О простеньком боте замолвите слово

Долженкова Мария Львовна,
кандидат технических наук, заведующий кафедрой ЭВМ
Нижегородова Маргарита Владимировна,
кандидат педагогических наук, доцент кафедры САУ
Шмакова Наталья Александровна,
старший преподаватель кафедры САУ

Культура написания программного кода

В 2001 году Гвидо ван Россум, создатель языка программирования Python, создал документ PEP 8 (Python Enhancement Proposal, «Предложение по усовершенствованию Python»), основная цель которого – улучшить читабельность и логичность кода на Python.

Какие ключевые принципы были заложены в этот документ?

– «Читаемость имеет значение».

Для того чтобы написать фрагмент программы вам может потребоваться несколько минут или целый день. Этот фрагмент кода может стать частью проекта, над которым вы работаете. И каждый раз, возвращаясь к нему, вам нужно будет вспомнить, что делает этот код и зачем вы его написали. Написание понятного, читаемого кода свидетельствует о профессионализме. И показывает, что вы понимаете, как правильно структурировать свой код.

– «Явное лучше неявного».

В процессе написания программы, вам нужно назвать переменные, функции, модули и так далее. Подбор удачных имен сэкономит вам время и силы в дальнейшем. По названию вы сможете понять, что представляет собой определенная переменная или функция. Избегайте использования неподходящих имен, которые могут привести к трудно локализуемым ошибкам. Никогда не используйте однобуквенные имена I, O или l, их можно ошибочно приняты за 1 и 0 при определенных шрифтах.

```
O = 2 # Выглядит так, будто вы пытаетесь присвоить нулю значение 2
```

Вот некоторые из распространенных стилей именования в коде на Python.

- При именовании функций используйте слово/слова в нижнем регистре.
- Используйте одну букву, слово или слова в верхнем регистре для именования констант.
- Давая имя переменной, используйте букву, слово или слова в нижнем регистре.

- Модули должны иметь короткие имена, состоящие из маленьких букв, так как имена модулей отображаются в имена файлов, а некоторые файловые системы нечувствительны к регистру и обрезают длинные имена.
- Для удобства чтения разделяйте слова подчеркиванием.

	Константы	Переменные	Функции	Модули
ХОРОШО	*MY_CONST* COUNT N	my_variable var j	*function* *my_function*	vkapi
ПЛОХО	My-Const Count n	My-Variable STR I	*FUNCTION* *My-function*	My-First-VK-API

Рисунок 1 – Примеры наименований

Лучше использовать информативные имена. Допустим Вам нужно написать фрагмент программы, осуществляющей считывание персональных данных пользователя. Можно использовать следующий код.

```
x = 'Иванов Иван'
y, z = x.split()
print(z, y, sep=', ') # 'Иван, Иванов'
```

Это будет работать, потом придется вспоминать, что собой представляют x, y и z. Более правильный выбор имен будет примерно таким.

```
name = 'Иванов Иван'
first_name, last_name = name.split()
print(last_name, first_name, sep = ', ') # 'Иван, Иванов'
```

– «Красивое лучше уродливого».

Расположение строк кода влияет на его восприятие. Вертикальные пробелы (то есть пустые строки) могут значительно улучшить читаемость вашего кода. Код, который сливается в сплошной блок, сложно просматривать. Но и слишком много пустых строк делают код очень разреженным. Существует два основных правила использования пустых строк.

Окружайте функции двумя пустыми строкам. Например, так.

```
def top_level_function():
    return None


def multiply_by_two(x):
    return x * 2
```

Используйте пустые строки внутри функций, чтобы показать четкие шаги и разграничить логику внутри функции. Пусть нам необходимо вычислить разницу

между суммой квадратов и квадратом суммы элементов списка. Логично разбить функцию на три фрагмента.

```
def calculate_variance(number_list):
        sum_list = 0
        for number in number_list:
                sum_list = sum_list + number

        sum_squares = 0
        for number in number_list:
                sum_squares = sum_squares + number ** 2

        return mean_squares - mean ** 2
```

