



Руководство по установке API Python дополнительное.

Роботы серии RC

www.robopro.pro/doc

Издание

03.2025

1.1 Назначение

Данное руководство является упрощённым дополнением к основному руководству, созданное для быстрого начала работы и начальной установки. Существует основное руководство, “**Руководство по программированию API Python 1_3_7**”, более подробно описывающее основные наборы команд и функций API. Основное руководство распространяется вместе с архивом API и доступно по ссылке.

<https://t.me/+okbXkLQi8PBLOTgy>

1.2 Целевая группа пользователей

Данная документация предназначена для пользователя со следующими знаниями:

- базовые знания по коллаборативной робототехнике, начальное понимание программирования;

1.3 Термины и определения

API — это набор правил и спецификаций, позволяющий одной программе взаимодействовать с другой программой или системой.

API для работы с коботами реализован на языке программирования Python и распространяется в виде пакета, содержащего папки с файлами на Python.

Оглавление

Шаг 1 – установка Visual studio Code	3
Шаг 2 – установка Python.	5
Шаг 3 – скачивание архива API.	8
Шаг 4 – подготовка к работе с API	10
4.1 Создание виртуальной среды.....	10
4.2 Создание и импорт программ для API.....	15
Примеры программ.....	17
1. 3_points_for_layer.py	17
2. move_1.py	19
3. all_func.py	21
4. Free_drive.py	23
5. Io_any_input_test.py.....	25

Шаг 1 – установка Visual studio Code.

Visual Studio Code (часто сокращенно VSCode) — это популярный кросс-платформенный текстовый редактор с открытым исходным кодом. Нам он нужен для внятного отображения кода и работой с папками API.

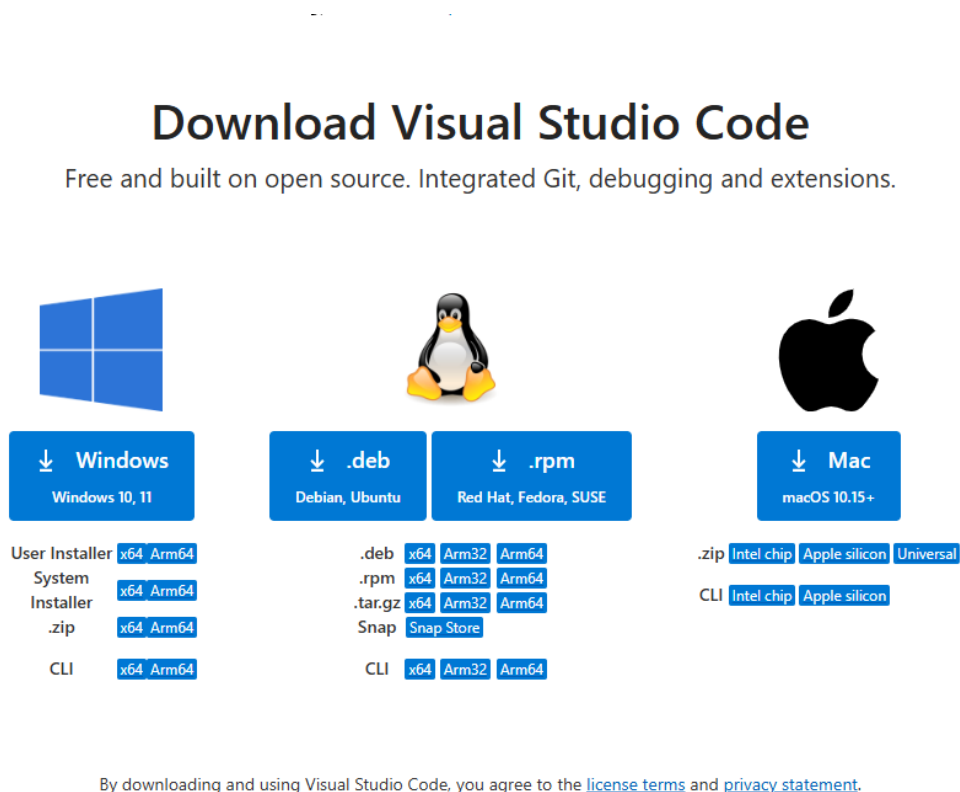
Установка на LINUX

Установка Visual Studio Code (VSCode) на Linux довольно проста и зависит от дистрибутива, который вы используете. Вот несколько способов установки VSCode на популярных дистрибутивах Linux:

1. Установка через официальный сайт

1. Перейдите на официальный сайт VSCode.

<https://code.visualstudio.com/>



2. Выберите версию для вашего дистрибутива (например, .deb для Debian/Ubuntu или .rpm для Fedora/RHEL).
3. Скачайте пакет.
4. Установите пакет, используя соответствующий менеджер пакетов вашей системы:
 - Для Ubuntu/Debian: `sudo dpkg -i <имя_пакета>.deb`
 - Для Fedora/RHEL: `sudo rpm -i <имя_пакета>.rpm`

2. Установка через Snap

1. Убедитесь, что у вас установлен Snapd. Если нет, установите его командой:

```
sudo apt install snapd
```

2. Установите VSCode:

```
sudo snap install code --classic
```

3. Установка через Flatpak

1. Убедитесь, что у вас установлен Flatpak. Если нет, установите его командой:

```
sudo apt install flatpak
```

2. Добавьте репозиторий Flathub:

```
flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo
```

3. Установите VSCode:

```
flatpak install flathub com.visualstudio.code
```

После установки, VSCode будет доступен в меню приложений вашей системы.

Установка на Windows

1. Загрузка установочного файла:

Перейдите на официальный сайт VSCode: <https://code.visualstudio.com/>

- Нажмите на кнопку Download for Windows, чтобы скачать установочный файл.

2. Запуск установщика:

- После загрузки запустите скачанный файл (VSCodeUserSetup-x.xx.x.exe). Появится мастер установки.

3. Процесс установки:

- Следуйте инструкциям мастера установки. Обычно предлагается выбрать место для установки программы и дополнительные опции, такие как добавление ярлыков на рабочий стол и панель быстрого запуска.
- Рекомендуется оставить все настройки по умолчанию, если у вас нет особых предпочтений.

4. Завершение установки:

- После завершения процесса установки нажмите Finish. Программа автоматически откроется, если вы не сняли соответствующую галочку.

Шаг 2 – установка Python.

Python на Windows:

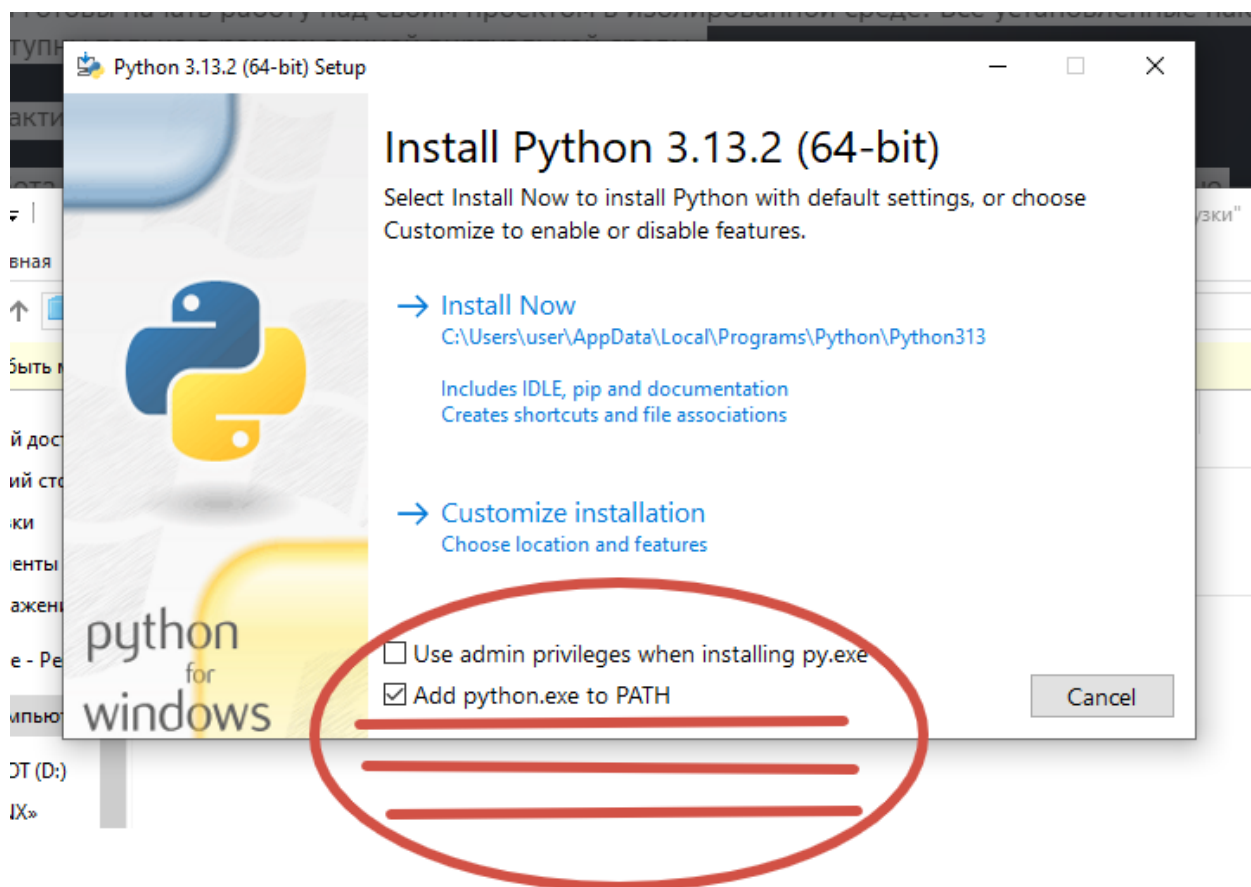
1. Загрузка установочного файла:

- Перейдите на официальный сайт Python: python.org/downloads/windows/.
- Выберите последнюю стабильную версию Python (на момент написания актуальной версией является Python 3.11.0).
- Загрузите установщик для вашей версии Windows (32-bit или 64-bit).

2. Запуск установщика:

- После загрузки запустите скачанный файл (python-3.x.x-amd64.exe или аналогичный).
- Появится окно установщика, в котором вы увидите приветственное сообщение.

3. Настройка установки:



- **Отметьте опцию Add Python to PATH (добавить Python в путь окружения)!**
Это позволит вам запускать Python из командной строки без необходимости указывать полный путь к исполняемому файлу.

- Также можно отметить опцию **Install launcher for all users** (установить лаунчер для всех пользователей), если хотите, чтобы Python был доступен для всех учетных записей на компьютере.

4. Процесс установки:

- Нажмите **Install Now** (Установить сейчас), чтобы начать установку.
- Подождите, пока установка завершится. Процесс занимает несколько минут.

5. Проверка установки:

- После завершения установки закройте установочное окно.
- Откройте командную строку (Windows PowerShell или CMD) и введите команду `python --version`.
- Если выводится версия Python, значит установка прошла успешно.

Установка Python на Ubuntu/Debian Linux

Откройте терминал командой `ctrl + shift + T` и вводите строки ниже.

1. Обновление репозитория:

```
sudo apt update
```

2. Установка Python 3:

```
sudo apt install python3
```

3. Проверка установки:

```
python3 --version
```

4. Установка pip (если необходим):

```
sudo apt install python3-pip
```

Установка Python на CentOS/Fedora

1. Обновление репозитория:

```
sudo dnf update
```

2. Установка Python 3:

```
sudo dnf install python3
```

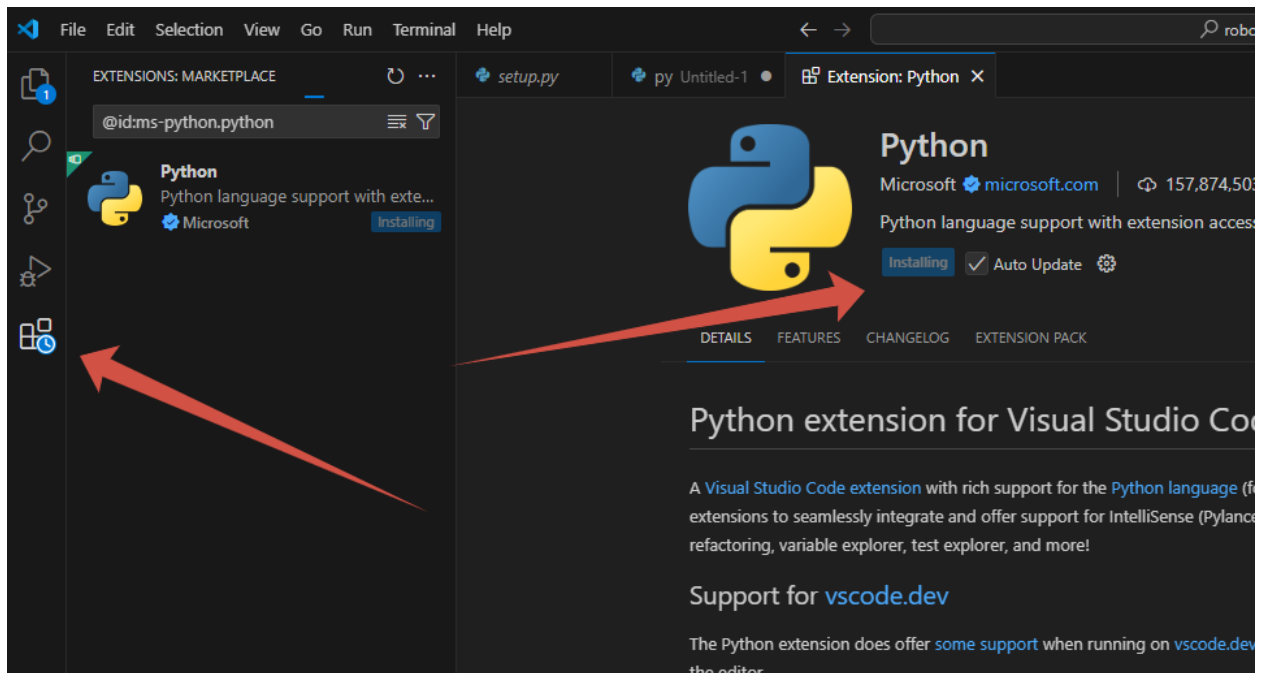
3. Проверка установки:

```
python3 --version
```

4. Установка pip (если необходим):

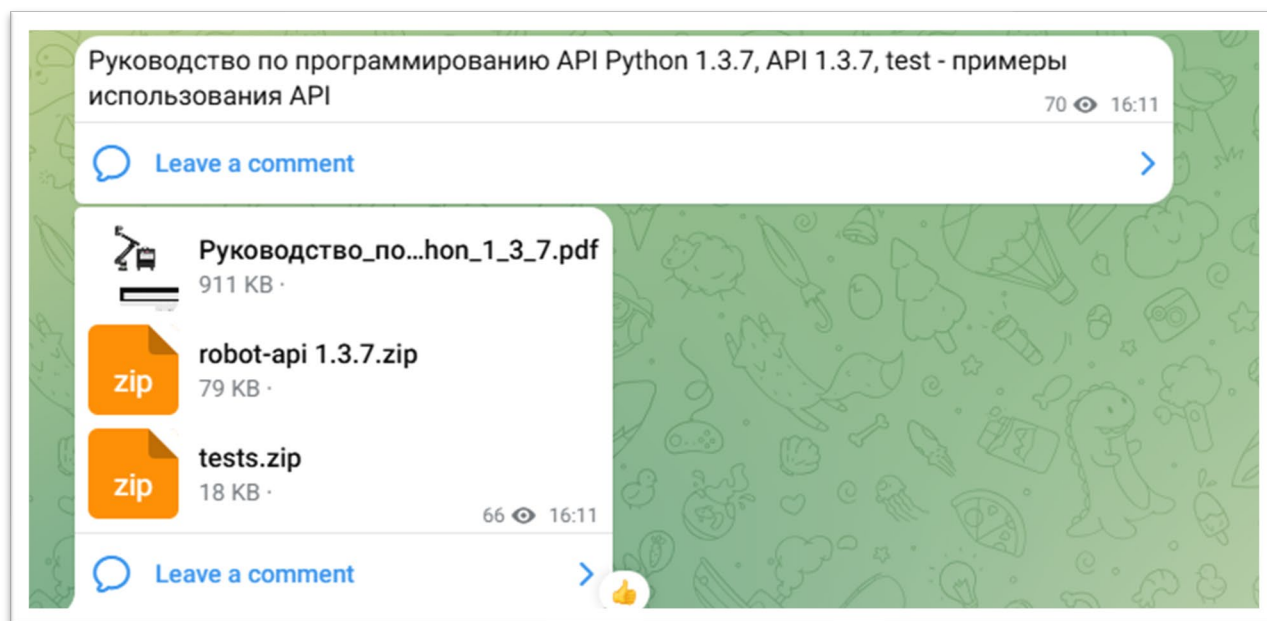
```
sudo dnf install python3-pip
```

Эти действия помогли нам установить python на компьютер, соталось активировать его в программе Vscode.



