**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

**ИНСТИТУТ ЦИФРЫ**

**ОТЧЁТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

«Асинхронность, чат-боты»

Студентки 2 курса, ФИТ-211 группы

**Колесник Полины Олеговны**

Направление 02.03.02 – «Фундаментальная информатика и информационные технологии»

Руководитель:

Доцент Зимин А. И.

Работа защищена

« »

“ ” 2023 г.

Кемерово 2023 г.

**ОТЧЁТ О ПРОДЕЛАННОЙ РАБОТЕ**

**1 задание**

import asyncio

async def factorial(name, number):

   f = 1

   for i in range(2, number + 1):

       print(f"Task {name}: Compute factorial({i})...")

       f \*= i

       await asyncio.sleep(0)

   print(f"Task {name}: factorial({number}) = {f}")

async def main():

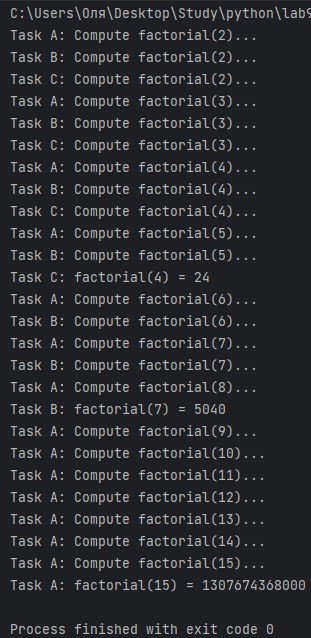
   tasks = [asyncio.create\_task(factorial("A", 15)), asyncio.create\_task(factorial("B", 7)),

            asyncio.create\_task(factorial("C", 4))]

   await asyncio.gather(\*tasks)

if \_\_name\_\_ == '\_\_main\_\_':

   asyncio.run(main())



**2 задание**

import requests

import asyncio

import os

async def get\_address(URL, field):

   result = requests.get(URL).json()

   return URL, result[field[URL]]

async def main():

   list = ['http://ip-api.com/json/', 'https://api.ipify.org/?format=json']

   slovar = {

       'https://api.ipify.org/?format=json': 'ip',

       'http://ip-api.com/json/': 'query'

   }

   tasks = [get\_address(item, slovar) for item in list]

   for future in asyncio.as\_completed(tasks):

       url, ip = await future

       print(f"Site: {url}: {ip}")

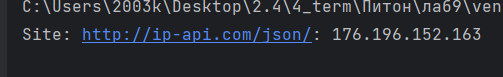
       break

if \_\_name\_\_ == '\_\_main\_\_':

   if os.name == 'nt':

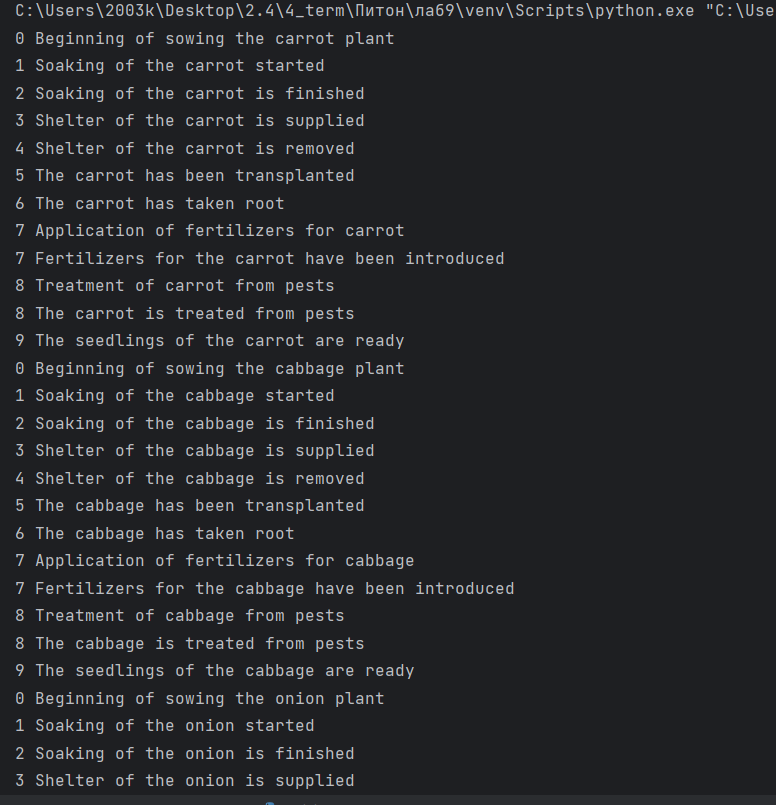
       asyncio.set\_event\_loop\_policy(asyncio.WindowsSelectorEventLoopPolicy())

   asyncio.run(main())



