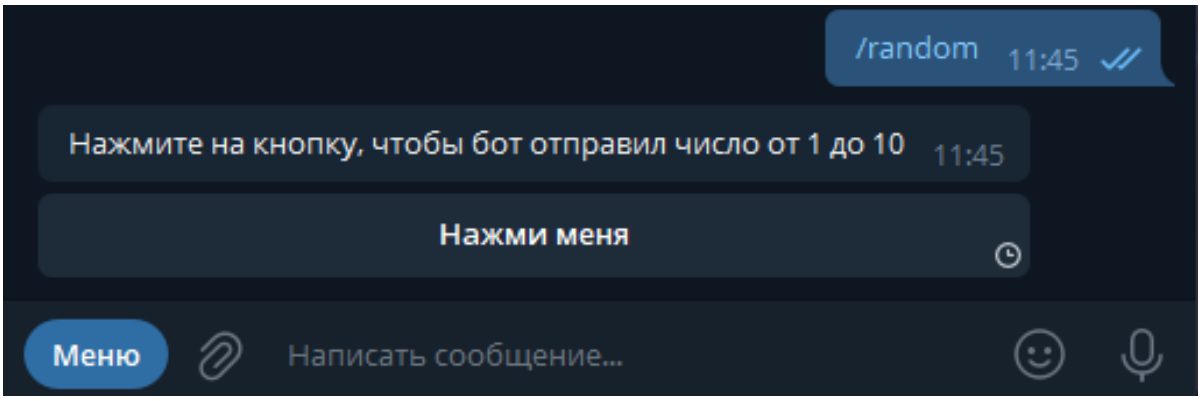
Теоретические материалы к занятию 3

Сегодня мы продолжим знакомство с клавиатурами.

С URL-кнопками больше обсуждать, по сути, нечего, поэтому перейдём к Callback-кнопкам. Это очень мощная штука, которую вы можете встретить практически везде. Кнопки-реакции у постов (лайки), меню у @BotFather и т.д. Суть в чём: у колбэк-кнопок есть специальное значение (data), по которому ваше приложение опознаёт, что нажато и что надо сделать. И выбор правильного data очень важен! Стоит также отметить, что, в отличие от обычных кнопок, нажатие на колбэк-кнопку позволяет сделать практически что угодно, от заказа еды до перезагрузки сервера.

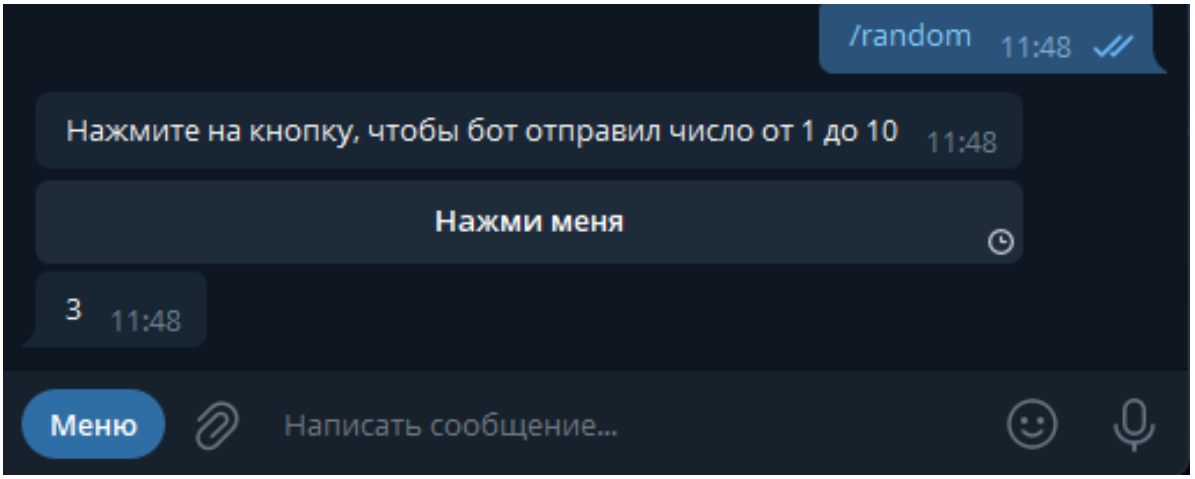
Напишем хэндлер, который по команде /random будет отправлять сообщение с колбэк-кнопкой:

```
@dp.message_handler(commands="random")
async def cmd_random(message: types.Message):
    keyboard = types.InlineKeyboardMarkup()
    keyboard.add(types.InlineKeyboardButton(text="Нажми меня",
callback_data="random_value"))
    await message.answer("Нажмите на кнопку, чтобы бот отправил
число от 1 до 10", reply_markup=keyboard)
```



