* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/165/groups/1257)

[Урок Чат-боты 3](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1668)

**Чат-боты 3**

1. [Введение](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1668/materials/4096#1)
2. [Асинхронное программирование](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1668/materials/4096#2)
3. [Подготовка](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1668/materials/4096#3)
4. [Простой бот](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1668/materials/4096#4)
5. [Объектно-ориентированный подход](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1668/materials/4096#5)
6. [Команды](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1668/materials/4096#6)
7. [Заключение](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1668/materials/4096#7)

**Аннотация**

*На заключительном уроке по созданию чат-ботов мы создадим бота для Discord, а также немного поговорим о таком понятии, как параллельное программирование.*

**Введение**

Сегодня мы создадим бота для Discord. Discord — бесплатный голосовой и текстовый чат, ориентированный в первую очередь на тех, кто играет в компьютерные игры. Однако его удобство, надежность и поддержка разных платформ позволяет использовать его не только в целях организации совместной игры. Есть много небольших команд разработчиков, которые используют Discord в рабочем процессе, в том числе и для автоматизации различных задач.

Для начала работы необходимо скачать клиент Discord с [официального сайта](https://discordapp.com/), установить его и зарегистрировать учетную запись, если ее еще нет.

Для работы с API Discord мы будем использовать библиотеку discord.py, которую сначала надо установить:

pip install discord.py

discord.py полностью реализует API Discord, и предоставляет возможность создавать собственных ботов как с использованием декораторов (как при создании своих обработчиков во flask), так и с использованием объектно-ориентированного подхода.

Документацию на библиотеку можно найти [тут](https://discordpy.readthedocs.io/en/latest/), а описание самого API — вот [тут](https://discordapp.com/developers/docs/intro).

Но прежде чем мы продолжим, надо упомянуть важный момент: discord.py — асинхронная библиотека. Это будет накладывать некоторый отпечаток на процесс разработки программ с ее использованием.

**Асинхронное программирование**

Все время до этого момента мы писали **синхронные** программы, то есть такие, которые выполняются поэтапно. Если в коде написан последовательный вызов двух функций, то в синхронной программе интерпретатор вызовет первую, дождется завершения ее выполнения, а только затем вызовет вторую.

Такой подход достаточно понятен в написании программы: у нас есть гарантия того, какой кусок кода выполнится после какого, гарантия, что какие-то ресурсы (например, стандартный поток ввода stdin) будут использованы именно теми функциями, от которых мы этого ожидаем в каждый конкретный момент времени. Однако такой подход далеко не всегда удобен.

Давайте представим, что операционная система нашего компьютера работает полностью синхронно. Мы выбрали файл и начали копировать его из одной папки в другую. При синхронном подходе на время копирования файлов наша работа с операционной системой заблокирована до завершения текущей операции. Согласитесь, звучит не очень.

Когда мы говорим об асинхронном программировании, то подразумеваем создание такого кода, который позволяет программе не ждать завершения некоторого процесса, а продолжать работу не зависимо от него. Например, мы начали скачивать в браузере какой-то файл и после этого переключились на другую вкладку и продолжили заниматься своими делами.

Сразу оговоримся: применение асинхронных механизмов при написании программы не обязательно сделает ее быстрее, а может даже и замедлить. Это связано с тем, что части асинхронного кода работают **в отдельных потоках**, которыми управляет процессор. Если у процессора только одно свободное ядро, то для выполнения асинхронной программы ему надо не только выделять время для каждого из потоков, но и тратить время на переключение между ними.

Изначально в Python для решения задач асинхронного программирования использовались **корутины**, основанные на генераторах. Но, начиная с Python 3.4, появился модуль asyncio, в котором реализованы механизмы асинхронного программирования. А в Python 3.5 появилась конструкция async/await.

