QUIZ

1) Создайте бд, в ней должна быть таблица “stations”, в таблице должны быть (как минимум) атрибуты “name”, “neighbours”, “distance_from_neighbours”. Создайте в ней 3 разные остановки.

2) Я скину файл. В этом файле зашифрована задача, которую надо решить. Каждая нечетная строка файла – одна буква/символ задачи (символы идут последовательно). Напишите код, который выведет условие задачи и напишите решение задачи.

RECAP

Создаем telegram-бота

Опишите шаги

```python
def message_handler(self, *custom_filters, commands=None, regexp=None, content_types=None, state=None,
                    run_task=None, **kwargs):
    """


    :param commands: list of commands
    :param regexp: REGEXP
    :param content_types: List of content types.
    :param custom_filters: list of custom filters
    :param kwargs:
    :param state:
    :param run_task: run callback in task (no wait results)
    :return: decorated function
    """

    def decorator(callback):
        self.register_message_handler(callback, *custom_filters,
                                      commands=commands, regexp=regexp, content_types=content_types,
                                      state=state, run_task=run_task, **kwargs)
        return callback

    return decorator
```

Регулярками (regex) в Python называются шаблоны, которые используются для поиска соответствующего фрагмента текста и сопоставления символов.

Грубо говоря, у нас есть input-поле, в которое должен вводиться email-адрес. Но пока мы не зададим проверку валидности введённого email-адреса, в этой строке может оказаться совершенно любой набор символов, а нам это не нужно.

```
r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)+$'
```

Синтаксис у регулярок необычный. Символы могут быть как буквами или цифрами, так и метасимволами, которые задают шаблон строки:

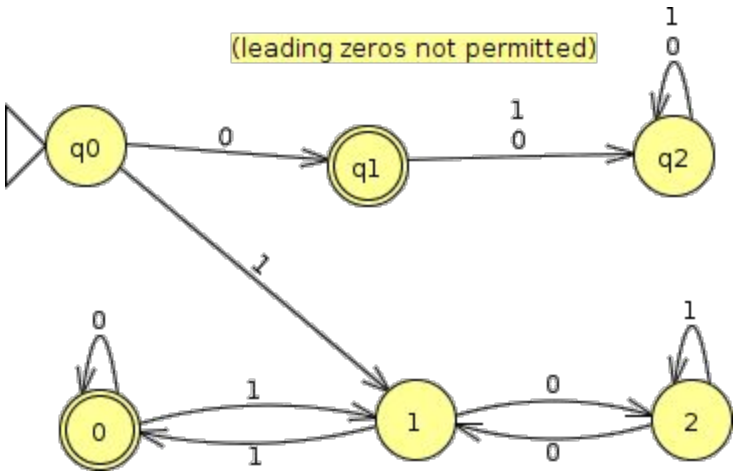
Спец. символ	Зачем нужен
.	Задает **один** произвольный символ
[ ]	Заменяет символ из квадратных скобок
-	Задает один символ, которого не должно быть в скобках
[^]	Задает один символ из **не** содержащихся в квадратных скобках
^	Обозначает начало последовательности
$	Обозначает окончание строки
*	Обозначает произвольное число повторений одного символа
?	Обозначает строго одно повторение символа
+	Обозначает один символ, который повторяется несколько раз
|	Логическое **ИЛИ**. Либо выражение до, либо выражение после символа
\	Экранирование. Для использования метасимволов в качестве обычных
( )	Группирует символы внутри
{ }	Указывается число повторений предыдущего символа

Про регулярки

https://tproger.ru/translations/regular-expression-python/

https://www.w3schools.com/python/python_regex.asp

StatesGroup



Подробнее о состояниях в aiogram

https://mastergroosha.github.io/aiogram-3-guide/fsm/

aiohttp

https://docs.aiohttp.org/en/stable/

```python
import asyncio
import json

import aiohttp

from config import WEATHER_TOKEN

async def get_coord(city_name, token):
    async with aiohttp.ClientSession() as session:
        async with session.get(f"http://api.openweathermap.org/geo/1.0/direct?q="
                               f"{city_name},"
                               f"Russian Federation"
                               f"&appid={token}") as resp:
            json_obj = json.loads(str(await resp.text()))
            lat = str(json_obj[0]["lat"])
            long = str(json_obj[0]["lon"])
            return lat, long