**3 задание**

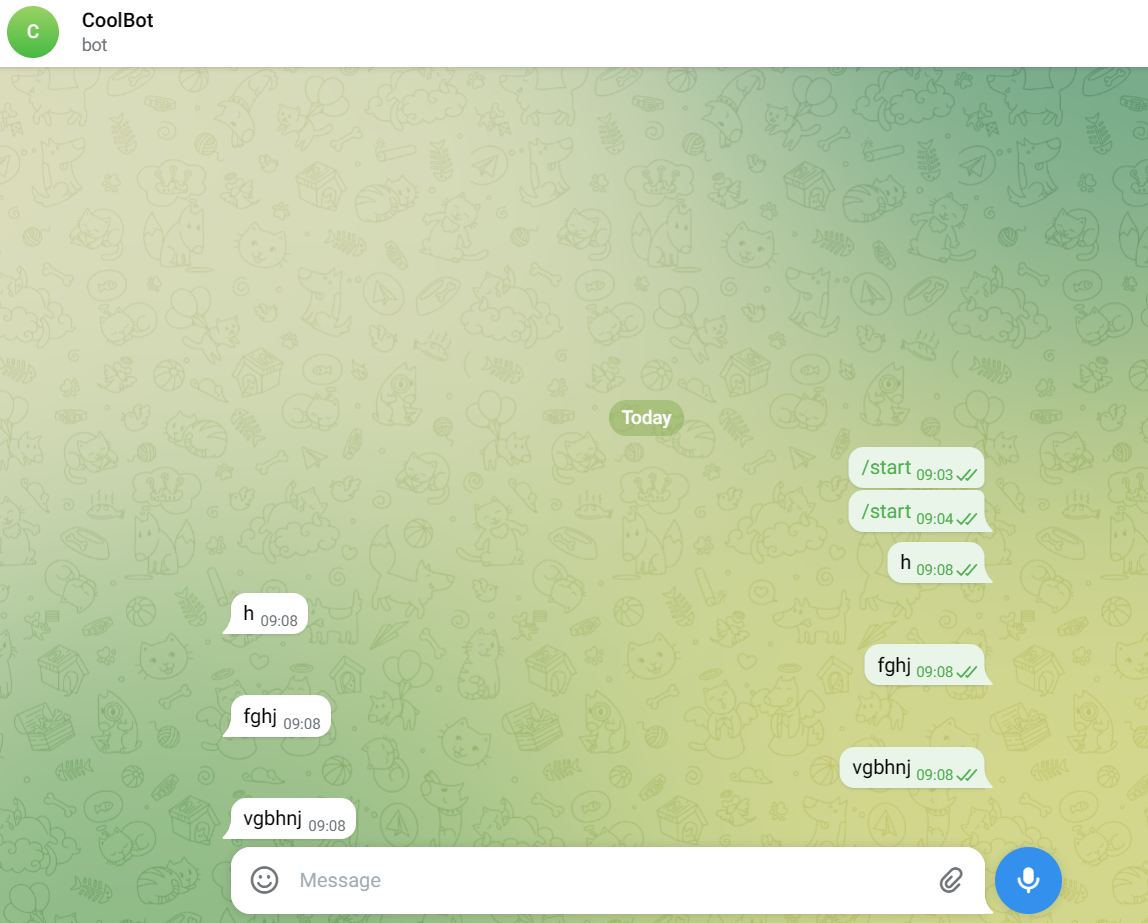
import asyncio  
from asyncio import create\_task  
  
  
async def soak(name, soakTime):  
 await asyncio.sleep(soakTime / 1000)  
 print(f"2 Soaking of the {name} is finished")  
  
  
async def sprout(name, sproutTime):  
 await asyncio.sleep(sproutTime / 1000)  
 print(f"4 Shelter of the {name} is removed")  
  
  
async def root(name, rootTime):  
 await asyncio.sleep(rootTime / 1000)  
 print(f"6 The {name} has taken root")  
  
  
async def top\_dressing(name):  
 await asyncio.sleep(3 / 1000)  
 print(f"7 Fertilizers for the {name} have been introduced")  
  
  
async def treatment(name):  
 await asyncio.sleep(5 / 1000)  
 print(f"8 The {name} is treated from pests")  
  
  
async def sowing(\*plants):  
 for plant in plants:  
 name, soakTime, sproutTime, rootTime = plant  
  
 print(f"0 Beginning of sowing the {name} plant")  
 print(f"1 Soaking of the {name} started")  
 await soak(name, soakTime)  
  
 print(f"3 Shelter of the {name} is supplied")  
 await sprout(name, sproutTime)  
  
 print(f"5 The {name} has been transplanted")  
 await root(name, rootTime)  
  
 print(f"7 Application of fertilizers for {name}")  
 await top\_dressing(name)  
  
 print(f"8 Treatment of {name} from pests")  
 await treatment(name)  
  
 print(f"9 The seedlings of the {name} are ready")  
  
  
data = [("carrot", 7, 18, 2), ("cabbage", 2, 6, 10), ("onion", 5, 12, 7)]  
asyncio.run(sowing(\*data))





