

Двенадцатое практическое занятие

Работа с серийным подключением

До настоящего времени мы осуществляли управление создаваемыми устройствами только через аппаратные средства (кнопки, потенциометры и т.п.), а данные с них получали выводя формируемые микроконтроллером значения на экран или в консоль `PuCharm`.

Но всегда ли этого достаточно? Очевидно, что нет! Как создать полноценный интерфейс? С трудом! Как сохранить и обработать получаемые данные? Не понятно...

Исправим это! Разберемся, как наш микроконтроллер взаимодействует с подключенным к нему компьютером и как полноценнее использовать эту связь!

Взаимодействие через `print()` и `input()`

Первое, с чем мы познакомились изучая язык программирования `python`, были функции консольного ввода и вывода:

- `input()`
- `print()`

Первая позволяла зарегистрировать последовательность нажатых на клавиатуре клавиш и сформировать по факту нажатия клавиши `enter` объект строки, с которым можно впоследствии работать в программе.

Вторая позволяет вывести в "консоль" сформированную по внутренним правилам работы функции строку (конкатенировать перечисленные строковые аргументы в функции, через разделитель `sep=" "`, добавив в конце последовательность `end="\n"` и т.д.).

Это все нам давно известно и привычно! Логика работы следующего кода должна быть очевидна нам с первого взгляда:

```
import time

n = 0
counter = 0
while True:
    print(f"{{counter}}_Hello_{{time.time()}}{ " " * n}{{"\b" * n}}")
    counter += 1
    time.sleep(1)
    # time.sleep(0.1)
```

```
# time.sleep(0.01)
# time.sleep(0.001)
```

Чего не скажешь о целесообразности возвращаться к таким примитивным вещам....
Но...

Попробуем изменить значение переменной `n`, добавляющей в конец строки равное количество пробелов и, стирающих их, "бэкспейсов".

Что произойдет при выводе сообщений в консоли? Пока стоит задержка в 1 секунду, вроде бы ничего...

Но что, если ее уменьшить? Мы сразу сможем заметить, что выводимые сообщения начинаю "запаздывать" и вот здесь уже необходимо определить причину этого.

Для этого обратимся к документации!

Принцип работы функции `print()`

Функция `print()` в `Python` — это встроенная функция, которая используется для вывода данных на стандартное устройство вывода (обычно это консоль или терминал). Она позволяет отображать текстовую информацию, значения переменных, результаты вычислений и многое другое.

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- `*objects` — объекты, которые нужно вывести (например, строки, числа, переменные).
- `sep` — разделитель между объектами (по умолчанию пробел).
- `end` — символ, который добавляется в конце вывода (по умолчанию новая строка `\n`).
- `file` — файл или поток, куда направляется вывод (по умолчанию `sys.stdout`, то есть консоль).
- `flush` — если `True`, то буфер вывода сбрасывается сразу после вывода (по умолчанию `False`).

Вот оно! Что это за такой поток `sys.stdout`, в который мы передаем сформированную строку?... Хотя строку ли?... С этим тоже, пожалуй стоит разобраться...

Взаимодействие через `sys.stdout` и `serial`

Что такое `sys.stdout`?

`sys.stdout` — это стандартный поток вывода (standard output) в `Python`, который используется для вывода данных на экран (консоль или терминал). Это объект

файлового типа, предоставляемый модулем `sys`, и он является частью стандартной библиотеки `Python`.

- По умолчанию он связан с консолью (терминалом), куда выводятся текстовые данные.
- Его можно перенаправить на другие объекты, например, файлы или строковые буферы.

В `micropython` модуль `sys` является также встроенным в [стандартную библиотеку](#).

Фактически следующий код делает абсолютно то же самое, что и предыдущий, только более явно:

```
import time
import sys

n = 0
counter = 0
while True:
    sys.stdout.write(f"{counter}_Hello_{time.time()}{" " * n}{"\b" *
n}\n".encode("utf-8"))
    counter += 1
    time.sleep(1)
    # time.sleep(0.1)
    # time.sleep(0.01)
    # time.sleep(0.001)
```

Разберем логику работы строки кода:

```
sys.stdout.write(f"{counter}_Hello_{time.time()}{" " * n}{"\b" *
n}\n".encode("utf-8"))
```

- мы обращаемся к статическому объекту `stdout` модуля `sys` (импортированного в преамбуле кода `import sys`);
- Вызываем метод `write` этого объекта, принимающий в качестве аргумента массив байт;
- Формируем из строки массив байт, применив к ней стандартный метод `encode()`, указав в нем требуемую кодировку (`UTF-8`)
- Отдельно отметим необходимость явного указания в конце строки символа переноса строки `\n`!

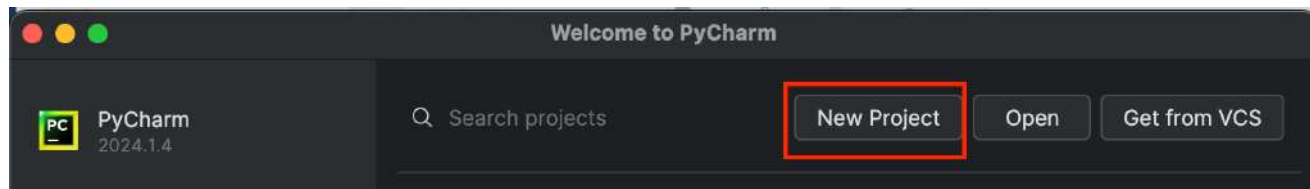
Выглядит немного громоздко, но теперь мы полностью управляем логикой передачи информации с нашего микроконтроллера!

Осталось немного - только получить и использовать ее!

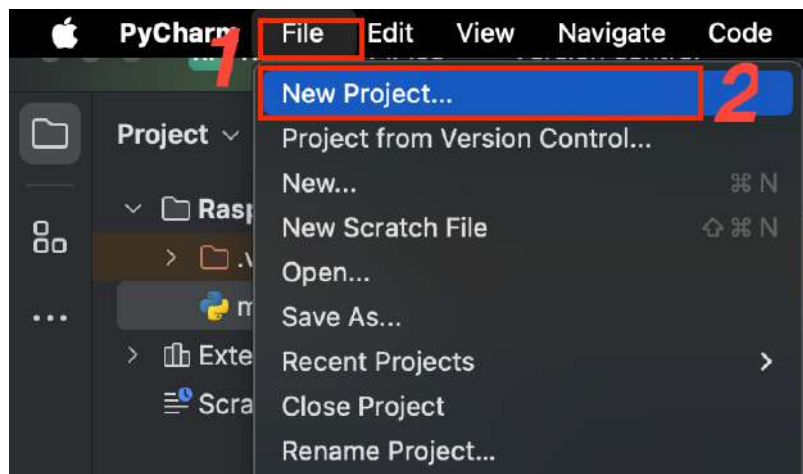
Для этого создадим новый проект в `Pycharm`!