С помощью ключевого слова async при создании функции можно указать интерпретатору, что она является корутиной и может выполняться асинхронно. С помощью ключевого слова await мы можем вызвать такую функцию. Давайте рассмотрим очень простой пример:

import asyncio

async def task(name):

print(f'{name} задача началась')

await asyncio.sleep(1)

print(f'{name} задача завершилась')

async def main():

await asyncio.gather(

task('Первая'),

task('Вторая'),

)

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

loop = asyncio.get\_event\_loop()

loop.run\_until\_complete(main())

Если мы запустим этот код, увидим такой вывод:

Первая задача началась

Вторая задача началась

Первая задача завершилась

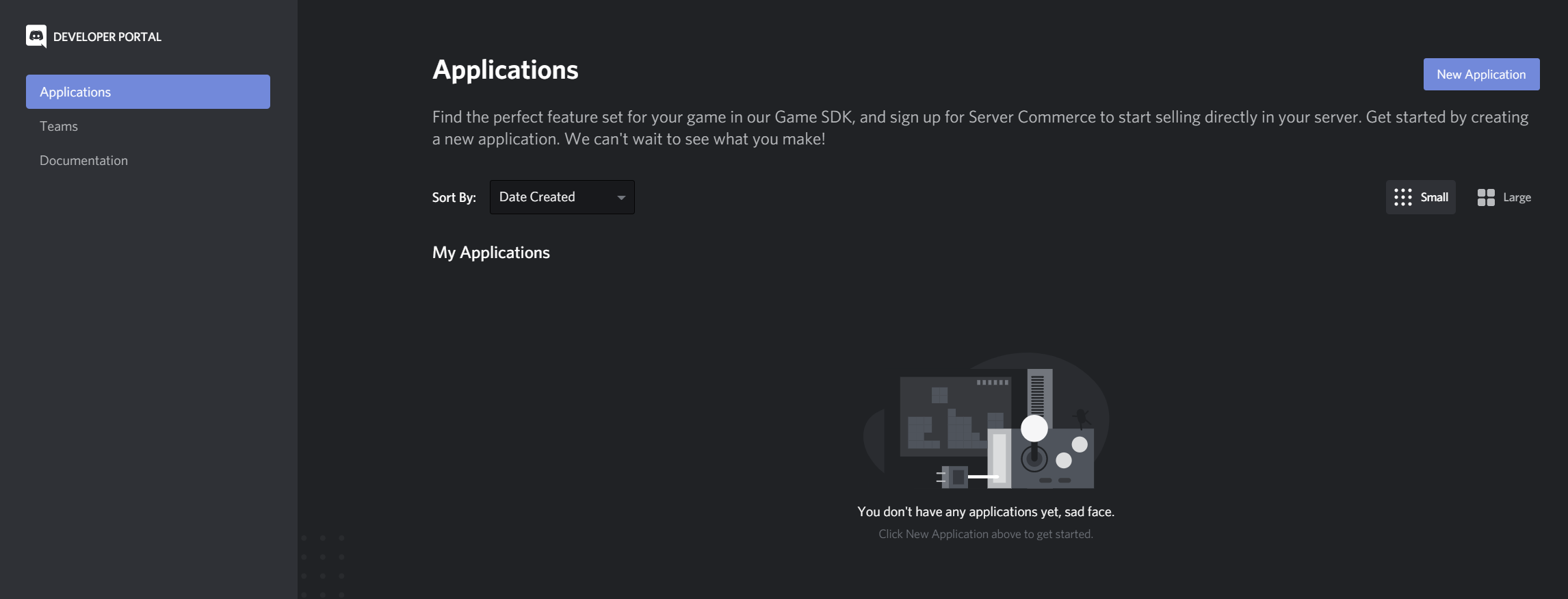
Вторая задача завершилась

Внутри модуля asyncio много всего интересного, о чем можно почитать в [документации](https://docs.python.org/3/library/asyncio.html), но для написания ботов для Discord нам будет достаточно этого небольшого введения.

