



# Мини-курс 3.0

# «Чатбот со Стоуном»

Лекция 3. **АРХИТЕКТУРА И КЛАВИАТУРА**

Курс лекций от Stone

2023



## Чем сегодня займемся?

- Создадим новый проект и познакомимся поближе с диспатчером, экзекутором (нет, БДСМ тут не при чем) и хэндлерами.
- И так построим свою архитектуру! С Блэкджеком и женщинами с низкой социальной ответственностью!!!
- Узнаем, что такое пакет и зачем там `__init__`
- Попробуем добавить простую клавиатуру

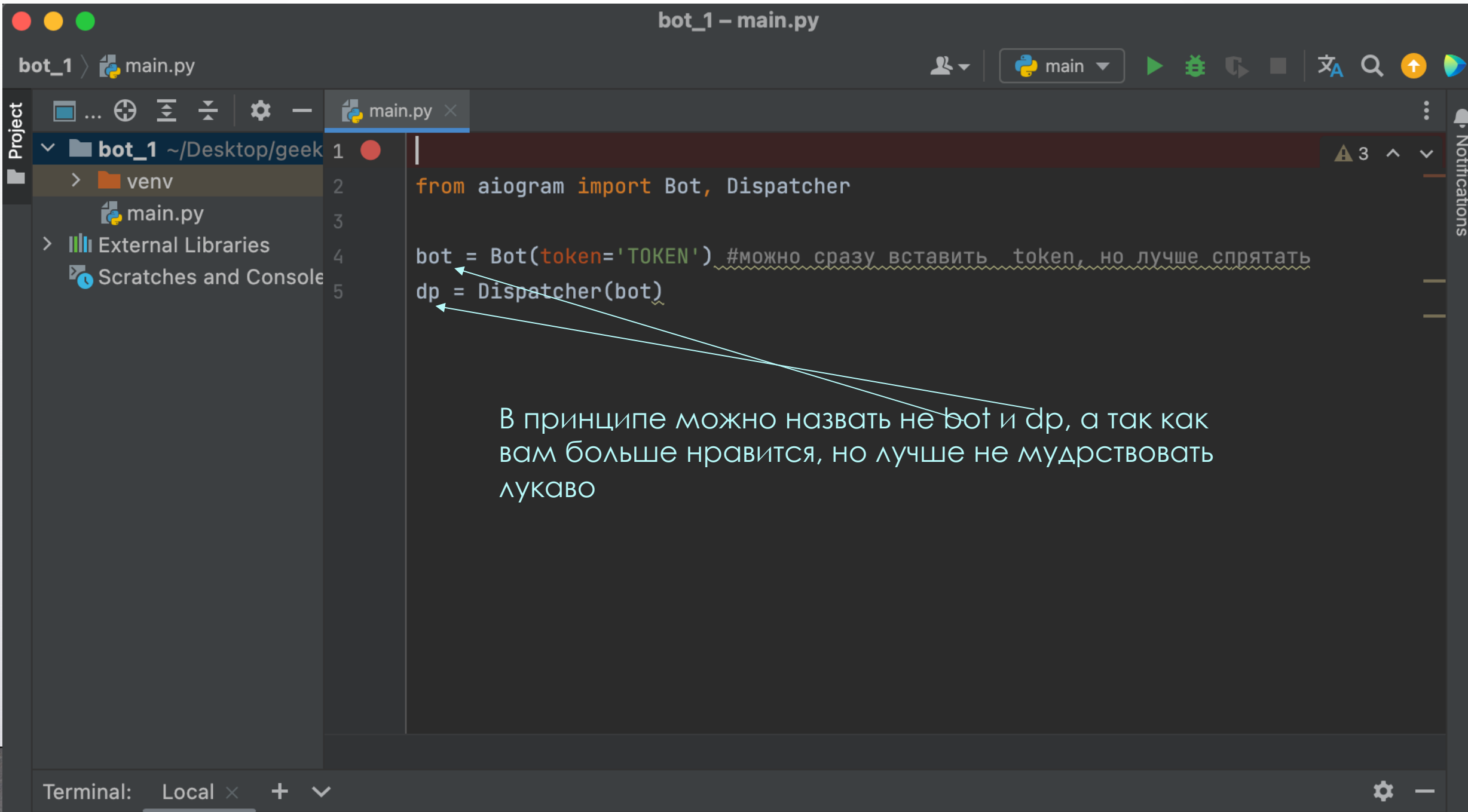
# Немного о последовательности создания бота

- Для начала создадим проект, придумаем ему название. И если вы не хотите работать в PyCharm, тогда позаботьтесь об окружении.
- Установите Aiogram любым удобным для вас способом.

