конфигурации и в поле Script path расположение Python файла с кодом программы. В завершение нажимаем на Apply, затем на ОК. Для второго способа необходимо сделать следующее: на экране приветствия в нижнем правом углу через Configure → Settings переходим в настройки. Затем переходим в раздел Project Interpreter. В верхнем правом углу есть кнопка с шестерёнкой, нажимаем на неё и выбираем Add, создавая новое окружение. И указываем расположение для нового окружения. Нажимаем на ОК. Далее в созданном окружении устанавливаем нужные пакеты. И выходим из настроек. В окне приветствия выбираем Create New Project. В мастере создания проекта, указываем имя расположения проекта в поле Location. Разворачиваем параметры окружения, щелкая по Project Interpreter, где выбираем Existing interpreter и указываем нужное нам окружение. Далее создаем конфигурацию запуска программы, также как создавали для раннее. После чего можно выполнить программу

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

Чтобы пользователи, которые скачивают какие-либо программы, скрипты, модули могли без проблем посмотреть, какие пакеты им нужно установить дополнительно для корректной работы. За описание о наличии какихлибо пакетов в среде как раз и отвечают файлы requirements.txt и environment.yml

Вывод: в результате выполнения работы были приобретены навыки по работе с менеджером пакетов pip и виртуальными окружениями с помощью языка программирования Python версии 3.x.