Создание нового проекта

Нажмите "Новый проект" при запуске программы:

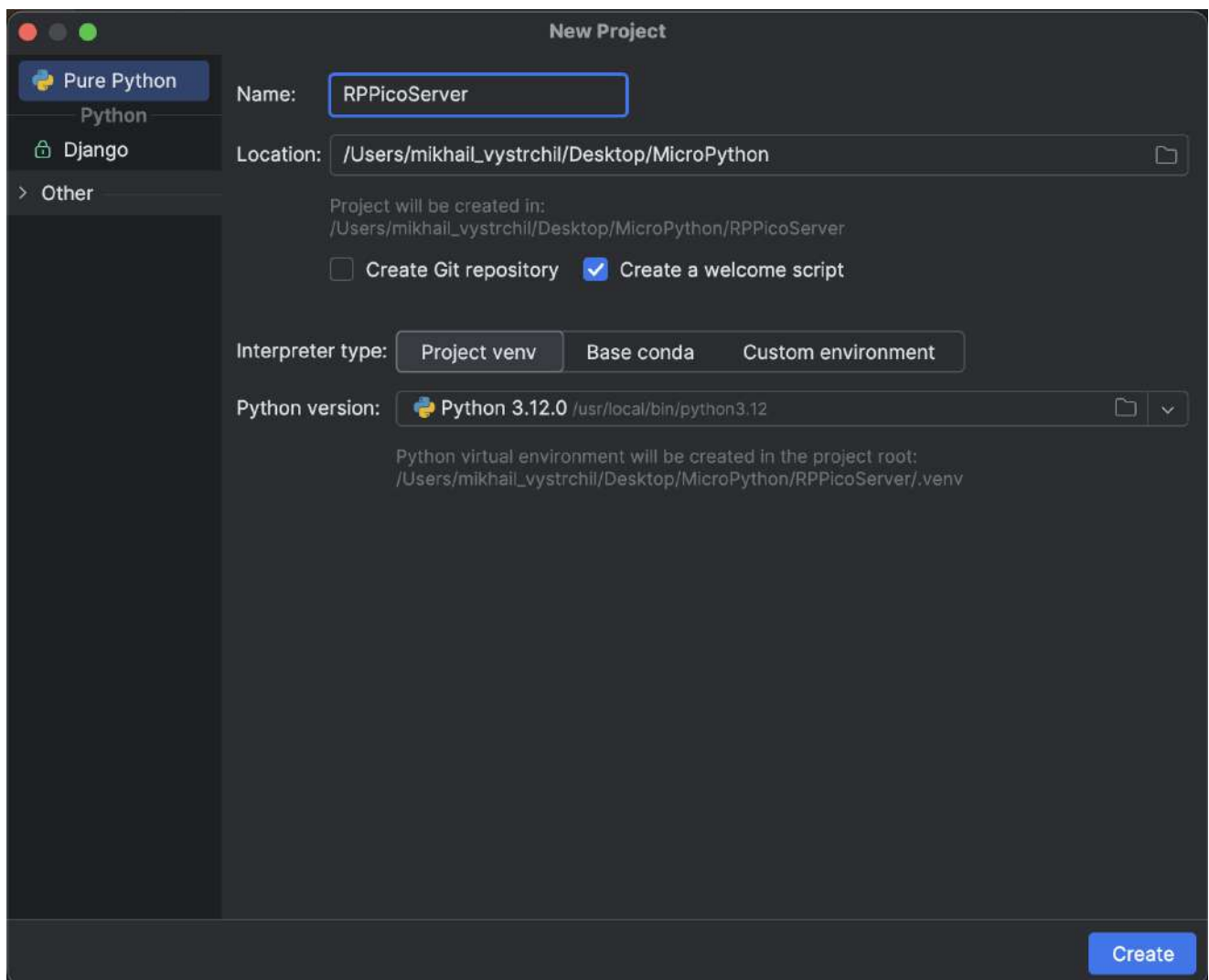


Или выберите "Новый проект" в меню "файл":



Обратите внимание, что это новый проект, никак напрямую не связанный ни с микроконтроллером, ни с `micropython`, вообще ни с чем!

Поэтому удостоьте его отдельным именем и местом инициализации!



В этом проекте мы будем работать с полноценным `python`, поэтому:

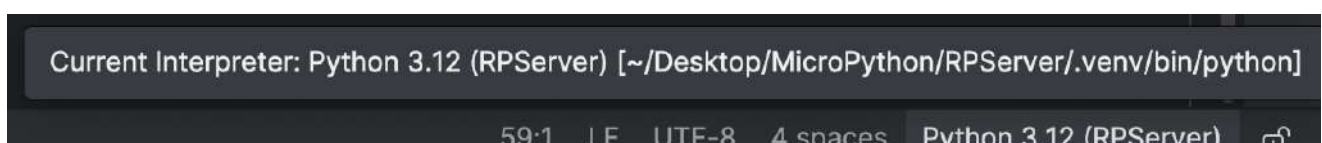
⚠ Никаких плагинов `micropython` подключать к нему не нужно!

А что нужно? Ну тут два варианта: или мы работаем в локальном виртуальном окружении `.venv` и будем устанавливать в него по новой все пакеты (напр: `matplotlib`, `numpy` и т.п.), или мы подключимся к основному системному интерпретатору языка, в котором все это уже есть!

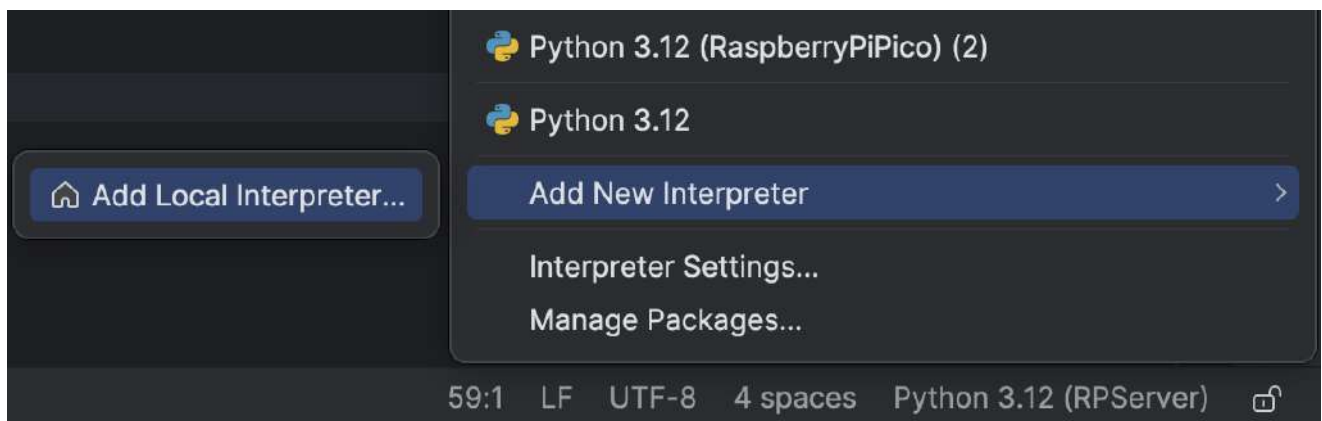
Если Вы работаете на своем ноутбуке, то этот выбор за Вами, если нет - тогда второй вариант!

