

Практическая №3

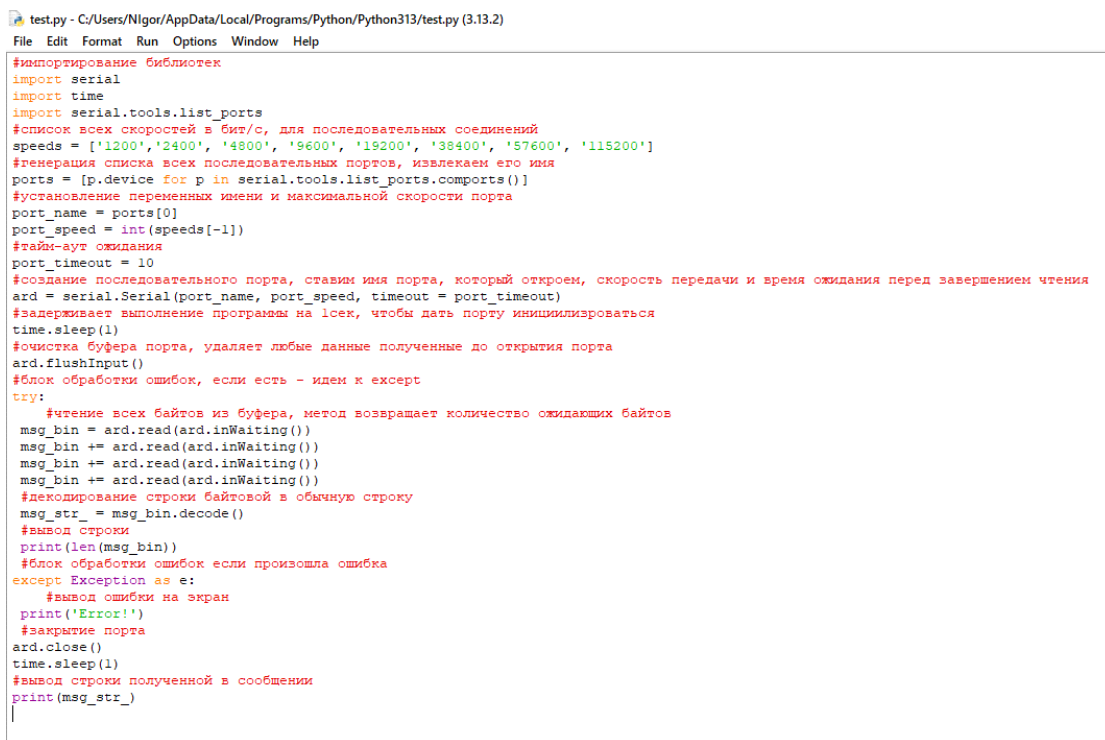
Реализация программы работы с последовательным портом средствами Python

Цель: закрепить навыки работы с Dockerfile, а также познакомиться с программой Anaconda и Jupyter в ней.

Ход работы

Нам необходимо создать файл python, разместив там программу, которая показывает, какие свободные последовательные порты доступны.

Мы создаем файл с помощью IDLE python, в котором вписываем программу из методических указаний, по условиям сказано описать каждую строку (рис. 1):