Для этого нужно найти Python в графе расширения, а затем установить через кнопку install.

Шаг 3 – скачивание архива API.

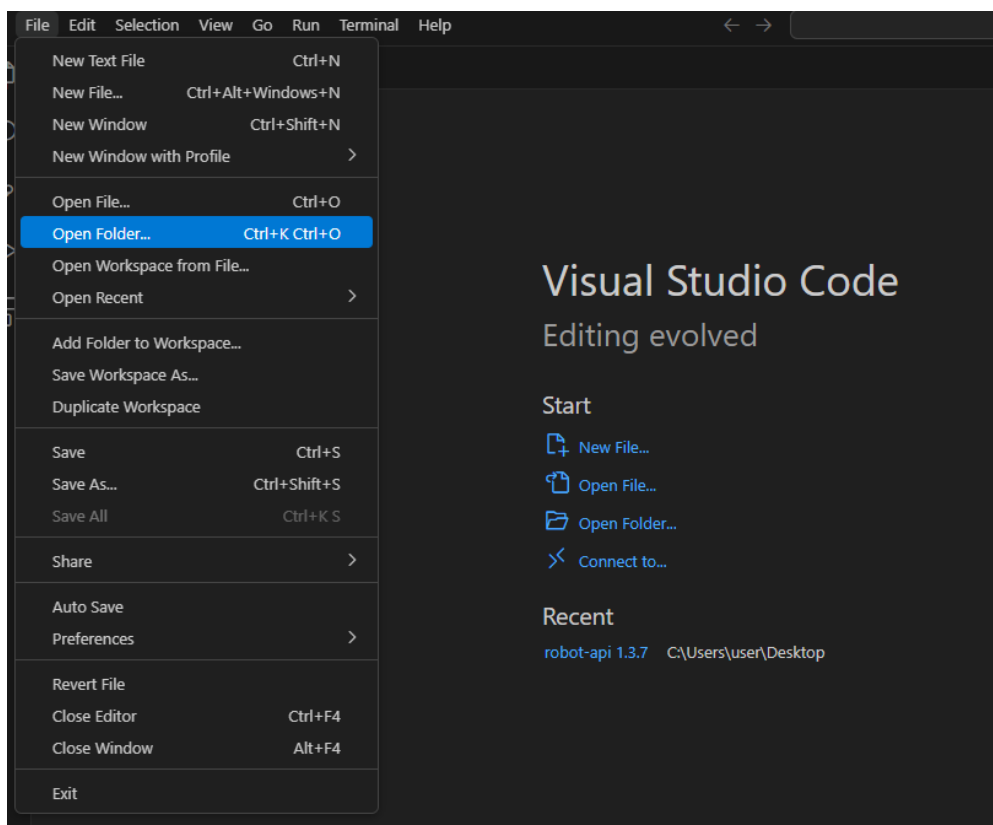


Найдите подходящий информационный ресурс RoboPro, скачайте ZIP-архивы и разархивируйте их, например, на рабочем столе.

<https://t.me/+okbXkLQi8PBIOTgy>

Используйте ссылку выше для перехода в телеграм-канал РобоПро, где можно найти необходимый архив.

Откройте папку robot-api 1.3.7 в программе Vscode.



1. Запустите VSCode:

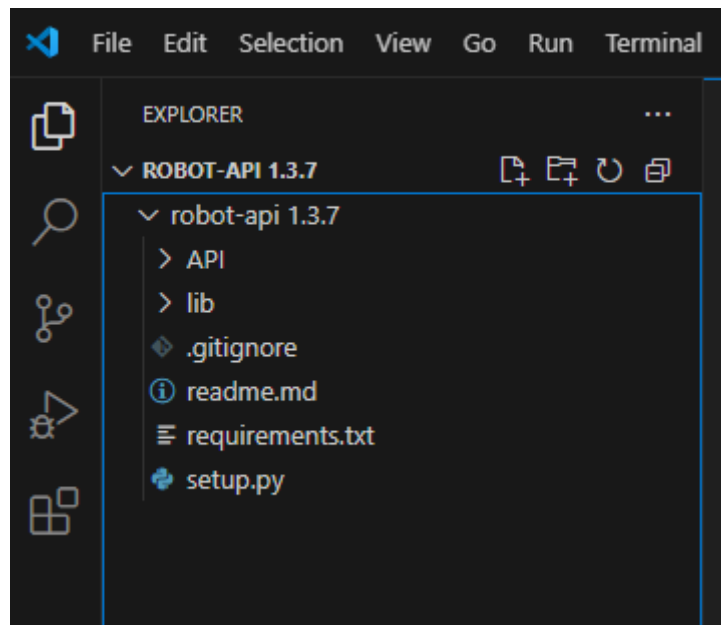
- Дважды щелкните по значку VSCode на рабочем столе или найдите программу в меню "Пуск" и запустите её.

2. Откройте папку в VSCode:

- В окне VSCode перейдите в меню **"File" → "Open Folder..."** (или используйте сочетание клавиш **Ctrl + K**, затем **Ctrl + O**).
- В открывшемся проводнике выберите папку robot-api-1.3.7 и нажмите **"Select Folder"**.

3. Работа с проектом:

- Теперь в левой панели VSCode (Explorer) вы увидите структуру папки robot-api-1.3.7.
- Вы можете начинать работу с файлами и директориями проекта.



Шаг 4 – подготовка к работе с API

4.1 Создание виртуальной среды.

Поскольку для работы API на Python необходимы дополнительные библиотеки, рекомендуется создать виртуальную среду, чтобы изолировать зависимости проекта и избежать возможных конфликтов с другими версиями пакетов или проектами.

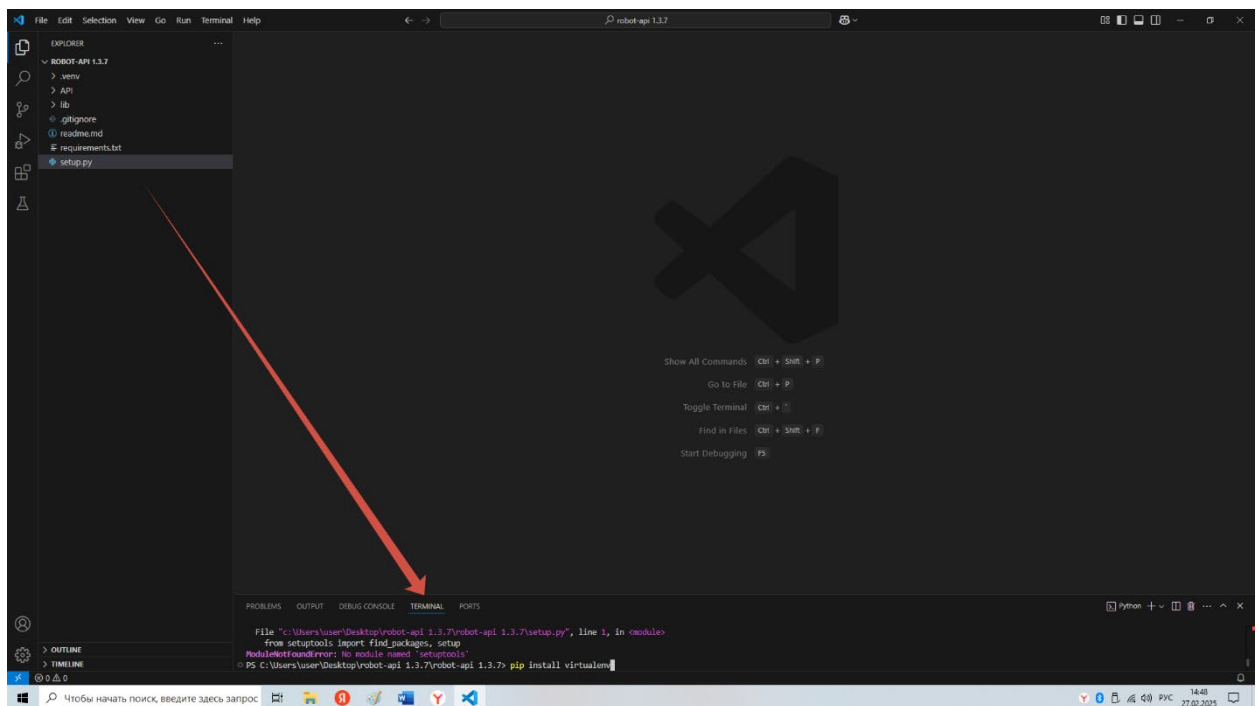
Виртуальная среда Python — это изолированная среда разработки, предназначенная для установки пакетов и управления зависимостями проекта. Она позволяет создавать отдельные окружения для каждого проекта.