Для этого:

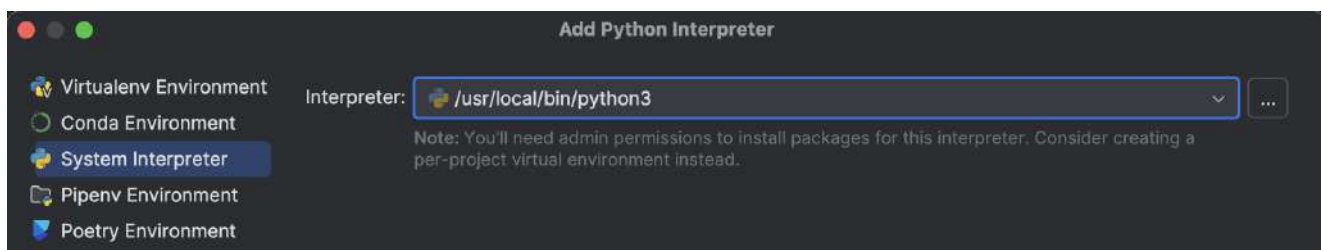
- В нижнем правом углу экрана щелкаем на текущий интерпретатор языка:



- Выбираем "добавить новый":



- И выбираем "системный интерпретатор":



Код программы


Создадим в новом проекте новый файл со следующим содержанием:

```
import serial

s = serial.Serial(port="/dev/tty.usbmodem1432301", baudrate=9600)

while True:
    data = s.readline()
    data = data.strip()
    data = data.decode(encoding="utf-8")
    print(type(data), data)
```

 Название файлов здесь могут быть произвольными - не обязательно `main.py`

 Модуль `serial` не является стандартным для библиотеки `python`. Если он отсутствует то необходимо его дополнительно установить через:

```
pip install pyserial
```

Разберем этот код:

- Импортируем модуль `serial`;
- После чего создаем объект класса `Serial` для работы с серийным подключением.

В качестве необходимого параметра при создании подключения необходимо указать адрес порта, к которому подключен микроконтроллер (в моем случае

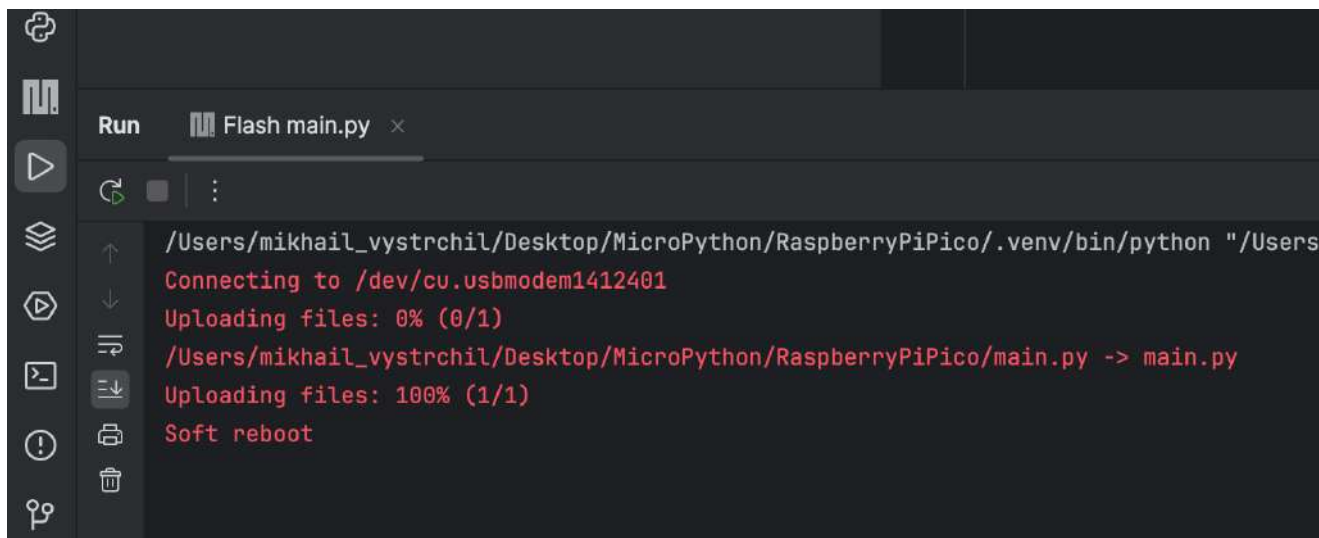
```
"/dev/tty.usbmodem1432301").
```

Узнать номер порта можно в диспетчере устройств на Windows (например COM5)

Или через терминальную команду на UNIX системах.

```
ls /dev/tty.*
```

❗ Номер порта указывается в терминале при загрузке скрипта на микроконтроллер:



```
/Users/mikhail_vystrchil/Desktop/MicroPython/RaspberryPiPico/.venv/bin/python "/Users
Connecting to /dev/cu.usbmodem1412401
Uploading files: 0% (0/1)
/Users/mikhail_vystrchil/Desktop/MicroPython/RaspberryPiPico/main.py -> main.py
Uploading files: 100% (1/1)
Soft reboot
```

Но вернемся к коду! Далее в цикле:

```
while True:
    data = s.readline()
    data = data.strip()
    data = data.decode(encoding="utf-8")
    print(type(data), data)
```

- Мы будем построчно читать информацию из созданного объекта подключения.

⚠ Метод `.readline()` будет формировать массив байтов до тех пор, пока не получит символ переноса на новую строку `"\n"`. Если его забыть при формировании сообщения со стороны микроконтроллера, то программа "зависнет"!

Далее:

- Мы удаляем лишние пробелы и символ переноса строки методом `.strip()`;
- И декодируем обратно массив байт в строку `.decode(encoding="utf-8")`;
- Выводим полученную строку в консоль стандартным методом `print()`.

Казалось бы, зачем так все усложнять? Ведь увидеть текст в консоли мы могли без всех этих заморочек...

Но главное, нам удалось получить сформированную на микроконтроллере информацию отдельно от плагина `micropython` и обработать ее на полноценном `python`!

Попробуем использовать ее более явно!

Построение графиков по получаемой с микроконтроллера информации

Представим следующую задачу:

Микроконтроллер собирает информацию с подключенных к нему датчиков, передает их в "сыром" виде на основной компьютер, а мы на нем визуализируем информацию.