Ограничивайте длину строк 79 символами (а длину строк документации и комментариев – 72 символами). В общем случае не используйте обратный слеш в качестве перехода на новую строку. Если код заключен в круглые, квадратные или фигурные скобки, Python «поймет», что это одно предложение. В таком случае для улучшения читаемости полезно использовать отступ.

```
style_object(self, width, htight, color = 'black',
                        design = None)
if (width == 0 and color == 'red' and
        height == 0):
        pass
```

Отступы чрезвычайно важны в Python. Уровень отступа строк кода в Python определяет, как группируются операторы. Рекомендуется использовать табуляцию в четыре пробела на каждый уровень отступа. Нежелательно смешивание табуляции и пробелов в отступах.

Не следует разделять строки с помощью точек с запятой и использовать точки с запятой для разделения команд, находящихся на одной строке. То есть не используйте стиль оформления как в нижеследующем примере.

```
s = 'Strint';
b = 15; c = 7.2;
```

Экономно используйте скобки. Например, они не нужны в выражении return или в условной конструкции, если не требуется организовать перенос строки.

ХОРОШО	ПЛОХО

`if command == 'end':`
`    return False`
`while count <= 10:`
`    count += 1`
	`if (command == 'end'):`
	`    return (False)`
	`while (count <= 10):`
	`    count += 1`

Рисунок 2 – Примеры форматирования кода

– «Если реализацию сложно объяснить – идея точно плоха».

Документирование кода помогает быстрее разобраться в нем. Поэтому по мере написания кода необходимо использовать комментарии. Из комментария должно быть понятно, что делает данный фрагмент кода и как он относится к остальной части программы.

Вот некоторые важные моменты, которые следует помнить при комментировании.

- Ограничьте длину комментариев 72 символами.
- Используйте полные предложения, начинайте их с заглавной буквы.
- Отступайте в комментариях блока до уровня кода, который они описывают.
- Разделяйте абзацы строкой, содержащей один символ #
- Не забывайте обновлять комментарии при изменении кода.

```
for i in range(0, 10):
    # Проходим цикл переменной i 10 раз и выводим на экран ее значение,
    # а затем знак переноса строки
    print(i, '\n')

def quadratic(a, b, c, x):
    # Посчитать решение квадратного уравнения при помощи
    # формулы дискриминанта.
    #
    # У квадратного уравнения всегда есть 2 корня: x_1 и x_2.
    x_1 = (-b + (b ** 2 - 4 * a * c) ** (1 / 2)) / (2 * a)
    x_2 = (-b - (b ** 2 - 4 * a * c) ** (1 / 2)) / (2 * a)
    return x_1, x_2
```

Строки документации (docstrings) пишутся в первой строке любой функции или модуля, заключаются в три пары двойных (""") или одинарных (''') кавычек. Завершающие многострочный комментарий кавычки, лучше располагать на отдельной строке.

```
def quadratic(a, b, c, x):
    """Решим квадратное уравнение через дискриминант.
    Форма квадратного уравнения:
    ax ** 2 + bx + c = 0
    У квадратного уравнения всегда 2 решения: x_1 & x_2.
    """
    x_1 = (-b + (b ** 2 - 4 * a * c) ** (1 / 2)) / (2 * a)
    x_2 = (-b - (b ** 2 - 4 * a * c) ** (1 / 2)) / (2 * a)
    return x_1, x_2
```

– «Разреженное лучше плотного».

Пробелы могут быть очень полезны в выражениях и операторах.

- Присваивания (=, +=, -= и так далее);
- сравнения (==, !=, >, <, >=, <=) и включения (is, is not, in, not in);
- логические операторы (and, not, or).

Все их нужно окружать пробелами с обеих сторон. При этом допустимо добавлять пробелы только вокруг операторов с самым низким приоритетом, особенно при выполнении математических манипуляций.

Не следует добавлять пробелы в следующих ситуациях.

- Сразу после открывающей скобки и непосредственно перед закрывающей;
- для отделения запятой, точки с запятой или двоеточия;
- перед открывающей скобкой, с которой начинается список аргументов вызова функции, индекс или срез;
- между запятой и закрывающей круглой скобкой;
- для выравнивания операторов присваивания.

ХОРОШО	ПЛОХО

`y = x ** 2 + 5`
`z = (x + y) * (x - y)`
`if x > 5 and x % 2 == 0:`
`    print('x больше 5 и чётное!')`
`my_list = [1, 2, 3]`
`print(x, y)`
`double(3)`
`list[3]`
`tuple = (1,)`
`var1 = 5`
`var2 = 6`
`some_long_var = 7`
	`y = x ** 2 + 5`
	`z = (x + y) * (x - y)`
	`if x > 5 and x % 2 == 0:`
	`    print('x больше 5 и чётное!')`
	`my_list = [ 1, 2, 3, ]`
	`print(x , y)`
	`double (3)`
	`list [3]`
	`tuple = (1, )`
	`var1 = 5`
	`var2 = 6`
	`some_long_var = 7`

Рисунок 3 – Еще один пример форматирования кода

Теперь вы знаете, как писать качественный, читаемый код на Python, используя рекомендации, изложенные в PEP 8. Хотя эти рекомендации могут показаться педантичными, следование им может действительно улучшить ваш код. Это особенно важно, когда вы собираетесь делиться своим кодом с потенциальными работодателями или соавторами.

Ваш Telegram бот

В своих лабораторных работах вы будете работать надо программой «Мое расписание». Вы научитесь создавать расписание на каждый день, определять, когда стоит то или иное занятие, узнавать план на день. Давайте теперь посмотрим, как можно будет превратить такую программу в настоящего Telegram-бота, которым сможете воспользоваться не только вы, но и ваши друзья у которых установлен Telegram.

Но для начала немного теории. Для того, чтобы понимать что мы делаем, разрабатывая Telegram-бота, давайте обсудим следующие темы.

Как организована передача информации между компьютерами в сети?

Как работает Telegram и его боты?

Мы должны понимать, что любой компьютер, подключенный к сети Internet, связан с любым другим компьютером, подключенным к этой сети. И между ними

может быть организована передача информации. Можно сказать, что Internet похож на почту, зная адрес вашего друга, вы можете отправить ему письмо. Так и компьютеры, общаясь между собой, обмениваться информацией: отправляют запросы и получают ответы.

Представьте, что вы набираете в адресной строке какой-то адрес сайта, например, yandex.ru. Что при том происходит между нажатием кнопки enter и отображением главной страницы сайта на экране Вашего компьютера?

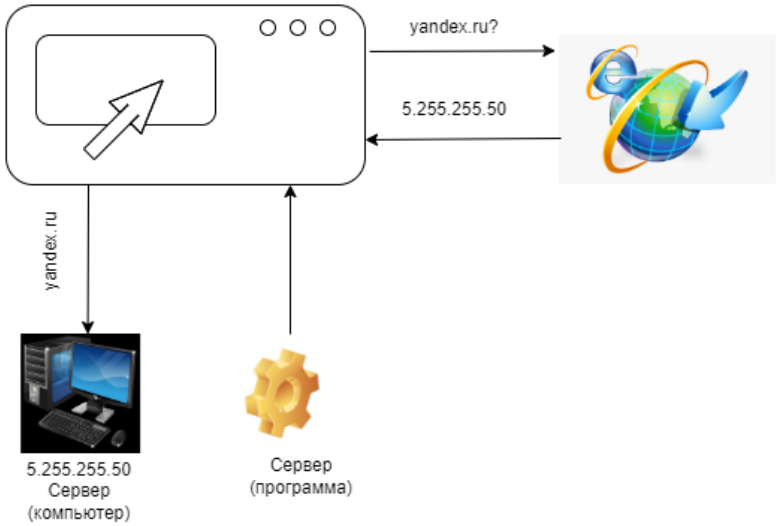

Рисунок 4 – Схема передачи данных

1) Браузер некоторым образом узнает адрес компьютера, который может ответить на запрос «Открой yandex.ru» (как это происходит в деталях – выходит за рамки нашего курса).

2) Браузер формирует запрос по специальным правилам и отправляет его по сети этому компьютеру.

3) На компьютере установлена специальная программа, которая умеет обрабатывать такие запросы. Она называется «сервер» (и сам компьютер тоже иногда называют сервером).

4) Программа (сервер) формирует ответ и отправляет его обратно на компьютер, отправивший запрос (он называется клиентом).

5) Браузер на компьютере-клиенте отображает пришедший ему ответ в соответствии с его типом и правилами.

Таким образом, мы получили модель, известную как «клиент-сервер», в которой клиент посылает запрос по особым правилам, а сервер формирует ответ на этот запрос.

Переписываемся в Telegram

Telegram работает по такой же модели.

В данном случае клиентом будет выступать ваш мессенджер. Он постоянно (несколько раз в секунду) обращается к серверу, который расположен в сети Internet с одним и тем же запросом: «Есть ли для меня новые сообщения?».

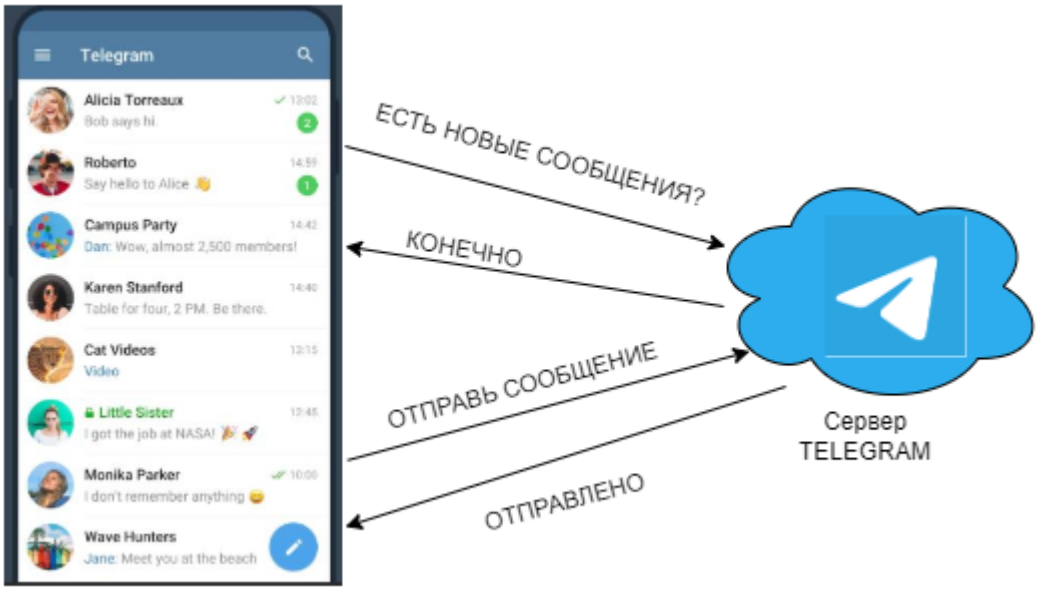

Рисунок 5 – Схема работы Telegram

Если такие сообщения есть, сервер посылает ответ: «Да, такие сообщения есть». И пересылает все эти сообщения, их текст, картинки, все, что было накоплено к этому моменту.

Итак, мы получили сообщение.

А что происходит, если мы пишем ответ? Наша клиентская программа формирует специальный запрос: «Отправь сообщение определенному контакту». Сервер отвечает: «Будет исполнено». И вы видите серую галочку в своем мессенджере. После этого, клиент вашего друга отправляет серверу свой запрос: «Есть ли для меня сообщения?». Сервер видит сообщение, которое вы отправили, и передает адресату текст вашего сообщения, а у вас появляется синяя галочка.

Давайте добавим в эту схему бота.

Бот – специальная программа, созданная, чтобы автоматически обрабатывать и отправлять сообщения. Пользователи взаимодействуют с ботом при помощи сообщений, отправляемых через обычные или групповые чаты.

1) Клиент отправляет сообщение боту. Запрос идет на Telegram-сервер.
2) Сервер сохраняет сообщение и ждет, пока к нему подключится бот.
3) Бот подключается и забирает сообщение.
4) Бот обрабатывает сообщение и отправляет ответ на сервер.
5) Как только появляется запрос клиента, сервер отдает ему ответ бота.

Так как эти запросы и ответы происходят несколько раз в секунду, мы почти не видим задержки передачи сообщений.

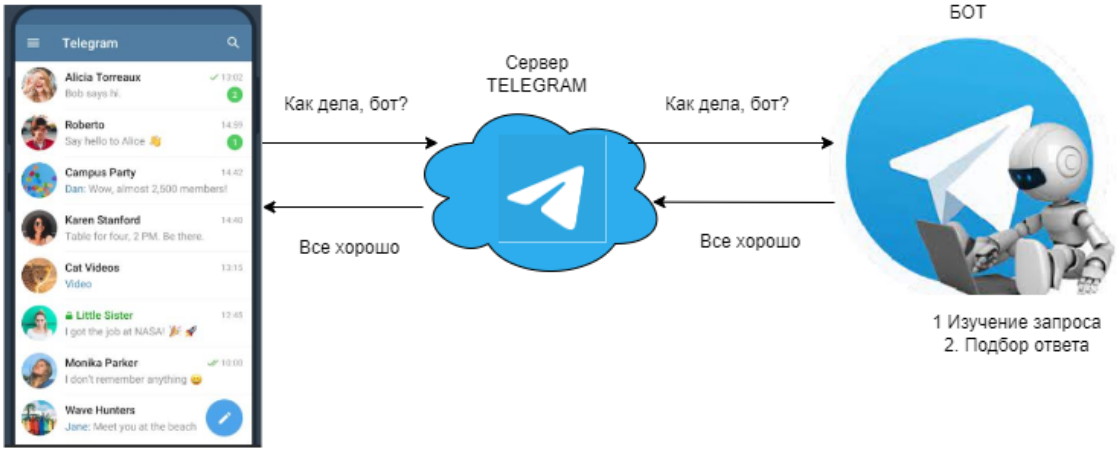

Рисунок 6 – Схема передачи сообщений с участием бота

Подготовка и создание Telegram бота

Чтобы бот мог работать его необходимо зарегистрировать в сети Telegram.

1) Находим в Telegram контакт @BotFather.
2) Отправляем ему /newbot.
3) Выбираем имя бота (имя бота может повторяться)
4) Выбираем никнейм бота (должен быть уникальным и заканчиваться на Bot).
5) Копируем токен.

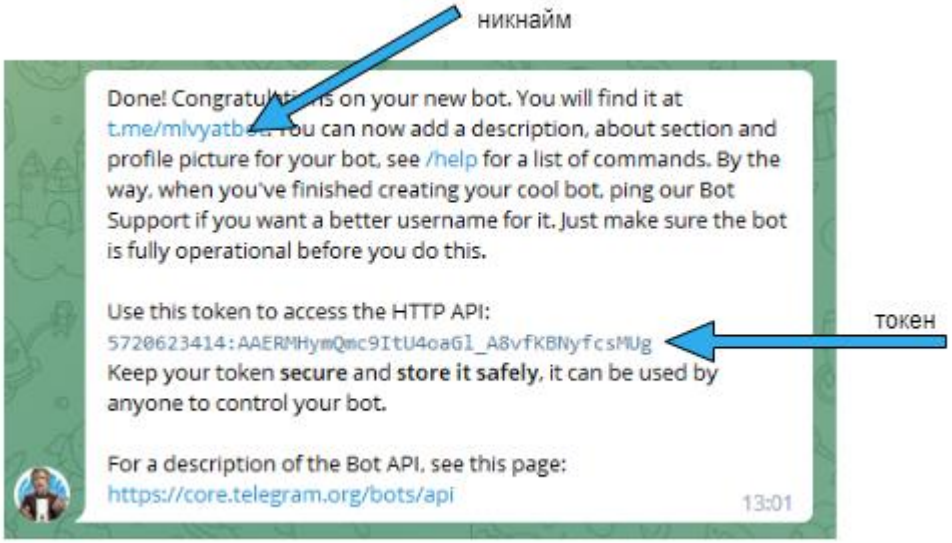

Рисунок 7 – Получение учетных данных для нового бота

Для создания бота воспользуемся облачной платформой PythonAnywhere, которая преимущественно предназначена для запуска приложений Python. Мы используем сайт pythonanywhere.com.

После регистрации создаем новый файл (обязательно .py).

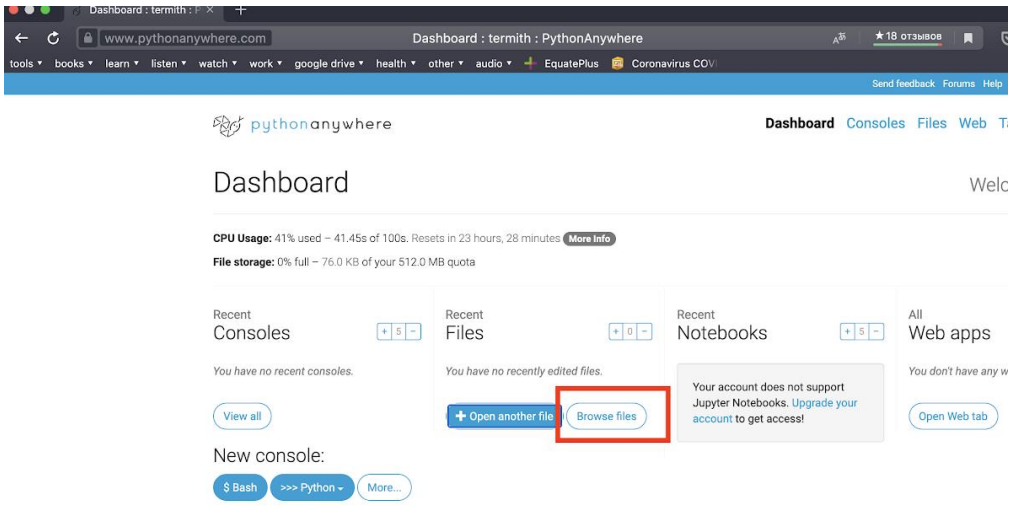

Рисунок 8 – Интерфейс PythonAnywhere

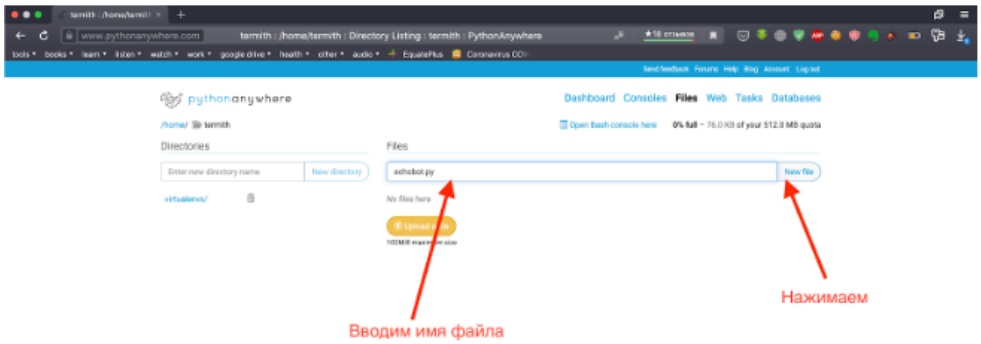

Рисунок 9 – Интерфейс PythonAnywhere

Для создания Telegram-ботов можно использовать различные библиотеки, которые берут на себя взаимодействие с серверами Telegram. Программисту остается лишь написание логики ответа на сообщения.

Таких библиотек для Python множество, мы будем использовать telebot (https://github.com/eternnoir/pyTelegramBotAPI).

Устанавливаем библиотеку. По кнопке «Run bash console» переходим в консоль. В консоли вводим следующий код для установки библиотеки.

```
pip3 install --user pytelegrambotapi
```

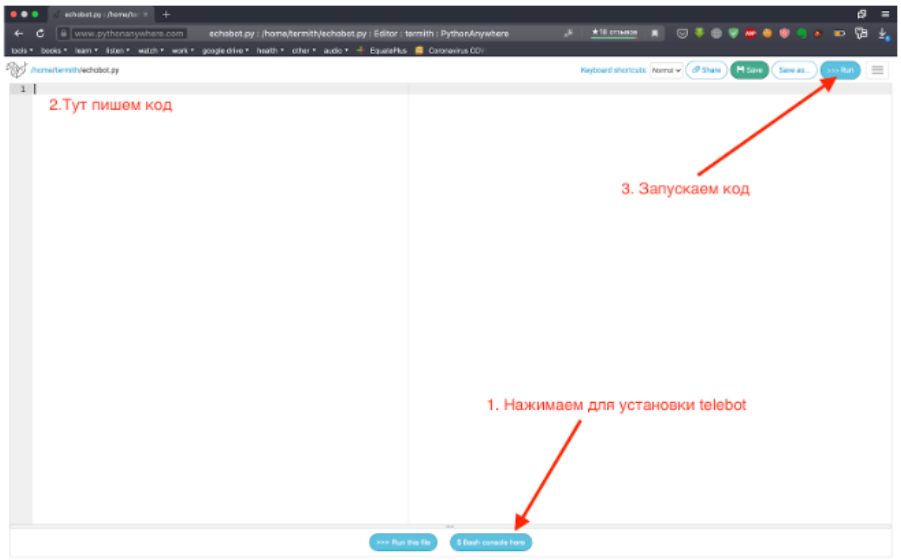

Рисунок 10 – Установка библиотеки

Как выглядит простейший бот? Пишем и запускаем следующий код.

```
import telebot
#Помещаем токен, который получили ранее, в переменную.
token = ''
bot = telebot.TeleBot(token)
#Создаем переменную bot.
#Это объект. Внутри него уже есть функции, которые мы будем вызывать
@bot.message_handler(content_types = ["text"])
#Регистрация функции обработчика сообщений типа «текст»
#Указывает, какие именно сообщения будет обрабатывать определяемая дальше функция.
def echo(message):
#Определение функции echo, принимающей один параметр – сообщение.
    bot.send_message(message.chat.id, message.text)
#Тело функции. Отправляем сообщение в тот же чат, откуда получили (message.chat.id)
#с тем же текстом (message.text).
bot.polling(none_stop = True)
#Отправка запросов к серверам Telegram, используя токен бота.
```

Именно токеном, а не именем, представляется бот на сервере. Если сообщения для бота есть, то вызывается обработка. В данном случае это функция echo.

Сделаем бота на основе программы-планировщика, рассмотренной в прошлой лекции.

```
while True:
    command = input("Введите команду")
    if command == 'ADD':
        add_task()
    elif command == 'SHOW':
        show_tasks()
    elif command == 'FIND':
        find_day();
    elif command == 'END':
```

```
            break
else:
    print('Такой команды нет')
```

У нас есть бесконечный цикл, в котором мы анализируем команды пользователя. Похожим образом будет устроен бот. Бесконечный цикл организуется с помощью функции bot.polling(none_stop = True). Мы постоянно будем запрашивать у сервера сообщения так же, как запрашивали у пользователя команду с помощью input().