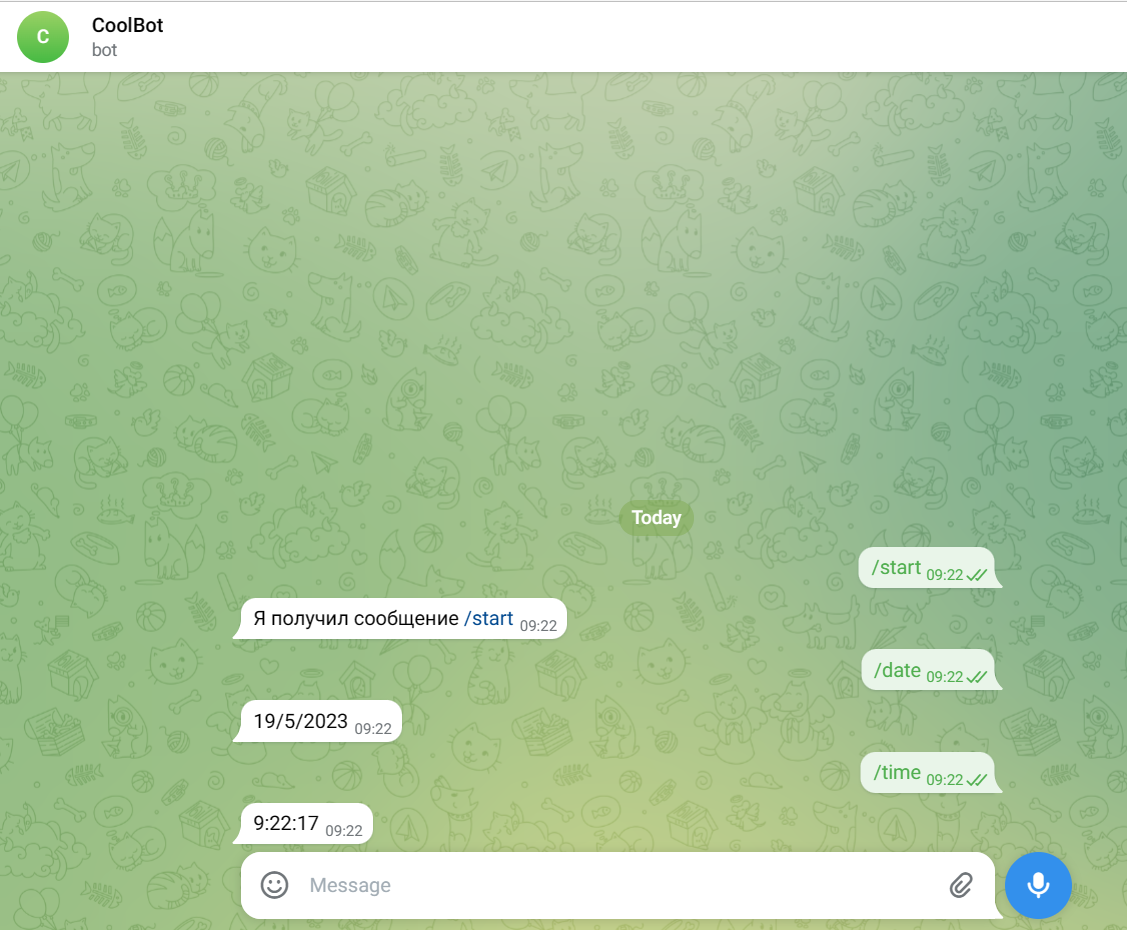
**4 задание**

from telegram.ext import Application, MessageHandler, filters  
  
API\_TOKEN = "6087748075:AAFeDWmJw0mz3dLivleidTu\_\_0nYt\_ou0Xs"  
  
  
async def echo(update, context):  
    await update.message.reply\_text(update.message.text)  
  
  
def main():  
    application = Application.builder().token(API\_TOKEN).build()  
  
    text\_handler = MessageHandler(filters.TEXT, echo)  
  
    application.add\_handler(text\_handler)  
  
    application.run\_polling()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    main()



**5 задание**

from telegram.ext import Application, MessageHandler, filters, CommandHandler  
import datetime  
  
API\_TOKEN = "6087748075:AAFeDWmJw0mz3dLivleidTu\_\_0nYt\_ou0Xs"  
  
  
async def echo(update, context):  
    await update.message.reply\_text("Я получил сообщение " + update.message.text)  
  
  
async def com\_time(update, context):  
    now = datetime.datetime.now()  
    hours = str(now.hour)  
    minutes = str(now.minute)  
    seconds = str(now.second)  
    await update.message.reply\_text(hours + ":" + minutes + ":" + seconds)  
  
  
async def com\_date(update, context):  
    now = datetime.datetime.now()  
    year = str(now.year)  
    month = str(now.month)  
    day = str(now.day)  
    await update.message.reply\_text(day + "/" + month + "/" + year)  
  
  
def main():  
    application = Application.builder().token(API\_TOKEN).build()  
    application.add\_handler(CommandHandler("time", com\_time))  
    application.add\_handler(CommandHandler("date", com\_date))  
    text\_handler = MessageHandler(filters.TEXT, echo)  
  
    application.add\_handler(text\_handler)  
  
    application.run\_polling()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    main()



**6 задание**

from random import randint

from telegram.ext import Application, Updater, CommandHandler, filters, MessageHandler, CallbackContext

from telegram import ReplyKeyboardMarkup, ReplyKeyboardRemove

API\_TOKEN = '6259887409:AAF8lqyviZQWKrekzCMgcqoW6mdg8grJy2M'

reply\_keyboard = [

   ['/dice', '/timer']

]

dice\_keyboard = [

   ["кинуть 1 шестигранный кубик", "кинуть 2 шестигранных кубика одновременно"],

   ["кинуть 20-гранный кубик", "вернуться назад"]

]

timer\_keyboard = [

   ["30 секунд", "1 минута"],

   ["5 минут", "вернуться назад"]

]

close\_keyboard = [

   ["/close"]

]

markup = ReplyKeyboardMarkup(reply\_keyboard, one\_time\_keyboard=False, resize\_keyboard=True)

markup\_dice = ReplyKeyboardMarkup(dice\_keyboard, one\_time\_keyboard=False, resize\_keyboard=True)

markup\_timer = ReplyKeyboardMarkup(timer\_keyboard, one\_time\_keyboard=False, resize\_keyboard=True)

markup\_close = ReplyKeyboardMarkup(close\_keyboard, one\_time\_keyboard=False, resize\_keyboard=True)

async def start(update, context):

   await update.message.reply\_text("start", reply\_markup=markup)

async def dice(update, context):

   await update.message.reply\_text("dice", reply\_markup=markup\_dice)

async def timer(update, context):

   await update.message.reply\_text("timer", reply\_markup=markup\_timer)

def roll\_cube(faces, count):

   nums = [randint(1, int(faces)) for \_ in range(count)]

   return nums

def remove\_job\_if\_exists(name, context):