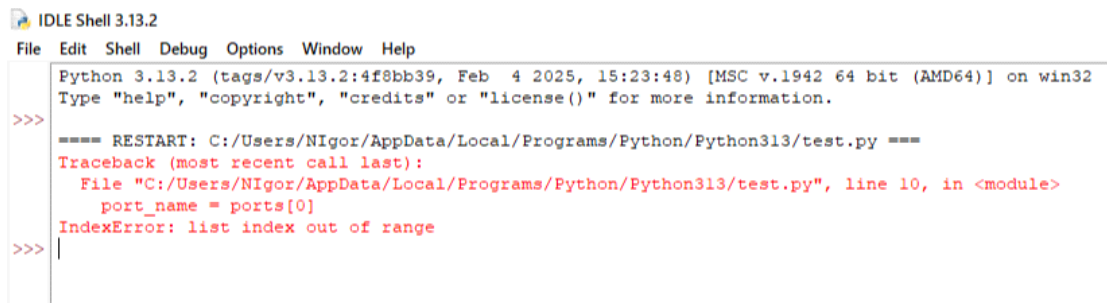


```
test.py - C:/Users/Nlgor/AppData/Local/Programs/Python/Python313/test.py (3.13.2)
File Edit Format Run Options Window Help

#импортирование библиотек
import serial
import time
import serial.tools.list_ports
#список всех скоростей в бит/с, для последовательных соединений
speeds = ['1200', '2400', '4800', '9600', '19200', '38400', '57600', '115200']
#генерация списка всех последовательных портов, извлекаем его имя
ports = [p.device for p in serial.tools.list_ports.comports()]
#установка переменных имени и максимальной скорости порта
port_name = ports[0]
port_speed = int(speeds[-1])
#тайм-аут ожидания
port_timeout = 10
#создание последовательного порта, ставим имя порта, который откроем, скорость передачи и время ожидания перед завершением чтения
ard = serial.Serial(port_name, port_speed, timeout = port_timeout)
#задерживает выполнение программы на 1сек, чтобы дать порту инициализироваться
time.sleep(1)
#очистка буфера порта, удаляет любые данные полученные до открытия порта
ard.flushInput()
#блок обработки ошибок, если есть - идем к эксепт
try:
    #чтение всех байтов из буфера, метод возвращает количество ожидающих байтов
    msg_bin = ard.read(ard.inWaiting())
    msg_bin += ard.read(ard.inWaiting())
    msg_bin += ard.read(ard.inWaiting())
    msg_bin += ard.read(ard.inWaiting())
    #декодирование строки байтовой в обычную строку
    msg_str_ = msg_bin.decode()
    #вывод строки
    print(len(msg_bin))
    #блок обработки ошибок если произошла ошибка
except Exception as e:
    #вывод ошибки на экран
    print('Error!')
    #закрытие порта
    ard.close()
    time.sleep(1)
    #вывод строки полученной в сообщении
    print(msg_str_)
```

Рисунок 1 – программа

При ее запуске, скорее всего, покажется 0, либо программа вообще выдаст ошибку о том, что последовательных портов нет. На рисунке 2 видим второй вариант.



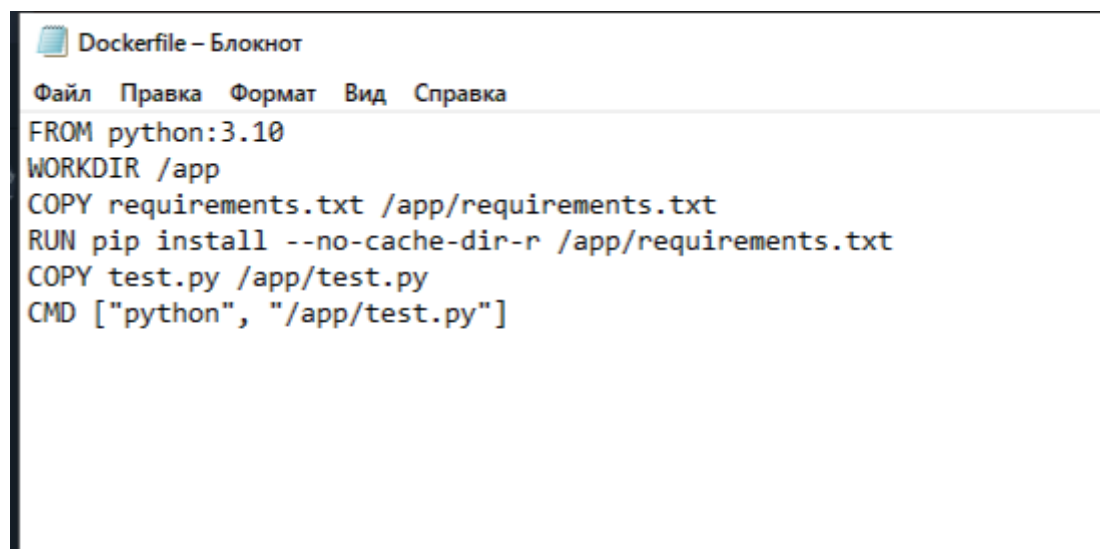
```
Python 3.13.2 (tags/v3.13.2:4f8bb39, Feb  4 2025, 15:23:48) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
==== RESTART: C:/Users/Nigor/AppData/Local/Programs/Python/Python313/test.py ====
Traceback (most recent call last):
  File "C:/Users/Nigor/AppData/Local/Programs/Python/Python313/test.py", line 10, in <module>
    port_name = ports[0]
IndexError: list index out of range
>>> |
```

