**Модуль pyTelegramBotAPI telebot**

Изначально перейдем в телеграмм и создадим там бота, с помощью системного бота тг FatherBot. Надо нажать new bot ввести название после надо ввести его название как ссылку, …bot в таком виде. И нам выдадут токен, он понадобится позже.

Для настройки фото и описания бота все делается интуитивно через команды в фазер боте.

import telebot

pip install pyTelegramBotAPI – подключаем

создаем объект класса где в бот передаем токен, потом через бот и будем работать

bot = telebot.TeleBot(Token)

этот декоратор говорит что мы будем обрабатывать команду пользователя старт

@bot.message\_handler(commands=["start"])

Под декоратором указывает функцию которая будет выполнятся когда пользователь введет команду из списка **commands=["start"]** причем со / типо **/start**

def Hello\_world(x):

    bot.send\_message(x.chat.id, 'Привет')

наша команда это функция название любое, в параметр берем то что пользователь будет вводить. Потом обращаемся к методу **bot.send.message**

в x параметре лежит вся инфа о нас о боте и о пользователе.

Ее посмтреть можно, возвратив не строку к примеру как мы делали привет. А сам x

def Hello\_world(x):

    bot.send\_message(x.chat.id, x)

и передаем id чата в первом аргументе, во втором смс которое мы будем выводить пользователю. А в третьем можем использовать форматирование строк.

**parse\_mode = ‘html’**

и баловаться с текстом вторым аргументом по html тэгам к примеру

bot.send\_message(x1.chat.id, '<b>Полезная</b> <u><em>информация</em></u>', parse\_mode = 'html')

где тэги:

<b> - жирный шрифт

<u> - подчеркнутая строка

<em> - курсив

Бот при первом же запуске не будет работать по этому ему надо в конце прописать

bot.polling(none\_stop=True)

команда значит, что код будет работать всегда.

Тут мы используем другой декоратор для обработки не команд со / а простых смс, просто не прописываем внутри декоратора **commands=[""])**

@bot.message\_handler()

def txt(messsage):

    tp = messsage.text.lower()

    print(tp)

    print(type(tp))

    if type(tp) == str:

        bot.send\_message(messsage.chat.id, "Текст")

**message.text** – дает нам обратится к тексту пользователя. И его текст, ахуй, мы можем записать в переменную и столько с ней начудить. Тут я решил проверить тип данных введенных пользователем если они стр выводим что это текст, конечно потом понял, что там даже цифры будут стр. и придумал как это исправить, парсить данные пользователя под исключениями. Но пока руки не дотянулись.

Есть еще прекрасный метод **bot.reply.to (“”)** – это мы не просто выводим пользователю свое смс, а даем ответ на его сообщение. Ну не кайф?

import webbrowser

@bot.message\_handler(commands=["chanel"])

def channel(message):

    webbrowser.open("https://www.youtube.com/channel/UCR9xBSOvFgVSoaINkK-B5\_w")

еще добавили модуль новый веб браузер позволяет открывать страницу в браузере. И мы сражу же обработали эту команду

Добавление кнопок

Вне функции или в функции не важно, создаем объект класса markup

markup = types.InlineKeyboardMarkup()

теперь там объявим кнопки, я их еще запихнул в массив и прописал функцию которая их сразу выводит. И ЧТОБЫ КНОПКИ НЕ ДУБЛИРОВАЛИСЬ МАРКАП НУЖНО ОЧИЩАТЬ, И СДЕЛАТЬ ЕГО КАК ГЛОБАЛЬНУЮ ПЕРЕМЕННУЮ, УФФ.

Button\_curse = types.InlineKeyboardButton("Курсы Валют", callback\_data='curse')

Button\_game = types.InlineKeyboardButton("Мини-игра", callback\_data='game')

Button\_parse = types.InlineKeyboardButton("Парсинг вк", callback\_data='parse')

Button\_photo\_bw = types.InlineKeyboardButton("Калькулятор", callback\_data='bw')

Button\_photo\_code = types.InlineKeyboardButton("Че-то", callback\_data='code')

def un\_pack\_button():

    global markup

    markup = types.InlineKeyboardMarkup()

    markup.row(Button\_parse)

    markup.row(Button\_curse, Button\_game)

    markup.row(Button\_photo\_bw, Button\_photo\_code)

Внутри функции которая отвечает за команду их выводим, от так

@bot.message\_handler(commands=["start"])

def Start(mes):

un\_pack\_button()

    bot.send\_message(mes.chat.id, "<b> Варианты действий: </b> ", reply\_markup=markup, parse\_mode="html")

callback\_data=” ”

отвечает за функцию которая будет отвечать за действие этой кнопки.

    markup.row(Button\_parse)

    markup.row(Button\_curse, Button\_game)

    markup.row(Button\_photo\_bw, Button\_photo\_code)

markup.row можно использовать вместо add, для того чтобы указать кнопкам в каком ряду стоять и сделать мини дазйн.ы

Вот так объявляем функцию для кнопок

@bot.callback\_query\_handler(func=lambda call: True)

def Photo\_bw(call):

    if call.data == 'bw':

        bot.send\_message(call.message.chat.id, "Работает")

Еслия вдруг захочу создать отдельные функции для кнопок чтобы их красиво вызвать в оработчике, запомни чорт что нужно прописывать во всех параметр call или какойто другой который будет в основной функции их вызова, вот так

def but\_bw(call):

    bot.send\_message(call.message.chat.id, "Работает")

def but\_curse(call):

    bot.send\_message(call.message.chat.id, "Работает")

def but\_game(call):

    bot.send\_message(call.message.chat.id, "Работает")

def but\_parse(call):

    bot.send\_message(call.message.chat.id, "Работает")

def but\_code(call):

    bot.send\_message(call.message.chat.id, "Работает")

@bot.callback\_query\_handler(func=lambda call: True)

def Photo\_bw(call):

    if call.data == 'bw':

        but\_bw(call)

    elif call.data == 'curse':

        but\_curse(call)

    elif call.data == 'game':

        but\_game(call)

    elif call.data == 'parse':

        but\_parse(call)

    elif call.data == 'code':

        but\_code(call)

    else:

        ...

Bot.delte.message(call.mes.chat.id, call.mes.message\_id)

Функция для удаления смс

Message\_id -это айди предпоследнего смс в чате

Bot.edit.message(“Новое смс”, call.mes.chat.id, call.mes.message\_id)

Функция для редактирования смс

call.mes.message\_id это айди смс можно писать -1 -2 -3 итак гонять по всем смс в чате.

file\_id = message.photo[-1].file\_id

        file\_info = bot.get\_file(file\_id)

        file\_path = file\_info.file\_path

        file = bot.download\_file(file\_path)

Вот такой костяк мы используем чтобы получить изображения пользователя

file = bot.download\_file(file\_path)

ссылка на расположение файла

Далее открываем фото в pillow с помощью модуля io для бинарного открытия

Конвертим

bw\_image = image.convert('L')

После создаем временый файл с фото

temp\_path = 'temp.jpg'

bw\_image.save(temp\_path)

отправляем его и удаляем временный файл

with open(temp\_path, 'rb') as photo:

bot.send\_photo(call.message.chat.id, photo)

os.remove(temp\_path)