Начнем с самой простой функции HELP. Она будет принимать сообщения и отправлять обратно в чат константу HELP с набором доступных команд. Перед каждым именем команды добавим символ «/», так как именно с него должна начинаться любая инструкция для бота.

```
HELP= '''
Список доступных команд:
/SHOW - напечатать все задачи на заданную дату
/ADD - добавить задачу
/FIND - узнать, когда запланирована задача
/HELP - напечатать help
'''
@bot.message_handler(commands = ['HELP'])
def help(message):
    bot.send_message(message.chat.id, HELP)
```

Давайте добавим команду ADD. Изменим формат команды. Будем сразу через пробел указывать день и задачу, которую мы назначаем.

```
/ADD 1 сдать зачет
```

С помощью функции split выбелим из команды нужные параметры.

```
_, day, task = message.text.split(maxsplit = 2)
```

Модифицируем функцию add_task().

```
def add_task(day, task):
    #запись новой задачи
    if day <= len(tasks) - 1:
        tasks[day].append(task)
    else:
        for i in range(len(tasks) - 1, day):
            tasks.append([])
        tasks[len(tasks) - 1].append(task)
```

И вызовем эту функцию из бота.

```
@bot.message_handler(commands = ['ADD'])
def add(message):
    _, day, task = message.text.split(maxsplit = 2)
    add_todo(date, task)
    bot.send_message(message.chat.id, f'Задача {task} добавлена на дату {day}')
```

Итоговый текст бота

```
HELP='''У меня есть система команд
 /ADD, если хотите добавить задачу,
 /SHOW, если хотите посмотреть задачи на день,
 /FIND, чтобы узнать день, на который запланирована задача.'''

def add_task(day, task):
	#запись новой задачи
	if day <= len(tasks) - 1:
		tasks[day].append(task)
	else:
		for i in range(len(tasks) - 1,day):
			tasks.append([])
	tasks[len(tasks) - 1].append(task)

def show_tasks(day):
	'''просмотр задачи на заданный день'''
	text = ''
	if day <= len(tasks) - 1:
		for i in tasks[day]:
			text = text + '[] ' + i + '\n'
	else:
		text = 'На этот день нет планов'
	return text

def find_day(task):
	'''определение в какой день запланирована задача'''
	flag = 0
	for i in range(len(tasks)):
		for j in tasks[i]:
			if j == task:
				ans = 'Задача запланирована на день ' + str(i)
				flag = 1
				break
		if flag == 1:
			break
	else:
		ans = 'Такая задача не запланирована'
	return ans

tasks = []
import telebot
import pickle

token = '5720623414:AAERMHymQmc9ItU4oaGl_A8vfKBNyfcsMUg'

bot = telebot.TeleBot(token)

@bot.message_handler(commands = ['HELP'])
def help(message):
	bot.send_message(message.chat.id, HELP)
```

```
@bot.message_handler(commands = ['FIND'])
def find(message):
        task = message.text.split()[1]
        res = find_day(task)
        bot.send_message(message.chat.id, res)

@bot.message_handler(commands = ['ADD'])
def add(message):
        _, day, task = message.text.split(maxsplit = 2)
        add_task(int(day), task)
        bot.send_message(message.chat.id, f'Задача {task} добавлена на дату {day}')

@bot.message_handler(commands = ['SHOW'])
def print_(message):
        day = int(message.text.split()[1])
        res = show_tasks(day)
        bot.send_message(message.chat.id, res)

bot.polling(none_stop = True)
```

Библиотека https://github.com/eternnoir/pyTelegramBotAPI (telebot) позволяет создавать более сложную логику для ботов. Например, на этом занятии мы использовали команды. Кроме этого в библиотеке доступны следующие конструкции.

- content_types – вид сообщения (текст, аудио и так далее);
- commands – список команд;
- regexp – регулярное выражение;
- func – произвольная функция, возвращающая True или False.

Более подробную информацию о работе с telebot можно найти по ссылке https://github.com/eternnoir/pyTelegramBotAPI#message-handlers.

Также с помощью этой библиотеки можно отправлять в чат не только текст, но и аудио, видео и даже стикеры.