Рисунок 2 - вылет программы

Далее нам нужно упаковать программу в Docker контейнер. Делаем следующее:

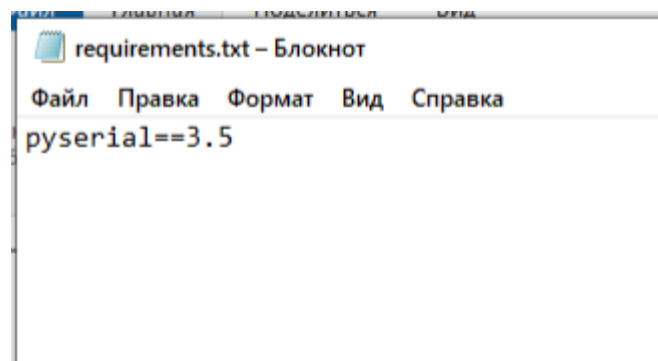
1. Устанавливаем Docker Desktop на ПК

2. Внутри PowerShell вписываем «cd chereshnya», «notepad Dockerfile», нам открывается блокнот этого файла, в который мы вписываем условия и «notepad requirements.txt», в который мы пишем наше требование к pyserial (рис. 2 и 3).



```
Dockerfile - Блокнот
Файл  Правка  Формат  Вид  Справка
FROM python:3.10
WORKDIR /app
COPY requirements.txt /app/requirements.txt
RUN pip install --no-cache-dir -r /app/requirements.txt
COPY test.py /app/test.py
CMD ["python", "/app/test.py"]
```

Рисунок 2 – Dockerfile



```
requirements.txt - Блокнот
Файл  Правка  Формат  Вид  Справка
pyserial==3.5
```

Рисунок 3 – requirements

3. Далее мы запускаем билд следующей командой:

docker build -t dockerfile .

```
PS C:\Users\Nlgor\chereshnya> docker build -t dockerfile .
[+] Building 1.8s (9/9) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 130B 0.0s
=> [internal] load metadata for docker.io/library/python:3.10 1.5s
=> [auth] library/python:pull token for registry-1.docker.io 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/3] FROM docker.io/library/python:3.10@sha256:76f22e4ce53774c1f5eb0ba145edb57b908e7aa329fee75eca69b511c1d0c 0.0s
=> => resolve docker.io/library/python:3.10@sha256:76f22e4ce53774c1f5eb0ba145edb57b908e7aa329fee75eca69b511c1d0c 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 2.25kB 0.0s
=> CACHED [2/3] WORKDIR /app 0.0s
=> [3/3] COPY test.py /app/test.py 0.0s
=> exporting to image 0.1s
=> => exporting layers 0.0s
=> => exporting manifest sha256:dd9a3ece5cd690d2e93e2eef4fb255f8212ed857073b48d39309610746d0b846 0.0s
=> => exporting config sha256:763084ce996d5f07b314473630128a043536da280fcc9a2d473fe80f518b687b 0.0s
=> => exporting attestation manifest sha256:ffe4190141a86d19ec5522e01e0b8fe44c6aade17d8f2b2a1b304cc3192fc37f 0.0s
=> => exporting manifest list sha256:6609eb923c5cebea133aaf7c955a1adb81a04f94ebfc63ed3b6ab7b021041a6d 0.0s
=> => naming to docker.io/library/dockerfile:latest 0.0s
=> => unpacking to docker.io/library/dockerfile:latest 0.0s
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/r48hax316e0h4075zw3z9atds
```