Шаг 1: Установите virtualenv

Убедитесь, что у вас установлен virtualenv. Этот инструмент используется для создания виртуальных сред в Python. Если у вас его еще нет, установите его с помощью команды:

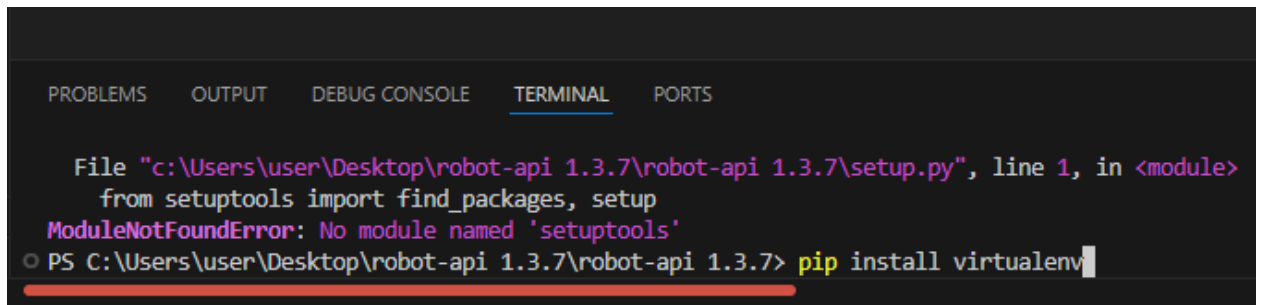
```
pip install virtualenv
```

команду нужно ввести в терминал, терминал показан стрелкой на рисунке ниже.



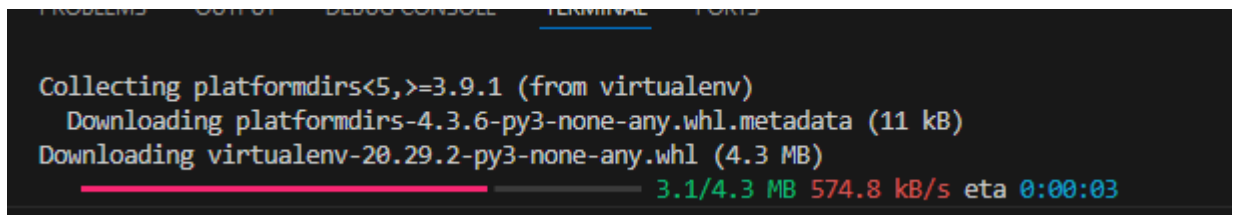
Перед вводом команд, убедитесь, что вы находитесь в директории проекта.

У вас должно отображаться все так, как подчеркнуто красным на рисунке ниже.



The screenshot shows a VS Code terminal window with the 'TERMINAL' tab selected. The terminal output shows an error: `File "c:\Users\user\Desktop\robot-api 1.3.7\robot-api 1.3.7\setup.py", line 1, in <module> from setuptools import find_packages, setup ModuleNotFoundError: No module named 'setuptools'`. Below the error, the command `PS C:\Users\user\Desktop\robot-api 1.3.7\robot-api 1.3.7> pip install virtualenv` is entered. The text `pip install virtualenv` is underlined in red.

После этого вы увидите процесс загрузки и virtualenv будет установлен.



The screenshot shows a VS Code terminal window with the 'TERMINAL' tab selected. The terminal output shows the installation progress of virtualenv: `Collecting platformdirs<5,>=3.9.1 (from virtualenv) Downloading platformdirs-4.3.6-py3-none-any.whl.metadata (11 kB) Downloading virtualenv-20.29.2-py3-none-any.whl (4.3 MB)`. A red progress bar is shown below the text, and the download speed and estimated time are displayed: `3.1/4.3 MB 574.8 kB/s eta 0:00:03`.

Шаг 2: Создайте виртуальную среду

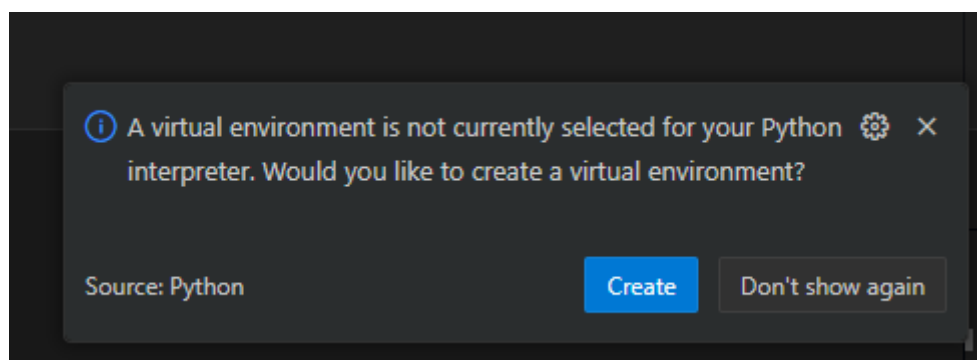
Создайте новую виртуальную среду для вашего проекта. Для этого выполните следующую команду в терминале (или командной строке):

virtualenv <имя_ виртуальной среды >

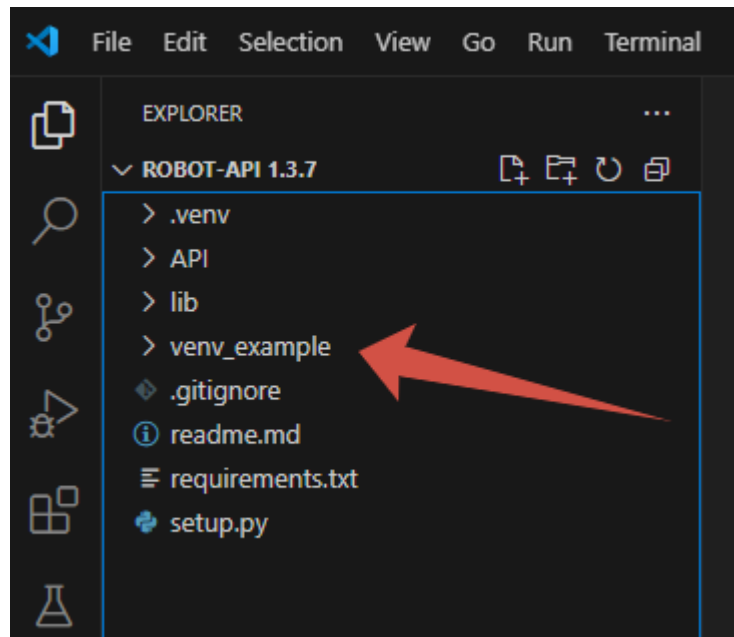
введем команду **virtualenv venv_example**, где venv_example это название-пример для нашей виртуальной среды.

Эта команда создаст директорию venv в текущей папке, где будет храниться ваша виртуальная среда.

Также создать виртуальную среду может предложить сама программа Vscode.



Виртуальная среда **venv_example** появится в директории проекта.