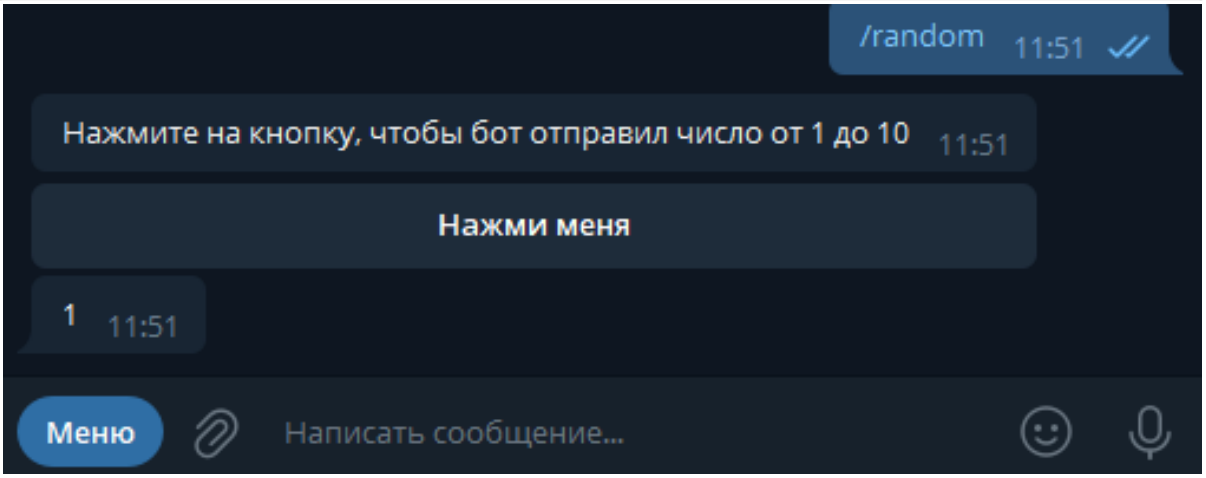
Но как же обработать нажатие? Если для обычных кнопок мы использовали message_handler для обработки входящих сообщений, то для инлайн - callback_query_handler для обработки колбэков. Ориентироваться будем на «значение» кнопки, т.е. на её data:

```
@dp.callback_query_handler(text="random_value")
async def send_random_value(call: types.CallbackQuery):
    await call.message.answer(str(randint(1, 10)))
```