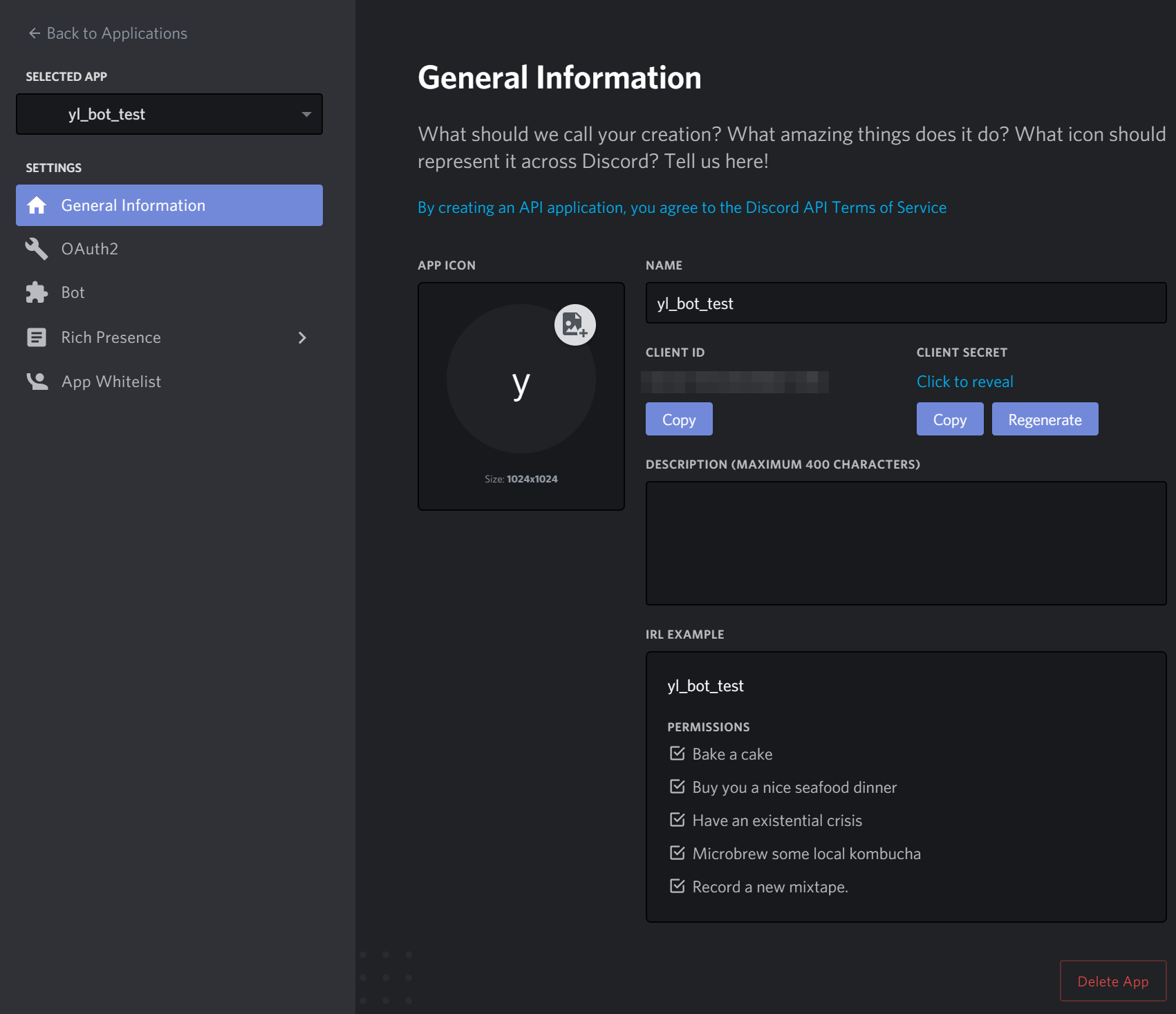
**Подготовка**

Чтобы получить доступ к API Discord, необходимо проделать несколько шагов и получить несколько токенов доступа. Во-первых, необходимо зарегистрироваться на самой платформе и подтвердить почту, которая была указана при регистрации.