Шаг 3: Активируйте виртуальную среду

Перед началом работы вам нужно активировать созданную виртуальную среду. В зависимости от вашей операционной системы, используйте одну из следующих команд:

- **Windows:**

`<имя_виртуальной среды>\Scripts\activate`

- **Linux/macOS:**

`source <имя_ виртуальной среды >/bin/activate`

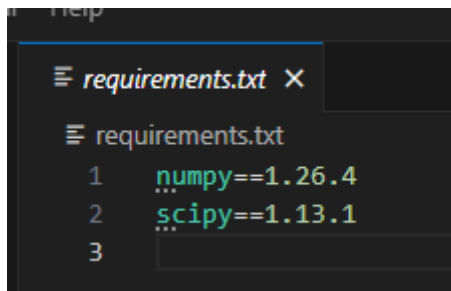
вместо <имя_ виртуальной среды > используйте название вашей виртуальной среды, в нашем случае **venv_example.**

```
PS C:\Users\user\Desktop\robot-api 1.3.7\robot-api 1.3.7> venv_example\Scripts\activate
(venv_example) PS C:\Users\user\Desktop\robot-api 1.3.7\robot-api 1.3.7> |
```

После активации вы увидите (venv) или иное название началом командной строки, что означает, что вы работаете в активной виртуальной среде.

Шаг 4: Установите необходимые пакеты

Теперь, когда виртуальная среда активирована, вы можете установить необходимые пакеты зависимостей из файла requirements.txt в виртуальной среде.

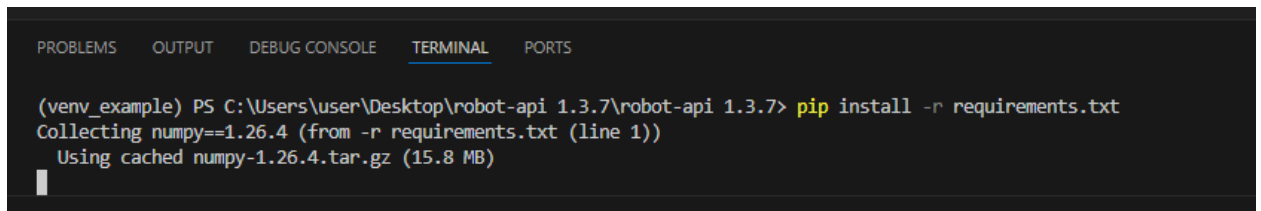


```
requirements.txt X
requirements.txt
1 numpy==1.26.4
2 scipy==1.13.1
3
```

В файле requirements.txt указаны необходимые к установке библиотеки numpy и scipy.

Используйте pip для установки:

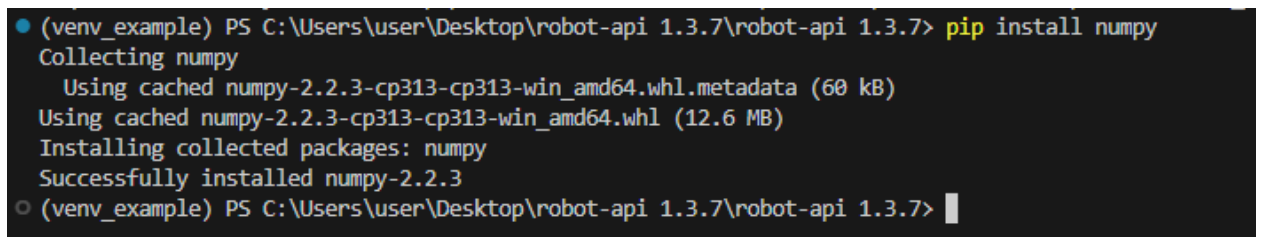
```
pip install -r requirements.txt
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(venv_example) PS C:\Users\user\Desktop\robot-api 1.3.7\robot-api 1.3.7> pip install -r requirements.txt
Collecting numpy==1.26.4 (from -r requirements.txt (line 1))
  Using cached numpy-1.26.4.tar.gz (15.8 MB)
```

В случае возникновения **ошибок** можно установить пакеты в ручную.

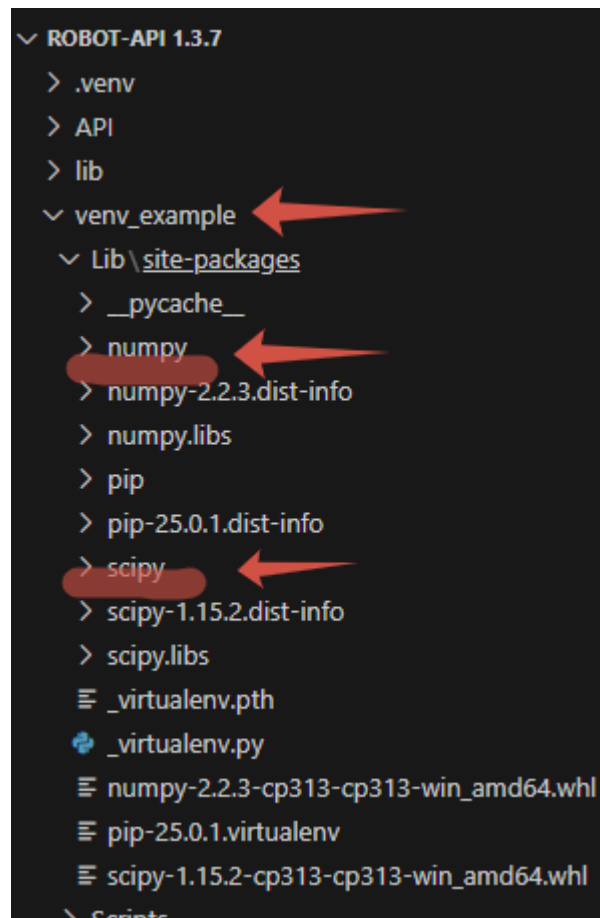
Используйте соответсвующие команды, например, pip install numpy и pip install scipy



```
• (venv_example) PS C:\Users\user\Desktop\robot-api 1.3.7\robot-api 1.3.7> pip install numpy
Collecting numpy
  Using cached numpy-2.2.3-cp313-cp313-win_amd64.whl.metadata (60 kB)
  Using cached numpy-2.2.3-cp313-cp313-win_amd64.whl (12.6 MB)
Installing collected packages: numpy
Successfully installed numpy-2.2.3
○ (venv_example) PS C:\Users\user\Desktop\robot-api 1.3.7\robot-api 1.3.7>
```

Если ваш API требует дополнительные библиотеки, установите их аналогичным образом.

Все установленные библиотеки будут видны в папке вашей виртуальной среды.



Шаг 5: Работайте с API

Теперь вы готовы начать работу над своим проектом в изолированной среде. Все установленные пакеты будут доступны только в рамках данной виртуальной среды.

Деактивация виртуальной среды

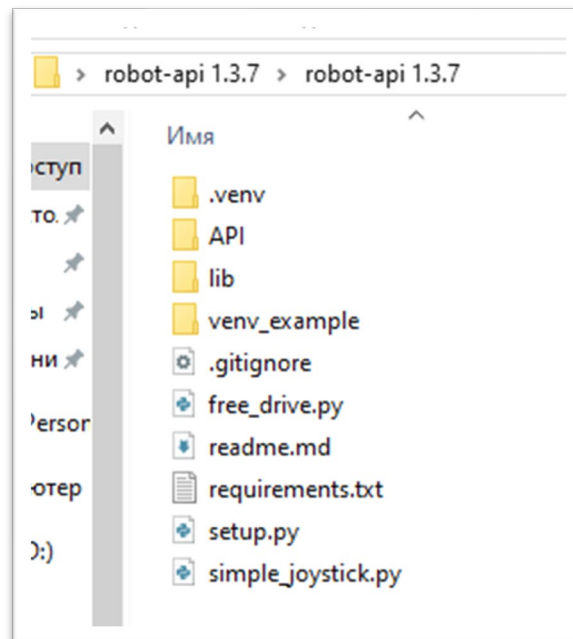
Когда работа над проектом завершится, вы можете деактивировать виртуальную среду с помощью следующей команды:

```
deactivate
```

Это вернет вас в глобальное окружение Python.