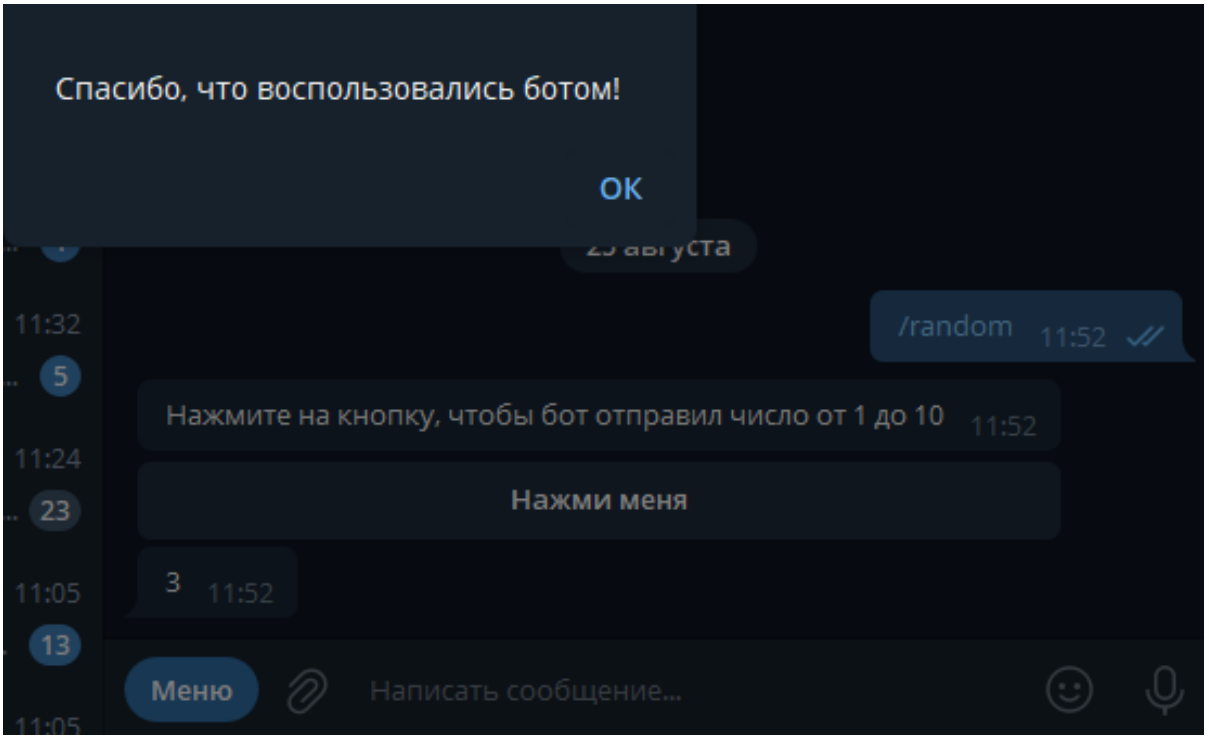


Все работает, но мы видим часы на кнопке. Кто помнит как их убрать и почему они появляются? Оказывается, сервер Telegram ждёт от нас подтверждения о доставке колбэка, иначе в течение 30 секунд будет показывать специальную иконку. Чтобы скрыть часики, нужно вызвать метод answer() у колбэка (или использовать метод API answer_callback_query()). В общем случае, в метод answer() можно ничего не передавать, но можно вызвать специальное окошко (всплывающее сверху или поверх экрана):

```
@dp.callback_query_handler(text="random_value")
async def send_random_value(call: types.CallbackQuery):
    await call.message.answer(str(randint(1, 10)))
    await call.answer()
```




```
@dp.callback_query_handler(text="random_value")
async def send_random_value(call: types.CallbackQuery):
    await call.message.answer(str(randint(1, 10)))
    await call.answer(text="Спасибо, что воспользовались ботом!",
show_alert=True)
```



В функции send_random_value мы вызывали метод answer() не у message, а у call.message. Это связано с тем, что колбэк-хэндлеры работают не с сообщениями (тип Message), а с колбэками (тип CallbackQuery), у которого другие поля, и само сообщение — всего лишь его часть. Учтите также, что message — это сообщение, к которому была прицеплена кнопка (т.е. отправитель такого сообщения — сам бот). Если хотите узнать, кто нажал на кнопку, смотрите поле from