*"""Удаляем задачу по имени.*

*Возвращаем True если задача была успешно удалена."""*

current\_jobs = context.job\_queue.get\_jobs\_by\_name(name)

   if not current\_jobs:

       return False

   for job in current\_jobs:

       job.schedule\_removal()

   return True

async def set\_timer(update, context, time):

   chat\_id = update.effective\_message.chat\_id

   due = int(time)

   job\_removed = remove\_job\_if\_exists(str(chat\_id), context)

   context.job\_queue.run\_once(task, due, chat\_id=chat\_id, name=str(chat\_id), data=due)

   text = f'Вернусь через {due} с.!'

   await update.effective\_message.reply\_text(text)

   #

   # chat\_id = update.effective\_message.chat\_id

   #

   #

   # if 'job' in context.chat\_data:

   #     old\_job = context.chat\_data['job']

   #     old\_job.schedule\_removal()

   # new\_job = context.job\_queue.run\_once(task, due, chat\_id=chat\_id, name=str(chat\_id), data=due)

   # context.chat\_data['job'] = new\_job

   # await update.message.reply\_text(f'Засек {due} секунд')

async def task(context):

   job = context.job

   await context.bot.send\_message(job.context, text='Указанное время истекло')

async def unset\_timer(update, context):

   if 'job' not in context.chat\_data:

       await update.message.reply\_text('Нет активного таймера', reply\_markup=markup\_timer)

       return

   job = context.chat\_data['job']

   job.schedule\_removal()

   del context.chat\_data['job']

   await update.message.reply\_text('Сбрасываю таймер', reply\_markup=markup\_timer)

async def ans(update, context):

   if update.message.text == "кинуть 1 шестигранный кубик":

       num = roll\_cube(6, 1)

       await update.message.reply\_text(\*num)

   elif update.message.text == "кинуть 2 шестигранных кубика одновременно":

       nums = list(map(str, roll\_cube(6, 2)))

       str\_nums = str(nums[0]) + ", " + str(nums[1])

       await update.message.reply\_text(str\_nums)

   elif update.message.text == "кинуть 20-гранный кубик":

       num = roll\_cube(20, 1)

       await update.message.reply\_text(\*num)

   elif update.message.text == "30 секунд":

       await set\_timer(update, context, 30)

   elif update.message.text == "1 минута":

       await set\_timer(update, context, 60)

   elif update.message.text == "5 минут":

       await set\_timer(update, context, 300)

   elif update.message.text == "вернуться назад":

       await update.message.reply\_text("вернуться назад", reply\_markup=markup)

application = Application.builder().token(API\_TOKEN).build()

application.add\_handler(CommandHandler("start", start))

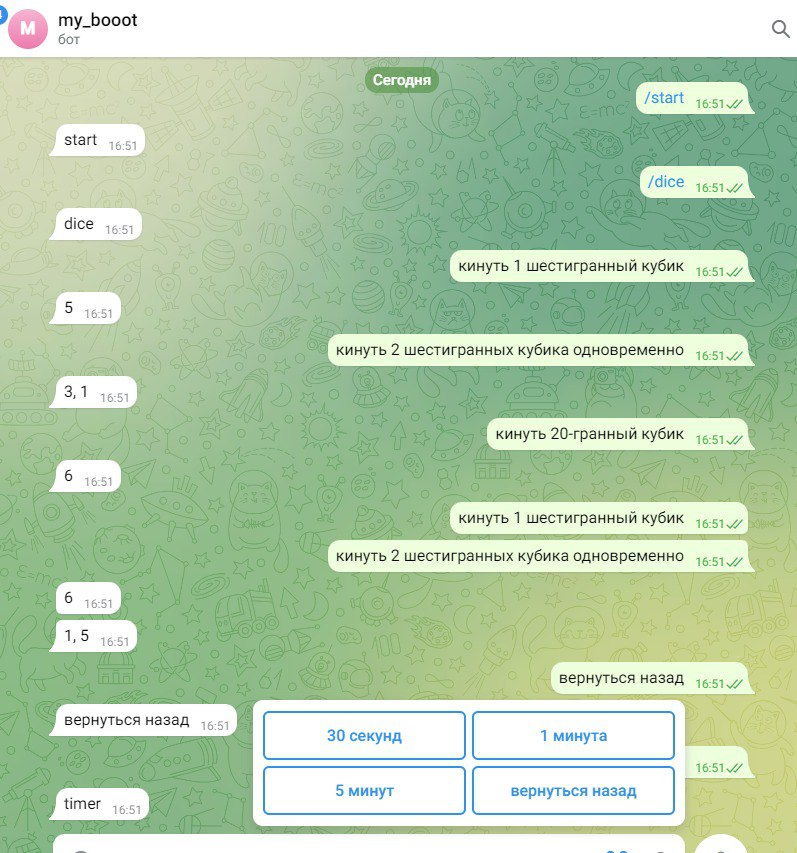
application.add\_handler(CommandHandler("dice", dice))

application.add\_handler(CommandHandler("timer", timer))

application.add\_handler(CommandHandler("close", unset\_timer))

application.add\_handler(MessageHandler(filters.TEXT, ans))

application.run\_polling()



**7 задание**

from telegram.ext import Application, CommandHandler, MessageHandler, filters

from telegram import ReplyKeyboardMarkup

API\_TOKEN = '6259887409:AAF8lqyviZQWKrekzCMgcqoW6mdg8grJy2M'

enter\_keyboard = [

   ['Перейти в зал №1']

]

room1\_keyboard = [

   ['Перейти в зал №2', 'Выход']

]

room2\_keyboard = [

   ['Перейти в зал №3']

]

room3\_keyboard = [

   ['Перейти в зал №1', 'Перейти в зал №4']

]

room4\_keyboard = [

   ['Перейти в зал №1']