Например: `pip install aiogram`

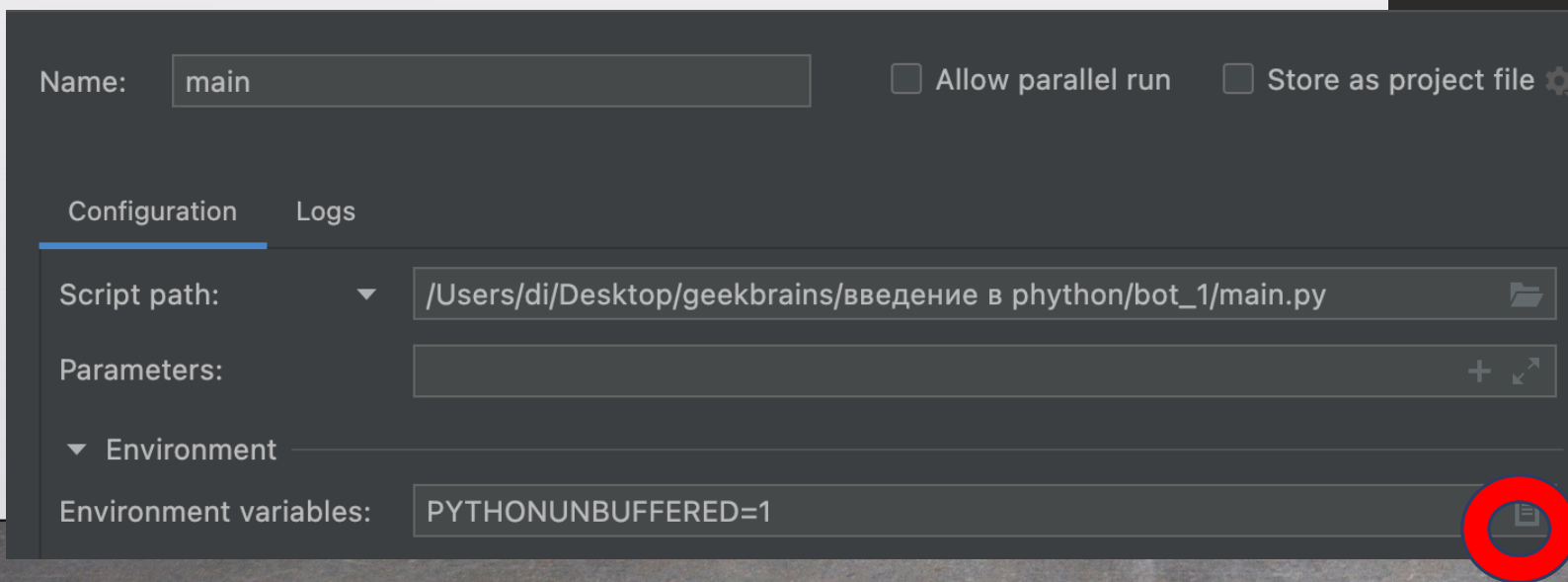
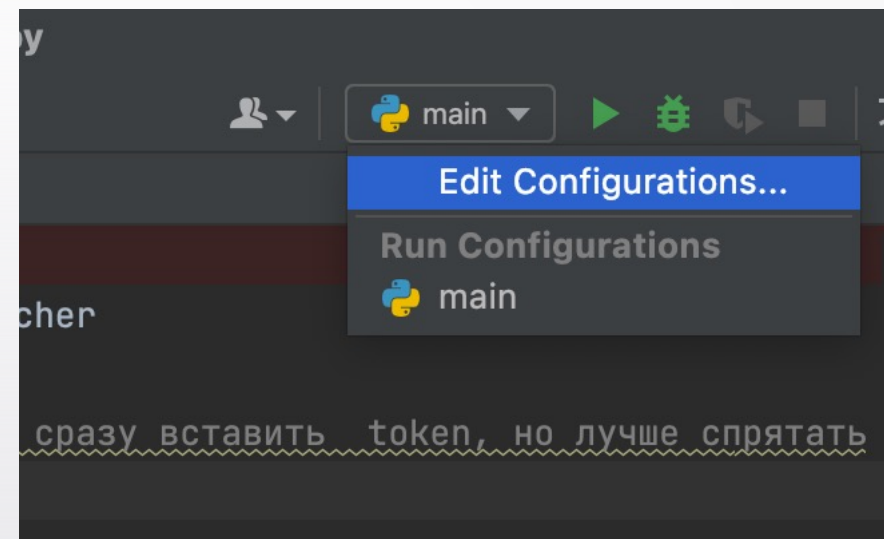
- Теперь нам нужно создать две сущности: собственно бота и диспатчера.
- Через бота мы привяжемся к нашему телеграм-боту и тут нам опять понадобится TOKEN
- Бота мы назовем bot
- А диспатчера dp
- Зачем же нужен диспатчер?





# Давайте спрячем Token

- Для начала давайте спрячем Token – есть несколько способов, мы спрячем в конфигурации
- Нажимаем на стрелочку рядом с main – будем менять конфигурации



Теперь нам нужна вот эта кнопка



## Environment Variables

User environment variables:

+ - [icon] [icon]	
Name	Value
PYTHONUNBUFFERED	1
TOKEN	23435678990000000000

Создаем свою переменную!

Называем ее TOKEN

А вот сюда записываем значение,  
То самое, что получили от BotFather

☒ Include system environment variables:

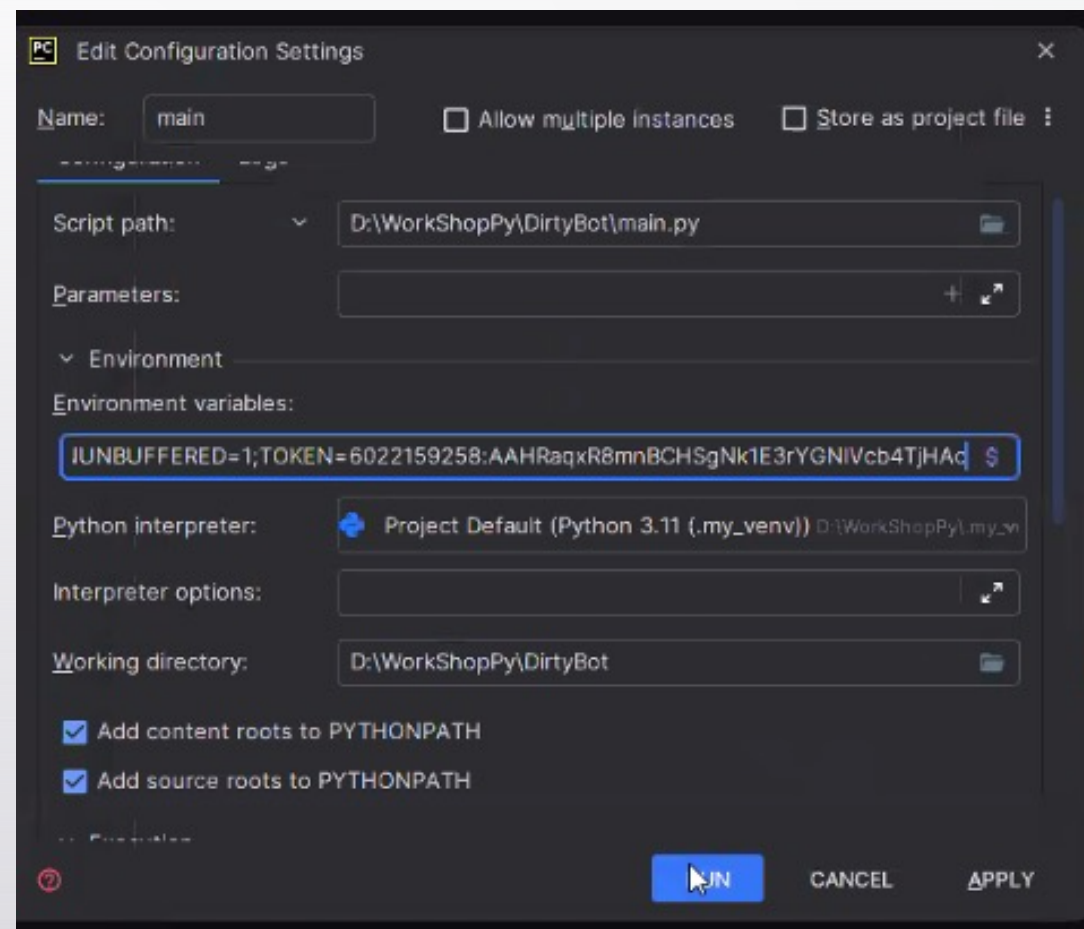
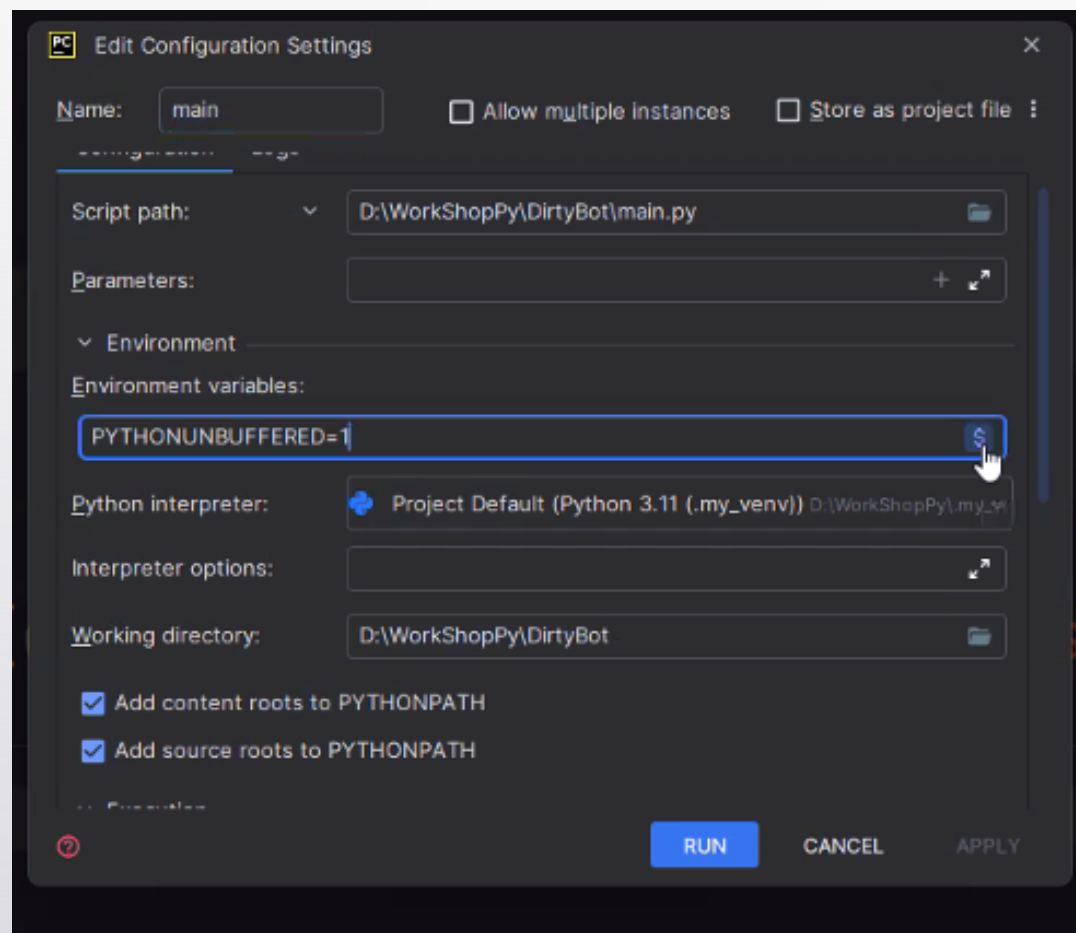
[icon] [icon]	
Name	Value
__CF_USER_TEXT_ENCODING	0x1F6:0x7:0x31
__CFBundleIdentifier	com.jetbrains.pycharm.ce
COMMAND_MODE	unix2003
HOME	/Users/di
LC_CTYPE	ru_BY.UTF-8
LOGNAME	di
OLDPWD	/
PATH	/usr/local/bin:/usr/bin:/bin:/usr/sbin:/

И не забудем нажать ок!  
Все, теперь осталось  
немного подправить код

Cancel

OK

Ну или вот так, если хотите, как на лекции

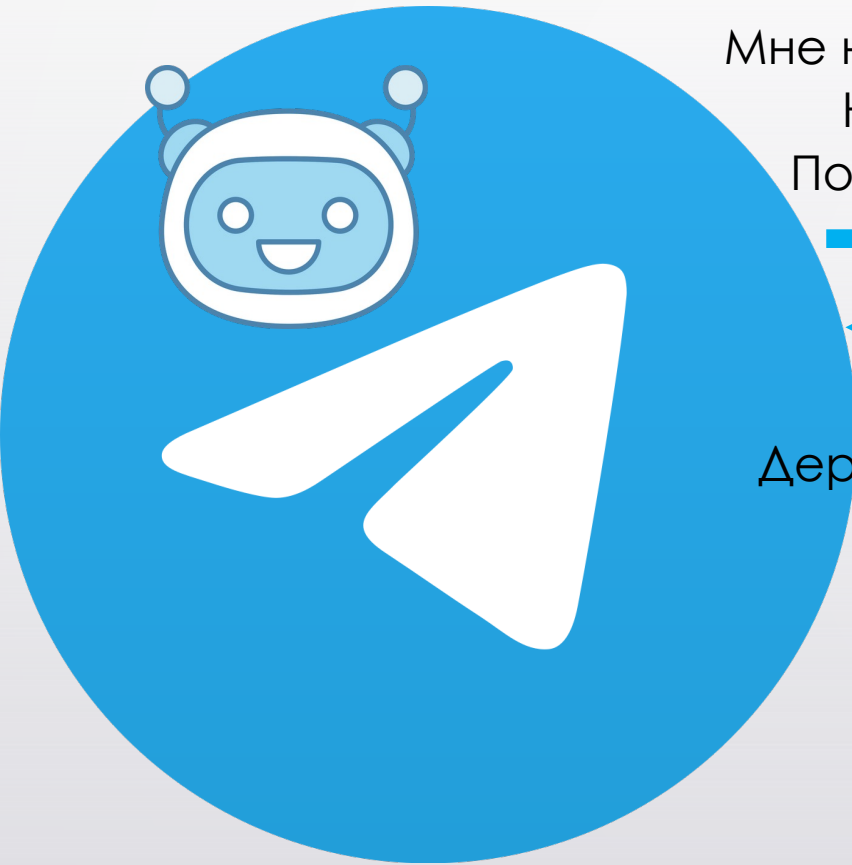


1 Если мы прячем наш токен, то нам потребуется его заполучить, а для этого нам понадобится os  
2 `import os`  
3 `from aiogram import Bot, Dispatcher`  
4  
5 `bot = Bot(token=os.getenv('TOKEN'))` # вот теперь мы все спрятали надежно!  
6 `dp = Dispatcher(bot)`

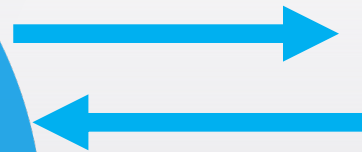
Все, теперь наш токен никто не украдет, идем писать бота!