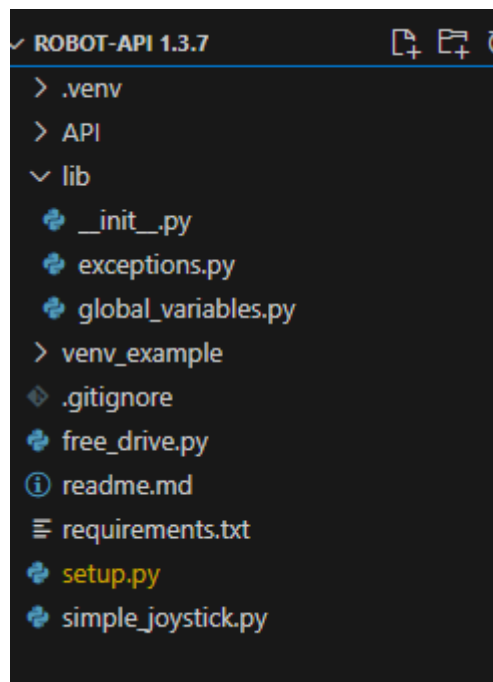
4.2 Создание и импорт программ для API

Переместите файл из папки tests. Возьмем для примера программу *simple_joystick.py*



Поместите ее на один уровень с API в директории.

Отображаться в директории проекта программа должна так. Это корректный уровень.

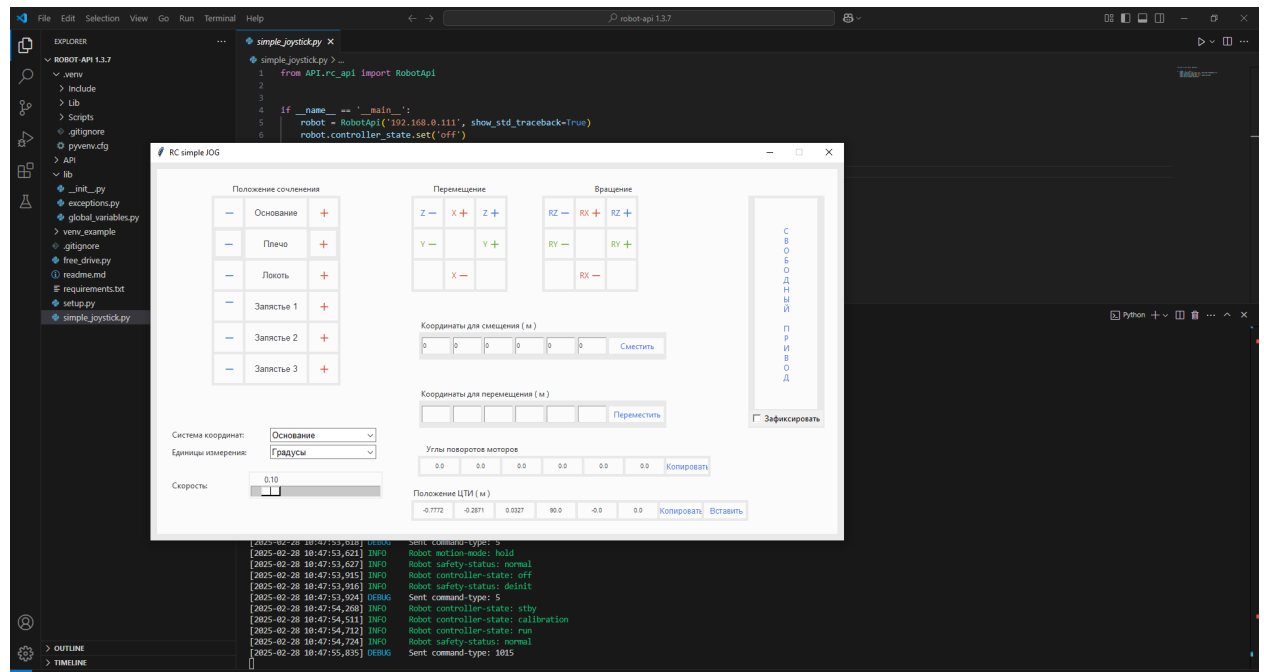


Запустите программу **simple_joystick.py**, введя команду в терминале.

python simple_joystick.py

```
[2025-02-27 17:28:30,300] DEBUG API private thread pool is closed (all threads are joined). threads is 0
(venv_example) PS C:\Users\user\Desktop\robot-api 1.3.7\robot-api 1.3.7> python simple_joystick.py
```

Обратите внимание, чтобы вы были в активной виртуальной среде, в которой есть все нужные для работы библиотеки, а также на то, используете ли вы корректную дирректорию перед командой.



Результатом выполнения этой программы является простой джойстик в виде окна с кнопками управления.

Примеры программ

1. 3_points_for_layer.py

Цель программы заключается в управлении движением робота с учетом определенной пользователем координатной системы.

```
3_points_for_layer.py 3 X
C: > Users > user > Desktop > tests > tests > 3_points_for_layer.py > ...
1  from API.rc_api import RobotApi
2  from API.source.ap_interface.motion.coordinate_system import CoordinateSystem
3  from API.source.features.mathematics.coordinate_system import (
4      calculate_plane_from_points
5  )
6
7
8  if __name__ == '__main__':
9      robot = RobotApi('192.168.0.190', show_std_traceback=True)
10     robot.tool.set(tool_end_point=(-0.00115, 0.00164, 0.14612, 0, 0, 0))
11     robot.controller_state.set('off')
12     robot.controller_state.set('run', await_sec=120)
13     calculated_coordinate_system = calculate_plane_from_points(
14         p0=[-0.62949, -0.48229, 0.00739],
15         pX=[-0.28411, -0.11155, 0.00842],
16         pY=[-0.14781, -0.92954, 0.00827],
17     )
18     robot._logger.info(calculated_coordinate_system)
19     user_coordinate_system = CoordinateSystem(calculated_coordinate_system)
20     # Установить скорость для перемещений в simple_joystick
21     robot.motion.simple_joystick(coordinate_system=user_coordinate_system)
22
```

Код - пример взаимодействия с роботизированным контроллером через интерфейс программирования приложений (API). Давайте разберём ключевые моменты кода:

1. Импорт модулей и классов:

RobotApi — основной класс для взаимодействия с роботом.

CoordinateSystem — структура для представления координатной системы.

calculate_plane_from_points — функция для вычисления плоскости на основе трёх точек.

2. Инициализация робота:

Вызывается конструктор класса **RobotApi**, который принимает IP-адрес робота и флаг **show_std_traceback**, отвечающий за вывод трассировки ошибок.

Устанавливаются параметры инструмента робота с помощью метода **tool.set()**.

3. Управление состоянием контроллера:

Сначала контроллер переводится в состояние "выключено" методом **controller_state.set('off')**.