]

markup\_enter = ReplyKeyboardMarkup(enter\_keyboard, one\_time\_keyboard=False, resize\_keyboard=True)

markup\_room1 = ReplyKeyboardMarkup(room1\_keyboard, one\_time\_keyboard=False, resize\_keyboard=True)

markup\_room2 = ReplyKeyboardMarkup(room2\_keyboard, one\_time\_keyboard=False, resize\_keyboard=True)

markup\_room3 = ReplyKeyboardMarkup(room3\_keyboard, one\_time\_keyboard=False, resize\_keyboard=True)

markup\_room4 = ReplyKeyboardMarkup(room4\_keyboard, one\_time\_keyboard=False, resize\_keyboard=True)

async def start(update, context):

   if update.message.text == '/start':

       await update.message.reply\_text(

           "Добро пожаловать!\n"

           "Пожалуйста, сдайте верхнюю одежду в гардероб\n",

           reply\_markup=markup\_enter)

       await update.message.reply\_text(

           "1 - Айвазовский")

   elif update.message.text == 'Перейти в зал №1':

       await update.message.reply\_text(

           "Вы зашли в 1 зал\n"

           "В данном зале представлены картины великого живописца Ивана Константиновича Айвазовского",

           reply\_markup=markup\_room1)

       await update.message.reply\_text(

           "2 - Клод Моне")

       await update.message.reply\_text(

           "Выход - выход")

   elif update.message.text == 'Перейти в зал №2':

       await update.message.reply\_text(

           "Вы зашли вo 2 зал\n"

           "В данном зале представлены картины Клода Моне",

           reply\_markup=markup\_room2)

       await update.message.reply\_text(

           "3 - Сальвадор Дали")

   elif update.message.text == 'Перейти в зал №3':

       await update.message.reply\_text(

           "Вы зашли в 3 зал\n"

           "В данном зале представлены картины испанского живописца Сальвадора Дали",

           reply\_markup=markup\_room3)

       await update.message.reply\_text(

           "4 - Леонардо да Винчи")

       await update.message.reply\_text(

           "1 - Айвазовский")

   elif update.message.text == 'Перейти в зал №4':

       await update.message.reply\_text(

           "Вы зашли в 4 зал\n"

           "В данном зале представлены картины итальянского художника Леонардо да Винчи",

           reply\_markup=markup\_room4)

       await update.message.reply\_text(

           "1 - Айвазовский")

   elif update.message.text == 'Выход':

       await update.message.reply\_text(

           "Всего доброго, не забудьте забрать верхнюю одежду в гардеробе")

def main():

   application = Application.builder().token(API\_TOKEN).build()

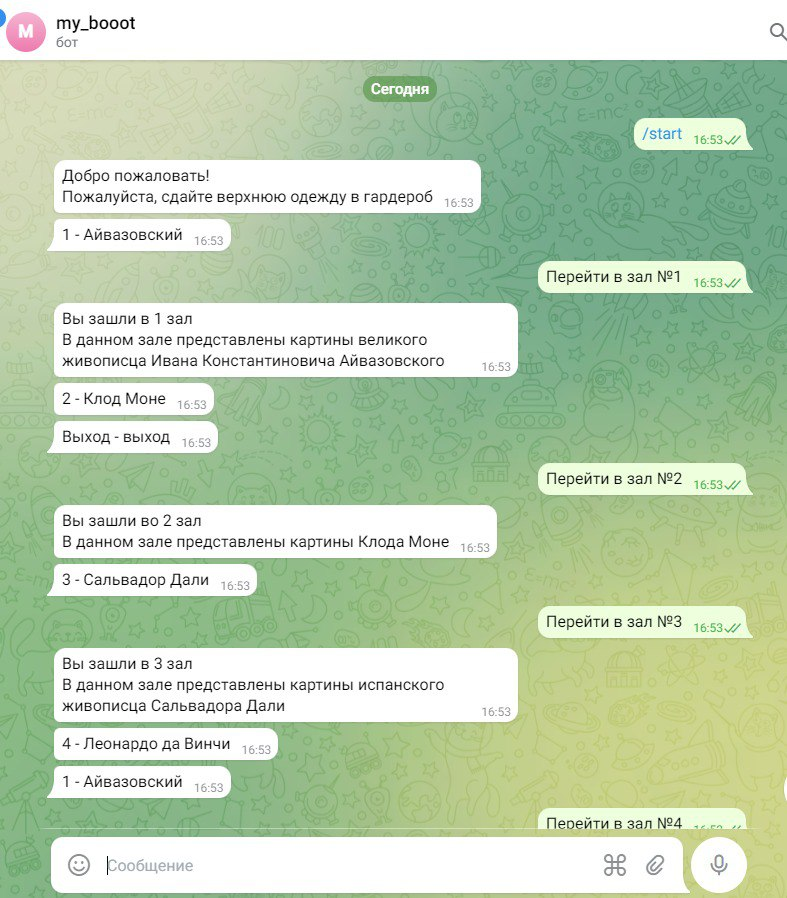
   application.add\_handler(CommandHandler("start", start))