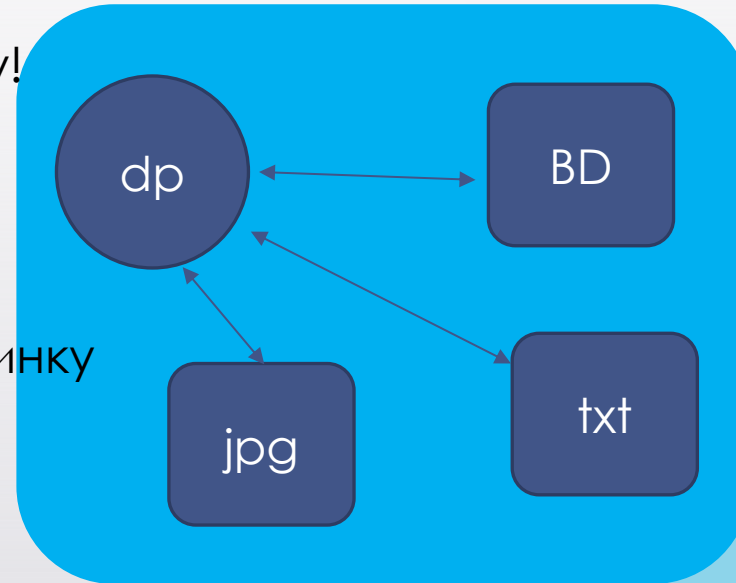
Мало нам бота, еще и диспатчер! Так что  
это за чудо-юдо многорукое?



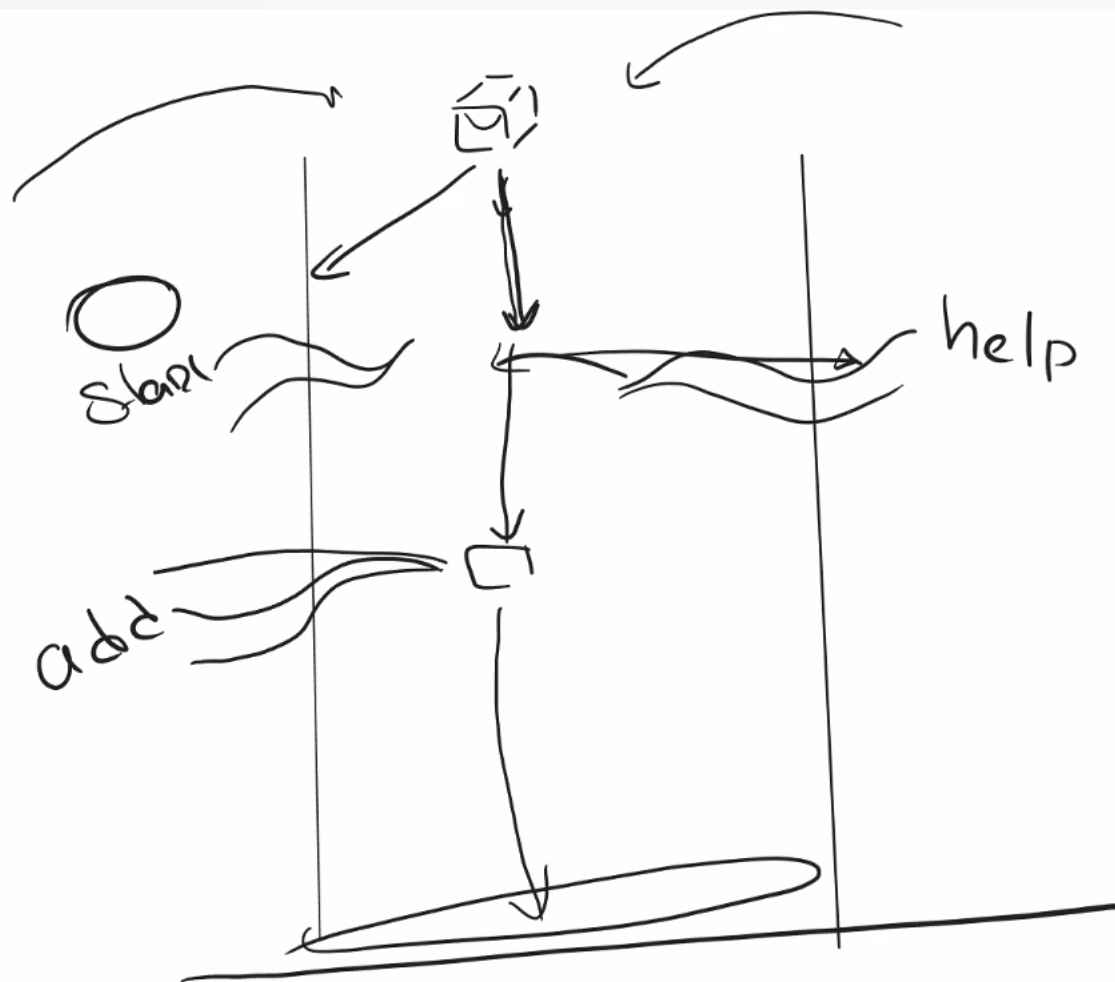
Мне нужна информация!  
Напиши мне!  
Покажи картинку!




Держи свою картинку  
Окей!  
Вот она!