Перейдём к примеру посложнее. Пусть пользователю предлагается сообщение с числом 0, а внизу три кнопки: +1, -1 и Подтвердить. Первыми двумя он может редактировать число, а последняя удаляет всю клавиатуру, фиксируя изменения. Хранить значения будем в памяти в словаре

```
import logging
from random import randint

from aiogram import Bot, Dispatcher, executor, types
from aiogram.dispatcher.filters import Text

from config import BOT_TOKEN

# Объект бота
bot = Bot(token=BOT_TOKEN, parse_mode=types.ParseMode.HTML)
# Диспетчер для бота
dp = Dispatcher(bot)
```

```
# Включаем логирование, чтобы не пропустить важные сообщения
logging.basicConfig(level=logging.INFO)


user_data = {}

def get_keyboard():
   # Генерация клавиатуры.
   buttons = [
       types.InlineKeyboardButton(text="-1",
callback_data="num_decr"),
       types.InlineKeyboardButton(text="+1",
callback_data="num_incr"),
       types.InlineKeyboardButton(text="Подтвердить",
callback_data="num_finish")
   ]
   # Благодаря row_width=2, в первом ряду будет две кнопки, а
оставшаяся одна
   # уйдёт на следующую строку
   keyboard = types.InlineKeyboardMarkup(row_width=2)
   keyboard.add(*buttons)
   return keyboard

async def update_num_text(message: types.Message, new_value:
int):
   # Общая функция для обновления текста с отправкой той же
клавиатуры
   await message.edit_text(f"Укажите число: {new_value}",
reply_markup=get_keyboard())

@dp.message_handler(commands="numbers")
async def cmd_numbers(message: types.Message):
   user_data[message.from_user.id] = 0
   await message.answer("Укажите число: 0",
reply_markup=get_keyboard())

@dp.callback_query_handler(Text(startswith="num_"))
async def callbacks_num(call: types.CallbackQuery):
   # Получаем текущее значение для пользователя, либо считаем его
равным 0
   user_value = user_data.get(call.from_user.id, 0)
   # Парсим строку и извлекаем действие, например `num_incr` ->
`incr`
   action = call.data.split("_")[1]
   if action == "incr":
       user_data[call.from_user.id] = user_value+1
       await update_num_text(call.message, user_value+1)
   elif action == "decr":
       user_data[call.from_user.id] = user_value-1
       await update_num_text(call.message, user_value-1)
   elif action == "finish":
       # Если бы мы не меняли сообщение, то можно было бы просто
удалить клавиатуру
       # вызовом await call.message.delete_reply_markup().
```

```
        # Но т.к. мы редактируем сообщение и не отправляем новую
клавиатуру,
        # то она будет удалена и так.
        await call.message.edit_text(f"Итого: {user_value}")
    # Не забываем отчитаться о получении колбэка
    await call.answer()



if __name__ == "__main__":
    # Запуск бота
    executor.start_polling(dp, skip_updates=True)
```

Учитель: В aiogram существует т.н. фабрика колбэков. Вы создаёте объект CallbackData, указываете ему префикс и произвольное количество доп. аргументов, которые в дальнейшем указываете при создании колбэка для кнопки.
Например, рассмотрим следующий объект

```
cb= CallbackData("post", "id", "action")
```

Тогда при создании кнопки вам надо указать её параметры так:

```
button =  types.InlineKeyboardButton(
    text="Лайкнуть",
    callback_data=cb.new(id=5, action="like")
)
```

В примере выше в кнопку запишется callback_data, равный post:5:like, а хэндлер на префикс post будет выглядеть так:

```
@dp.callback_query_handler(cb.filter())
async def callbacks(call: types.CallbackQuery, callback_data:
dict):
    post_id = callback_data["id"]
    action = callback_data["action"]
```

