QUIZ

1) Напишите две корутины, каждая из которых работает со списком (аргумент). Первая корутина должна увеличивать каждый элемент списка на 1 после чего вызывает функцию sleep(1), вторая – уменьшать. Создайте список с 5-ю элементами, event_loop и запустите корутины для созданного списка.
2) Создайте репозиторий в github на двоих. Дайте права доступа, создайте ветку bot, залейте туда свои решения первого задания.

Получаем токен для бота

Получим токен у @BotFather

Для этого напишем этому боту и следуя его инструкциям получим токен.

Создадим переменную token в которой будем хранить токен от @BotFather

```
TOKEN = '1178765389:AAFhPyETaraYuFQsXDh5KwWcd6RlU1nePOM'
```

Пишем первого echo bot

```python
from aiogram import Bot, Dispatcher, executor, types

API_TOKEN = '1178765389:AAFhPyETaraYuFQsXDh5KwWcd6RlU1nePOM'

bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot)


@dp.message_handler()
async def echo(message: types.Message):
    await message.answer(message.text)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

Добавим информцио о боте в логи

```python
import logging
from aiogram import Bot, Dispatcher, executor, types

TOKEN = '1178765389:AAFhPyETaraYuFQsXDh5KwWcd6RlU1nePOM'

logging.basicConfig(level=logging.INFO)

bot = Bot(token=TOKEN)
dp = Dispatcher(bot)


@dp.message_handler()
async def echo(message: types.Message):
    await message.answer(message.text)


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

Можем добавлять команды для бота

```python
@dp.message_handler(commands=["start"])
async def start(message: types.Message):
    await message.answer("Привет, я тестовый бот!")
```

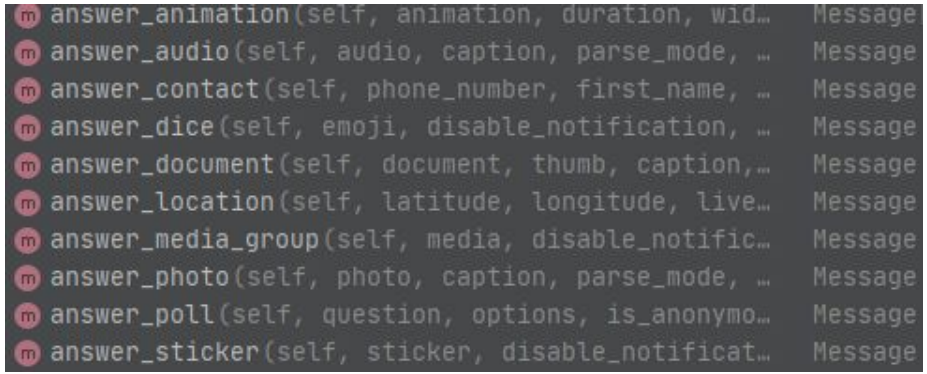
```python
import datetime


@dp.message_handler(commands=["time"])
async def time(message: types.Message):
    await message.answer(datetime.datetime.now().time())
```

Немного методов и переменных у типа Message

message.text это что написал пользователь в бота (Строка)

message.chat.id это уникальный номер чата пользователя. Можно его сохранять в базу данных как пользователя

message.answer отвечает на сообщение пользователя. Можно отправлять разные файлы, фото, аудио, GIF но об этом немного позже



Усложняем логику

Это все тот же питон, только теперь у вас другой вид ввода информации. Например давайте отвечать приветик если в сообщении есть слово привет

```
@dp.message_handler()
async def echo(message: types.Message):
    if "привет" in message.text.lower().split():
        await message.answer("Приветик)")
    else:
        await message.answer(message.text)
```

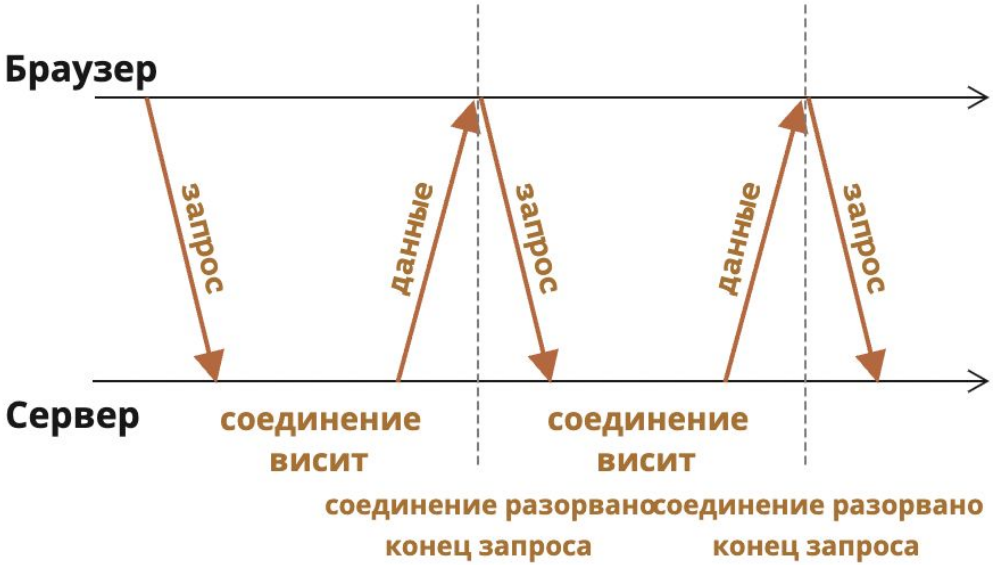
Going deeper

```python
class BaseBot:
    """
    Base class for bot. It's raw bot.
    """
    _ctx_timeout = ContextVar('TelegramRequestTimeout')
    _ctx_token = ContextVar('BotDifferentToken')

    def __init__(
            self,
            token: base.String,
            loop: Optional[Union[asyncio.BaseEventLoop, asyncio.AbstractEventLoop]] = None,
            connections_limit: Optional[base.Integer] = None,
            proxy: Optional[base.String] = None,
            proxy_auth: Optional[aiohttp.BasicAuth] = None,
            validate_token: Optional[base.Boolean] = True,
            parse_mode: typing.Optional[base.String] = None,
            disable_web_page_preview: Optional[base.Boolean] = None,
            protect_content: Optional[base.Boolean] = None,
            timeout: typing.Optional[typing.Union[base.Integer, base.Float, aiohttp.ClientTimeout]] = None,
            server: TelegramAPIServer = TELEGRAM_PRODUCTION
    ):
```

```python
class Dispatcher(DataMixin, ContextInstanceMixin):
    """
    Simple Updates dispatcher

    It will process incoming updates: messages, edited messages, channel posts, edited channel posts,
    inline queries, chosen inline results, callback queries, shipping queries, pre-checkout queries.
    """

    def __init__(self, bot, loop=None, storage: typing.Optional[BaseStorage] = None,
                 run_tasks_by_default: bool = False,
                 throttling_rate_limit=DEFAULT_RATE_LIMIT, no_throttle_error=False,
                 filters_factory=None):
```

```python
async def start_polling(self,
                        timeout=20,
                        relax=0.1,
                        limit=None,
                        reset_webhook=None,
                        fast: bool = True,
                        error_sleep: int = 5,
                        allowed_updates: typing.Optional[typing.List[str]] = None):
    """
    Start long-polling

    :param timeout:
    :param relax:
    :param limit:
    :param reset_webhook:
    :param fast:
    :param error_sleep:
    :param allowed_updates:
    :return:
    """
    if self._polling:
        raise RuntimeError('Polling already started')

    log.info('Start polling.')

    # context.set_value(MODE, LONG_POLLING)
    Dispatcher.set_current(self)
    Bot.set_current(self.bot)
```

1. Запрос отправляется на сервер.
2. Сервер не закрывает соединение, пока у него не возникнет сообщение для отсылки.
3. Когда появляется сообщение – сервер отвечает на запрос, посылая его.
4. Браузер немедленно делает новый запрос.



```python
def message_handler(self, *custom_filters, commands=None, regexp=None, content_types=None, state=None,
                    run_task=None, **kwargs):
    """


    :param commands: list of commands
    :param regexp: REGEXP
    :param content_types: List of content types.
    :param custom_filters: list of custom filters
    :param kwargs:
    :param state:
    :param run_task: run callback in task (no wait results)
    :return: decorated function
    """

    def decorator(callback):
        self.register_message_handler(callback, *custom_filters,
                                      commands=commands, regexp=regexp, content_types=content_types,
                                      state=state, run_task=run_task, **kwargs)
        return callback

    return decorator
```

Регулярками (regex) в Python называются шаблоны, которые используются для поиска соответствующего фрагмента текста и сопоставления символов.

Грубо говоря, у нас есть input-поле, в которое должен вводиться email-адрес. Но пока мы не зададим проверку валидности введённого email-адреса, в этой строке может оказаться совершенно любой набор символов, а нам это не нужно.

```
r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)+$'
```

Синтаксис у регулярок необычный. Символы могут быть как буквами или цифрами, так и метасимволами, которые задают шаблон строки:

Спец. символ	Зачем нужен
.	Задает **один** произвольный символ
[ ]	Заменяет символ из квадратных скобок
-	Задает один символ, которого не должно быть в скобках
[^]	Задает один символ из **не** содержащихся в квадратных скобках
^	Обозначает начало последовательности
$	Обозначает окончание строки
*	Обозначает произвольное число повторений одного символа
?	Обозначает строго одно повторение символа
+	Обозначает один символ, который повторяется несколько раз
|	Логическое **ИЛИ**. Либо выражение до, либо выражение после символа
\	Экранирование. Для использования метасимволов в качестве обычных
( )	Группирует символы внутри
{ }	Указывается число повторений предыдущего символа

Про регулярки

https://tproger.ru/translations/regular-expression-python/

https://www.w3schools.com/python/python_regex.asp

Отправляем картинки

```python
import logging

from aiogram import Bot, Dispatcher, executor, types

API_TOKEN = '1178765389:AAFhPyETaraYuFQsXDh5KwWcd6RlU1nePOM'

logging.basicConfig(level=logging.INFO)

bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot)


@dp.message_handler(regexp='(^cat[s]?$|puss)')
async def cats(message: types.Message):
    with open('cat.jpg', 'rb') as photo:
        await message.reply_photo(photo, caption='Котики 😻')


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

Так мы можем отправлять документы, аудио и много всего другого

```python
import logging

from aiogram import Bot, Dispatcher, executor, types

API_TOKEN = '1178765389:AAFhPyETaraYuFQsXDh5KwWcd6RlU1nePOM'

logging.basicConfig(level=logging.INFO)

bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot)


@dp.message_handler()
async def cats(message: types.Message):
    with open('sound.wav', 'rb') as audio:
        await message.answer_audio(audio, caption='Звуки ♪')


if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

```python
@dp.message_handler()
async def document(message: types.Message):
    with open('text.txt', 'rb') as doc:
        await message.answer_document(doc, caption='Документ 📄')
```

```
answer_dice(self, emoji, disable_notification, …   Message
answer_document(self, document, thumb, caption,…   Message
answer_animation(self, animation, duration, wid…   Message
answer_audio(self, audio, caption, parse_mode, …   Message
answer_contact(self, phone_number, first_name, …   Message
answer_location(self, latitude, longitude, live…   Message
answer_media_group(self, media, disable_notific…   Message
answer_photo(self, photo, caption, parse_mode, …   Message
answer_poll(self, question, options, is_anonymo…   Message
answer_sticker(self, sticker, disable_notificat…   Message
answer_venue(self, latitude, longitude, title, …   Message
answer_video(self, video, duration, width, heig…   Message
answer_video_note(self, video_note, duration, l…   Message
answer_voice(self, voice, caption, parse_mode, …   Message
```