## А это гениальный рисунок от Стоуна!



- Если помните, бот постоянно ходит за обновлениями от TG
- А хэндлеры – это щупальца, которые реагируют на определенные раздражители (команды, слова и т.д.)
- Очередность очень важна!
- Поэтому хэндлер, которого Стоун зовет жадным ублюдком (он ловит вообще все, надо располагать в самом низу)



## Как же прописывать команды для Диспатчера?

- Не так сложно, как можно подумать!
- Используем декоратор `@dp.message_handler(commands=['тут_команда', 'и_даже_не_одна'])`
- Кроме команды можно задавать определенные слова или фильтр, или даже ничего – тогда Хэндлер будет реагировать на любую активность.
- Все! Теперь мы отрастили Диспетчеру новую 'руку'. Кстати, команды можно и не писать, тогда любое сообщение юзера будет улавливаться как инициатор для начала выполнения функции.
- После декоратора пишем функцию, которая будет выполняться на эту команду. Не забываем про `async` (асинхронность) и `await` (подожди, программа, а вдруг к тебе обращаются). Про асинхронность немного позже

```
main.py x
1
2 import os
3 from aiogram import Bot, Dispatcher
4 from aiogram.types import Message #Поскольку нам предстоит иметь дело с message импортируем тип
5
6 bot = Bot(token=os.getenv('TOKEN'))
7 dp = Dispatcher(bot)
8
9 @dp.message_handler(commands=['start']) #декоратор, команда start
10 async def start_bot(message: Message): #принимаем сообщение
11     print(message) #печатаем его в консоли
```

В общем и целом, функций может быть множество, а логика их написания стандартная – ветвление, циклы и прочие радости Python



# А кто же такой экзекютер?

- Не смотря на романтическое название, экзекютер – это скорее посыльный. Он собирает с сервера телеграмма запросы в соответствии с инструкцией, которую вы ему дадите.
- Чтобы его запустить для начала импортируем его
- `from aiogram.utils import executor`
- Затем запустим, указав диспетчера
- `executor.start_polling(dispatcher=dp)`
- Можно прописать, чтобы он «выкидывал» накопленные сообщения `executor.start_polling(dispatcher=dp, skip_updates=True)`
- И можно еще добавить, что ему делать при запуске бота

Сейчас  
посмотрим,  
что тут у нас!





```
3 from aiogram import Bot, Dispatcher
4 from aiogram.types import Message
5 from aiogram.utils import executor #Экзекутора надо импортировать
6
7 bot = Bot(token=os.getenv('TOKEN'))
8 dp = Dispatcher(bot)
9
10 async def on_startup(_): #Пропишем команду, что ему делать при запуске бота
11     print('Бот запущен') #Например написать об этом в консоль
12     await dp.bot.send_message(chat_id=40920000, text='Хозяин! Бот запущен') # id того, кому отправляем
13
14 @dp.message_handler(commands=['start']) #декоратор, команда start
15 async def start_bot(message: Message): #принимаем сообщение
16     print(message) #печатаем его в консоли
17
18 executor.start_polling(dispatcher=dp, skip_updates=True, on_startup=on_startup)
```



# Обещанная асинхронность

- Как вы уже знаете, Python - язык интерпретируемый, а не компилируемый. А потому, кто первый – тот и первый: команды кода считываются ПОСЛЕДОВАТЕЛЬНО.
- Однако, при работе чатботов часто требуется «многозадачность», особенно с учетом того, что бот должен быть всегда готов!
- И что же делать? Вот тут-то и нужна асинхронность!
- Асинхронность — это **возможность выполнения программой задач и процессов без ожидания их завершения**. То есть если предыдущий процесс все еще находится на этапе выполнения, асинхронная программа может легко перейти к обработке следующих задач.
- Давайте приведем аналогию!

# Асинхронность и пельмени



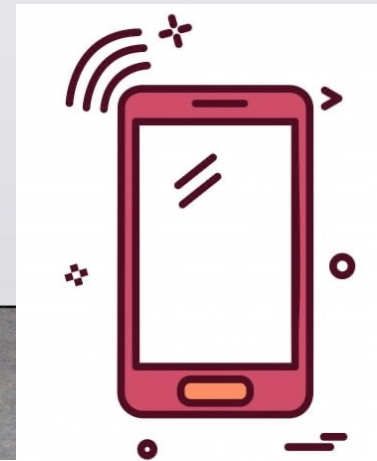
Кофеек  
готов?



Проверю  
пельмени!



Нет ли  
звонка?



Итак, чтобы заставить функцию крутиться асинхронно нужно выполнить два условия

- Перед определением функции написать `async`
- Внутри функции `await`

```
@dp.message_handler(text='мама')
async def help_command(message: Message):
    await message.answer(f'Ты смотри маму он вспомнил')
```





## И пару слов об архитектуре

- Пока наш бот совсем простой и в принципе, можно все уместить в один файл. Но лучше так не делать! Мы же хотим, чтобы наш бот рос большим, а для этого хорошо бы чтобы у него было удобное строение и навигация.
- Итак, какие же файлы и папки нам пригодятся?
- `main.py` содержит необходимый минимум для запуска бота и обязательно ссылку на хэндлеры
- `create_bot.py` или `loader.py` содержит информацию о боте и диспатчере, и на этот файл будут ссылаться все работающие с ботом файлы
- Целая папка с хэндлерами, где содержаться все файлы с функциями, а также `__init__.py` который поможет в управлении пакетом
- Ну а после создадим папку под клавиатуры



# Для чего же нам нужны пакеты?

- Пакеты нам очень нужны! Мы в них все запакуем, ведь это так удобно, что нам не придется импортировать огромное количество файлов!
- В файле `__init__` будет достаточно прописать все файлы (вот тут для хэндлеров важна очередность!!) и добавить в список `__all__`



```
main.py  __init__.py  help.py  loader.py  start.py
1  from .start import dp
2  from .help import dp
3
4  __all__ = ['dp']
```

Вот так выглядит файл `__init__.py` в папке хэндлеры

```
from aiogram import Bot, Dispatcher
from aiogram.utils import executor
from Handlers import dp

I

async def on_start(_):
    print('Бот запущен!')

executor.start_polling(dp, skip_updates=True, on_startup=on_start)
```

А вот так теперь выглядит `main` – никаких лишних команд, главное – импортируйте свою папку с хэндлерами

Простое – это красивое!



## Еще пару слов об архитектуре

- По мере развития бота, архитектура будет дополняться новыми файлами и папками. Главное, чтобы и вы, и бот знали, где что лежит
- Вот ссылка на репозиторий, где можно все это подробно посмотреть <https://github.com/STONE17th/DP-Bot.git>
- А теперь самое время заняться кнопками



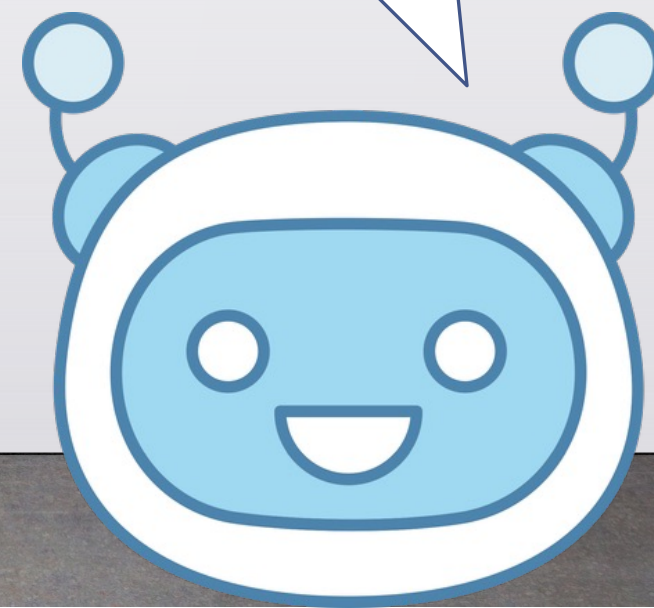
# Виды клавиатур

Клавиатуры по сути это набор кнопок, нажимая на которые мы отправляем боту определенную инструкцию.