Рисунок 4 - build

4. Далее мы запускаем Docker-контейнер командой:

docker run -d --name dockerfile dockerfile

5. При запуске произошла та же самая ошибка с портами, что, собственно говоря и не удивительно.

```
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/rbnkc3v1atjrfrk1c7vkag28s
PS C:\Users\Nlgor\chereshnya> docker run -d --name dockerfile dockerfile
03b786a8676ae08522b66f587a8e348f0a0e1075cdc3e85df263325d288822c3
PS C:\Users\Nlgor\chereshnya> docker logs 03b786a8676ae08522b66f587a8e348f0a0e1075cdc3e85df263325d288822c3
Traceback (most recent call last):
  File "/app/test.py", line 10, in <module>
    port_name = ports[0]
IndexError: list index out of range
PS C:\Users\Nlgor\chereshnya>
```

Рисунок 5 – Вход Docker в test.py

Далее вторая часть, нам нужно установить Anaconda. После установки заходим в «**jupyter Notebook**». Следуя всем инструкциям из методических указаний, мы запускаем программу, которая генерирует случайные числа и строит по ним график (рис. 5):

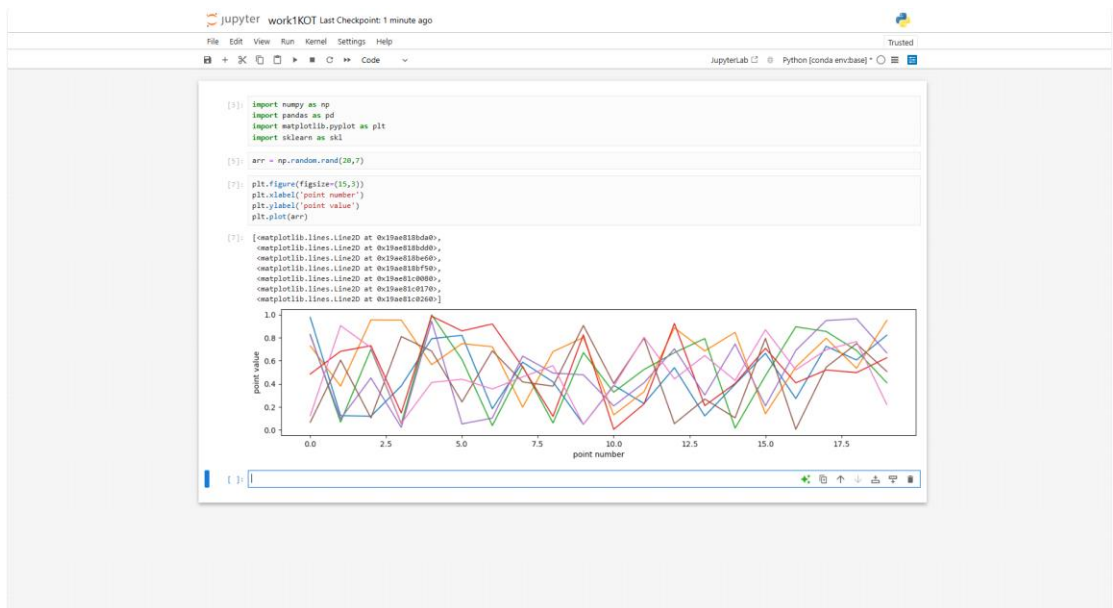


Рисунок 6 – график случайных чисел

Вывод: в результате практической работы мы закрепили навыки работы с Dockerfile, изучили Anaconda и построили там с помощью jupyter Notebook график случайных чисел.