В предыдущем примере с числами мы грамотно выбрали callback_data, поэтому смогли легко записать все обработчики в один хэндлер. Но можно логически разнести обработку инкремента и декремента от обработки нажатия на кнопку "Подтвердить". Для этого в фильтре можно указать желаемые значения какого-либо параметра. Давайте перепишем наш пример с использованием фабрики:

```
callback_numbers = CallbackData("fabnum", "action")
```

```
def get_keyboard_fab():
    buttons = [
        types.InlineKeyboardButton(text="-1",
callback_data=callback_numbers.new(action="decr")),
        types.InlineKeyboardButton(text="+1",
callback_data=callback_numbers.new(action="incr")),
        types.InlineKeyboardButton(text="Подтвердить",
callback_data=callback_numbers.new(action="finish"))
    ]
    keyboard = types.InlineKeyboardMarkup(row_width=2)
    keyboard.add(*buttons)
    return keyboard


async def update_num_text_fab(message: types.Message, new_value:
int):
    with suppress(MessageNotModified):
        await message.edit_text(f"Укажите число: {new_value}",
reply_markup=get_keyboard_fab())


@dp.message_handler(commands="numbers_fab")
async def cmd_numbers(message: types.Message):
    user_data[message.from_user.id] = 0
    await message.answer("Укажите число: 0",
reply_markup=get_keyboard_fab())


@dp.callback_query_handler(callback_numbers.filter(action=["incr"
, "decr"]))
async def callbacks_num_change_fab(call: types.CallbackQuery,
callback_data: dict):
    user_value = user_data.get(call.from_user.id, 0)
    action = callback_data["action"]
    if action == "incr":
        user_data[call.from_user.id] = user_value + 1
        await update_num_text_fab(call.message, user_value + 1)
    elif action == "decr":
        user_data[call.from_user.id] = user_value - 1
        await update_num_text_fab(call.message, user_value - 1)
    await call.answer()


@dp.callback_query_handler(callback_numbers.filter(action=["finis
h"]))
async def callbacks_num_finish_fab(call: types.CallbackQuery):
    user_value = user_data.get(call.from_user.id, 0)
    await call.message.edit_text(f"Итого: {user_value}")
    await call.answer()
```

С клавиатурами с вами мы разобрались. Теперь давайте начнем писать какого нибудь бота заказа еды, как пример. Понятное дело он будет максимально упрощен, но мы с каждой новой темой будем его дорабатывать, чтобы в конце у вас получился полноценный проект Возьмем самое простое меню для нашего бота: пицца и сопутствующие закуски.

Для начала добавим в нашу заготовку команды по умолчанию.

```
import asyncio
import logging

from aiogram import Bot, Dispatcher, executor, types
from aiogram.dispatcher.filters import CommandStart
from aiogram.types import BotCommandScopeDefault, BotCommand

from config import BOT_TOKEN
from filters import AdminFilter

bot = Bot(token=BOT_TOKEN, parse_mode=types.ParseMode.HTML)
dp = Dispatcher(bot)
logging.basicConfig(level=logging.INFO)

from aiogram import types


async def set_default_commands(bot: Bot):
   return await bot.set_my_commands(
       commands=[
           BotCommand('menu', 'Вывести меню'),
           BotCommand('help', 'Помощь'),
           BotCommand('support', 'Поддержка'),
       ],
       scope=BotCommandScopeDefault(),

   )


@dp.message_handler(AdminFilter(), CommandStart())
async def admin_start(message: types.Message):
   await message.reply('Команды установлены')
   await set_default_commands(message.bot)


if __name__ == "__main__":
   # Запуск бота
   executor.start_polling(dp, skip_updates=True)
```

В файле filters.py мы пропишем свой фильтр

```
class AdminFilter(BoundFilter):
    async def check(self, message: types.Message):
        member = str(message.from_user.id)
        return member in ['1865314469', ]
```

Данный фильтр проверяет находимся ли мы в списках с id администраторов и возвращает True или False соответственно.

Добавим клавиатуру на главное меню

```
def get_menu():
    menu_kb = InlineKeyboardMarkup(row_width=2)
    pizza_button = InlineKeyboardButton(text='Пицца 🍕',
callback_data='pizza_cat')
    snacks_button = InlineKeyboardButton(text='Закуски 🍟',
callback_data='snacks_cat')
    menu_kb.insert(pizza_button)
    menu_kb.insert(snacks_button)
    return menu_kb
```

и создадим обработчик

```
@dp.message_handler(commands='menu')
async def menu_bot(message: types.Message):
    await message.answer('Добро пожаловать в доставку еды Crosta',
reply_markup=get_menu())
```

Отлично начало нашего бота положено. Далее мы будем дополнять нашего бота на новых занятиях.