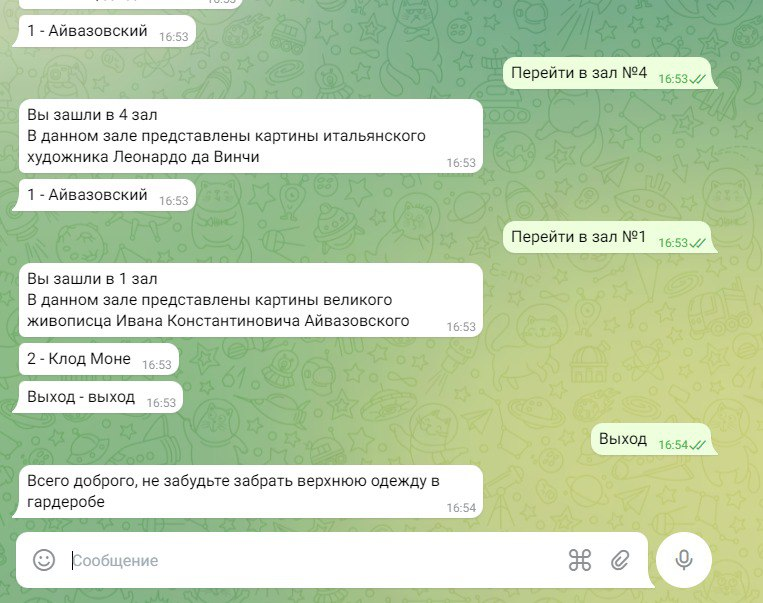
   application.add\_handler(MessageHandler(filters.TEXT, start))

   application.run\_polling()

if \_\_name\_\_ == '\_\_main\_\_':

   main()





**9 задание**

from telegram.ext import Application, MessageHandler, filters, CommandHandler

import requests

API\_TOKEN = "6022531502:AAF2lrV4vQsglVuMV9A\_X-GQAn63nlc10lo"

def get\_location(coords):

   PARAMS = {

       "apikey": "7a0332fa-0401-4a46-ac3c-da05786df316",

       "format": "json",

       "lang": "ru\_RU",

       "kind": "house",

       "geocode": coords

   }

   spisok = PARAMS["geocode"].split(", ")

   try:

       response2 = requests.get(f"https://static-maps.yandex.ru/1.x/?l=map&pt={spisok[0]},{spisok[1]},pm2rdm&z=15")

       map\_file = "map.png"

       with open(map\_file, "wb") as file:

           file.write(response2.content)

       response1 = requests.get(url="https://geocode-maps.yandex.ru/1.x/", params=PARAMS)

       json\_data = response1.json()

       address\_str = json\_data["response"]["GeoObjectCollection"]["featureMember"][0]["GeoObject"]["metaDataProperty"][

           "GeocoderMetaData"]["AddressDetails"]["Country"]["AddressLine"]

       return address\_str

   except Exception as e:

       return "Не могу определить адрес по этим координатам.\n\nОтправьте мне координаты (долгота, широта):"

async def text\_photo(update, context):

   coords = update.message.text

   address\_str = get\_location(coords)

   await context.bot.send\_photo(chat\_id=update.message.chat\_id, photo=open('map.png', 'rb'), caption=address\_str)

def main():

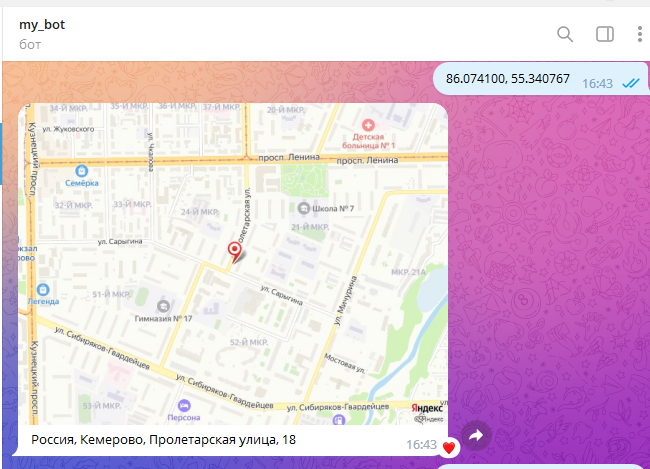
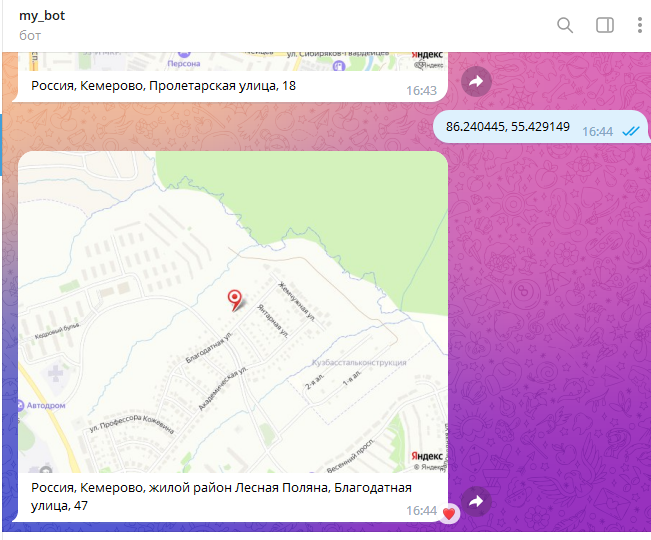
   application = Application.builder().token(API\_TOKEN).build()

   application.add\_handler(MessageHandler(filters.TEXT, text\_photo))

   application.run\_polling()

if \_\_name\_\_ == '\_\_main\_\_':

   main()



**12 задание**

import discord

import requests

config = {

   'token': 'MTEwOTc2MjQxMTI5ODI5NTg0OQ.Gbi8L\_.DCURActB9AIRWIp\_S\_LCBmyZ5Zf6Z9QhBRUsMI'

}

class HWBot(discord.Client):

   async def on\_ready(self):

       print(f'{self.user} Привет! Я подключился и готов показать котика или собачку')

   async def on\_message(self, message):

       if message.author == self.user:

           return

       if "собака" in message.content.lower():

           response = requests.get("https://dog.ceo/api/breeds/image/random").json()

           dog\_url = response['message']

           embed = discord.Embed(title="рандомная собака!", color=discord.Color.purple())

           embed.set\_image(url=dog\_url)

           await message.channel.send(embed=embed)

       elif "кот" in message.content.lower():

           response = requests.get("https://api.thecatapi.com/v1/images/search").json()

           cat\_url = response[0]['url']

           random\_cat\_png = discord.Embed(title="рандомный кот!", color=discord.Color.purple())

           random\_cat\_png.set\_image(url=cat\_url)

           await message.channel.send(embed=random\_cat\_png)