Первый вариант

Код на микроконтроллере

```
import random
import time
import sys

min_val = 0
max_val = 100

while True:
    random_number = random.randint(min_val, max_val)
    sys.stdout.write(f"{random_number}\n".encode("utf-8"))
    time.sleep(0.1)
```

Тут все просто:

- Десять раз в секунду мы генерируем случайное число в пределах от 0 до 100;
- Формируем из полученного числа строку, добавляем символ переноса строки;
- Кодируем полученную строку в массив байт;
- И передаем на внешний компьютер.

Код на компьютере

```
import serial
import matplotlib.pyplot as plt
import matplotlib.animation as animation
```



```

from CONFIG import RP_PORT

s = serial.Serial(port=RP_PORT, baudrate=115200)

fig, ax = plt.subplots()

x = 0
x_lst, y_lst = [], []

def update(i):
    global x
    data = s.readline().strip().decode(encoding="utf-8")
    data = int(data)
    x_lst.append(x)
    x += 1
    y_lst.append(data)
    ax.clear()
    ax.plot(x_lst[-100:], y_lst[-100:])

ani = animation.FuncAnimation(fig, update, interval=1)
plt.show()

```

Для отрисовки графика в реальном времени воспользуемся пакетом `animation` модуля `matplotlib`

Здесь и далее название порта к которому подключен микроконтроллер записано в константе `RP_PORT` в файле `CONFIG`.

После необходимых импортов:

- Создаем объект подключения.

```
s = serial.Serial(port=RP_PORT, baudrate=115200)
```

- Создаем окно и оси для построения графика.

```
fig, ax = plt.subplots()
```

- Иницилируем переменные в которых будут сохраняться данные для отрисовки графика.

```

x = 0
x_lst, y_lst = [], []

```

- Создаем функцию `update(i)` определяющую логику отрисовки графика на каждом "кадре".

```
def update(i):  
    global x  
    data = s.readline().strip().decode(encoding="utf-8")  
    data = int(data)  
    x_lst.append(x)  
    y_lst.append(data)  
    x += 1  
    ax.clear()  
    ax.plot(x_lst[-100:], y_lst[-100:])
```

Внутри этой функции:

- Считываем информацию из серийного подключения.

```
data = s.readline().strip().decode(encoding="utf-8")
```

- Приводим ее к целочисленному виду.
- Добавляем текущее значение счетчика `x` и полученное число `data` в соответствующие списки `x_lst` и `y_lst`.
- Инкриминируем значение счетчика `x`.

⚠ Обратите внимание, что для доступа к этой переменной вне функции необходимо использовать указание `global x`, так как мы ПЕРЕОПРЕДЕЛЯЕМ значение переменной, для `x_lst` и `y_lst` `global` не нужен, так как мы добавляем значения в списки, а не создаем новые!

- Стираем с оси `ax` все, что было на ней нарисовано.

```
ax.clear()
```

- Рисуем линейный график по последним 100 значениям.

```
ax.plot(x_lst[-100:], y_lst[-100:])
```

Уже вне функции создаем специальный объект для анимированного графика:

```
ani = animation.FuncAnimation(fig, update, interval=1)
```

Передав в него тот объект фигуры (`fig`), в котором будет выполняться отрисовка, объект функции `update`, определяющий логику изменения графика и задав интервал

в 1 миллисекунду между обновлением графика.

Вызываем график на отрисовку:

```
plt.show()
```

Второй вариант

А можем мы одновременно построить график по нескольким источникам данных?

Конечно!

Для этого сгенерируем на микроконтроллере пару случайных чисел и передадим их!

Код на микроконтроллере

```
import random
import time
import sys

min_val = 0
max_val = 100

while True:
    random_number_1 = random.randint(min_val, max_val)
    random_number_2 = random.randint(min_val, max_val)
    data_str = f"{random_number_1} {random_number_2}\n"
    sys.stdout.write(data_str.encode("utf-8"))
    time.sleep(0.1)
```

Код на компьютере

```
import serial
import matplotlib.pyplot as plt
import matplotlib.animation as animation

from CONFIG import RP_PORT

s = serial.Serial(port=RP_PORT, baudrate=115200)

fig, ax = plt.subplots()

x = 0
x_lst, y_1_lst, y_2_lst = [], [], []

def update(i):
    global x
```

```
data = s.readline().strip().decode(encoding="utf-8")
y_1, y_2 = [int(y) for y in data.split()]
x_lst.append(x)
x += 1
y_1_lst.append(y_1)
y_2_lst.append(y_2)
ax.clear()
ax.plot(x_lst[-25:], y_1_lst[-25:])
ax.plot(x_lst[-25:], y_2_lst[-25:])

ani = animation.FuncAnimation(fig, update, interval=1)
plt.show()
```

В целом, все работает абсолютно также, но должно начаться формирование потенциальной проблемы:

Как понять и договориться о формате передаваемых и читаемых данных?

Ведь пока структура данных определяется просто *нашим условным знанием, что мы передали с контроллера!*

Как упростить эту задачу при масштабировании, когда передаваемых чисел будет много?

Нужно использовать более явные структуры данных, например словари, но как их представить в виде строки, а, еще важнее, легко прочесть обратно?

Ответ есть - JSON!

Третий вариант - сериализация данных через JSON

Сериализация данных

Сериализация — это процесс преобразования объектов или структур данных в формат, который можно сохранить или передать (например, JSON, XML, бинарный формат).

JSON (JavaScript Object Notation)— это текстовый формат для хранения и передачи данных. Он простой, легковесный и легко читается как людьми, так и машинами. Используется для обмена данными между клиентом и сервером, а также для хранения настроек и конфигураций.

Основные особенности:

- Данные представляются в виде пар **ключ-значение**.
- Поддерживает типы данных: строки, числа, массивы, объекты, булевы значения и `null`.

- Пример JSON:

```
{
  "name": "Alice",
  "age": 25,
  "isStudent": true,
  "skills": ["Python", "JavaScript"]
}
```

Сериализация в JSON:

Python → JSON:

Используется метод `json.dumps()` для преобразования Python-объекта в JSON-строку.

```
import json

data = {
    "name": "Alice",
    "age": 25,
    "isStudent": True
}
json_string = json.dumps(data)
print(json_string)  # {"name": "Alice", "age": 25, "isStudent": true}
```

JSON → Python:

Используется метод `json.loads()` для преобразования JSON-строки в Python-объект.

```
json_string = '{"name": "Alice", "age": 25, "isStudent": true}'
data = json.loads(json_string)
print(data["name"])  # Alice
```

Используем все это при передаче данных!