При помощи Aiogram можно создавать два типа клавиатур:

- 1) Стандартная – по сути шаблон для отправки команд, на которые заточены наши хэндлеры, фактически она просто заменяет ручной ввод команды.
- 2) Inline- клавиатура – это уже настоящая клавиатура, нажатие на которую дает боту куда больше информации и возможности. Звучит сложно, но разберем на примерах.

Нажми на  
кнопку –  
получишь  
результат!

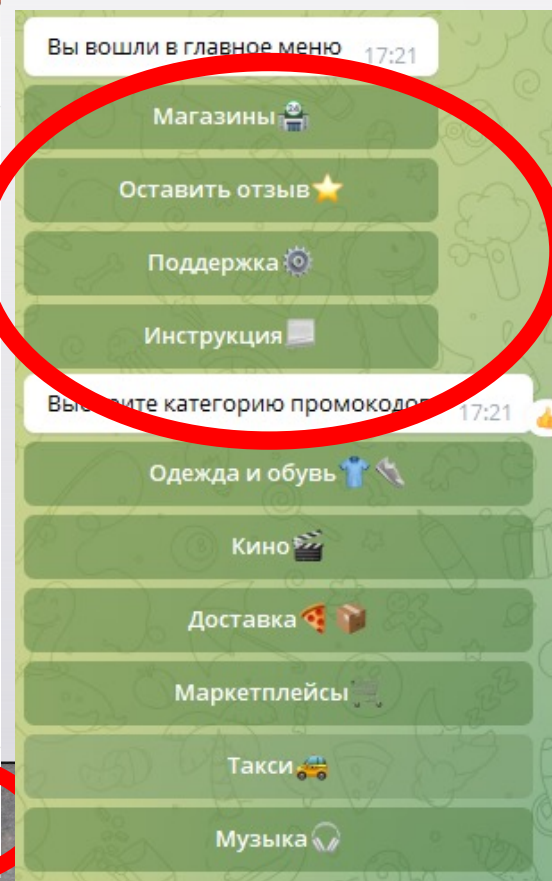
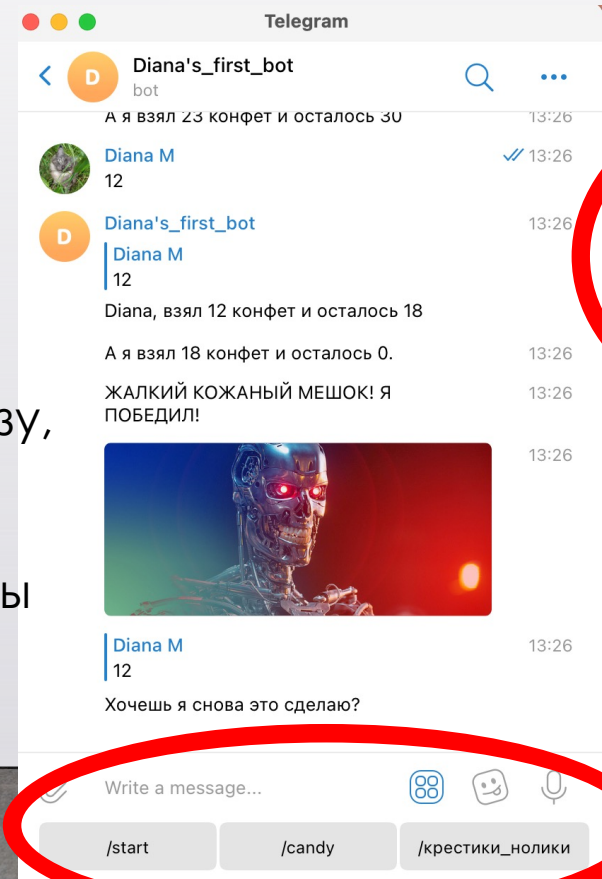


# Как отличить, какую клавиатуру мне предлагает бот?

- Нет ничего проще!

Вот это стандартная клавиатура

- Она расположена внизу, под строкой ввода
- Клавиши называются строго так, как команды для соответствующих хэндлеров



А вот это Inline клавиатура

- Она располагается под сообщением.
- Клавиши могут называться как угодно, содержать эмоджи, менять название после нажатия и вообще их функционал гораздо шире



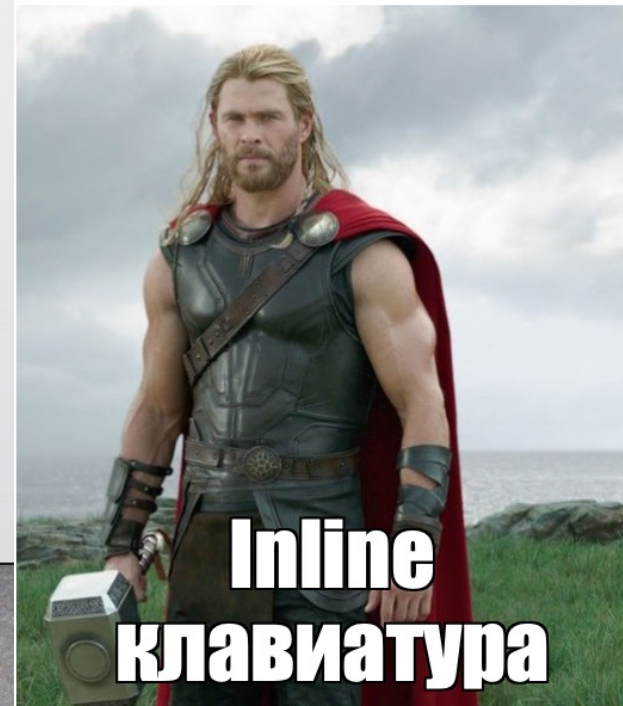
# Пойдем от простого к сложному

Итак, давайте попробуем создать простую клавиатуру.