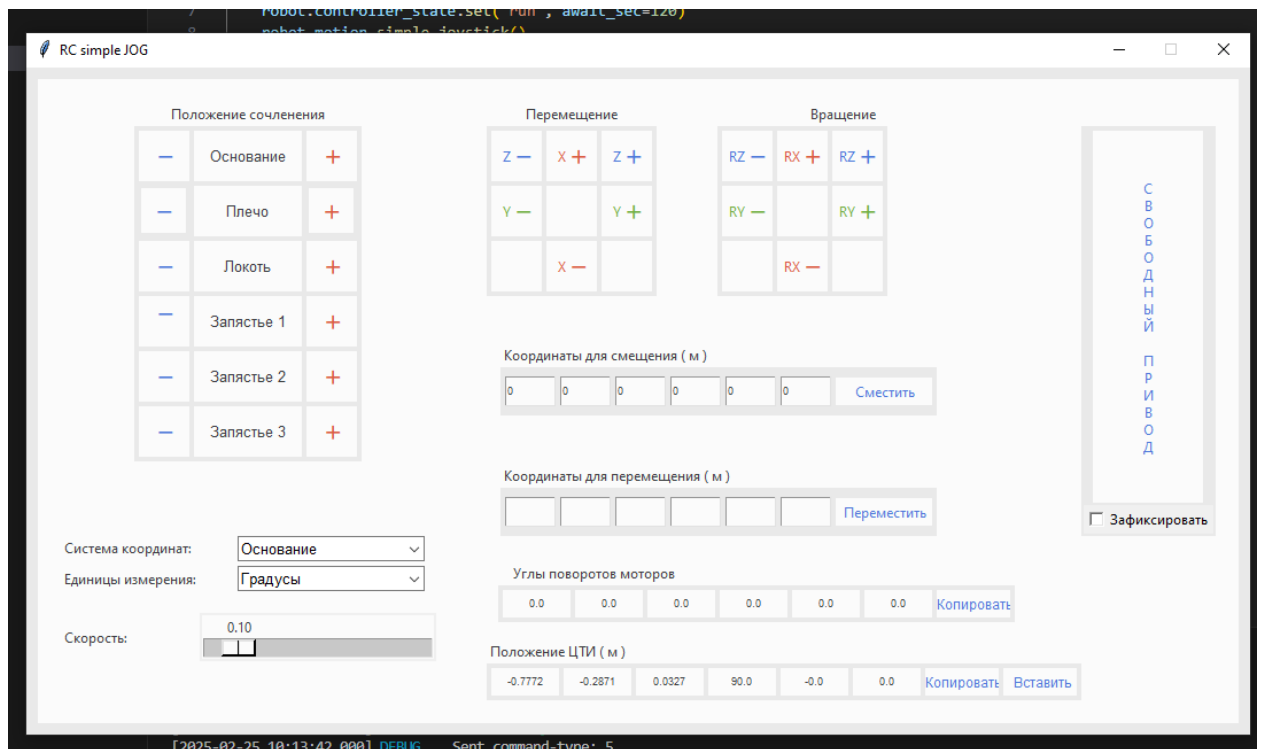
Затем контроллер включается методом **controller_state.set('run', await_sec=120)**, который ожидает 120 секунд перед следующим действием.

4. Вычисление координатной системы:

Функция **calculate_plane_from_points()** вычисляет плоскость, проходящую через три заданные точки. Возвращаемые координаты используются для создания объекта **CoordinateSystem**.

5. Использование пользовательской координатной системы:

Объект **CoordinateSystem** передается методу **motion.simple_joystick()**, который управляет движением робота относительно указанной координатной системы.



2. move_1.py

Цель программы. Код управляет движением робота, задавая последовательность угловых и линейных перемещений.

```
move_1.py X
robot-api 1.3.7 > move_1.py > ...
1  import sys
2  import time
3
4
5  from API.rc_api import RobotApi
6
7  if __name__ == '__main__':
8      robot = RobotApi('192.168.0.111', show_std_traceback=True)
9
10     robot.payload.set(mass=0, tcp_mass_center=(0, 0, 0))
11
12     robot.motion.scale_setup.set(velocity=0.1, acceleration=0.1)
13
14     # Переводим контроллер робота в состояние "run", робот деактивирует
15     # электромеханические тормоза и находится в режиме сервоудержания
16     robot.controller_state.set('run', await_sec=120)
17
18     while True:
19         # Добавляем точки в ядро управления роботом:
20         robot.motion.joint.add_new_waypoint(
21             angle_pose=(0, -115, 120, -100, -90, 0),
22             speed=10,
23             accel=10,
24             units='deg'
25         )
26
27         # Запускаем перемещение по точкам
28         robot.motion.mode.set('move')
29
30         # Ожидаем когда буфер точек будет равен 0
31         robot.motion.wait_waypoint_completion(0)
32
33         # Выполняем перемещение по линейно траектории траектории
34         robot.motion.linear.add_new_waypoint(
35             tcp_pose=(-0.44, -0.16, 0.337, -175, 0, 90),
36             speed=0.2,
37             accel=0.2
38         )
39         robot.motion.mode.set('move')
40         robot.motion.wait_waypoint_completion(0)
41
```

1. Импортируемые модули:

sys: Стандартный модуль Python, предоставляющий доступ к различным системным параметрам и функциям.

time: Модуль для работы с временем.

RobotApi: Пользовательский модуль, обеспечивающий связь с роботом.

2. Основная логика программы:

Создание объекта **RobotApi**:

Конструктор **RobotApi** получает IP-адрес робота и флаг **show_std_traceback**, который, вероятно, контролирует вывод сообщений об ошибках.

3. Настройка полезной нагрузки:

Метод **payload.set** устанавливает параметры полезной нагрузки, такие как масса и центр масс **TCP** (Tool Center Point).

4. Настройка скорости и ускорения движения:

Метод **scale_setup.set** задает параметры скорости и ускорения для движений робота.

5. Перевод контроллера в режим "run":

Метод **controller_state.set('run', await_sec=120)** переводит контроллер робота в рабочее состояние и ожидает 120 секунд перед выполнением последующих команд.

6. Циклическое добавление точек движения:

В бесконечном цикле **while True**: происходит добавление новых точек для перемещения робота.

Метод **joint.add_new_waypoint** добавляет точку для перемещения суставов робота. Указываются углы поворота каждого сустава в градусах, а также скорость и ускорение.

Метод **mode.set('move')** запускает выполнение добавленной траектории.

Метод **wait_waypoint_completion(0)** ожидает завершения выполнения всей последовательности точек.

7. Линейное перемещение:

Метод **linear.add_new_waypoint** добавляет точку для линейного перемещения робота. Указывается конечная позиция TCP в декартовых координатах, а также скорость и ускорение.

После добавления точки выполняется запуск движения с ожиданием завершения.

3. all_func.py

Программа позволяет подключиться к роботу, задать ЦТИ и нагрузку, перемещаться по указанным точкам и управлять джойстиком.

```
all_func.py 2 x
C:\Users\user\Desktop>tests>tests>all_func.py>...
1 import sys
2 import logging
3 from API.rc_api import RobotApi
4 from API.source.models.classes.enum_classes.state_classes import (
5     InComingControllerState as Ics, InComingSafetyStatus as Iss
6 )
7
8 if __name__ == '__main__':
9
10     # Создаем точку входа для управления роботом и настройки логирования
11     robot = RobotApi(
12         '192.168.0.53',
13         enable_logger=True,
14         log_std_level=logging.INFO,
15         enable_logfile=True,
16         logfile_level=logging.INFO
17     )
18
19     # Проверка состояния на наличие ошибки и попытка сброса ошибки
20     if (
21         robot.safety_status.get() == Iss.fault.name
22         or robot.controller_state.get() == Ics.failure.name
23     ):
24         robot.controller_state.set('off')
25
26     # Задать параметры инструмента
27     # Установка координат ЦТИ
28     robot.tool.set((0, 0, 0, 0, 0, 0), units='deg')
29     # Установка нагрузки инструмента
30     robot.payload.set(mass=0, tcp_mass_center=(0, 0, 0))
31
32     # Задать ограничения ускорения и скорости, устанавливаем множитель (до 1)
33     robot.motion.scale_setup.set(velocity=1, acceleration=1)
34
35     # Перевод робота в состояние gup, подача питания и включение режима
36     # сервоудержания
37     robot.controller_state.set('run', await_sec=120)
38
39     # В бесконечном цикле создаем основное тело программы управления
40     while True:
41
42         # Задаем точки перемещения joint
43         robot.motion.joint.add_new_waypoint(
44             angle_pose=(0, -90, 0, -90, 0, 0),
45             speed=20,
46             accel=20,
47             blend=0,
48             units='deg'
49         )
50         robot.motion.joint.add_new_waypoint(
51             angle_pose=(0, -120, 120, -90, -90, 0),
52             speed=20,
53             accel=20,
54             blend=0,
55             units='deg'
56         )
57         robot.motion.joint.add_new_waypoint(
58             angle_pose=(0, -90, 0, -90, 0, 0),
59             speed=20,
60             accel=20,
61             blend=0.5,
62             units='deg'
63         )
64         robot.motion.joint.add_new_waypoint(
65             angle_pose=(0, -120, 120, -90, -90, 0),
66             speed=20,
67             accel=20,
68             blend=0.5,
69             units='deg'
70         )
71
72         # Запускаем перемещение по добавленным точкам
73         robot.motion.mode.set('move')
74
75         # Ожидаем пока робот выполнит перемещения (скрипт находится в ожидании)
76         robot.motion.wait_waypoint_completion(0)
77
78         # Установим параметры скорости и ускорения для всех точек
79         robot.motion.set_motion_config(
80             units='deg',
81             joint_speed=30,
82             joint_acceleration=30,
83             linear_speed=1,
84             linear_acceleration=1
85         )
86         robot.motion.joint.add_new_waypoint(
87             angle_pose=(0, -90, 0, -90, 0, 0)
88         )
89         robot.motion.joint.add_new_waypoint(
90             angle_pose=(0, -120, 120, -90, -90, 0)
91         )
92         robot.motion.joint.jog_once(0, '+')
93
94         # Запускаем окно для перемещения
95         robot.motion.simple_joystick()
96         input('')
97         sys.exit
```

1. Импортируемые модули:

sys: Стандартный модуль Python, позволяющий взаимодействовать с различными системными параметрами и функциями.

time: Модуль для работы с временем, используемый для задержек и синхронизации операций.