Немного классных фишек aiogram

```python
from aiogram.dispatcher import filters


@dp.message_handler(filters.RegexpCommandsFilter(regexp_commands=['item_([0-9]*)']))
async def send_welcome(message: types.Message, regexp_command):
    await message.reply(f"You have requested an item with id <code>{regexp_command.group(1)}</code>")
```

```python
@dp.message_handler(text=['text1', 'text2'])  # message.text == text1 or message.text == text2 or

@dp.message_handler(text_contains='container1')
@dp.message_handler(text_contains='container2')

@dp.message_handler(text_contains=['str1', 'str2'])

@dp.message_handler(text_startswith=['prefix1', 'prefix2'])

@dp.message_handler(text_endswith=['postfix1', 'postfix2'])
```

Давайте сделаем клавиатуру!

```python
@dp.message_handler(commands='start')
async def start_cmd_handler(message: types.Message):
    keyboard_markup = types.ReplyKeyboardMarkup(row_width=3)

    buttons_text = ('Yes!', 'No!')
    keyboard_markup.row(*(types.KeyboardButton(text) for text in buttons_text))

    more_buttons_text = ("I don't know", "Who am i?", "Where am i?", "Who is there?")
    keyboard_markup.add(*(types.KeyboardButton(text) for text in more_buttons_text))

    await message.reply("Hi!\nHow are you?", reply_markup=keyboard_markup)
```

И тогда нам нужно обрабатывать от нее ответы

```python
@dp.message_handler()
async def all_msg_handler(message: types.Message):
    button_text = message.text
    logger.debug('The answer is %r', button_text)  # print the text we've got

    if button_text == 'Yes!':
        reply_text = "That's great"
    elif button_text == 'No!':
        reply_text = "Oh no! Why?"
    else:
        reply_text = "Keep calm...Everything is fine"

    await message.answer(reply_text, reply_markup=types.ReplyKeyboardRemove())
```

Inline клавиатура

```python
@dp.message_handler(commands='start')
async def start_cmd_handler(message: types.Message):
    keyboard_markup = types.InlineKeyboardMarkup(row_width=3)
    # default row_width is 3, so here we can omit it actually
    text_and_data = (
        ("I'm ok !", 'cool'),
        ('Bad :(', 'bad'),
    )
    # in real life for the callback_data the callback data factory should be used
    # here the raw string is used for the simplicity
    row_btns = (types.InlineKeyboardButton(text, callback_data=data) for text, data in text_and_data)

    keyboard_markup.row(*row_btns)
    keyboard_markup.add(
        # url buttons have no callback data
        types.InlineKeyboardButton('Just link', url='https://bit.ly/37TTiGu'),
    )

    await message.reply("Hi!\nHow are you?", reply_markup=keyboard_markup)
```

Обрабатываем ответы от inline клавиатуры

```python
# Use multiple registrators. Handler will execute when one of the filters is OK
@dp.callback_query_handler(text='cool1')
@dp.callback_query_handler(text='bad')
async def inline_kb_answer_callback_handler(query: types.CallbackQuery):
    answer_data = query.data
    await query.answer(f'You answered with {answer_data!r}')

    if answer_data == 'cool1':
        text = 'Great, me too!'
    else:
        text = 'Oh no...Why so?'

    await bot.send_message(query.from_user.id, text)
```

Бот для чатов

https://docs.aiogram.dev/en/latest/telegram/types/chat.html

Полезная ссылка

https://docs.aiogram.dev/en/latest/telegram/types/message.html

Домашнее задание

В созданной репозитории добавьте файл README.md
Создайте простенького бота с кнопками, который может отправлять картинки и выводить текущее время в разных городах мира.
Подумайте над идеей своего бота