1. Помним, что у нашего бота есть архитектура, поэтому начнем мы с того, что создадим папку(а точнее пакет), которую так и назовем - Keyboard

2. А теперь, создадим еще один пакет— standart (Зачем так много папок? А у нас будет много клавиатур, которые удобно держать в одном месте)

3. А вот теперь создадим файл с расширением ru, который назовем в честь нашей будущей клавиатуры, например main\_menu





## Ну, файлы делать мы умеем. Как же создать клавиатуру

- И тут мы снова обратимся к Aiogram и попросим у него соответствующие инструменты.
- `from aiogram.types import KeyboardButton, ReplyKeyboardMarkup, ReplyKeyboardRemove`
- `ReplyKeyboardMarkup` – для создания клавиатуры
- `KeyboardButton` – для создания кнопок
- `ReplyKeyboardRemove` – для удаления клавиатуры

## Создаем клавиатуру

- Для начала создадим саму клавиатуру, то поле, в котором будем размещать кнопки, и тот объект, к которому мы будем обращаться при необходимости
- `kb_menu = ReplyKeyboardMarkup()`
- Название, как шляпка у старой дамы, должно отображать суть, гармонировать с содержимым и быть не слишком громоздким.





## А теперь создаем кнопки...

- По сути мы пока создали пустую клавиатуру, чистый лист, так сказать. И теперь нам нужны кнопки, на которые будет нажимать пользователь.
- `btn_start = KeyboardButton(text='/start')`  
`btn_print = KeyboardButton(text='/print')`  
`btn_frak = KeyboardButton(text='/freak')`  
`btn_exit = KeyboardButton(text='/exit')`
- Удобно, когда название отражает команду, в скобках мы собственно указываем команду именно так, как она и должна писаться – со слэшем .
- И именно это и будет написано на кнопке!

////////////////////////////////////

# Думаете готово? Ничего подобного! Кнопки теперь надо разместить в клавиатуре!

- Давайте нарисуем схему!

Это клавиатура и пока она сама по себе

А это кнопки, и пока они сами по себе

/start

/freak

/print

/exit

А теперь мы поместили кнопки на клавиатуру



## dirty\_python – main\_menu.py

dirty\_python &gt; keyboard &gt; standart &gt; main\_menu.py



main ▾

Project  
Pull Requests

- main.py ×
- main\_menu.py ×
- print\_message.py
- start.py
- take\_it\_all.py
- keyboard
  - standart
    - \_\_init\_\_.py
    - main\_menu.py
    - \_\_init\_\_.py
- venv
  - create\_bot.py
  - handlers.py
  - main.py
- External Libraries
- dirty\_python

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12

```
from aiogram.types import KeyboardButton, ReplyKeyboardMarkup, ReplyKeyboardRemo  
  
kb_menu = ReplyKeyboardMarkup(resize_keyboard=True, one_time_keyboard=True)  
  
btn_start = KeyboardButton(text='/start')  
btn_print = KeyboardButton(text='/vacation')  
btn_location = KeyboardButton(text='/location', request_location=True)  
btn_contact = KeyboardButton(text='/contact', request_contact=True)  
  
kb_menu.row(btn_start, btn_print)  
kb_menu.add(btn_location)  
kb_menu.add(btn_contact)
```

Notifications

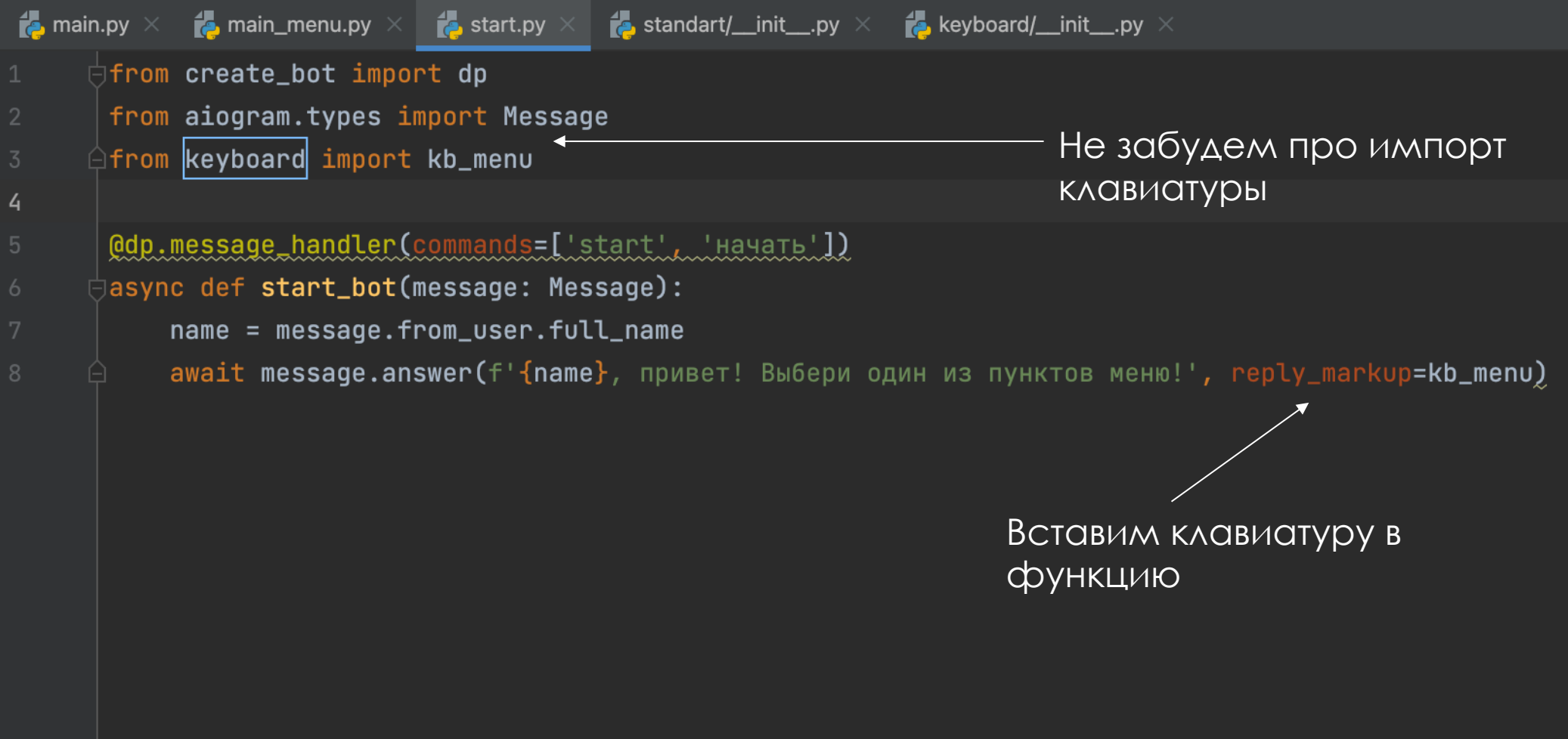
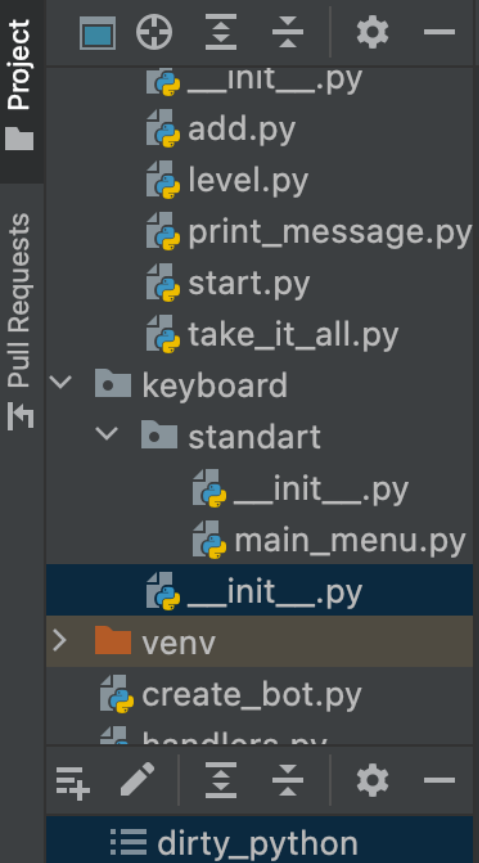
## И все? Ну-ну, не надо торопиться, клавиатуру еще надо подключить