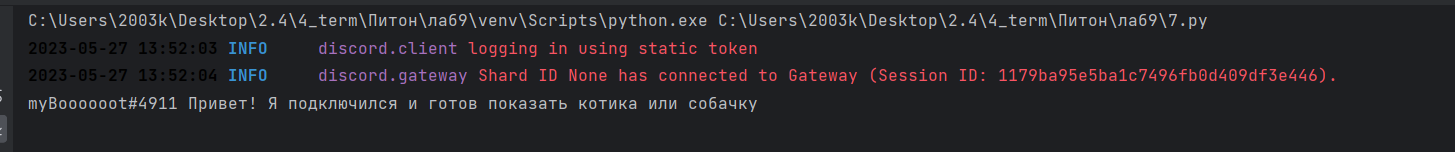
intents = discord.Intents.default()

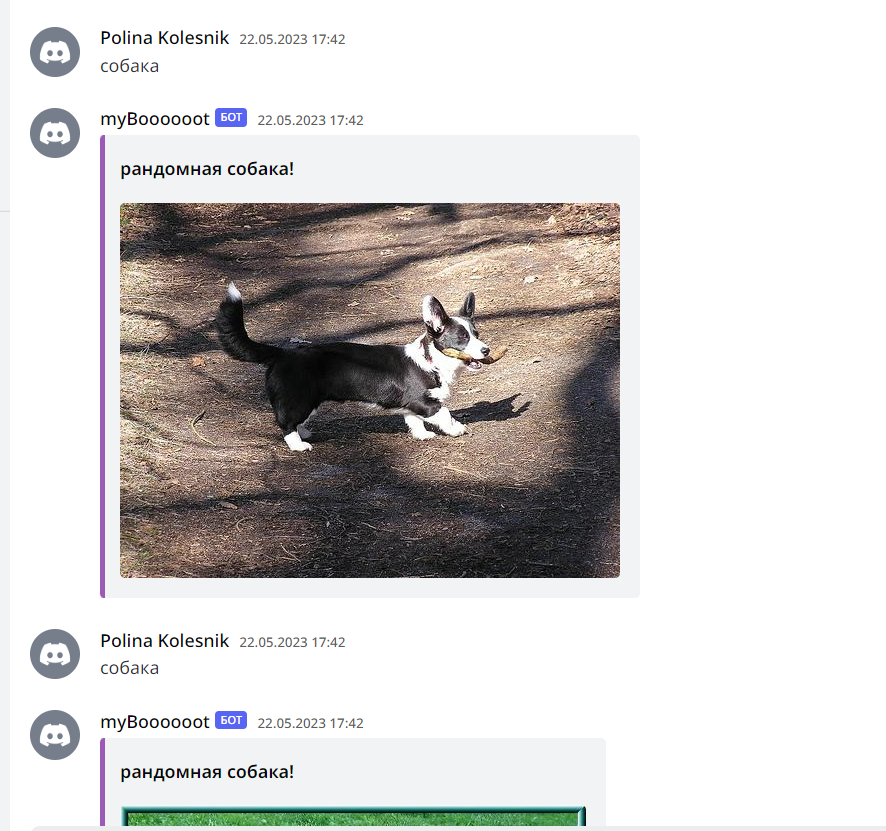
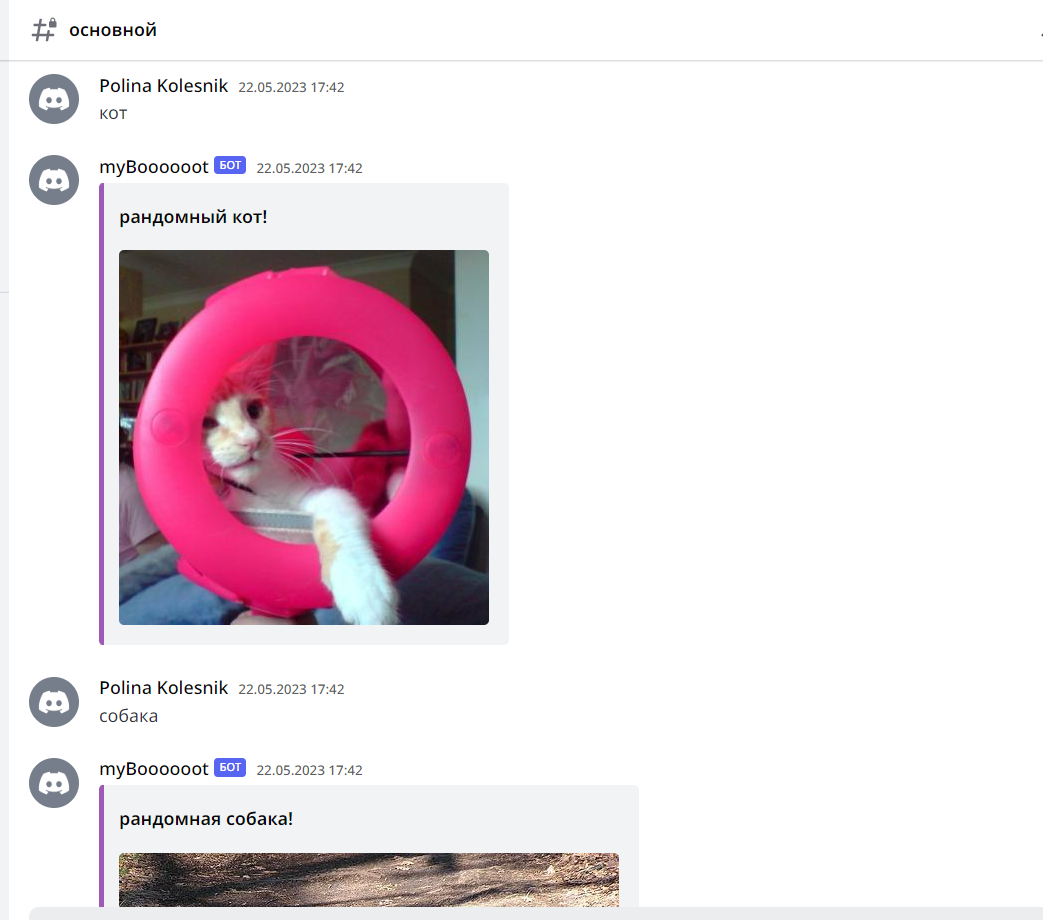
intents.members = True