RobotApi: Пользовательский модуль, отвечающий за взаимодействие с роботом, отправляя команды и получая данные.

2. Основная логика программы:

Создание объекта RobotApi:

Конструктор **RobotApi** принимает IP-адрес робота и параметр **show_std_traceback**, который управляет отображением трассировки ошибок.

3. Настройка полезной нагрузки:

Метод **payload.set()** позволяет установить параметры полезной нагрузки, такие как её масса и положение центра масс относительно TCP (Tool Center Point).

4. Настройка скорости и ускорения движения:

С помощью метода **scale_setup.set()** задаются параметры скорости и ускорения для перемещений робота. Это помогает контролировать плавность и безопасность движений.

5. Перевод контроллера в режим "run":

Команда **controller_state.set('run', await_sec=120)** переводит контроллер робота в рабочий режим, ожидая 120 секунд перед началом выполнения дальнейших действий.

6. Циклическое добавление точек движения:

Внутри бесконечного цикла **while True:** добавляется последовательность точек для перемещения робота.

Метод **joint.add_new_waypoint()** определяет новую позицию для суставов робота, указывая углы поворотов в градусах, а также параметры скорости и ускорения.

После добавления всех точек вызывается метод **mode.set('move')**, чтобы запустить выполнение траектории.

Для завершения выполнения маршрута используется метод **wait_waypoint_completion(0)**, который ждёт, пока робот достигнет последней точки.

7. Линейное перемещение:

Метод **linear.add_new_waypoint()** добавляет точку для линейного перемещения робота. Указывается целевая позиция TCP в пространстве, а также скорость и ускорение.

4. Free_drive.py

Код инициализирует подключение к роботу, настраивает полезную нагрузку и переходит в режим свободного вождения на определённое количество времени.

```
free_drive.py 2 X
C: > Users > user > Desktop > tests > tests > free_drive.py > ...
1  from API.rc_api import RobotApi
2  from API.source.features.tools import sleep
3
4
5  if __name__ == '__main__':
6      robot = RobotApi('192.168.0.53', show_std_traceback=True)
7      robot.controller_state.set('run', await_sec=120)
8      # Установка нагрузки инструмента,
9      # некорректно заданная нагрузка может привести к неадекватной работе в
10     # режиме free_drive
11     robot.payload.set(mass=0, tcp_mass_center=(0, 0, 0))
12
13     # Выставляем таймер в секундах, free_drive использовать только в рамках
14     # данной функции
15     for i in sleep(200):
16         robot.motion.free_drive()
17
```

1. Импортируемые модули:

API.rc_api.RobotApi: Этот модуль содержит класс RobotApi, который используется для управления роботом. Этот класс обеспечивает связь с физическим роботом и позволяет отправлять команды и получать данные.

API.source.features.tools.sleep: Функция sleep из модуля tools предназначена для создания пауз между операциями. Она заменяет стандартный модуль time.sleep() и используется для ожидания между командами управления роботом.

2. Основная логика программы:

Создание объекта RobotApi:

```
robot = RobotApi('192.168.0.53', show_std_traceback=True)
```

Объект robot создаётся путём вызова конструктора класса RobotApi. Аргументы:

IP-адрес '192.168.0.53': Используется для установления соединения с роботом по сети.

Параметр show_std_traceback=True: Включает вывод трассировки ошибок в случае возникновения исключений.

Перевод робота в режим "run":

```
robot.controller_state.set('run', await_sec=120)
```

Эта команда переводит контроллер робота в состояние готовности ("run").

Параметр `await_sec=120` указывает, что программа будет ожидать 120 секунд, пока робот перейдёт в это состояние.

3. Настройка полезной нагрузки:

Установка массы и центра масс инструмента:

```
robot.payload.set(mass=0, tcp_mass_center=(0, 0, 0))
```

Этот метод устанавливает параметры полезной нагрузки, прикреплённой к концу манипулятора (инструмента). Здесь задана нулевая масса (`mass=0`) и центр масс TCP установлен в точке `(0, 0, 0)`.

4. Режим свободного вождения (Free Drive):

Ожидание и включение режима Free Drive:

```
for i in sleep(200):
```

```
    robot.motion.free_drive()
```

Цикл `for` выполняется 200 раз, каждый раз вызывая функцию `robot.motion.free_drive()`, которая включает режим Free Drive. В этом режиме оператор может вручную управлять движениями робота.

5. io_any_input_test.py

Этот код инициализирует подключение к роботу, переводит его в рабочее состояние и постоянно отслеживает изменения в цифровых входах, фиксируя эти события в журнале.

```
io_any_input_test.py 1 X
C: > Users > user > Desktop > tests > tests > io_any_input_test.py > ...
1  from API.rc_api import RobotApi
2
3  if __name__ == '__main__':
4      robot = RobotApi('192.168.0.63')
5      robot.controller_state.set('run', await_sec=120)
6
7      while True:
8          robot.io.digital.wait_any_input()
9          if robot.io.digital.get_input(0):
10             robot._logger.info('Index 0 has been changed')
11          if robot.io.digital.get_input(1):
12             robot._logger.info('Index 1 has been changed')
13          input()
14
```

код управляет состоянием робота, переводя его в режим "run" и ожидая любые изменения на цифровых входах. Когда изменяется состояние какого-либо входа, информация об этом фиксируется в логе. Программа продолжает работу бесконечно, пока её не прервут вручную.

1. Импортируемые модули:

- **API.rc_api.RobotApi:** Этот модуль содержит класс RobotApi, который используется для управления роботом. Класс предоставляет методы для отправки команд и получения данных от физического робота.

2. Основная логика программы:

Создание объекта RobotApi:

```
robot = RobotApi('192.168.0.63')
```

Создаётся объект robot — это экземпляр класса RobotApi, который представляет собой интерфейс для управления роботом. В качестве аргумента передаётся IP-адрес робота ('192.168.0.63'), что позволяет установить сетевое соединение с устройством.

Перевод робота в режим "run":

```
robot.controller_state.set('run', await_sec=120)
```

Метод `controller_state.set()` переводит контроллер робота в состояние "run". Это значит, что робот переходит в режим готовности к выполнению операций.

Параметр `await_sec=120` указывает, что программа будет ожидать 120 секунд, пока робот достигнет нужного состояния.

Циклический опрос цифровых входов:

```
while True:
```

```
    robot.io.digital.wait_any_input()
```

Бесконечный цикл `while True`: запускает процесс постоянного опроса цифровых входов робота. Метод `wait_any_input()` блокирует выполнение программы до тех пор, пока не произойдёт какое-либо изменение на одном из цифровых входов.

Обработка изменений на цифровых входах:

```
if robot.io.digital.get_input(0):
```

```
    robot._logger.info('Index 0 has been changed')
```

После того как произошло изменение на цифровом входе, программа проверяет состояние конкретного входа с помощью метода `get_input(index)`. Если состояние входа с индексом 0 изменилось, в лог записывается сообщение "Index 0 has been changed".

```
if robot.io.digital.get_input(1):
```

```
    robot._logger.info('Index 1 has been changed')
```

Аналогичная проверка происходит для входа с индексом 1. При изменении состояния входа с индексом 1, программа записывает в лог сообщение "Index 1 has been changed".

Остановка программы:

```
input()
```

Функция `input()` вызывает ожидание ввода с клавиатуры. Эта строка служит для приостановки выполнения программы до тех пор, пока пользователь не введёт какую-либо команду или не нажмет Enter. Это полезно для тестирования или временного прекращения работы программы.