- Для начала, в `__init__.py` (папки `standart`) пропишем следующий код, чтобы подключить клавиатуру:

```
from .main_menu import kb_menu
```

```
__all__ = ['kb_menu']
```

- И тоже самое добавим в файл `__init__.py` только уже для папки `keybord`
- Теперь выберем ту функцию, вызов которой будет сопровождаться вызовом клавиатуры. Если это `main_menu` логично присоединить его к стартовому хэндлеру

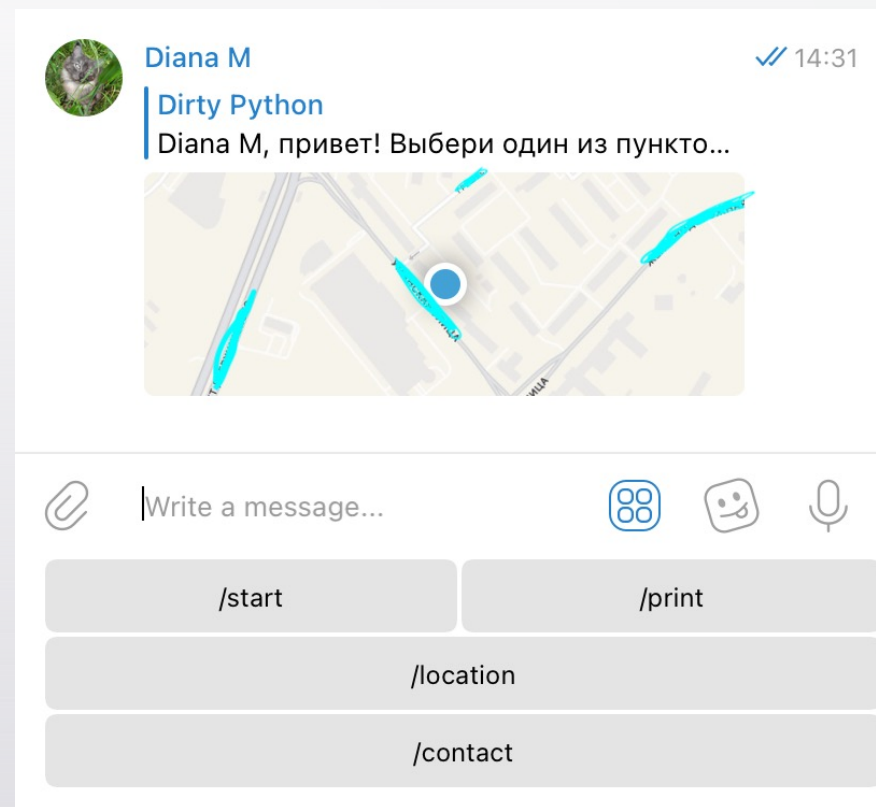


Не забудем про импорт клавиатуры

Вставим клавиатуру в функцию

# Специальные кнопки location и contact

- У стандартной клавиатуры есть две специальные кнопки, которые позволяют получить контакт пользователя и его локацию (доступно в телефонной версии)
- Для этого нужно указать параметр
- `btn_location = KeyboardButton(text='/location', request_location=True)`  
`btn_contact = KeyboardButton(text='/contact', request_contact=True)`
- Как обрабатывать эти данные и что с ними делать – тема отдельного разговора (приходите на семинар))





## И еще немного про стандартные клавиатуры

- Создавайте их столько, сколько вам нужно (не забывайте добавлять их `__init__.py` и импортировать в соответствующие файлы!)
- Одну и ту же клавиатуру вы можете подключать к разным функциям
- Если вам лень прописывать каждую кнопку – подключите циклы

```
in.py × main_menu.py × start.py × list_group.py × standart/__init__.py × keyboard/__init__.py
from aiogram.types import KeyboardButton, ReplyKeyboardMarkup, ReplyKeyboardRemove

kb_print = ReplyKeyboardMarkup(resize_keyboard=True, one_time_keyboard=True)

btn_print = []
city = ['Анапа', 'Геленджик', 'Сочи', 'Новороссийск']

for town in city:
    btn_print.append(KeyboardButton(text=town))

kb_print.row(*btn_print)
```

## И что дальше?

- Для начала попробуйте сделать своего бота с клавиатурой (не надо ничего мудрить – наша цель научиться создавать клавиатуры, а не писать супер-крутого бота)
- А теперь попробуйте менять клавиатуры в зависимости от того, что было нажато раньше
- Попробуйте разнообразить внешний вид своих клавиатур. Например создайте верхний ряд с одной кнопкой, а нижний с тремя. Или наоборот
- Посмотрите, как отрабатывают кнопки `request_location` и `request_contact` – они могут пригодиться, особенно если планируется работа с клиентами
- Если вам этого мало – напишите кликер: две кнопочки и функции для подсчета кликов и обнуления.