intents.message\_content = True

client = HWBot(intents=intents)

client.run(config['token'])





**13 задание**

from time import sleep

import discord

from discord.ext import commands

config = {

   'token': 'MTEwOTc2MjQxMTI5ODI5NTg0OQ.Gbi8L\_.DCURActB9AIRWIp\_S\_LCBmyZ5Zf6Z9QhBRUsMI'

}

intents = discord.Intents.default()

intents.members = True

intents.message\_content = True

bot = commands.Bot(command\_prefix='#!', intents=intents)

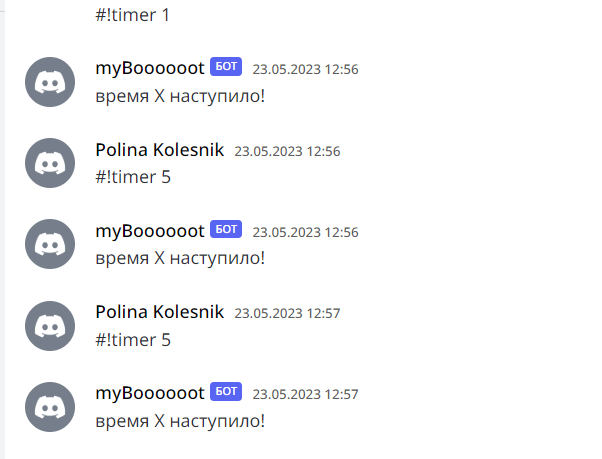
@bot.command(name='timer')

async def timer(ctx, seconds):

   sleep(int(seconds))

   await ctx.send("время X наступило!")

bot.run(config['token'])



**15 задание**

import discord

import requests

import json

intents = discord.Intents.default()

intents.members = True

intents.message\_content = True

client = discord.Client(intents=intents)

api\_key = '10410733-f10c-46a0-b1b2-d7ea273d38a0'

location = 'Moscow'

help\_message = '''Текущая погода и прогноз на несколько дней.

Команды:

!help\_bot - помощь

!place <место> - изменить место прогноза

!current - получить текущую погоду

!forecast <дней> - получить прогноз на указанное количество дней'''

def get\_weather\_data(location):

   url = f'https://api.weather.yandex.ru/v2/forecast'

   headers = {'X-Yandex-API-Key': api\_key}

   response = requests.get(url, headers=headers)

   data = json.loads(response.text)

   return data

def format\_weather\_data(data):

   fact = data['fact']

   temp = fact['temp']

   pressure = fact['pressure\_mm']

   humidity = fact['humidity']

   wind\_speed = fact['wind\_speed']

   wind\_dir = fact['wind\_dir']

   return f'Температура: {temp}°C\n' \

          f'Давление: {pressure} мм рт. ст.\n' \

          f'Влажность: {humidity}%\n' \

          f'Ветер: {wind\_speed} м/с, {wind\_dir}'

def get\_forecast\_data(location, days):

   url = f'https://api.weather.yandex.ru/v1/forecast?&lang=en\_US&limit={days}' \

         f'&hours=false&extra=false'

   headers = {'X-Yandex-API-Key': api\_key}

   response = requests.get(url, headers=headers)

   data = json.loads(response.text)

   forecast = ''

   for day in data['forecasts']:

       date = day['date']

       parts = day['parts']

       day\_temp = parts['day']['temp\_avg']

       day\_prec = parts['day']['prec\_mm']

       night\_temp = parts['night']['temp\_avg']

       night\_prec = parts['night']['prec\_mm']

       forecast += f'{date}: Днем {day\_temp}°C, ' \

                   f'{day\_prec} мм осадков. Ночью {night\_temp}°C, {night\_prec} мм осадков.\n'

   return forecast

@client.event

async def on\_message(message):

   global location

   if message.author == client.user:

       return

   if message.content.startswith('!help\_bot'):

       await message.channel.send(help\_message)

   elif message.content.startswith('!place'):

       location = message.content.split(' ')[1]

       await message.channel.send(f'Место прогноза изменено на {location}')

   elif message.content.startswith('!current'):

       data = get\_weather\_data(location)

       weather = format\_weather\_data(data)

       await message.channel.send(f'Погода в {location}:\n{weather}')

   elif message.content.startswith('!forecast'):

       days = int(message.content.split(' ')[1])

       if days > 7:

           await message.channel.send('Прогноз доступен только на 7 дней')

       else:

           forecast = get\_forecast\_data(location, days)

           await message.channel.send(f'Прогноз в {location} на {days} дней:\n{forecast}')

client.run('MTEwOTc2MjQxMTI5ODI5NTg0OQ.Gbi8L\_.DCURActB9AIRWIp\_S\_LCBmyZ5Zf6Z9QhBRUsMI')