Код на микроконтроллере

Рассчитаем значения синусоиды и косинусоиды на микроконтроллере, сформируем по полученным данным словарь `data`, сериализуем словарь в строку `data_str`, используя модуль `json`, и передадим ее после кодировки и добавления символа переноса строки на компьютер:

```
import time
import sys
import math
```

```

import json

x_deg = 0

while True:
    x_deg = x_deg
    x_rad = math.radians(x_deg)
    y_sin = math.sin(x_rad)
    y_cos = math.cos(x_rad)

    data = {"x_deg": x_deg,
            "x_rad": x_rad,
            "y_sin": y_sin,
            "y_cos": y_cos,
            }

    data_str = json.dumps(data)
    sys.stdout.write(f"{data_str}\n".encode("utf-8"))
    x_deg += 1
    time.sleep(0.01)

```

Код на компьютере

Добавим в функции `update(i)` строку:

```
data = json.loads(data)
```

возвращающую объект словаря из прочитанный json строки и построим требуемые графики:

```

import serial
import matplotlib.pyplot as plt # $ pip install matplotlib
import matplotlib.animation as animation
import json

from CONFIG import RP_PORT

s = serial.Serial(port=RP_PORT, baudrate=115200)

x, y_cos, y_sin = [], [], []
fig, ax = plt.subplots()

def update(i):
    data = s.readline().strip()
    data = data.decode(encoding="utf-8")
    data = json.loads(data)

    x.append(data["x_deg"])

```

```

y_cos.append(data["y_cos"])
y_sin.append(data["y_sin"])

ax.clear()
ax.plot(x[-360:], y_cos[-360:], label=r"$\cos$")
ax.plot(x[-360:], y_sin[-360:], label=r"$\sin$")
ax.grid()
ax.legend()

ani = animation.FuncAnimation(fig, update, interval=1)
plt.show()

```

Управление подключенными к микроконтроллеру устройствами с компьютера

Реализуем возможность управлением светодиодами с внешнего устройства!

Классы для светодиодов

Для этого создадим и запишем в память микроконтроллера (по аналогии с прошлым занятием) файл `led.py` следующего содержания:

```

from machine import PWM, Pin

class Led:
    BRIGHTNESS_SCALE = 255

    def __init__(self, led_pin):
        self.led = PWM(Pin(led_pin, Pin.OUT), freq=1000)
        self.brightness = 0

    def on(self):
        self.led.duty_u16(65535)

    def off(self):
        self.led.duty_u16(0)

    def turn_to_brightness(self, brightness):
        if brightness <= 0:
            brightness = 0
        elif brightness > self.BRIGHTNESS_SCALE:
            brightness = self.BRIGHTNESS_SCALE
        duty = int(65535 / self.BRIGHTNESS_SCALE * brightness)
        self.led.duty_u16(duty)
        self.brightness = brightness

```

```

class RGBLed:

    BRIGHTNESS_SCALE = 255

    def __init__(self, red_pin, green_pin, blue_pin):
        self.red = Led(red_pin)
        self.green = Led(green_pin)
        self.blue = Led(blue_pin)
        self.color_tuple = (0, 0, 0)

    def turn_on_color(self, color_tuple):
        self.red.turn_to_brightness(color_tuple[0])
        self.green.turn_to_brightness(color_tuple[1])
        self.blue.turn_to_brightness(color_tuple[2])
        self.color_tuple = color_tuple

    def turn_to_brightness(self, brightness):
        if brightness <= 0:
            brightness = 0
        elif brightness > self.BRIGHTNESS_SCALE:
            brightness = self.BRIGHTNESS_SCALE
        total_summ_color = 0
        for idx, led in enumerate([self.red, self.green, self.blue]):
            led_brightness = self.color_tuple[idx] * brightness /
self.BRIGHTNESS_SCALE
            led.turn_to_brightness(led_brightness)
            total_summ_color += self.color_tuple[idx]
        if total_summ_color == 0:
            self.turn_on_color([brightness] * 3)

    def on(self):
        self.turn_on_color([255, 255, 255])

    def off(self):
        self.turn_on_color([0, 0, 0])

```

Ссылка на [файл](#).

Управление обычным светодиодом

Для микроконтроллера

Схема подключения

Код программы

Со стороны микроконтроллера напишем следующий код, в котором создадим объект светодиода `led` класса `Led`, подключенного к 16 пину (`LED_PIN`).


```

import sys

from led import Led

LED_PIN = 16

led = Led(led_pin=LED_PIN)

while True:
    command = sys.stdin.readline().strip().lower()
    if command == "on":
        led.on()
        sys.stdout.write("Диод включен!\n".encode("utf-8"))
    elif command == "off":
        led.off()
        sys.stdout.write("Диод выключен!\n".encode("utf-8"))
    elif command[0] == "b":
        try:
            brightness = int(command[1:])
            led.turn_to_brightness(brightness)
            sys.stdout.write(f"Диод включен на яркость {brightness/255*100:.2f}% от максимальной!\n".encode("utf-8"))
        except Exception as e:
            sys.stdout.write(f"Неверная команда!\n".encode("utf-8"))
    else:
        sys.stdout.write("Неверная команда!\n".encode("utf-8"))

```

Далее в бесконечном цикле:

Читаем строку данных, поступивших в стандартный поток входа

`sys.stdin.readline()`. Удаляем из нее все лишние пробелы и переносы строк и приводим в нижнему регистру (`.lower()`)

```
command = sys.stdin.readline().strip().lower()
```

- ❗ Приведение строки к нижнему регистру позволяет избежать возможных опечаток со стороны пользователя. Например "On", "ON", "oN" и "on" во всех случаях будут приведены к строке "on".

Полученная команда `command` проходит через ряд логических проверок в которой:

- в случае `"on"` - светодиод загорается на максимальную яркость;
- `"off"` - выключается;
- `"b0"` .. `"b255"` - если строка начинается с "b" и продолжается целым числом, оно интерпретируется как требуемая яркость светодиода (0 -выключен, 255 - максимальная яркость).

После выполнения каждой из команд (или не выполнения ни одной) обратно в консоль выводится сообщение о статусе выполненной команды.

⚠ Обратите внимание, что, если выполнение программы прервется какой-либо ошибкой на стороне микроконтроллера, отследить ее и восстановить его работу можно будет только его перезагрузкой, так что особенно внимательно отнеситесь к проверке и отладке кода.

Для компьютера

Со стороны компьютера все проще:

```
import serial

from CONFIG import RP_PORT

s = serial.Serial(port=RP_PORT, baudrate=115200)

while True:
    command = input("Command: ")
    command = f"{command}\n".encode("UTF-8")
    s.write(command)
    answer = s.readline().strip().decode("UTF-8")
    print(answer)
```

Мы просто:

- Создаем серийное подключение;
- Ждем в цикле команду от пользователя (через стандартную функцию `input("Command: ")`);
- Добавляем перенос строки и кодируем ее в массив байт;
- Передаем ее через серийное соединение на микроконтроллер;
- Ждем ответа с микроконтроллера;
- выводим его в консоль через функцию `print(answer)`