После этого надо зайти на [портал для разработчиков](https://discordapp.com/developers/), авторизоваться там и нажать кнопку New application. Discord устроен так, что для создания бота необходимо сначала создать приложение для получения доступа к API, а потом к этому приложению можно будет добавить бота.

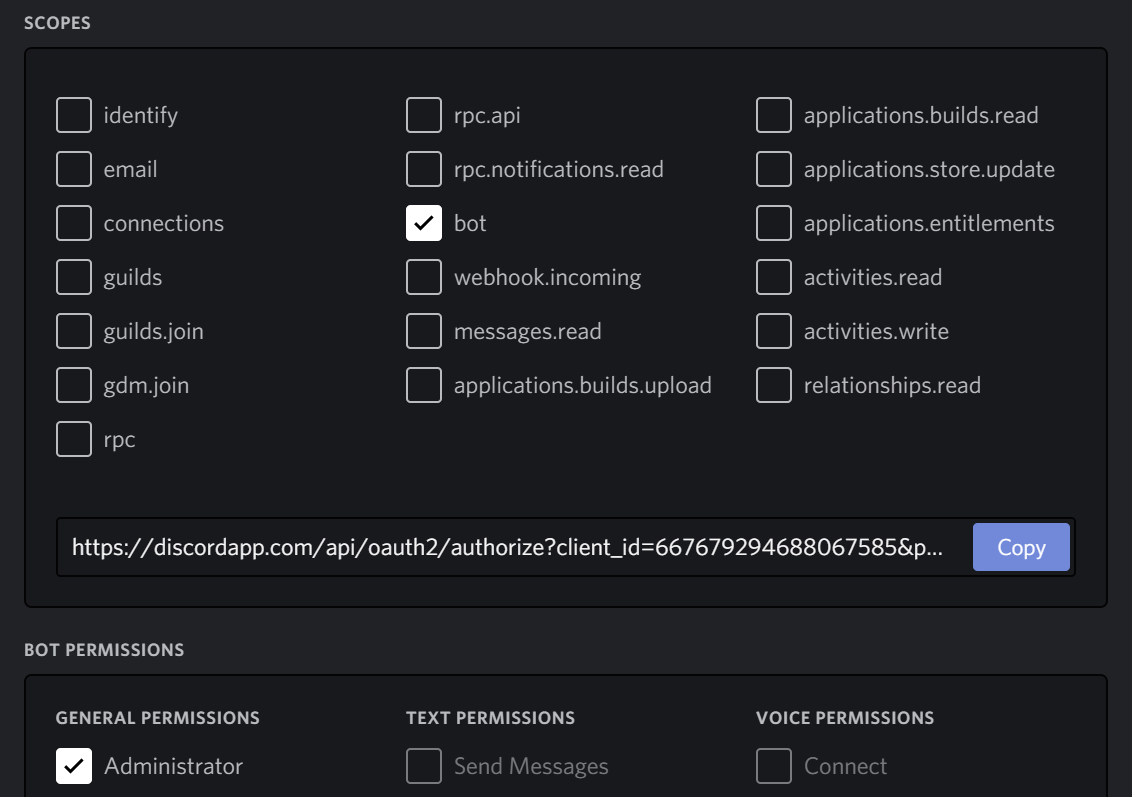


После ввода имени приложения мы перейдем в настройки приложения.



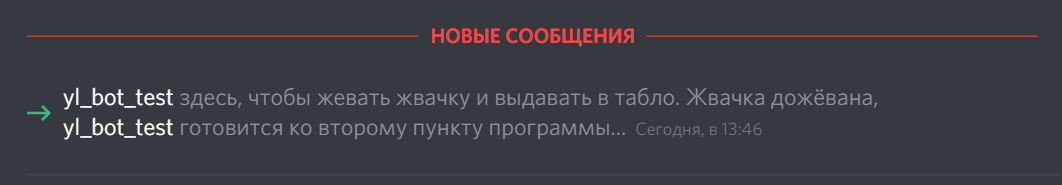
Затем для добавления к своему приложению бота перейдите на вкладку Bot и нажмите кнопку Add Bot. Можно сделать публичного бота, которого смогут добавлять все желающие, либо приватного, которого сможете добавлять только вы.