```

```python
async def get_weather(city, token):
    lat, long = await get_coord(city, token)
    print(lat, long)
    async with aiohttp.ClientSession() as session:
        async with session.get(f"https://api.openweathermap.org/data/2.5/weather?"
                               f"lat={lat}"
                               f"&lon={long}"
                               f"&appid={token}&units=metric") as resp:
            print(resp.status)
            print(await resp.text())
            json_obj = json.loads(str(await resp.text()))
            temp = json_obj["main"]["temp"]
            feels_like = json_obj["main"]["feels_like"]
            humidity = json_obj["main"]["humidity"]
            wind_speed = json_obj["wind"]["speed"]
            print(temp, feels_like, humidity, wind_speed)
```

Подключимся к любой API и выгрузим какие-нибудь данные

```
40  async def get_currency(s):
41      async with aiohttp.ClientSession() as session:
42          async with session.get(f"https://api.exchangerate.host/latestbase=USD") as resp: # response
43              print(await resp.text())
44              json_obj = json.loads(str(await resp.text()))
45              return (json_obj["rates"][s])
```

В этом нам помогут

https://catalog.api.2gis.ru/doc/2.0/transport/#/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82/get_2_0_transport_route_search

или

https://yandex.ru/dev/rasp/

DB, SQLalchemy

```python
from sqlalchemy import create_engine

from sqlalchemy import MetaData, Table, String, Integer, Column, Text, DateTime, Boolean
from datetime import datetime
from sqlalchemy import insert, select

metadata = MetaData()

blog = Table('blog', metadata,
             Column('id', Integer()),
             Column('name', String(50))
             )

print(blog.columns)

engine = create_engine('sqlite:///sqlite3.db')

metadata.create_all(engine)

ins = blog.insert().values(
    id="10",
    name="Damir"
)
#
```

```python
from sqlalchemy.ext.automap import automap_base
from sqlalchemy.orm import Session
from sqlalchemy import create_engine
from sqlalchemy.sql import select
from sqlalchemy import create_engine, MetaData, Table

engine = create_engine('sqlite:///sqlite3.db', echo=True)
Base = automap_base()
Base.prepare(engine, reflect=True)

# Users = Base.classes.users
session = Session(engine)

metadata = MetaData()
blog = Table('blog', metadata,
                          autoload_with=engine)
cols = blog.c

ins = blog.insert().values(
    id="11",
    name="Insaf"
)

sel = select(blog).where()

conn = engine.connect()
conn.execute(ins)

query = blog.select()
result = conn.execute(query)

for row in result:
    print(row)
```

Python Notebook

https://jupyter.org/

https://jupyter.org/try-jupyter/lab/

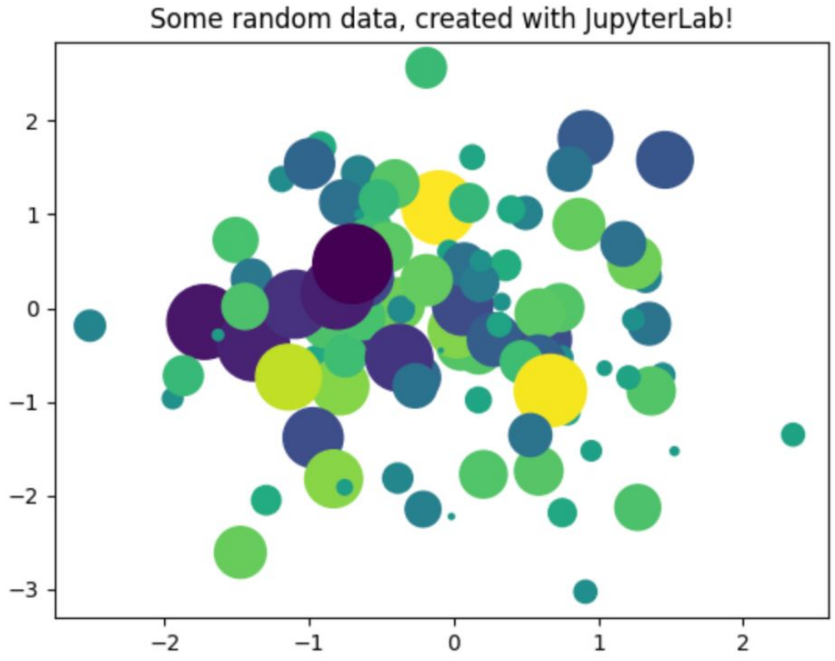
https://jupyter.org/install

Jupyter Notebooks — это стандарт сообщества для общения и выполнения интерактивных вычислений. Это документ, который сочетает в себе вычисления, выходные данные, пояснительный текст, математику, изображения и мультимедийные представления объектов.

[1]:
```
from matplotlib import pyplot as plt
import numpy as np

# Generate 100 random data points along 3 dimensions
x, y, scale = np.random.randn(3, 100)
fig, ax = plt.subplots()

# Map each onto a scatterplot we'll create with Matplotlib
ax.scatter(x=x, y=y, c=scale, s=np.abs(scale)*500)
ax.set(title="Some random data, created with JupyterLab!")
plt.show()
```



DataSpell

VS Code

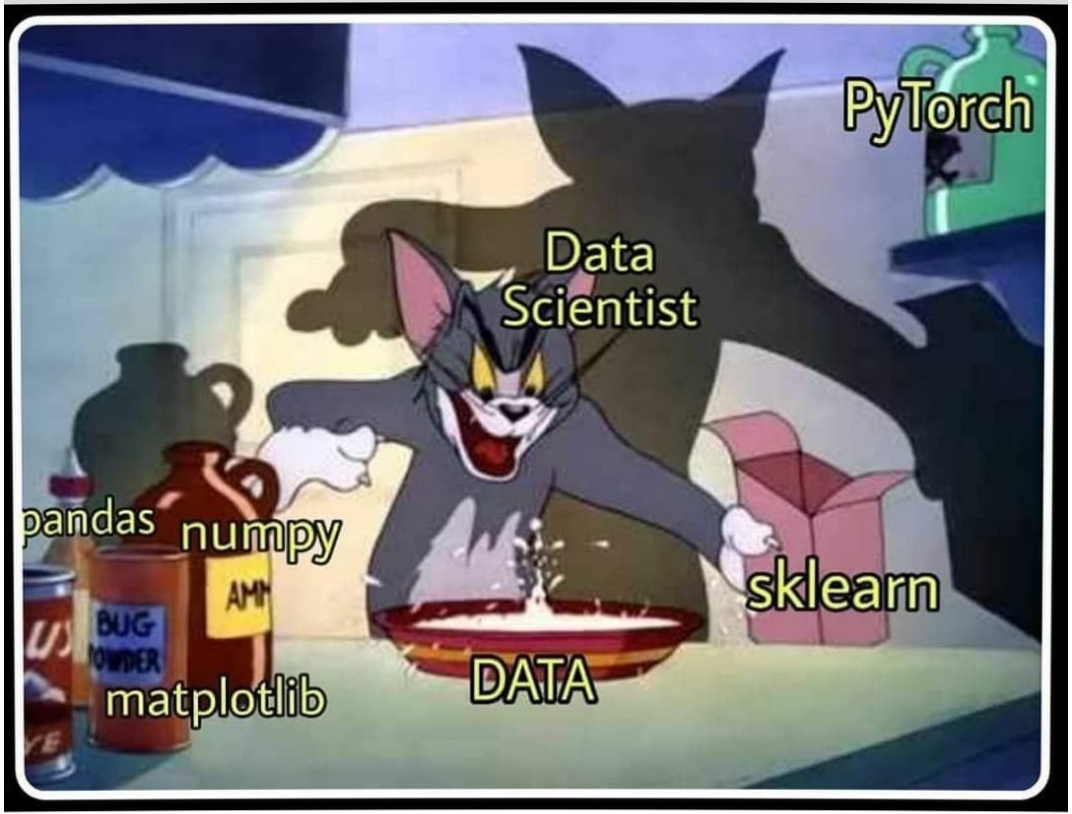
and many others
(including **JupyterLab**)