Управление RGB светодиодом

Для микроконтроллера

Схема подключения

Код программы

```
import sys

from led import RGBLed
```

```

RED_PIN = 13
GREEN_PIN = 14
BLUE_PIN = 15

rgb_led = RGBLed(red_pin=RED_PIN, green_pin=GREEN_PIN, blue_pin=BLUE_PIN)

while True:
    command = sys.stdin.readline().strip()
    if command.lower() == "on":
        rgb_led.on()
        sys.stdout.write("Turned on!\n".encode("utf-8"))
    elif command.lower() == "off":
        rgb_led.off()
        sys.stdout.write("Turned off!\n".encode("utf-8"))
    elif command.lower()[0] == "b":
        try:
            brightness = int(command[1:])
            brightness = brightness if brightness < 255 else 255
            brightness = brightness if brightness > 0 else 0
            rgb_led.turn_to_brightness(brightness)
            sys.stdout.write(f"Turned on brightness
{brightness/255*100:.2f}%!\n".encode("utf-8"))
        except Exception as e:
            sys.stdout.write(f"Wrong command -
{e.__class__}!\n".encode("utf-8"))
    else:
        try:
            data = command.split(",")
            if len(data) != 3:
                raise ValueError
            color_tuple = tuple(map(int, data))
            rgb_led.turn_on_color(color_tuple=color_tuple)
            sys.stdout.write(f"Turned on RGB color
{color_tuple}!\n".encode("utf-8"))
        except Exception as e:
            sys.stdout.write(f"Wrong command -
{e.__class__}!\n".encode("utf-8"))

```

В целом представленный код работает аналогично коду для обычного светодиода, за исключением возможности парсинга строки, содержащей перечисления трех чисел через запятую (типа `255, 0, 125`) как цвета светодиода

Для компьютера

Со стороны компьютера остается тем же самым:

```
import serial
```

```
from CONFIG import RP_PORT

s = serial.Serial(port=RP_PORT, baudrate=115200)

while True:
    command = input("Command: ")
    command = f"{command}\n".encode("UTF-8")
    s.write(command)
    answer = s.readline().strip().decode("UTF-8")
    print(answer)
```

Управление светодиодом через Telegram бота

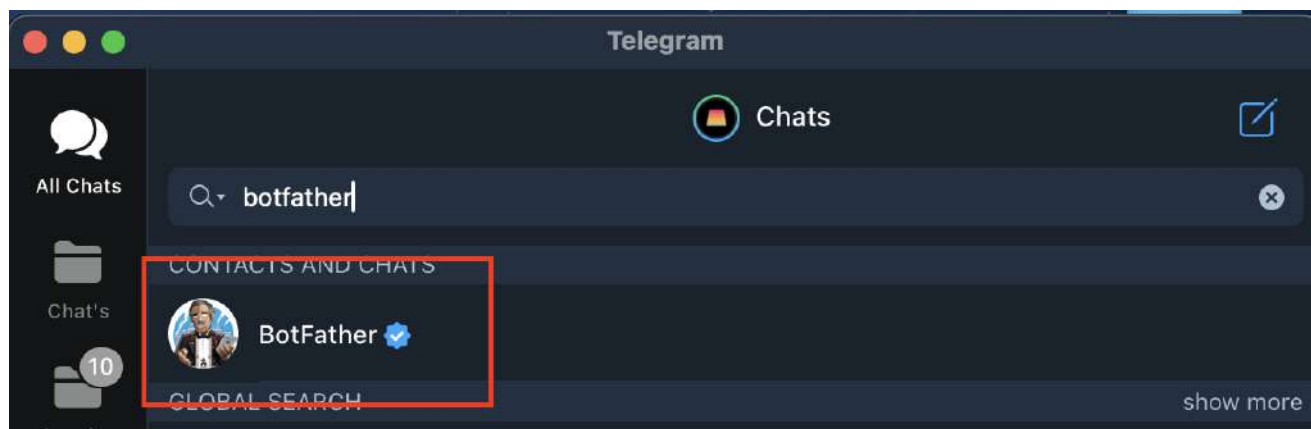
⚠️ Работа с телеграм ботом требует создание и запуск на компьютере сервера с понятным и постоянным доступом в интернет и, вероятнее всего, не сработает с стационарных компьютеров Горного Университета.

Подготовка бота определяется решением двух задач:

- Создание телеграм бота;
- Настройка и запуск сервера для работы с ним.

Создание бота

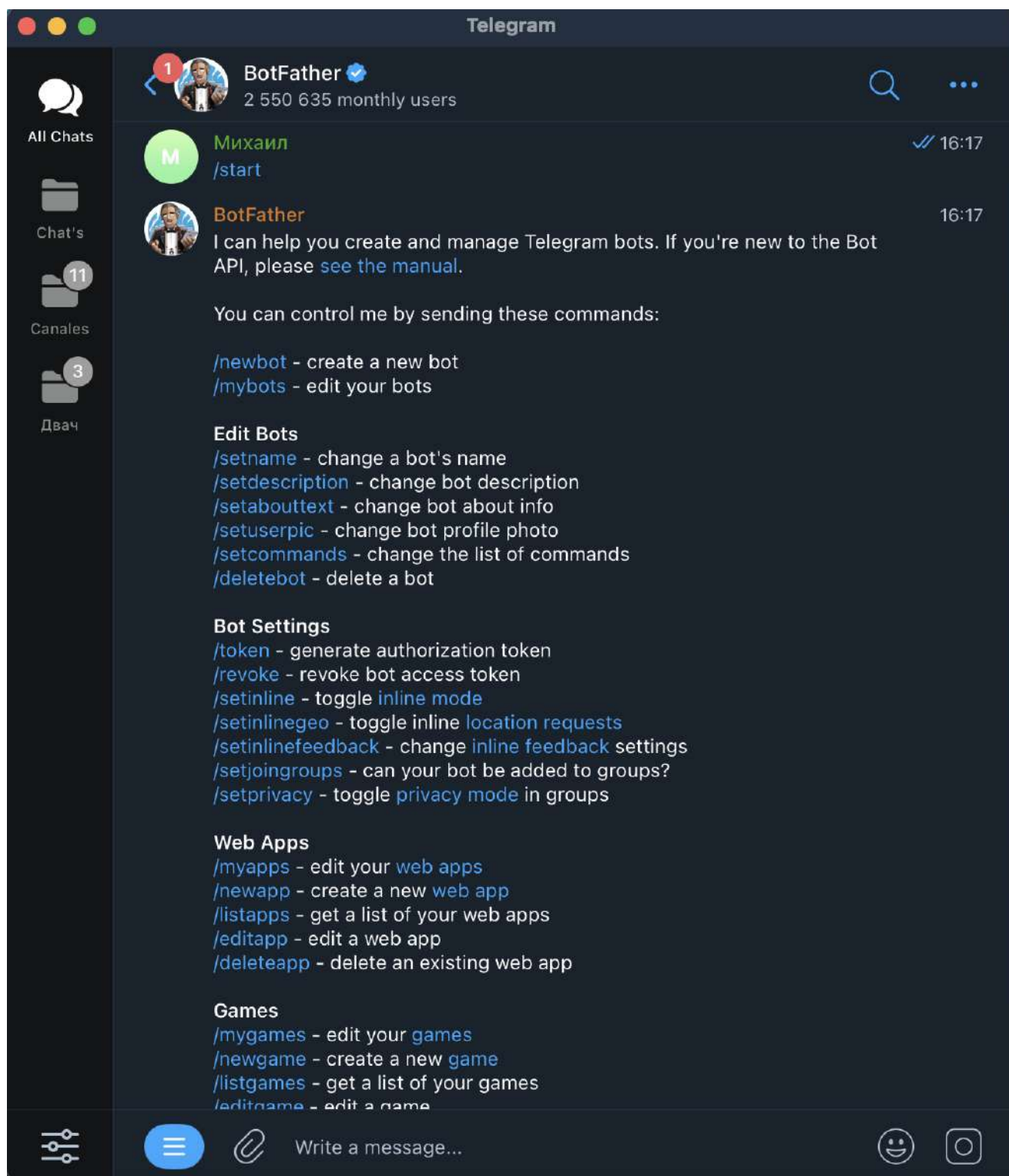
Создание нового бота происходит с уважением через стандартного телеграмм бота BotFather:



Вся эта часть никак не связана с `python` и выполняется через любой телеграм клиент.

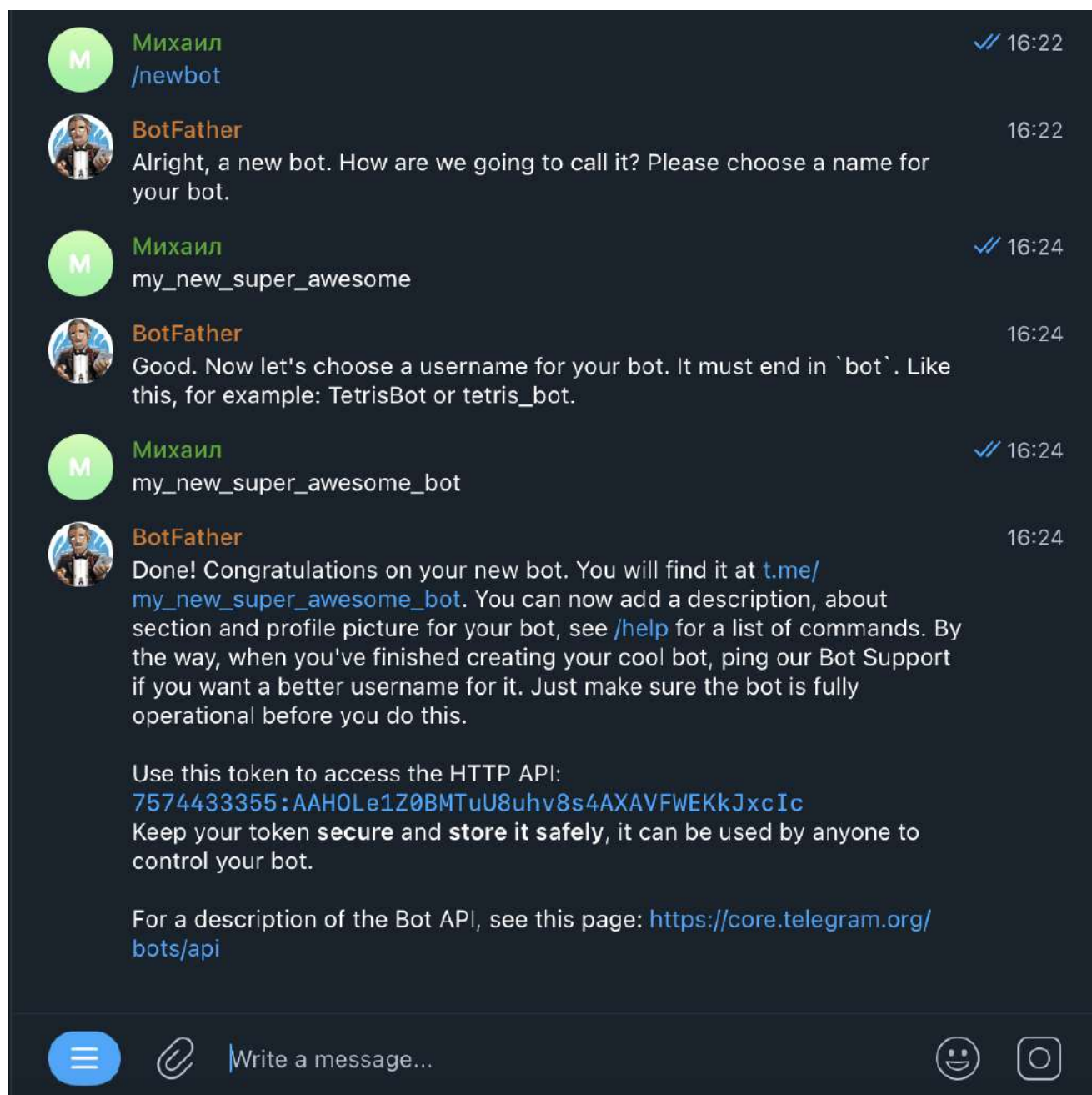
Запускаем BotFather и выбираем команду `/newbot`

ℹ️ Далее через этот бот Вы сможете редактировать и настраивать свои боты



После запуска команды на создание нового бота необходимо дать ему уникальное имя.

⚠ имя бота должно заканчиваться на `bot` !



В результате вы получите ссылку на своего бота:

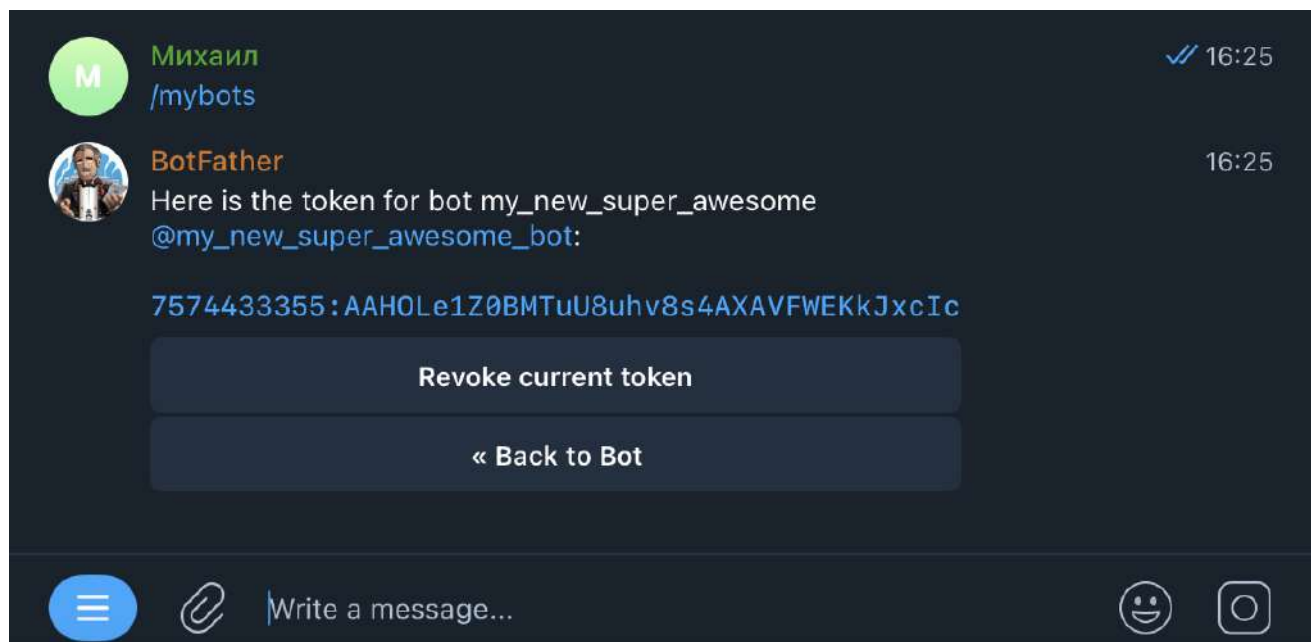
@my_new_super_awesome_bot

и **СЕКРЕТНЫЙ** токен, через который будет определяться право управления ботом со стороны Вашего сервера!

7574433355:AAH0Le1Z0BMTuU8uhv8s4AXAVFEKkJxcIc

⚠ Слова СЕКРЕТНЫЙ токен присутствуют в тексте не случайно! Этот токен нельзя нигде и никому показывать и пересылать иначе Ваш бот начнет жить своей жизнью и результат этого может сказаться на Вас самым печальным образом!

В настройках бота можно переопределить скомпрометировавший себя токен (я это уже сделал) поэтому показываю Вам старый без опаски!



❗ Для наглядности я буду показывать старый токен в коде, как будто он все еще действует.

Создание сервера для бота

Для создания сервера для бота необходимо скачать дополнительный официальный пакет [pyTelegramBotAPI](#)

```
pip install pyTelegramBotAPI
```

❗ На странице пакета есть документация на русском и английском языке

Не вдаваясь в детали, напишем следующий код:

```
import telebot
import serial

from CONFIG import RP_PORT

bot = telebot.TeleBot("7574433355:AAH0Le1Z0BMTuU8uhv8s4AXAVFEKkJxcIc")
s = serial.Serial(port=RP_PORT, baudrate=115200)

print("TeleBot ready!")

@bot.message_handler()
def echo_message(message):
    print(f"User_message: {message.text}")
    s.write(f"{message.text}\n".encode("utf-8"))
```



```

    answer = s.readline().strip().decode("UTF-8")
    print(f"RP_answer: {answer}\n")
    bot.reply_to(message, answer)

bot.infinity_polling()

```

В котором:

- В начале мы подключаем необходимые пакеты:

```

import telebot
import serial

```

- Создаем объекты для серийного подключения и телеграмм бота:

```

from CONFIG import RP_PORT

bot = telebot.TeleBot("7574433355:AAH0Le1Z0BMTuU8uhv8s4AXAVFEKkJxcIc")
s = serial.Serial(port=RP_PORT, baudrate=115200)

```

❗ Было бы неплохой идеей записать токен бота в константу и вынести ее в отдельный "секретный" файл, который не записывается в интернет и не показывается чужим!

```

from CONFIG import RP_PORT
from SECRET import API_TOKEN

bot = telebot.TeleBot(API_TOKEN)
s = serial.Serial(port=RP_PORT, baudrate=115200)

```

- Функция `echo_message(message)` определяет реакцию бота на поступающие в него сообщения:

```

@bot.message_handler()
def echo_message(message):
    print(f"User_message: {message.text}")
    s.write(f"{message.text}\n".encode("utf-8"))

    answer = s.readline().strip().decode("UTF-8")
    print(f"RP_answer: {answer}\n")
    bot.reply_to(message, answer)

```

Декоратор `@bot.message_handler()` без указания дополнительных параметров будет запускать декорируемую функцию при любом сообщении, передавая в нее объект

сообщения `message`.

В самой функции:

- Мы выводим в консоли текст сообщения в формате `User_message: {message.text}`;
- Добавляем к тексту сообщения перенос строки, кодируем и отправляем в микроконтроллер;
- Читаем ответ с микроконтроллера;
- Выводим его в консоль в формате `RP_answer: {answer}\n`;
- Отправляем его пользователю в качестве ответа на его сообщение.

Заметим, что сам код на микроконтроллере остался без изменений - ему абсолютно все равно, что происходит с внешней стороны - главное получить корректное текстовое сообщение!

Управление светодиодом через графический интерфейс

Чтобы у Вас не осталось чувства горького разочарования в случае отсутствия реализации телеграм бота, реализуем управление светодиодом через графический интерфейс!

Созданию графических интерфейсов мы посвятим (если успеем) отдельное занятие, пока воспримем предлагаемый код созерцательно и утилитарно:

```
import sys

import serial
from PyQt6.QtWidgets import QApplication, QWidget, QPushButton,
QVBoxLayout, QSizePolicy

from CONFIG import RP_PORT

class LedSwitchApp(QWidget):
    def __init__(self, serial_connection):
        super().__init__()
        # Настройка окна
        self.setWindowTitle("LedSwitcher")
        self.setGeometry(100, 100, 300, 200)
        # Создание кнопки
        self.button = QPushButton("Включить", self)
        self.button.setCheckable(True) # Кнопка может быть "нажата" или
"отжата"
        self.button.clicked.connect(self.toggle_button)
```

```

self.button.setSizePolicy(
    QSizePolicy.Policy.Expanding,
    QSizePolicy.Policy.Expanding)
# Настройка макета
layout = QVBoxLayout()
layout.addWidget(self.button)
self.setLayout(layout)
# Состояние кнопки (включено/выключено)
self.is_on = False
self.serial_connection = serial_connection

def toggle_button(self):
    # Изменение состояния кнопки
    self.is_on = not self.is_on
    if self.is_on:
        self.button.setText("Выключить")
        command = "ON\n".encode("UTF-8")
        self.serial_connection.write(command)
        answer =
self.serial_connection.readline().strip().decode("UTF-8")
        print(answer)
    else:
        self.button.setText("Включить")
        command = "OFF\n".encode("UTF-8")
        self.serial_connection.write(command)
        answer =
self.serial_connection.readline().strip().decode("UTF-8")
        print(answer)

if __name__ == "__main__":
    serial_connection = serial.Serial(port=RP_PORT, baudrate=115200)

    app = QApplication(sys.argv)
    window = LedSwitchApp(serial_connection=serial_connection)
    window.show()
    sys.exit(app.exec())

```

Ссылка на [файл с кодом](#)

❗ Для создания графического интерфейса необходимо установить дополнительный модель PyQt6. Скорее всего Вы уже поставили его для получения графиков с matplotlib во внешнем окне, если нет - то установите его:

```
pip install PyQt6
```