Остался последний штрих — добавить боту прав. Для этого воспользуемся вкладкой OAuth2. Выберем в Scopes — Bot, а в Bot Permissions — Administrator.



После этого перейдите по ссылке и добавьте бота на сервер. Это может быть личный сервер или любой, который вы создали в приложении Discord.

Бот добавился и написал автоматическое приветственное сообщение:



**Простой бот**

Самый простой бот для Discord выглядит следующим образом:

import discord

TOKEN = "BOT\_TOKEN"

client = discord.Client()

client.run(TOKEN)

Этот бот, конечно, ничего не делает, но можно явно посмотреть на ту структуру, которую предлагает нам модуль. Согласитесь, она очень похожа на flask. Так же, как и во flask, обработчики событий можно создавать с использованием декоратора. Тут это декоратор client.event.

Давайте добавим между созданием клиента и его запуском обработчик события on\_ready, который выполнится после того, как бот будет запущен и подключится к Discord, то есть появится «В сети».

@client.event

async def on\_ready():

print(f'{client.user} подключен к Discord!')

for guild in client.guilds:

print(

f'{client.user} подключились к чату:\n'

f'{guild.name}(id: {guild.id})'

)

Этот обработчик пройдет по всем гильдиям (чатам), в которые добавлен бот, и выведет информацию об этом. Обратите внимание: наша функция асинхронная и определена с использованием ключевого слова async. Собственно, на этом все особенности в данном случае и заканчиваются.

В принципе, можно продолжать писать ботов с использованием декораторов, но библиотека discord.py позволяет использовать и объектно-ориентированный подход.

**Объектно-ориентированный подход**

Давайте перепишем этот простой пример с использованием объектно-ориентированного подхода:

class YLBotClient(discord.Client):

async def on\_ready(self):

print(f'{self.user} has connected to Discord!')

for guild in self.guilds:

print(

f'{self.user} подключились к чату:\n'

f'{guild.name}(id: {guild.id})')

client = YLBotClient()

client.run(TOKEN)

Код получился еще проще и понятнее. Давайте добавим несколько обработчиков событий. Для начала напишем обработчик, который будет приветствовать нового пользователя личным сообщением. Обработчик этого события будет принимать еще один аргумент — объект нового пользователя.

async def on\_member\_join(self, member):

await member.create\_dm()

await member.dm\_channel.send(

f'Привет, {member.name}!'

)

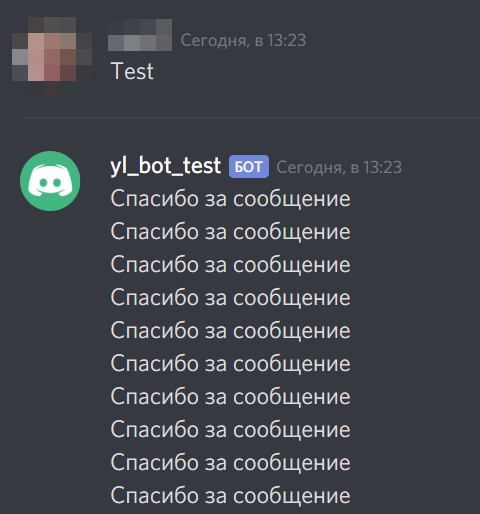
В тот момент, когда к нам приходит новый пользователь, бот создает чат для прямой отправки сообщений этому пользователю и посылает в него сообщение. Обратите внимание: все вызовы внутренних функций и методов объектов модуля discord.py должны происходить с указанием ключевого слова await.

Теперь давайте добавим реакцию бота на какое-либо сообщение. Для этого необходимо в класс добавить метод on\_message, который в качестве параметра принимает объект сообщения.

async def on\_message(self, message):

await message.channel.send("Спасибо за сообщение")

Мы получаем сообщение и отправляем свое в чат, в котором это сообщение было получено. Давайте запустим и посмотрим, что произойдет.