Быстрый старт

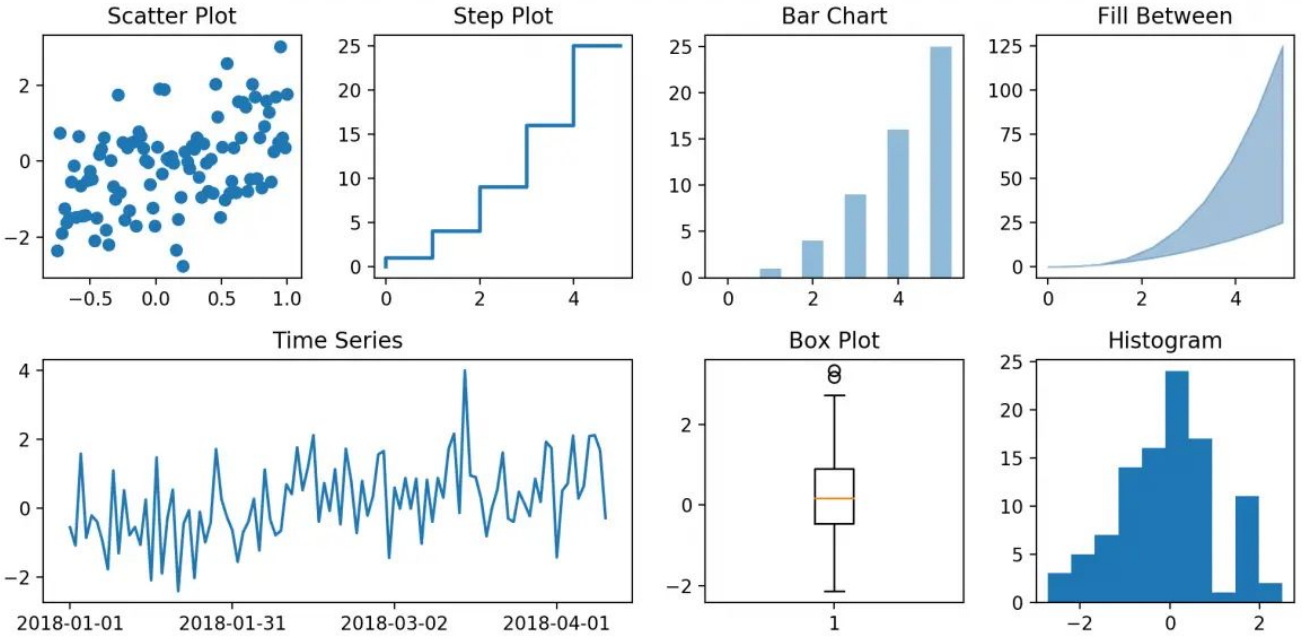
https://colab.research.google.com/

Jupyter Notebook и .ipynb

https://colab.research.google.com/



pip install matplotlib



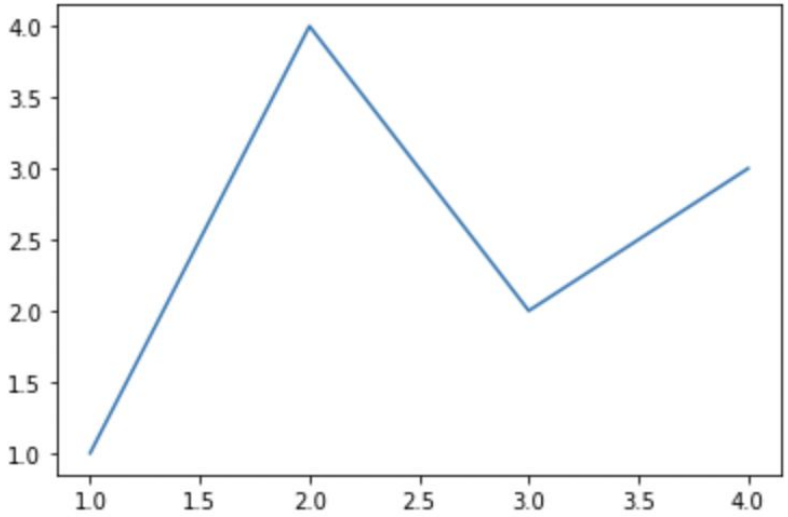
+ Код + Текст

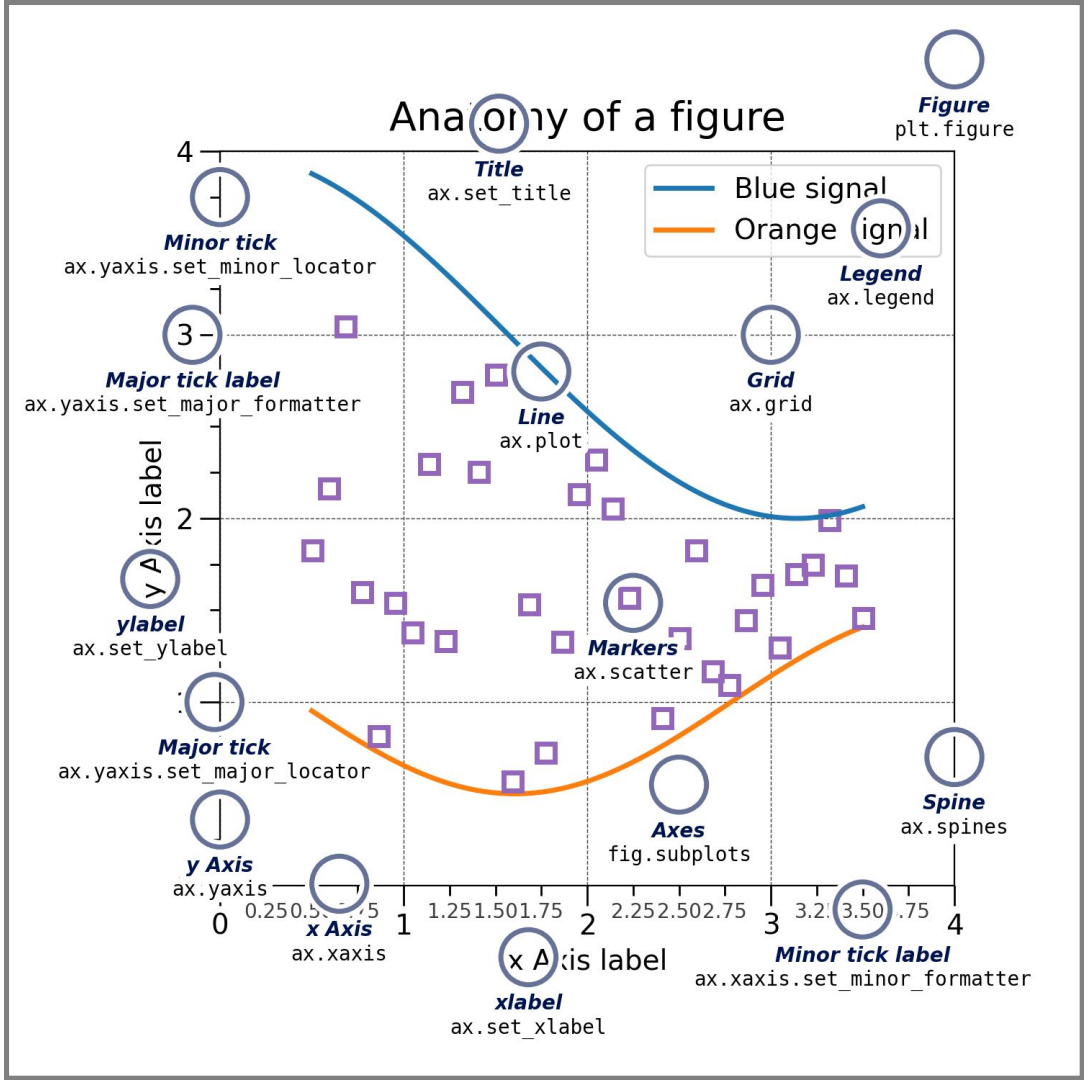
0 сек.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot some data on the axes.
```

```
[<matplotlib.lines.Line2D at 0x7f84d8063760>]
```




Anatomy of a figure
Figure
plt.figure
Title
ax.set_title
Minor tick
ax.yaxis.set_minor_locator
Blue signal
Orange signal
Legend
ax.legend
Major tick label
ax.yaxis.set_major_formatter
Grid
ax.grid
Line
ax.plot
y Axis label
ylabel
ax.set_ylabel
Markers
ax.scatter
Major tick
ax.yaxis.set_major_locator
Spine
ax.spines
Axes
fig.subplots
y Axis
ax.yaxis
x Axis
ax.xaxis
x Axis label
xlabel
ax.set_xlabel
Minor tick label
ax.xaxis.set_minor_formatter

Младшие21.ipynb

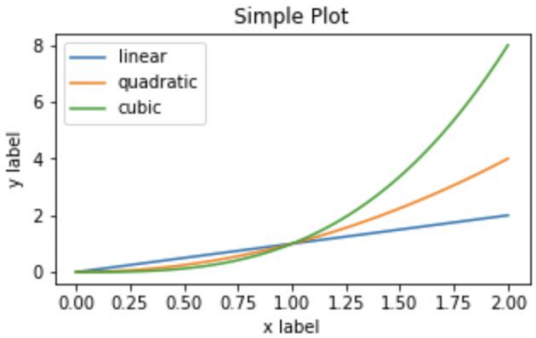
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка

+ Код + Текст

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)  # Sample data.

plt.figure(figsize=(5, 2.7))
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```



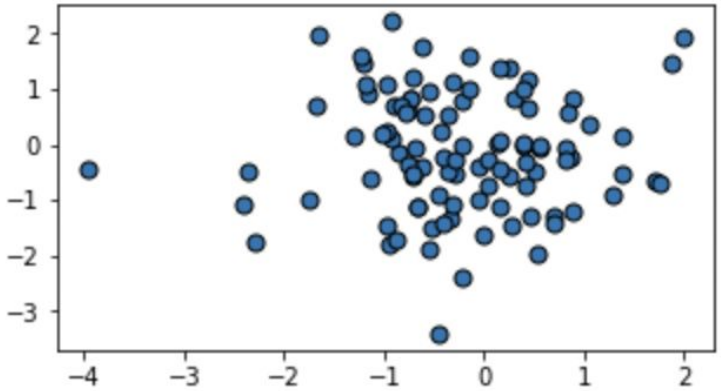
+ Код + Текст

0 сек.

```
import matplotlib.pyplot as plt
import numpy as np

data1, data2, data3, data4 = np.random.randn(4, 100)  # make 4 random data sets
fig, ax = plt.subplots(figsize=(5, 2.7))
ax.scatter(data1, data2, s=50, facecolor='C0', edgecolor='k')
```

```
<matplotlib.collections.PathCollection at 0x7f84b38a2370>
```



https://colab.research.google.com/drive/17IAtYobgZHnzqsZe2FKMEYW1Q05b47UU#scrollTo=PaH_PJ-e-Mig

```
[ ]  import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     %matplotlib inline
```

```
[ ]  dataset = pd.read_csv('a1_dataset.csv')
     dataset.shape
```

```
(925, 8)
```

1.2 Training (50 %)

Splitting dataset on train/test with +- the same number of positive labels.

```
from sklearn.model_selection import train_test_split

print('% of positive samples in whole data:', sum(dataset['target'] == 1) / len(dataset))

random_state = 0

X = dataset.iloc[:, 1:].values
y = dataset["target"].values

x_train, x_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, stratify=dataset['target'], random_state=random_state)

print('% of positive samples in train set:', sum(y_train== 1) / len(x_train))
print('% of positive samples in test set:', sum(y_test== 1) / len(x_test))
```

```
% of positive samples in whole data: 0.4962162162162162
% of positive samples in train set: 0.49594594594594593
% of positive samples in test set: 0.4972972972972973
```

Домашнее задание

1) нарисуйте графики синуса и косинуса, используйте numpy matplotlib.pyplot
2) добавьте к предыдущим графикам немного шума (рандомные значения в пределах +-0.1)
3) Проект