Наш бот начнет отправлять сообщения бесконечно. Это связано с тем, что сообщения от самого бота также вызывают событие on\_message. Чтобы этого не происходило, можно добавить проверку на автора сообщения:

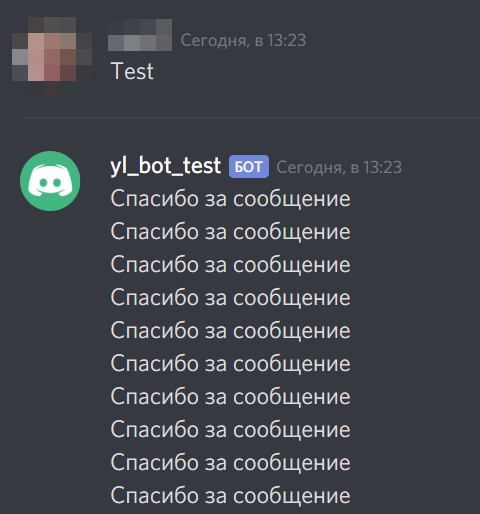
if message.author == self.user:

return

Разумеется внутри обработчика мы можем получить доступ к содержанию сообщения message. Давайте проверим, что пользователь с нами поздоровался:

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

await message.channel.send("И тебе привет")



**Команды**

Помимо простого анализа текста в Discord, как и в Telegram, можно создавать свои команды. При этом немного изменится структура программы: надо создавать уже не объект класса Client, а объект класса Bot, и использовать декоратор bot.command. Давайте сделаем бота, который будет возвращать случайное число от минимального до максимального, указанного пользователем.

Импортируем из модуля discord.ext модуль commands.

Сначала сделаем это с помощью отдельной функции:

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

@bot.command(name='randint')

async def my\_randint(ctx, min\_int, max\_int):

num = random.randint(int(min\_int), int(max\_int))

await ctx.send(num)

bot.run(TOKEN)

При создании бота необходимо указать подстроку command\_prefix, с которой будут начинаться все команды. Параметры команды должны идти в сообщении через пробел после названия команды, но первым аргументом передается **контекст**, из которого мы можем получить чат и другую полезную информацию.

А теперь сделаем имитацию броска шестигранных кубиков, но уже с использованием объектно-ориентированного подхода:

dashes = ['\u2680', '\u2681', '\u2682', '\u2683', '\u2684', '\u2685']

class RandomThings(commands.Cog):

def \_\_init\_\_(self, bot):

self.bot = bot

@commands.command(name='roll\_dice')

async def roll\_dice(self, ctx, count):

res = [random.choice(dashes) for \_ in range(int(count))]

await ctx.send(" ".join(res))

@commands.command(name='randint')

async def my\_randint(self, ctx, min\_int, max\_int):

num = random.randint(int(min\_int), int(max\_int))

await ctx.send(num)

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

bot.add\_cog(RandomThings(bot))

bot.run(TOKEN)

Обратите внимание: в таком случае нам надо наследоваться не от класса Bot, а от класса Cog, определять команды там, а потом добавлять их в бота методом bot.add\_cog.

**Заключение**

Возможности API Discord достаточно широкие, есть даже возможность подключаться к прослушиванию аудио-сообщений от пользователей канала. Простор для экспериментов действительно огромен.

Помимо рассмотренных, есть еще ряд платформ с хорошим API и библиотекам к нему на Python. К части из них доступ получить просто и мгновенно, где-то (например, для доступа к API Twitter) вас попросят написать мини-пояснение на тему «как вы хотите использовать API», где-то — пройти дополнительную идентификацию и подтверждение личности.

Кроме того, рекомендуем посмотреть в сторону параллельного программирования. Такую парадигму поддерживает множество классных библиотек, например, веб-фреймворк [Tornado](https://www.tornadoweb.org/en/stable/index.html).

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты