Введение в программирование робота на Python

Начало обучения программированию с RoboIntellect (для модели RM-001)

Содержание

Введение

Дорогой читатель!

Если ты читаешь эту инструкцию, значит ты получил доступ к роборуке производства ООО НПО “Робоинтеллект”, узнал, что она является программируемым устройством, и тебе стало интересно, как писать собственные программы для роборуки и других робототехнических устройств. Этот документ поможет тебе начать, если до этого момента у тебя не было возможности получить опыт в программировании.

В процессе чтения этой инструкции мы выясним, как подготовить компьютер к написанию программ, разберём несколько примеров, и даже напишем свои собственные программы.

Программное обеспечение роборуки включает в себя RI SDK. RI SDK - это расширяемая библиотека, предназначенная для программирования робототехнических устройств. Она позволяет программисту писать код для компонентов автоматизируемых систем, сосредотачивая внимание на функциональных задачах, а не на особенностях работы компонентов.

RI SDK написана на языке программирования Golang, но может быть интегрирована в программы и на других языках, таких как C, C++, Python, PHP, и прочие. Эта инструкция написана на основе примеров на Python.

Это руководство начинается с инструкции по развертыванию на компьютере рабочего окружения для программирования роборуки на Python, включающего в себя интерпретатор, стандартную библиотеку, и редактор исходного кода. Затем следует пример пример вызова готовой программы, написанной на Python.

Подготовка рабочего окружения для программирования на Python

Для выполнения программ на Python потребуется установить на компьютер интерпретатор Python и IDE.

Установка интерпретатора

Загрузка файла инсталлятор Python

Для установки интерпретатор Python на компьютер необходимо сначала скачать инсталлятор языка с официального сайта.

Для этого необходимо перейти по ссылке https://www.python.org/downloads/, и кликнуть по кнопке Download Python 3.10.2 (версия может со временем измениться) (см. рисунок 1).

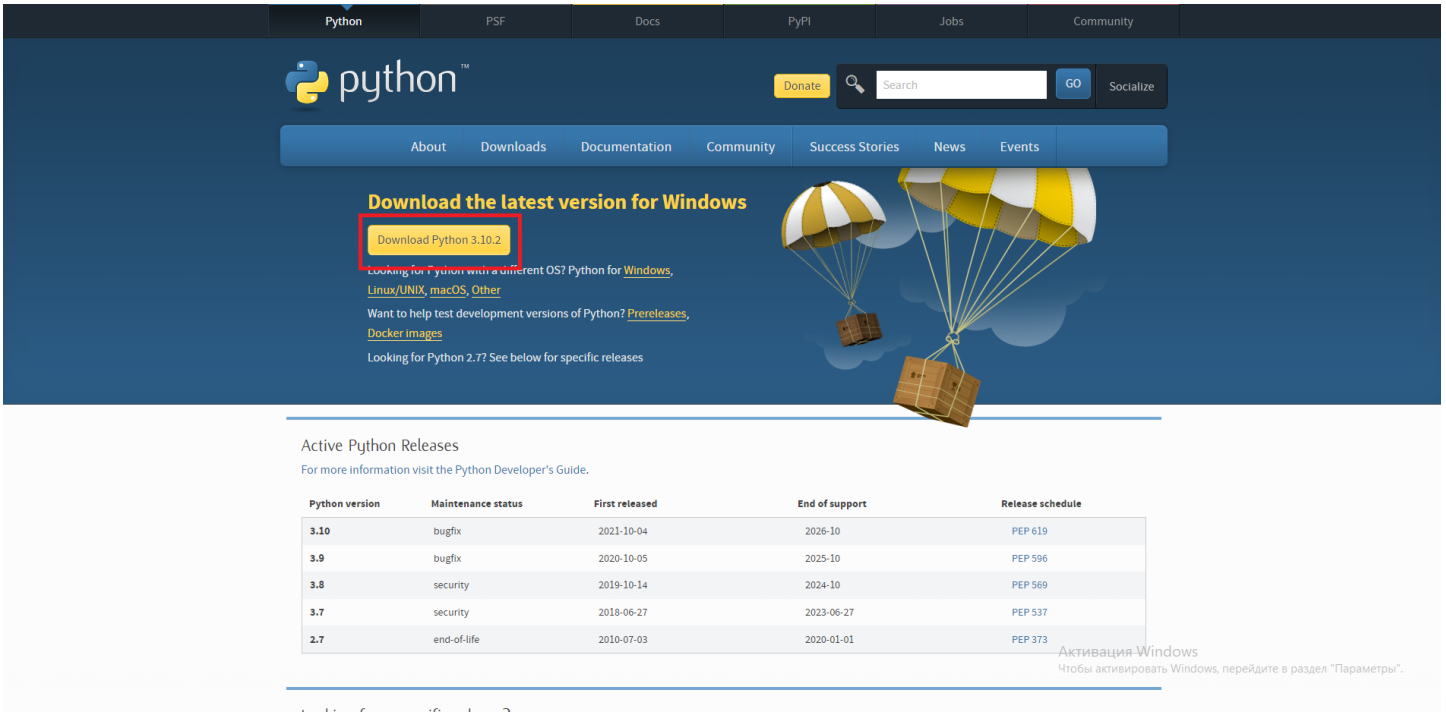


Рисунок 1. Страница для скачивания инсталлятора языка

После клика по кнопке, которая выделена красным цветом на рисунке 1, начнётся загрузка файла инсталлятора Python. Загрузка выполняется по пути, указанному в настройках браузера. По-умолчанию это будет C:\Users_имя_пользователя_ОС_\Downloads.

Установка интерпретатора

После загрузки инсталлятора необходимо перейти в директорию, в которую был загружен файл, и двойным кликом левой кнопки мыши запустить процесс установки интерпретатора Python. В результате откроется первый экран окна процесса установки интерпретатора.

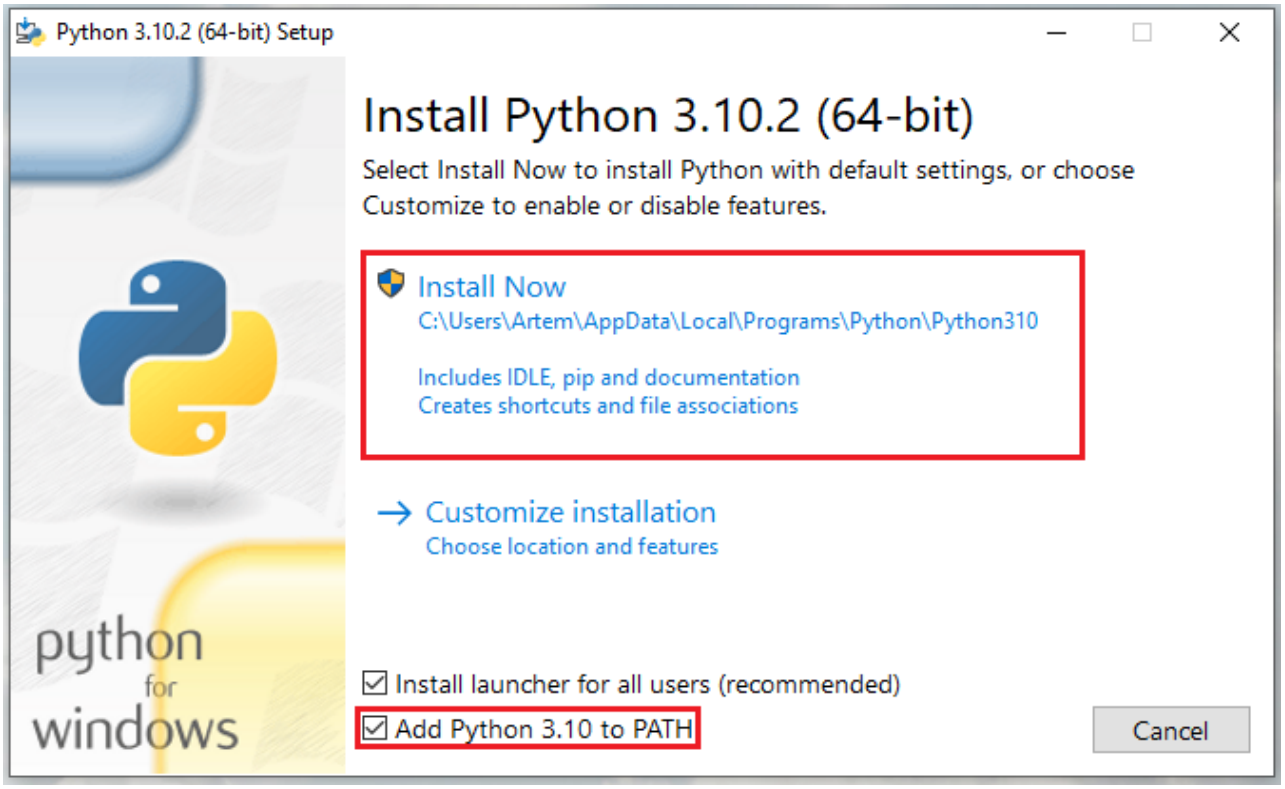


Рисунок 2. Первый экран окна процесса установки интерпретатора Python

Для старта процесса установки необходимо убедиться, что установлен флаг Add Python 3.10 to PATH (внизу на рисунке 2), и кликнуть по кнопке Install Now (вверху на рисунке 2).
Если не установить флаг Add Python 3.10 to PATH, то после установки интерпретатора потребуется путь к нему в переменную окружения PATH задавать вручную.
После клика по кнопке Install Now появится диалоговое окно с подтверждением начала процесса установки. Необходимо нажать “Да”. Начнётся процесс установки интерпретатора, который обычно занимает несколько минут.

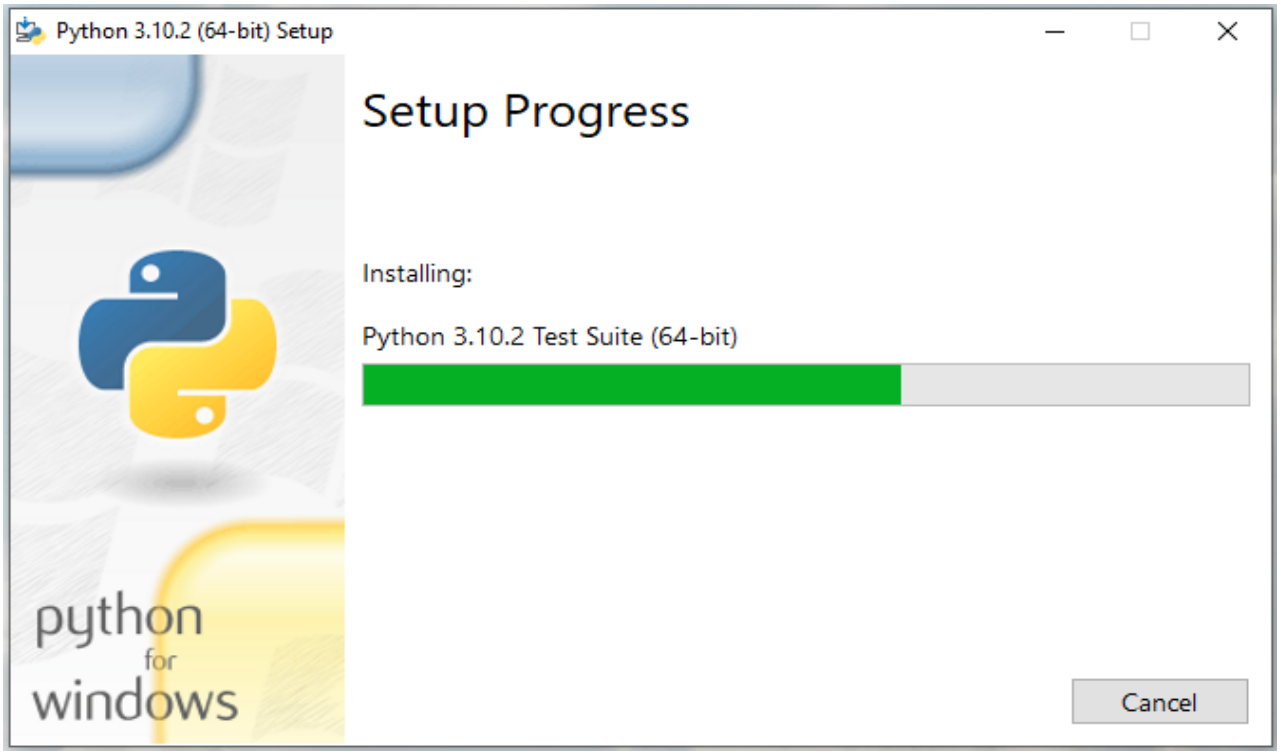


Рисунок 3. Процесс установки интерпретатора Python

После завершения установки появится окно с подтверждением. Необходимо нажать кнопку Close (см. рисунок 4).

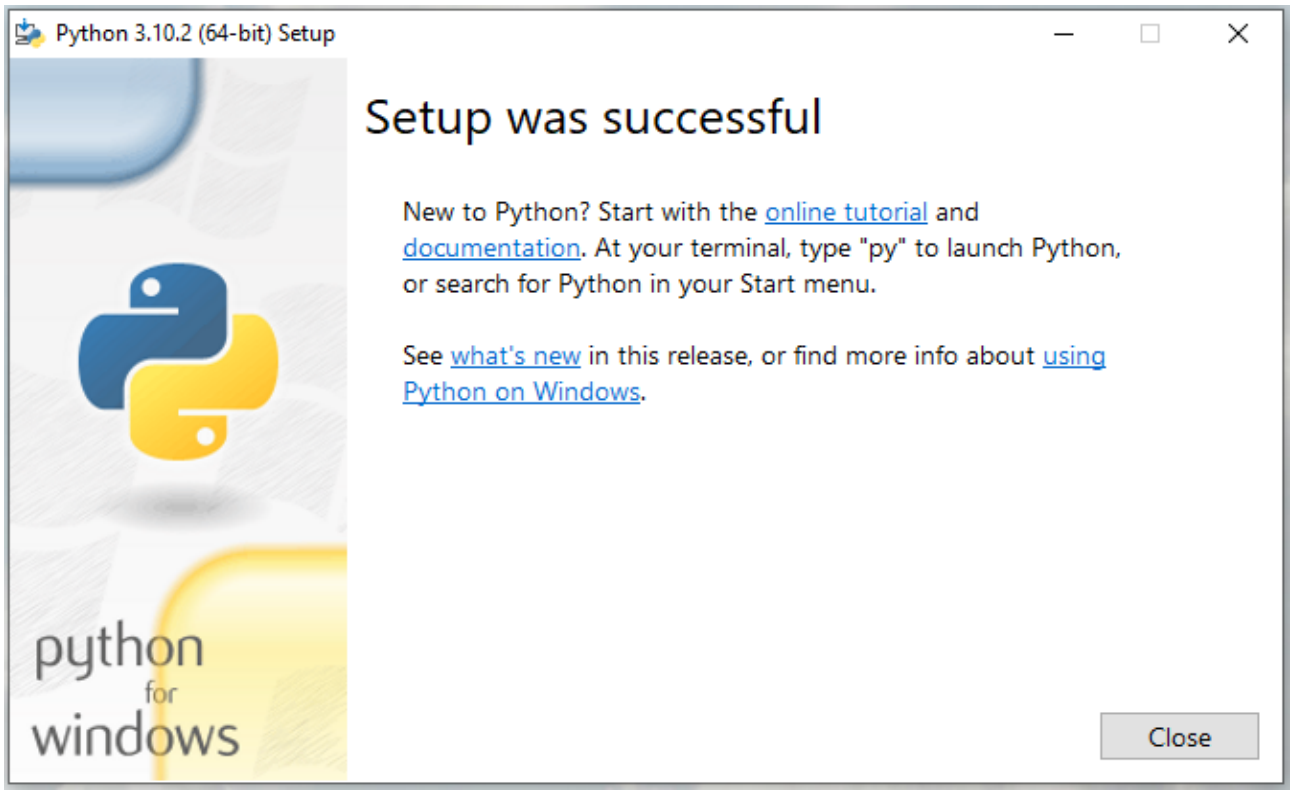


Рисунок 4. Подтверждение завершения установки Python.

На этом окне размещено несколько полезных ссылок на документацию по Python. Если на первом экране Вы не установили флаг Add Python 3.10 to PATH (см. рисунок 1), то следует перейти к разделу “Настройка переменных окружения для Python”. В противном случае можно этот раздел пропустить.

Настройка переменных окружения для Python

Если на этапе установки интерпретатора Вы не установили флаг Add Python 3.10 to PATH (см. рисунок 1), то в процессе не были автоматически заданы переменные окружения, необходимые для корректного выполнения программ на Python.

В этом разделе инструкции Вы найдёте, как установить требуемые переменные окружения вручную.

Внимание! Если Вы установили флаг Add Python 3.10 to PATH на первом экране окна установки интерпретатора, то данный раздел инструкции необходимо пропустить.

Для начала нажмите “Пуск”, и кликните по иконке “Параметры” (см. рисунок 5).

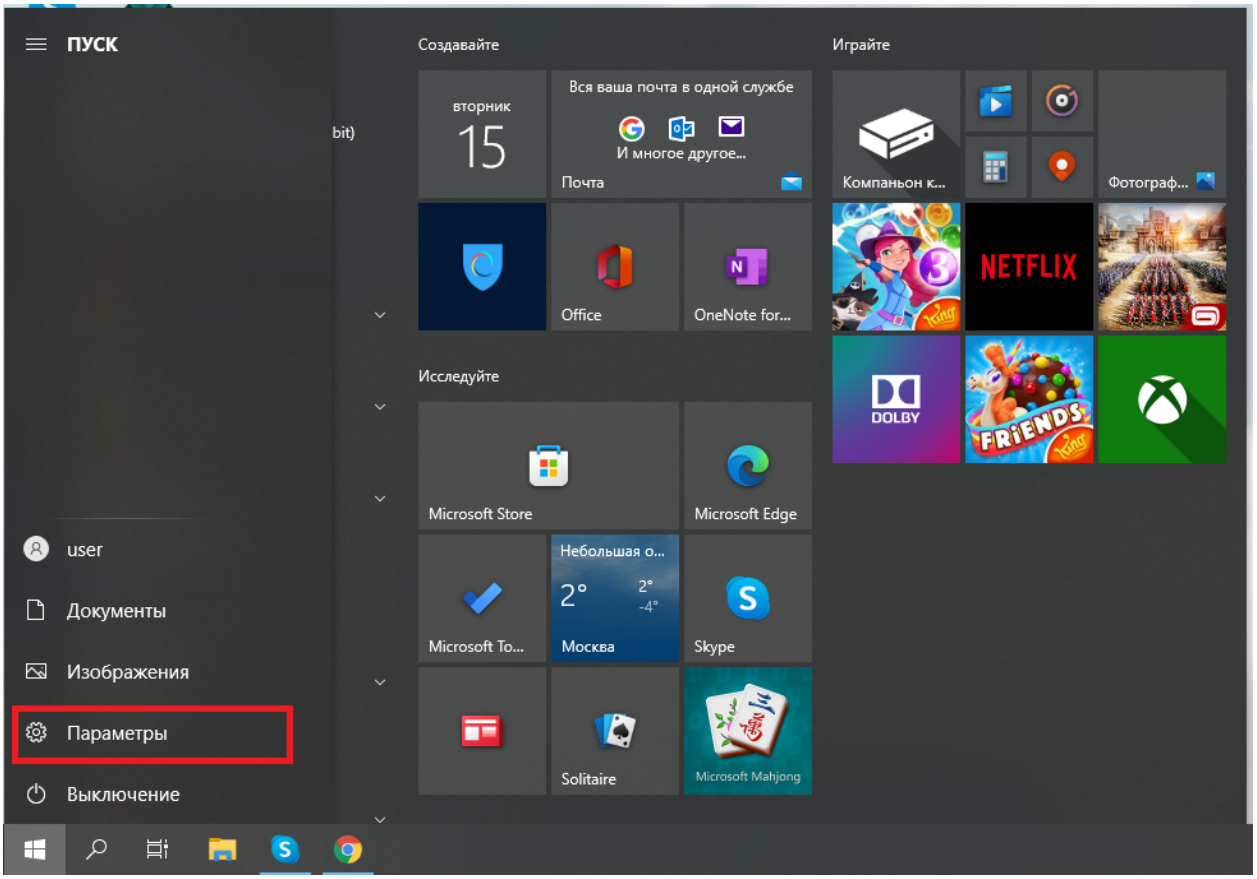


Рисунок 5. Меню “Пуск”

Откроется модальное окно параметров операционной системы. На этом окне необходимо выбрать раздел “Система” (см. рисунок 6).

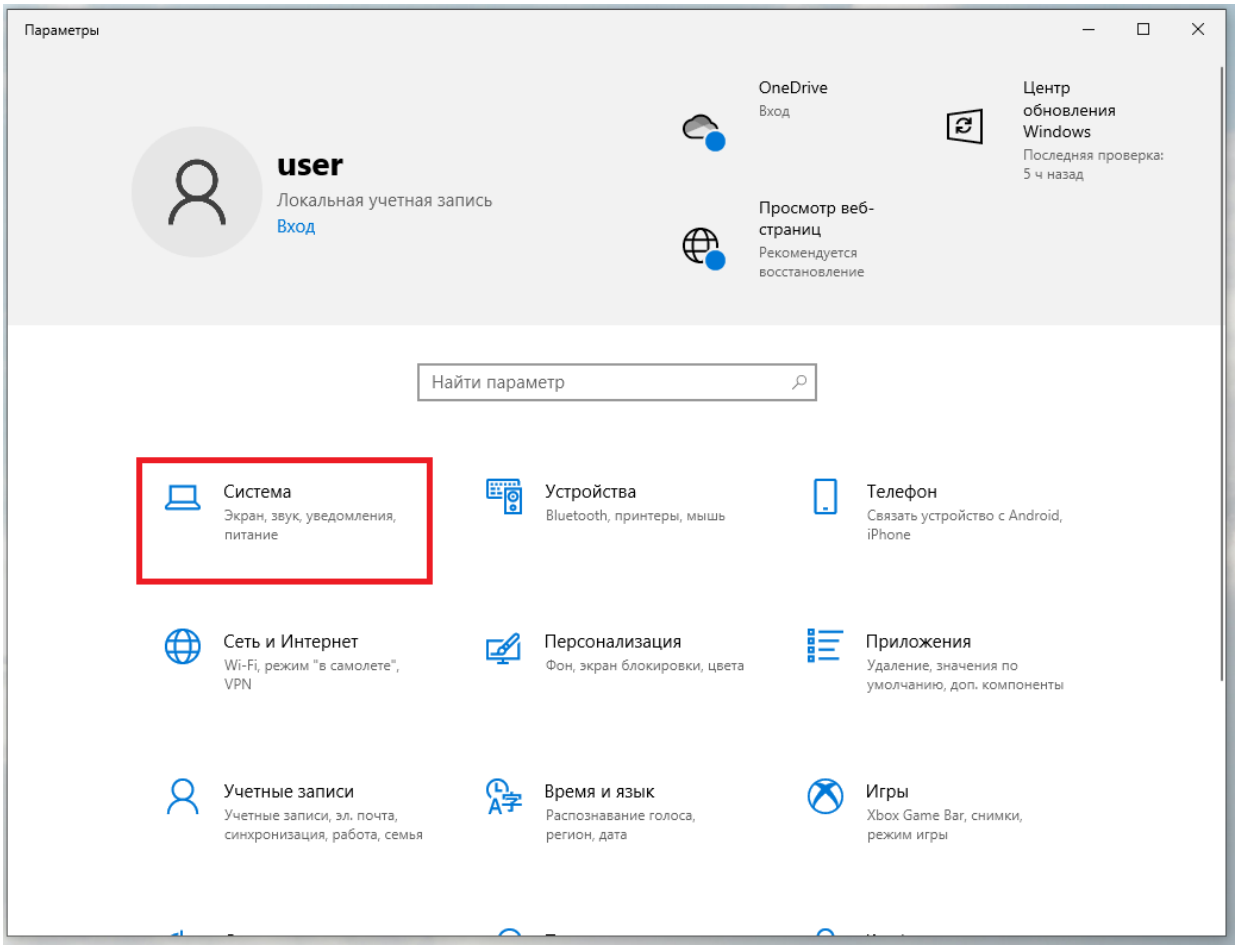


Рисунок 6. Параметризация ОС

После перехода в раздел “Система” откроется окно, представленное на рисунке 7. Необходимо перейти в раздел “О программе” (1), затем кликнуть по ссылке “Дополнительные параметры системы” (2), и в открывшемся модальном окне кликнуть по кнопке “Переменные среды” (3).

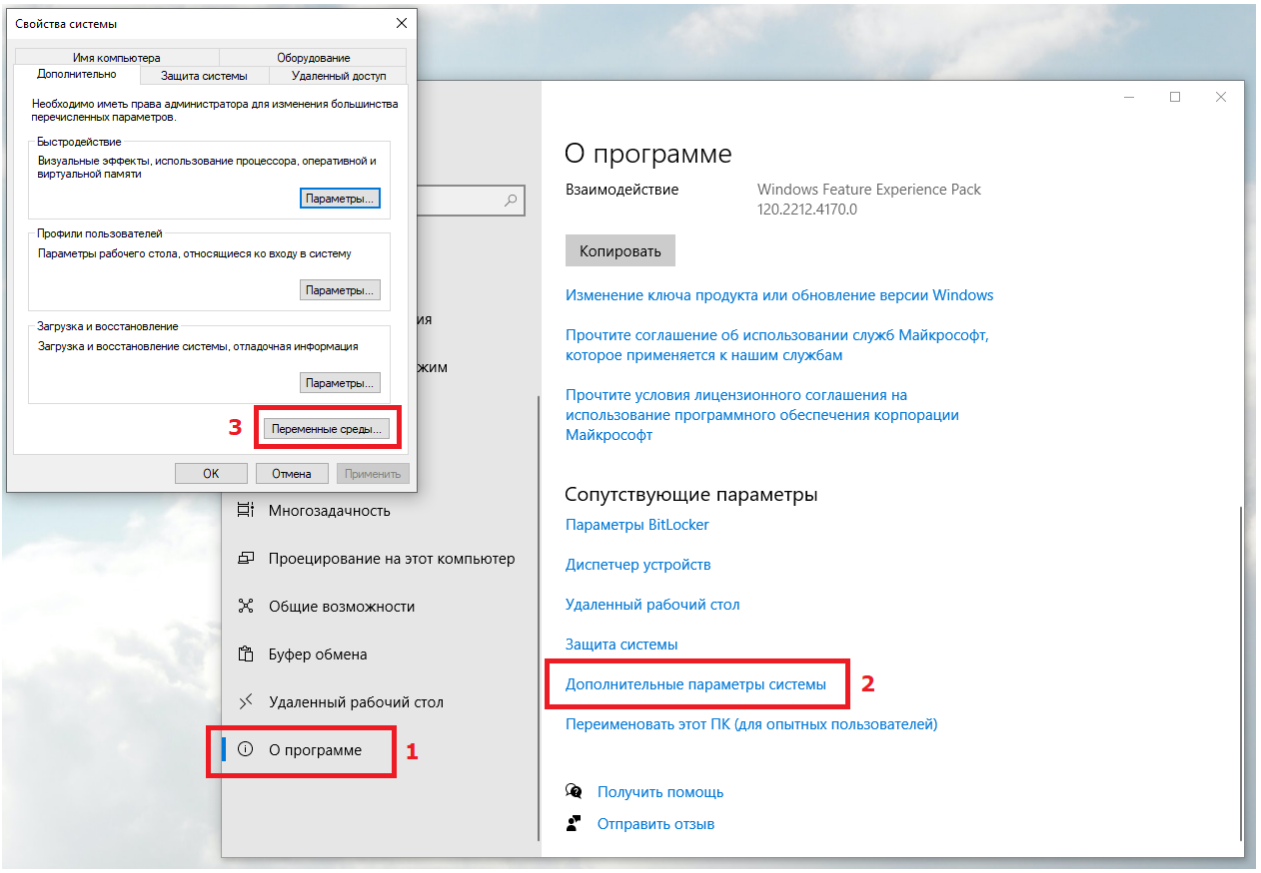


Рисунок 7. Последовательные переходы к редактированию переменных окружения

На открывшемся модальном окне редактирования переменных окружения необходимо выбрать переменную Path в разделе переменных для пользователя, и нажать кнопку “Изменить” (см. рисунок 8)

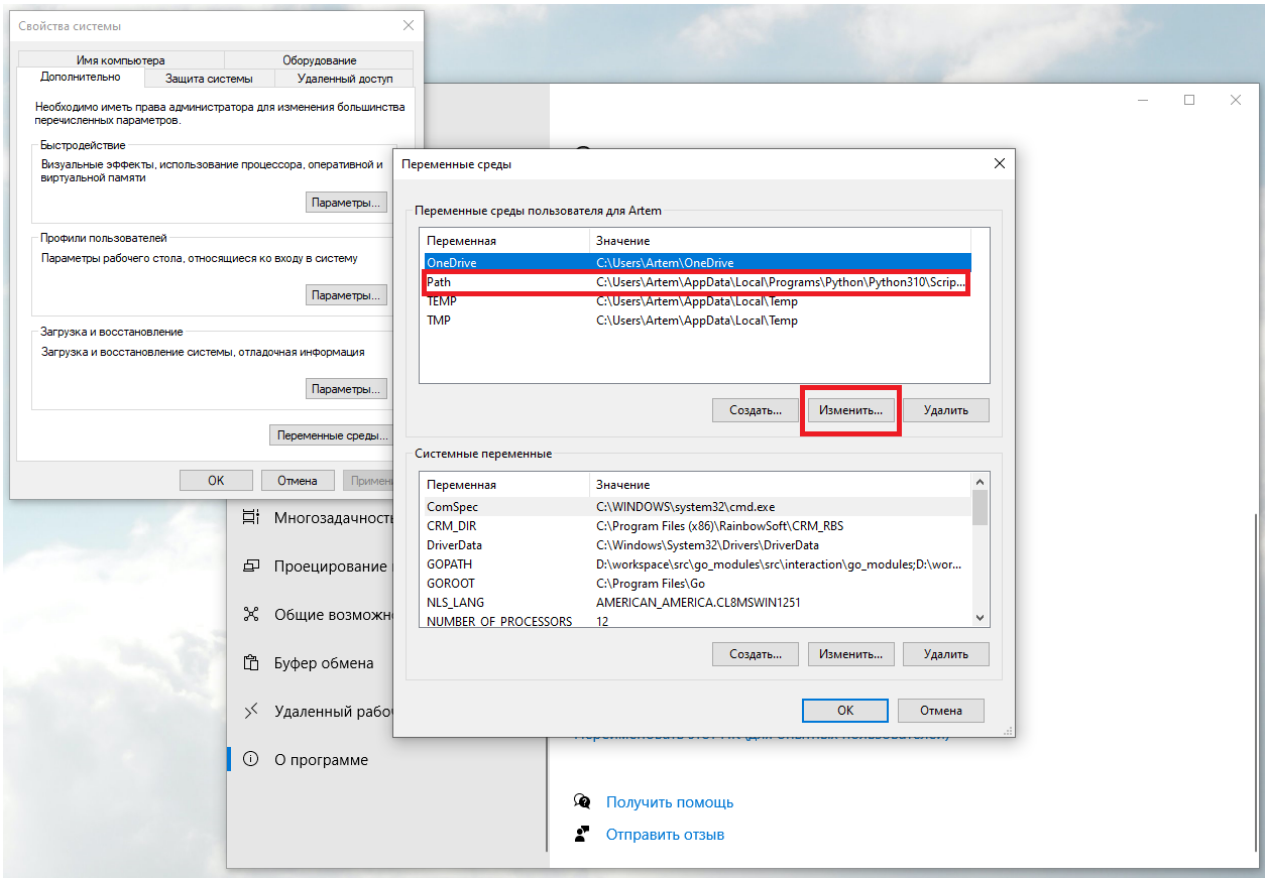


Рисунок 8. Выбор переменной окружения для редактирования

После выбора переменной Path и клика по кнопке “Изменить” откроется модальное окно редактирования переменной (см. рисунок 9). С помощью кнопки “Создать” необходимо добавить в Path два значения: путь к корневой директории с интерпретатором Python, и путь к внутренним скриптам Python.
Первая - это обычно C:\Users_имя_пользователя_ОС_\AppData\Local\Programs\Python\Python310\
а вторая - это
C:\Users_имя_пользователя_ОС_\AppData\Local\Programs\Python\Python310\Scripts\

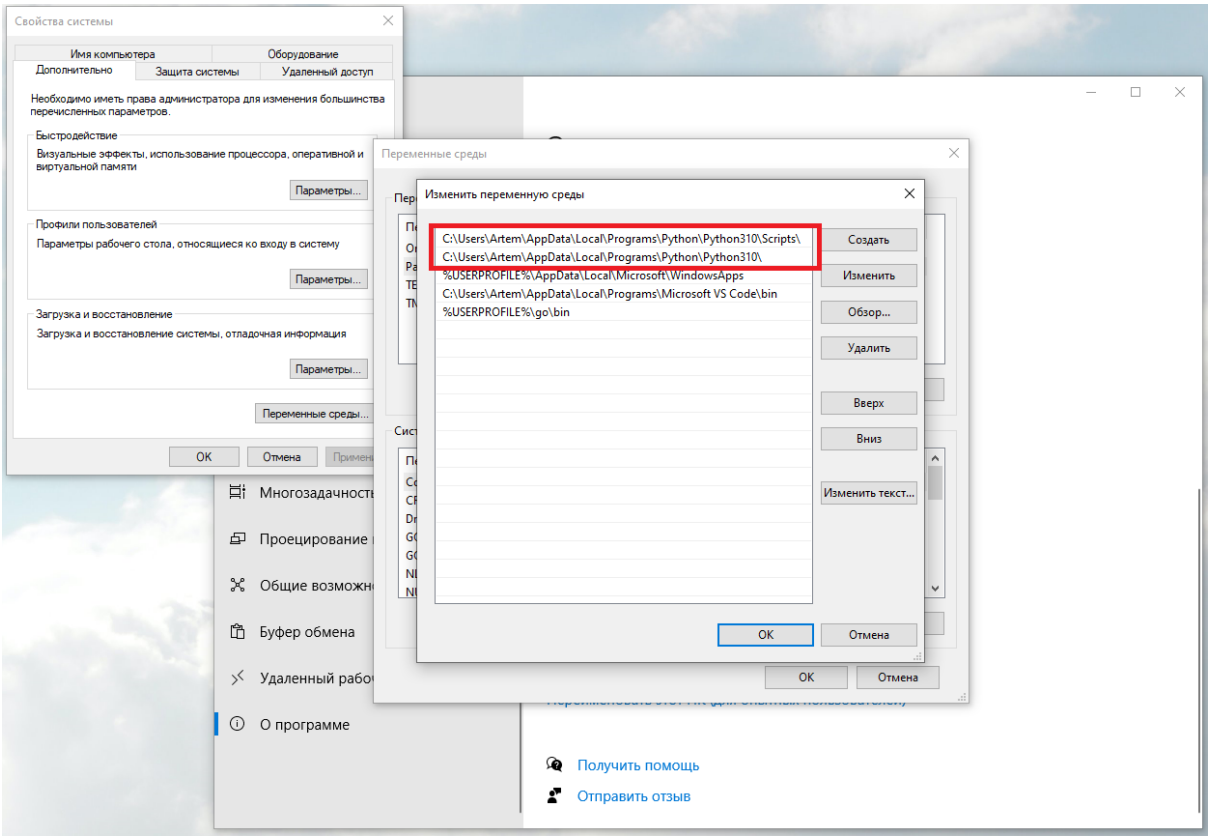


Рисунок 9. Редактирование переменной Path.

Перед редактированием сначала убедиться в корректности указанных путей. Для этого можно найти указанные директории с помощью программы “Проводник”. Следует учитывать, что AppData является скрытой директорией, и сперва необходимо включить просмотр скрытых папок.

Установка IDE

Для написания исходного кода можно применять любой текстовый редактор, но обычно используют специальные “интегрированные среды разработки”, integrated development environment, то есть IDE. От простых редакторов IDE отличается тем, что предоставляет программисту не только редактор текста, но и подсветку синтаксиса, средства отладки, средства работы с системами контроля версий, и многое другое.
Так как эта инструкция предназначена для “новичков”, то и IDE будет использоваться начального уровня - Thonny.

Загрузка инсталлятора IDE Thonny

Загрузить инсталлятор Thonny можно на странице по адресу https://thonny.org/. На ней необходимо кликнуть по ссылке Windows на панели загрузки.

После клика по ссылке, которая выделена красным цветом на рисунке 1, начнётся загрузка файла инсталлятора Thonny. Загрузка выполняется по пути, указанному в настройках браузера. По-умолчанию это будет C:\Users_имя_пользователя_ОС_\Downloads.

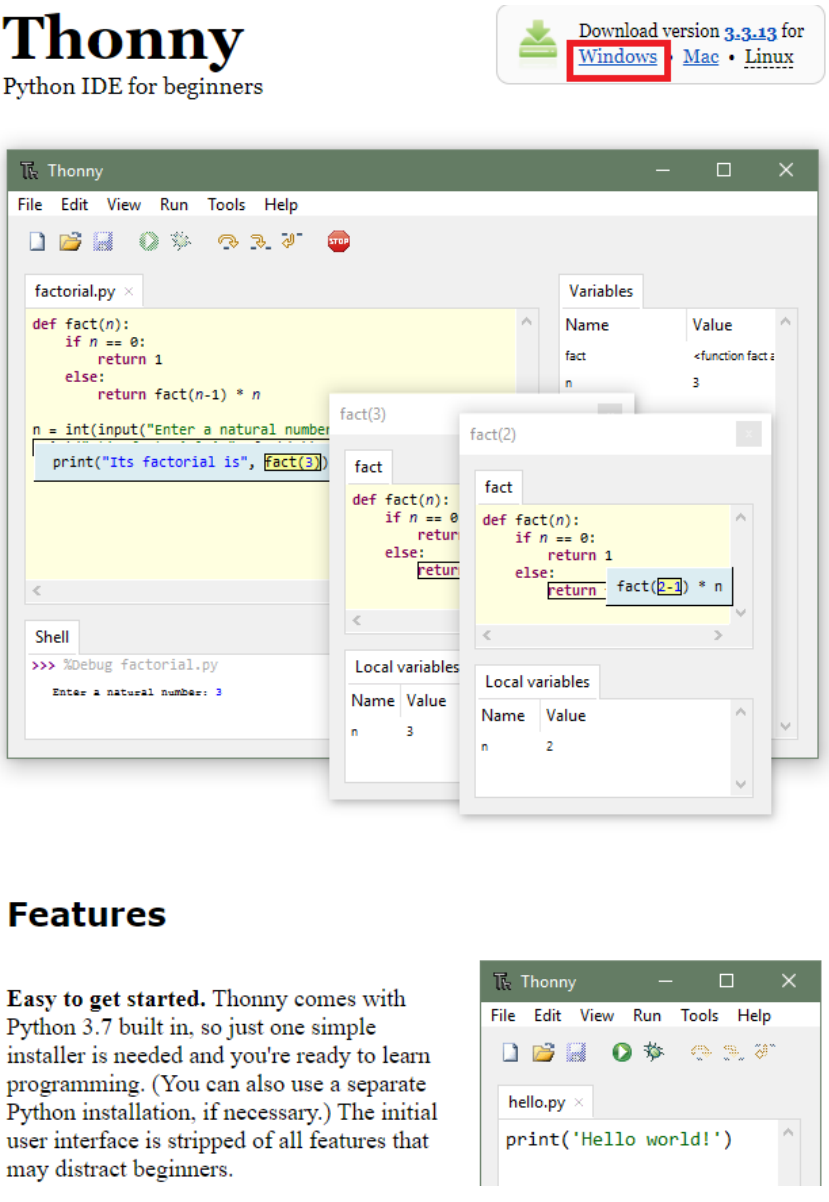


Рисунок 10. Страница загрузки IDE Thonny.

Установка IDE Thonny

После загрузки инсталлятора необходимо перейти в директорию, в которую был загружен файл, и двойным кликом левой кнопки мыши запустить процесс установки Thonny. В результате откроется первый экран окна процесса установки IDE (см. рисунок 11).

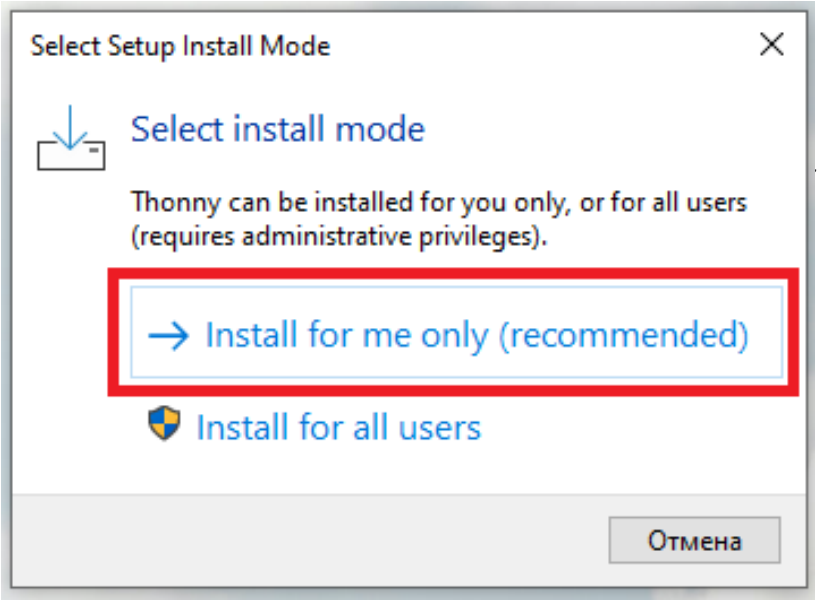


Рисунок 11. Стартовый экран окна установки IDE Thonny.

Thonny использует лицензию MIT, что означает свободное и полностью бесплатное использование данной программы.
На следующем экране, после подтверждения лицензионного соглашения, инсталлятор предлагает указать путь установки Thonny (см. рисунок 13).

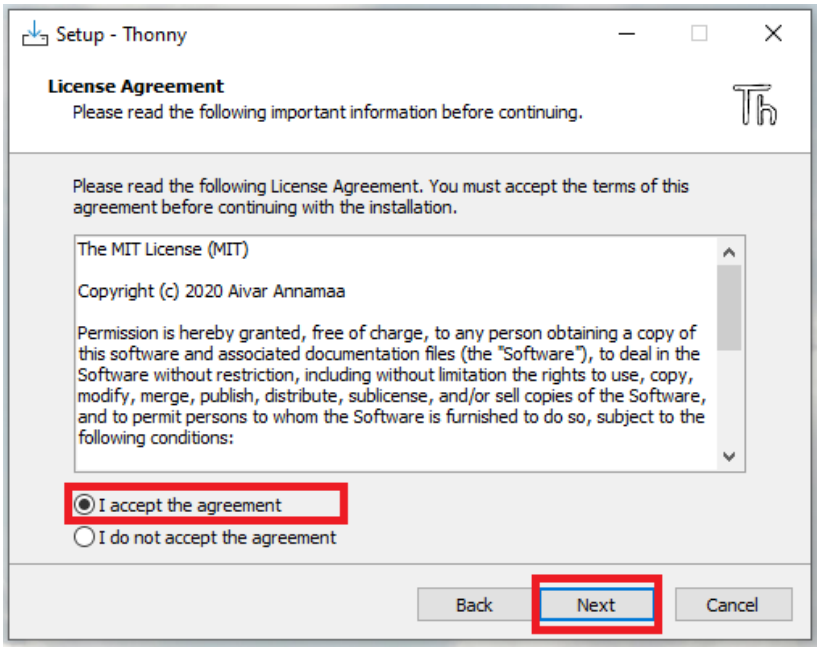


Рисунок 12. Лицензионное соглашение IDE Thonny

Рекомендуется оставить путь по-умолчанию, но при желании его можно изменить.

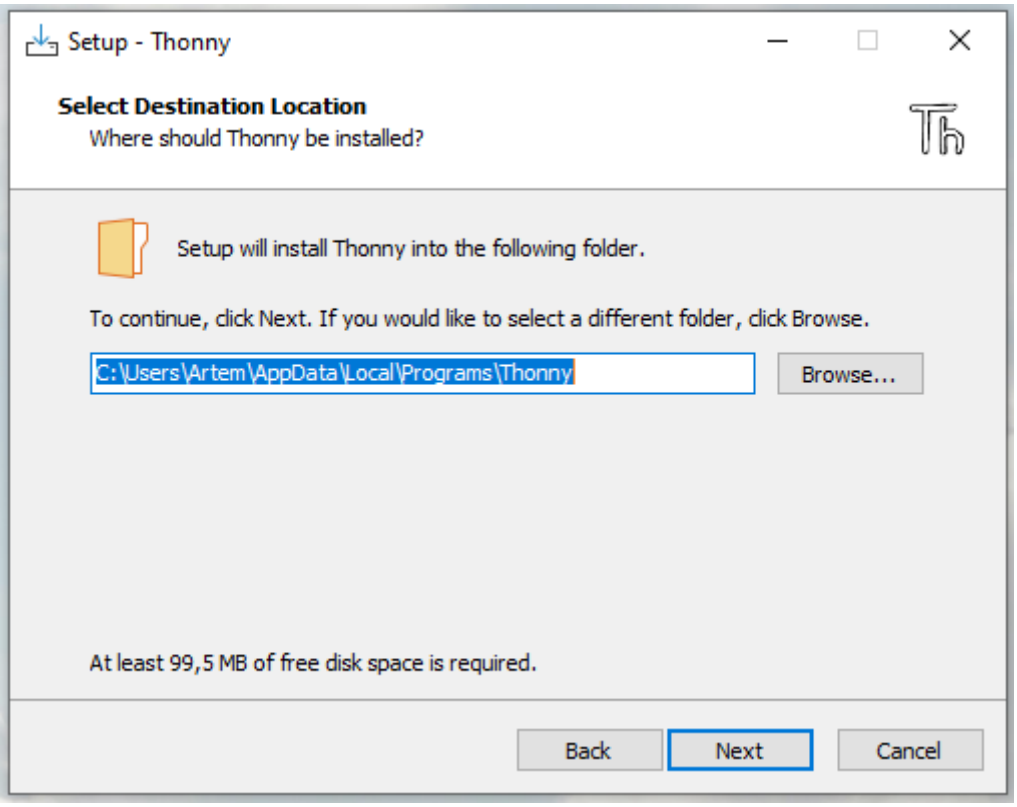


Рисунок 13. Экран определения пути установки Thonny.

На следующем экране выбора названия для пункта в меню “Пуск” необходимо всё оставить без изменений, и нажать кнопку “Next”.

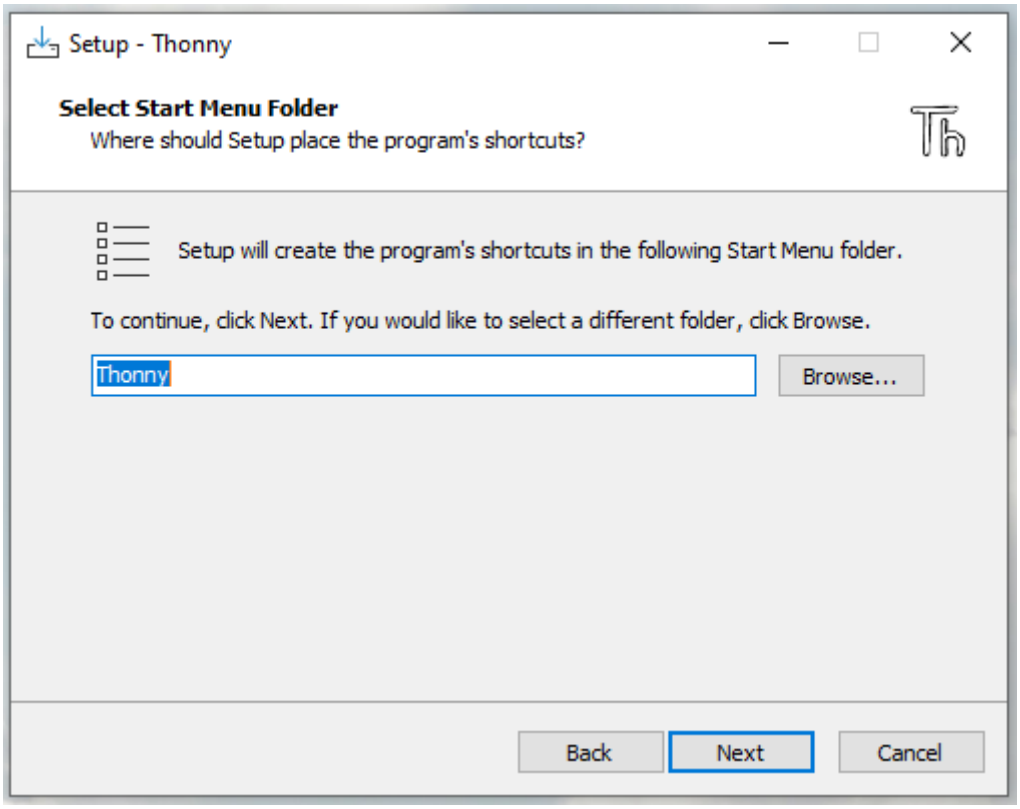


Рисунок 14. Выбор названия для пункта меню “Пуск”

На экране подтверждения создания ярлыка на рабочем столе рекомендуется установить флаг “Create desktop icon” (см. рисунок 15).

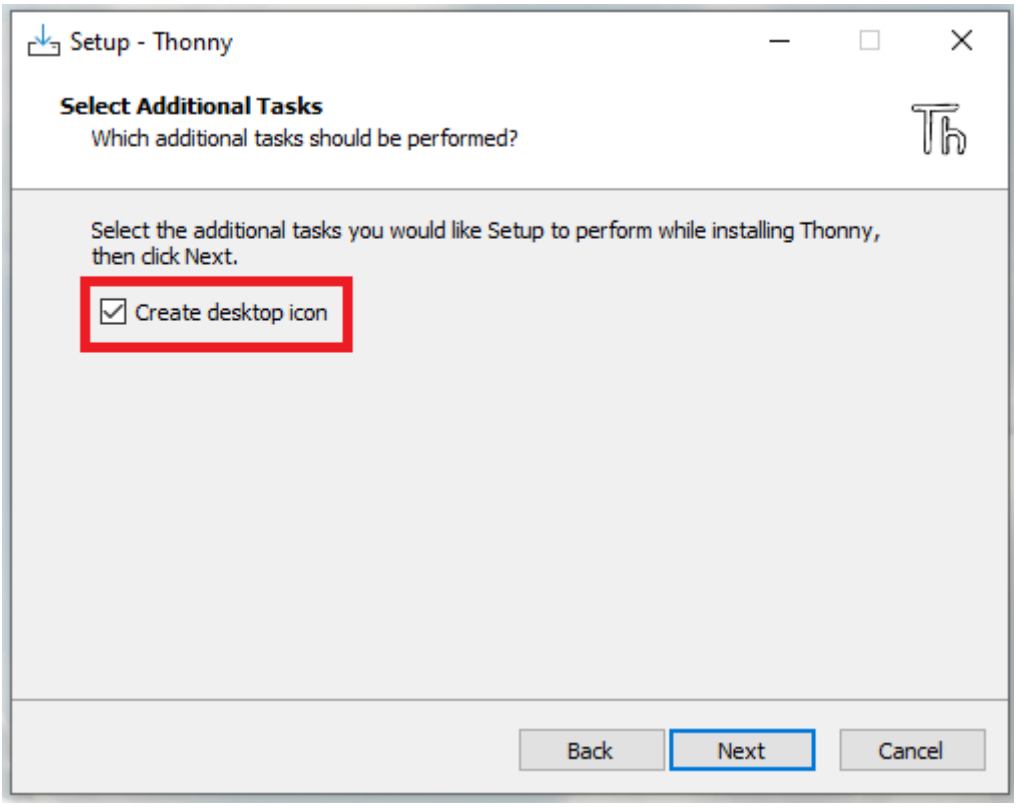


Рисунок 15. Подтверждение создания ярлыка на рабочем столе

На последнем экране подтверждения параметров инсталляции (см. рисунок 16) необходимо убедиться в корректности путей установки, и нажать кнопку “Install”.

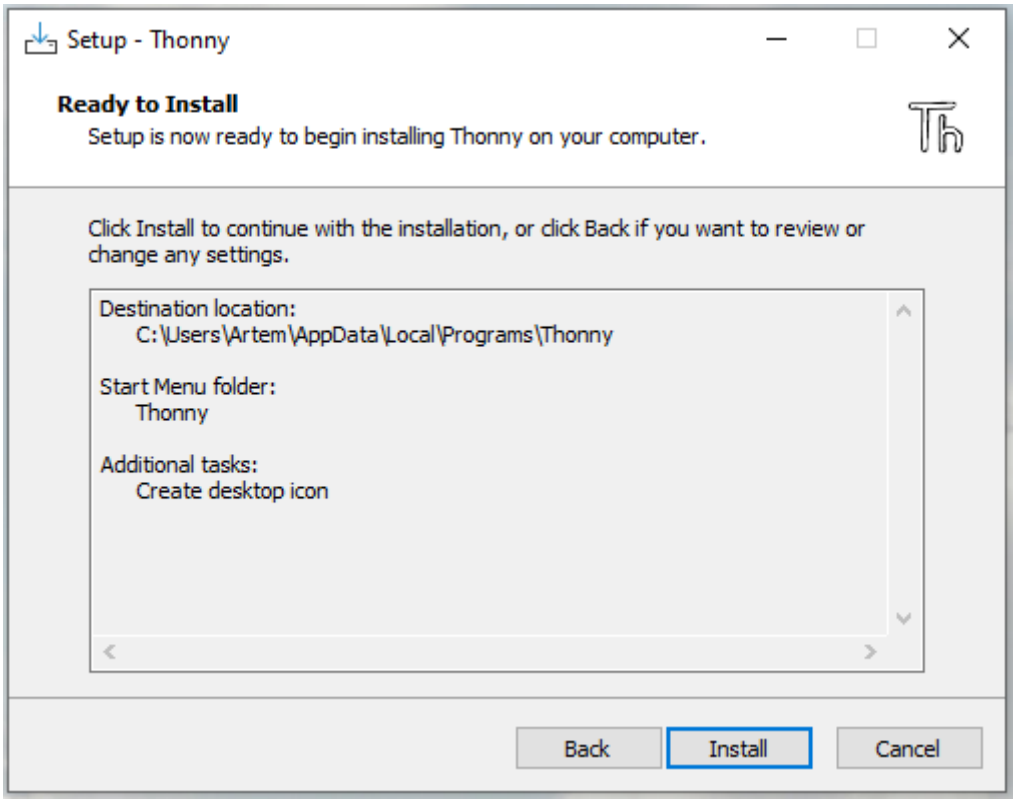


Рисунок 16. Подтверждение настроек инсталляции

После подтверждения настроек инсталляции начнётся процесс установки Thonny, по завершении которого будет выведен экран с рисунка 17.

Рисунок 17. Установка Thonny завершена

Настройка IDE Thonny

При первом запуске IDE Thonny пользователю предлагается выполнить первоначальную настройку программы (см. рисунок 18).

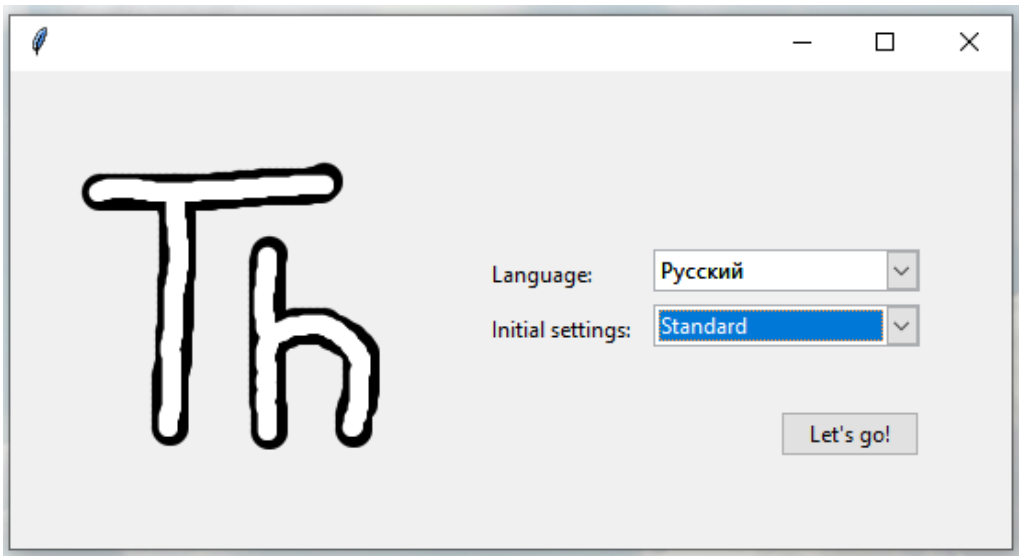


Рисунок 18. Первоначальная настройка IDE Thonny

На этом окне можно выбрать язык интерфейса IDE и задать шаблон настроек для среды IDE. Необходимо установить параметр Initial settings в положение Standart. Язык можно выбрать в соответствие со своими предпочтениями.

Настройка интерпретатора, который используется Thonny

Thonny поставляется с собственной версией интерпретатора Python, который используется по-умолчанию. Эта версия является 32-разрядной. В примерах используется последняя версия 64-разрядного интерпретатора, поэтому следует изменить изменить версию используемого Thonny интерпретатор через настройки (см. рисунок 19).

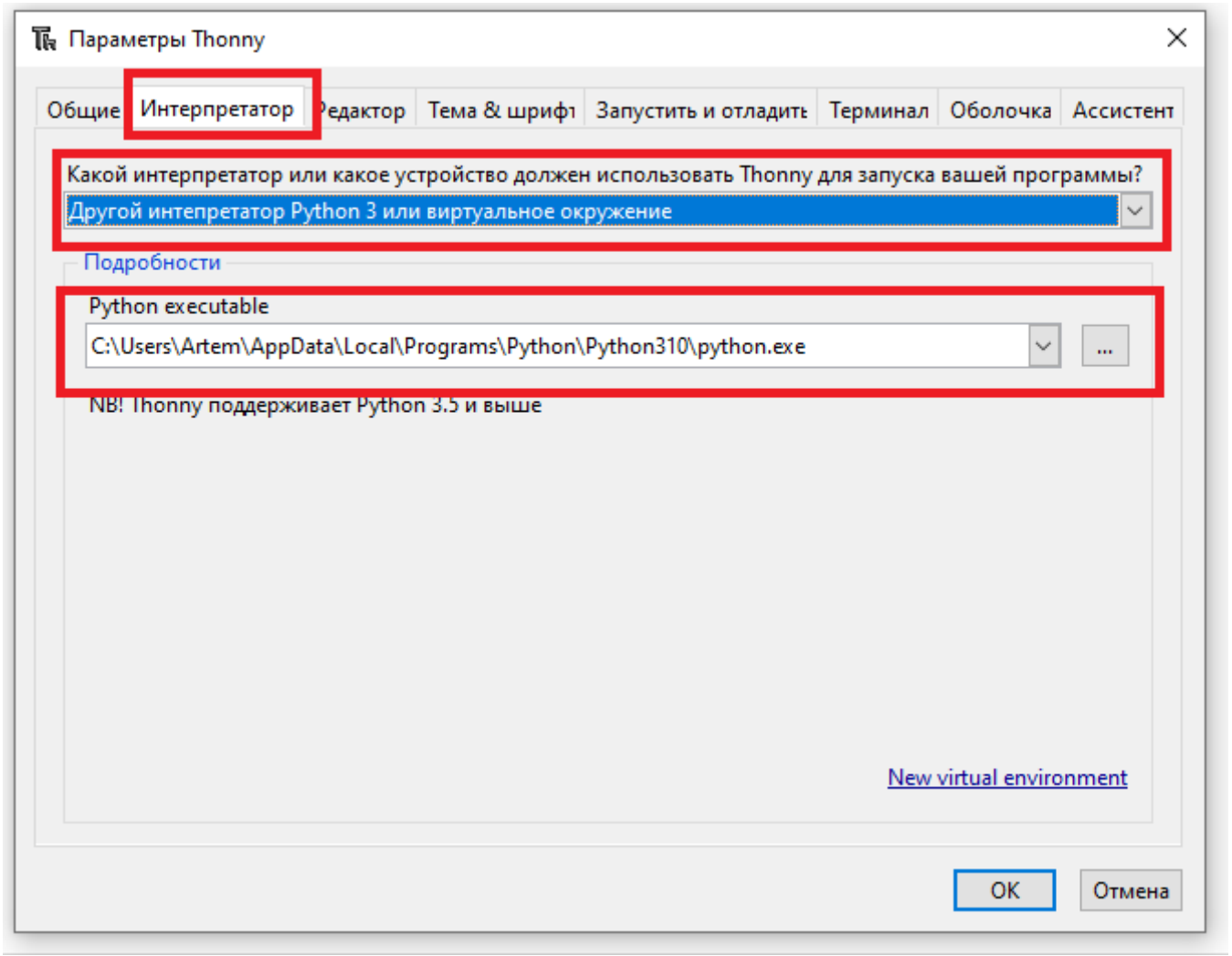


Рисунок 19. Настройка интерпретатора, который используется Thonny

Через пункт меню “Инструменты” необходимо открыть окно “Параметры”, и перейти на вкладку “Интерпретатор”. На этой вкладке необходимо в выпадающем списке выбрать вариант “Другой интерпретатор Python 3 или виртуальное окружение”, и в появившемся поле ввода ввести путь к исполняемому файлу Python. По-умолчанию это
C:\Users_имя_пользователя_ОС_\AppData\Local\Programs\Python\Python310\python.exe
Если переменные окружения ОС настроены правильно, это значение подставится автоматически.

Интерфейс IDE Thonny

В этом разделе приведена краткая справка по элементам графического интерфейса Thonny (см. рисунок 20).

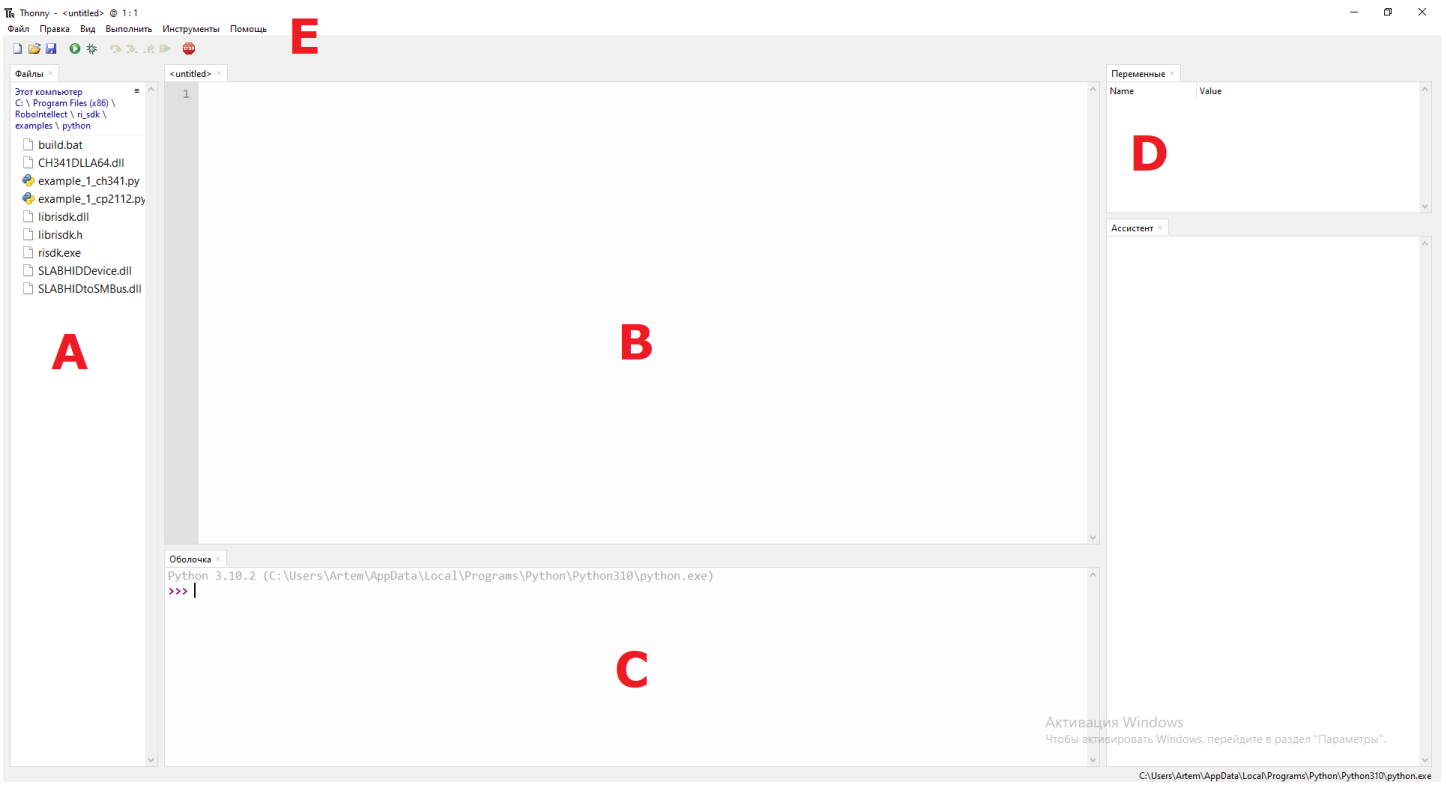


Рисунок 20. Графический интерфейс IDE Thonny

Слева находится панель, на которой выводится дерево файловой системы (A), через которое можно удобно открывать файлы исходного кода. Предварительно эту панель необходимо включить с помощью пункта меню “Вид”.

Меню программы размещено сверху основного интерфейса (E). Меню позволяет настраивать внешний вид IDE, работать с файловой системой, и предоставляет доступ к инструментам редактирования и отладки кода.

Посередине размещается область редактирования исходного кода (B). Весь исходный код программы будет выводиться на этой панели с включенной подсветкой синтаксиса.

Внизу размещается панель консоли интерпретатора (C). С помощью неё можно выполнять отдельные команды языка Python, и наблюдать ход их выполнения. Кроме того, на нижней панели можно выводить интерфейсы дополнительных инструментов IDE, которые включаются с помощью пункта меню “Вид”.

Справа находится навигатор по переменным (D), объявленным в теле исходного кода программы.

Проверка работоспособности окружения Python (helloworld)

Для проверки работоспособности всего окружения рекомендуется в панели консоли Python ввести примитивную команду, и выполнить её, нажав enter после ввода. Такой командой является функция print() (см. рисунок 21).

Внимание! Версия Python и IDE Thonny, указанные в этой инструкции или приведенные на рисунках, могут отличаться от версий ПО, установленных на Вашем компьютере.

Рисунок 21. Вывод функции print(“Hello world”)

Если после нажатия клавиши enter был выведен текст Hello world! в консоли, значит интерпретатор и IDE настроены корректно, и окружение пригодно для написания программ.

Установка RI SDK

Высокоуровневое программирование робототехнических устройств возможно благодаря использованию расширяемой библиотеки RI SDK.
RI SDK предоставляет простой и понятный программный интерфейс (API) для функционального управления компонентами робота. API включает в себя функции для управления сервоприводами (вращение, поворот, и т.д.), светодиодами (мигание, мерцание, и пр.), и датчиком тока (чтение текущих электрических характеристик РТУ).
RI SDK устанавливается вместе с программным обеспечением робота, которое можно скачать по ссылке.

Установка программного обеспечения роборуки

Установить ПО для роборуки, включая RI SDK можно, пройдя по ссылке https://robointellect.ru/models (см. рисунок 22).

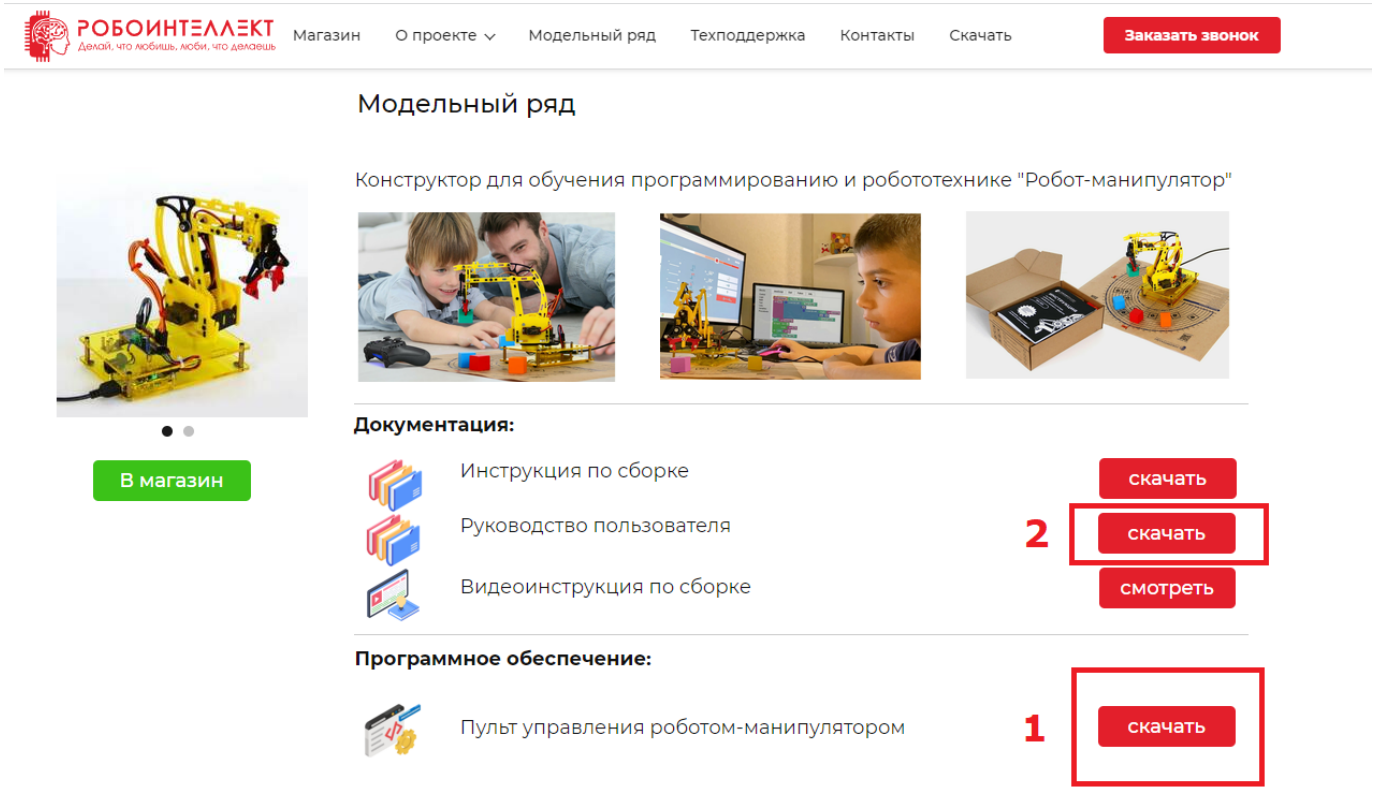


Рисунок 22. Страница для загрузки ПО роборуки и документации к нему

Для этого необходимо кликнуть по кнопке “Скачать” напротив раздела “Пульт управления роботом-манипулятором” (1). После клика по кнопке, начнётся загрузка файла инсталлятора ПО роборуки. Загрузка выполняется по пути, указанному в настройках браузера. По-умолчанию это будет C:\Users_имя_пользователя_ОС_\Downloads.
Инструкция по установке ПО роборуки доступна при клике по кнопке “Скачать” напротив раздела “Руководство пользователя”, раздел “5. Установка программного обеспечения для управления роботом”.

Запуск примеров готовых программ

В составе ПО роборуки поставляется примеры готовых программ на различных языках, в том числе на Python. Это программы размещены в директории C:\Program Files (x86)\RoboIntellect\ri_sdk\examples (см. рисунок 23).

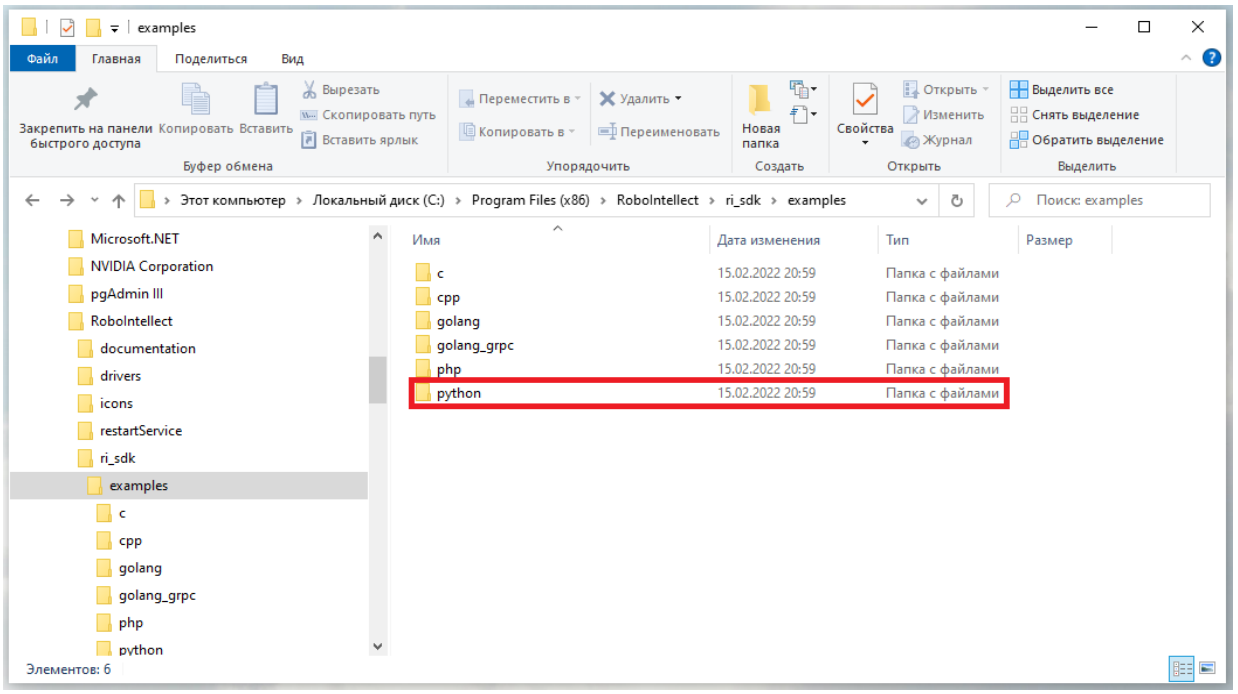


Рисунок 23. Директория с примерами программ на различных языках

Сама расширяемая библиотека хранится по адресу C:\Program Files (x86)\RoboIntellect\ri_sdk. Версия для 32 разрядной Windows лежит C:\Program Files (x86)\RoboIntellect\ri_sdk\ri_sdk_x86, а версия для 64 разрядной в C:\Program Files (x86)\RoboIntellect\ri_sdk\ri_sdk_x64.

Внимание! Для запуска примера программы на Python необходимо скопировать содержимое директории с RI SDK соответствующей разрядности в директорию C:\Program Files (x86)\RoboIntellect\ri_sdk\examples\python (см. рисунок 24).

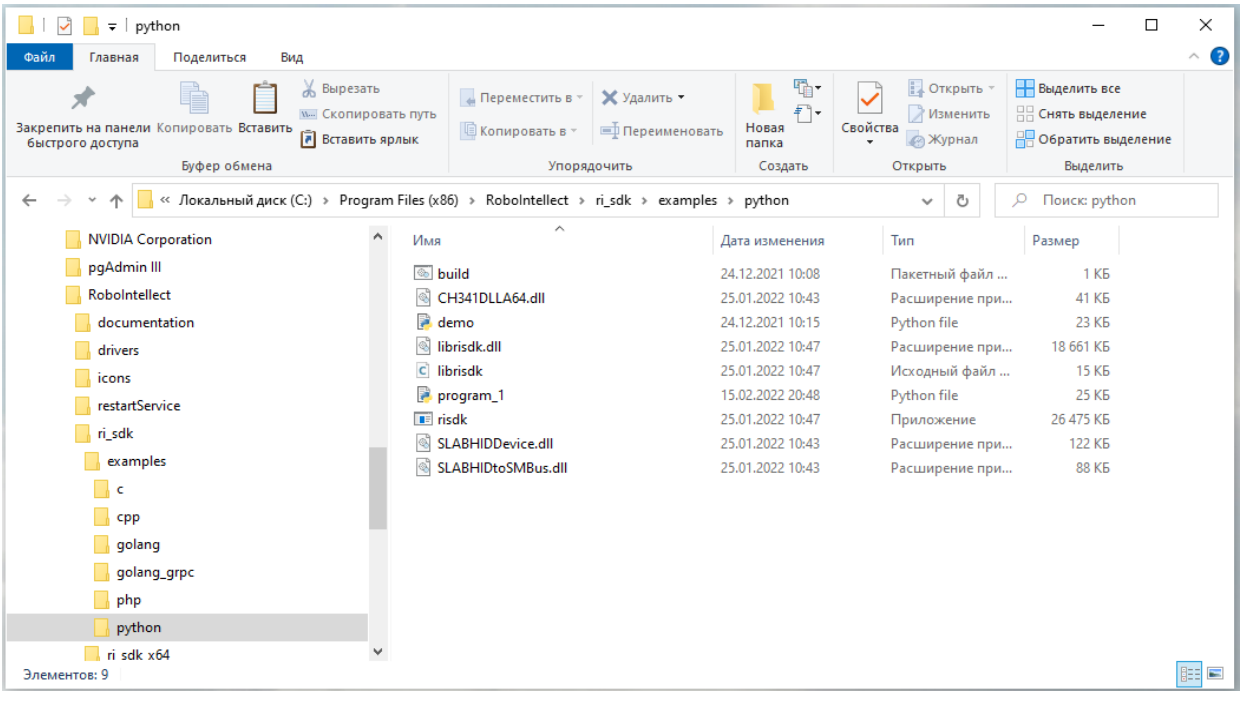
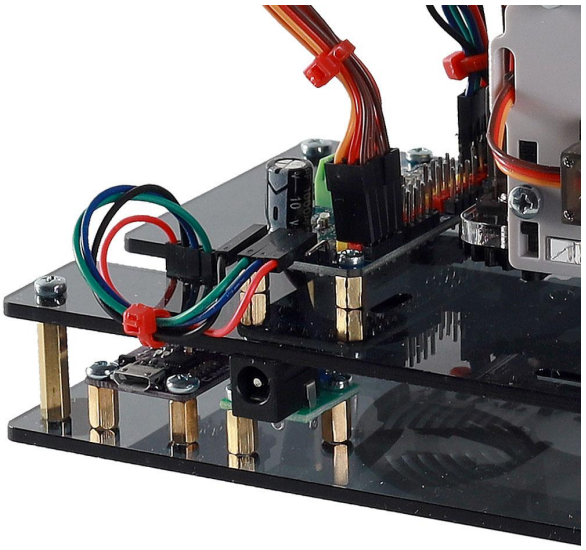


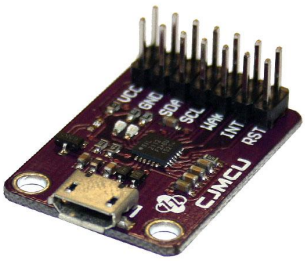
Рисунок 24. Содержимое директории с примером на Python после копирования библиотеки

Запуск программы - пример №1 (не helloworld)

Для запуска готового примера программы необходимо выполнить несколько шагов. Во-первых, необходимо включить роборуку в электрическую цепь, и затем подключить её к компьютеру через любой USB порт.
Во-вторых, необходимо открыть исходный код готового примера через IDE Thonny, используя проводник ОС, или навигатор интерфейса самой IDE. Исходный код примера на Python - это файл C:\Program Files (x86)\RoboIntellect\ri_sdk\examples\python\example_1. Обратите внимание на то, что в разных моделях роботов могут использоваться разные модели контроллеров, и поэтому необходимо задавать разные параметры инициализации I2C адаптеров - cp2112, если у вас робот RM001 и ch341, если у вас робот RM001M2. По ссылке доступно видео с описанием процесса запуска.

Как понять, какая модель I2C адаптера используется в Вашей модели роборуки? Если Ваш робот модели «РобоИнтеллект» RM 001 (четыре степени свободы) собран из конструктора (жёлтого, серого, белого цвета), то в нём используется преобразователь интерфейса USB – I2C на основе микросхемы CP2112.





Преобразователь интерфейса USB в I2C cp2112 с разъёмом microUSB

Рисунок 25. Адаптер I2C cp2112

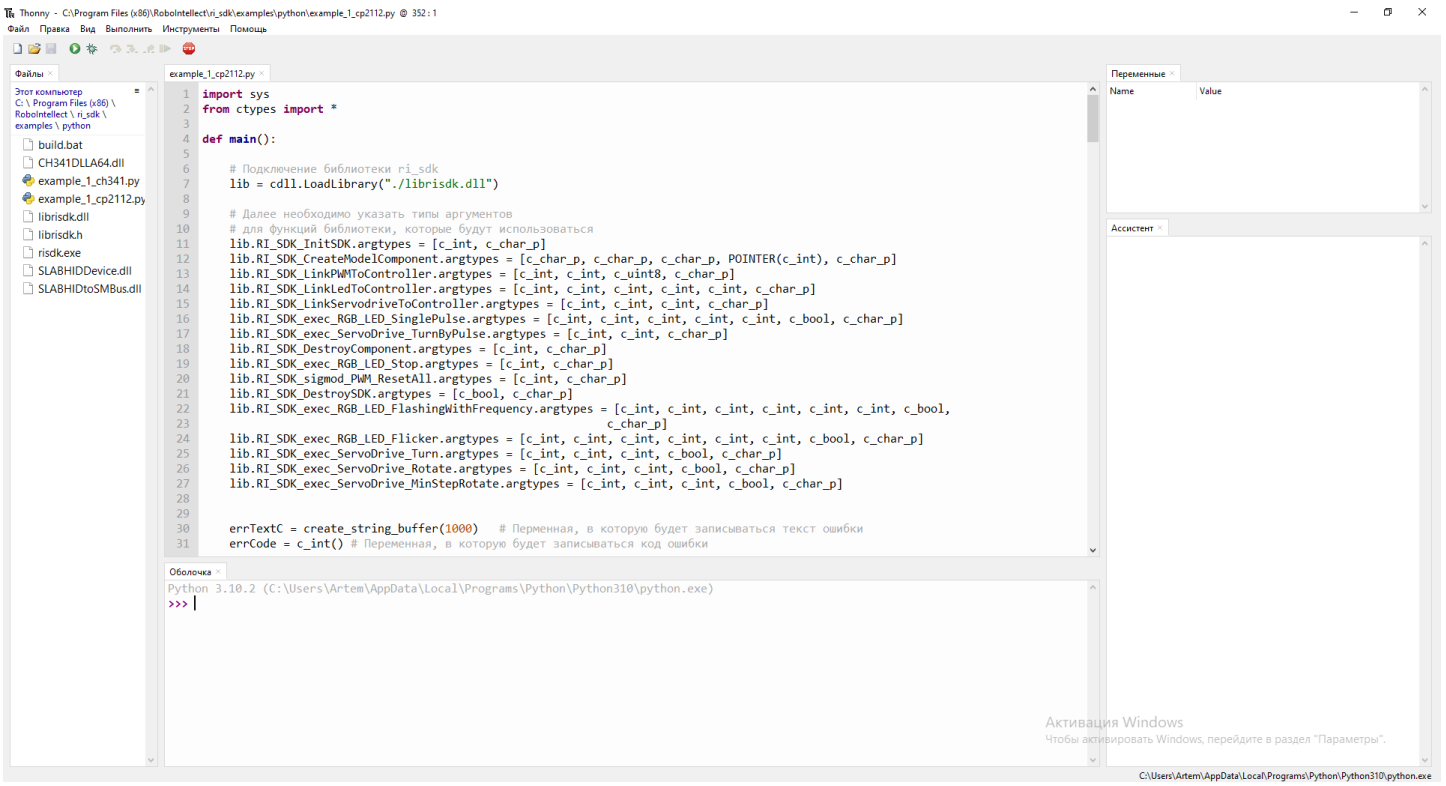


Рисунок 26. Интерфейс Thonny после открытия примера готовой программы на Python

И, наконец, после открытия исходного кода примера можно выбрать в меню пункт “Выполнить текущий скрипт” (горячая клавиша F5), что приведёт к выполнению примера программы.

Внимание! Особенность работы I2C адаптера CH341 заключается в том, что он не поддерживает одновременную работу нескольких подключений. Поэтому, перед выполнением примеров или собственных программ необходимо отключить приложение “Пульт управления РТУ”. Для этого нужно в меню “Пуск” найти папку RoboIntellect, и кликнуть по ярлыку “Остановка пульта управления РТУ”. После этого можно выполнить пример. Если не отключить предварительно пульт, то пример выполнится с ошибкой. В данный момент ведутся работы по обеспечению одновременного программного подключения по I2C для адаптеров ch341.

Рекомендуется внимательно ознакомиться с комментариями в исходном коде примера. Давайте кратко разберем, как работает данный пример программы.

1. Программа начинается с подключения библиотеки RI SDK и объявления типов аргументов функций RI SDK, которые будут вызываться в примере.

```python
import sys
from ctypes import *

def main():

    # Подключение библиотеки ri_sdk
    lib = cdll.LoadLibrary("./librisdk.dll")

    # Далее необходимо указать типы аргументов
    # для функций библиотеки, которые будут использоваться
    lib.RI_SDK_InitSDK.argtypes = [c_int, c_char_p]
    lib.RI_SDK_CreateModelComponent.argtypes = [c_char_p, c_char_p, c_char_p, POINTER(c_int), c_char_p]
    lib.RI_SDK_LinkPWMToController.argtypes = [c_int, c_int, c_uint8, c_char_p]
    lib.RI_SDK_LinkLedToController.argtypes = [c_int, c_int, c_int, c_int, c_int, c_char_p]
    lib.RI_SDK_LinkServodriveToController.argtypes = [c_int, c_int, c_int, c_char_p]
    lib.RI_SDK_exec_RGB_LED_SinglePulse.argtypes = [c_int, c_int, c_int, c_int, c_int, c_bool, c_char_p]
    lib.RI_SDK_exec_ServoDrive_TurnByPulse.argtypes = [c_int, c_int, c_char_p]
    lib.RI_SDK_DestroyComponent.argtypes = [c_int, c_char_p]
    lib.RI_SDK_exec_RGB_LED_Stop.argtypes = [c_int, c_char_p]
    lib.RI_SDK_sigmod_PWM_ResetAll.argtypes = [c_int, c_char_p]
    lib.RI_SDK_DestroySDK.argtypes = [c_bool, c_char_p]
    lib.RI_SDK_exec_RGB_LED_FlashingWithFrequency.argtypes = [c_int, c_int, c_int, c_int, c_int, c_int, c_bool,
                                                              c_char_p]
    lib.RI_SDK_exec_RGB_LED_Flicker.argtypes = [c_int, c_int, c_int, c_int, c_int, c_int, c_bool, c_char_p]
    lib.RI_SDK_exec_ServoDrive_Turn.argtypes = [c_int, c_int, c_int, c_bool, c_char_p]
    lib.RI_SDK_exec_ServoDrive_Rotate.argtypes = [c_int, c_int, c_int, c_bool, c_char_p]
    lib.RI_SDK_exec_ServoDrive_MinStepRotate.argtypes = [c_int, c_int, c_int, c_bool, c_char_p]

```

2. Затем инициализируются объекты I2C адаптера и ШИМ, которые связываются друг с другом.

```python
    errCode = lib.RI_SDK_CreateModelComponent("connector".encode(), "i2c_adapter".encode(), "cp2112".encode(),
                                              i2c, errTextC)
    if errCode != 0:
        print(errCode, errTextC.raw.decode())
        sys.exit(2)

```

3. После этого инициализируются объекты сервоприводов модели mg90s, которые связываются с ШИМ на указанных портах. Аналогичные действия выполняются для светодиода. Ниже приведён пример инициализации компонента.

```python
    errCode = lib.RI_SDK_CreateModelComponent("executor".encode(), "servodrive".encode(), "mg90s".encode(),
                                              servo_1, errTextC)
    if errCode != 0:
        print(errCode, errTextC.raw.decode())
        sys.exit(2)
```

4. После выполнения инициализации всех компонентов, поочередно вызываются функции сервоприводов - поворот, вращение, поворот на минимальный шаг, поворот с заданным импульсом. Ниже приведен пример поворота сервопривода на заданный угол. **Особого внимания заслуживает параметр ряда функций async (на рисунке ниже в функцию передаётся значение False). Он отвечает за асинхронное выполнение вызываемой функции. Это позволяет выполнять несколько действий одновременно и параллельно для нескольких компонентов. Например, одновременно вращать несколько сервоприводов. В первом примере все действия выполняются синхронно для упрощения понимания.**

```python
    errCode = lib.RI_SDK_exec_ServoDrive_Turn(servo_1, 90, 100, c_bool(False), errTextC)
    if errCode != 0:
        print(errCode, errTextC.raw.decode())
        sys.exit(2)
```

5. Для светодиода выполняются функции мигания, мерцания, и свечения заданным цветом.

```
261     errCode = lib.RI_SDK_exec_RGB_LED_Flicker(led, 0, 255, 0, 500, 10, c_bool(False), errTextC)
262     if errCode != 0:
263         print(errCode, errTextC.raw.decode())
264         sys.exit(2)
```

6. После этого демонстрация завершается, и необходимо выполнить корректное завершение программы. Для этого сбрасываются сигналы со всех портов ШИМ, затем выполняется поочередное удаление всех ранее созданных компонентов для очистки памяти. На примере ниже показан вызов функции сброса сигналов с портов ШИМ и освобождение памяти, выделенной для компонента сервопривода.

```
288     errCode = lib.RI_SDK_sigmod_PWM_ResetAll(pwm, errTextC)
289     if errCode != 0:
290         print(errCode, errTextC.raw.decode())
291         sys.exit(2)
292
293     # Для удаления компонентов можно использовать метод RI_SDK_DestroyComponent
294     # В него достаточно передать дескрипотор компонента, который необходимо удалить
295
296     # Удаление сервоприводов
297     errCode = lib.RI_SDK_DestroyComponent(servo_1, errTextC)
298     if errCode != 0:
299         print(errCode, errTextC.raw.decode())
300         sys.exit(2)
```

7. И, наконец, выполняется удаление внутренних системных компонентов библиотеки RI SDK.

```
342     errCode = lib.RI_SDK_DestroySDK(c_bool(True), errTextC)
343     if errCode != 0:
344         print(errCode, errTextC.raw.decode())
345         sys.exit(2)
346
347     print("Success")
```

При выполнении примеров и собственных программ могут возникать различные ошибки. Ниже приведены некоторые из них:
130001 error in binding PWM to adapter: Connection creation error: Error creating I2C connection: Operation HidSmbus_Open failed - ошибка возникает, если запускать программу при включенном пульте управления. Решение - временно отключить пульт управления (см. предупреждение выше в этом разделе). Кроме того, такая ошибка возникает, если робот не подключен к компьютеру, или I2C адаптер неправильно подключен к ШИМ.

Запуск программы, написанной с помощью функций - пример №2.

На примере №2 ознакомимся с концепцией использования функций в различных языках программирования, и в том числе в Python.

Функция в программировании - это объявленный с помощью специального синтаксиса фрагмент исходного кода, который можно неоднократно использовать в различных участках программы с помощью оператора вызова функции. Это позволяет, во-первых, избежать дублирования кода, а во-вторых лучшим образом структурировать программу для упрощения написания кода и его последующей отладки. Функции - это "кирпичи" из которых строятся большинство программ. Более того, механизм импорта модулей Python позволяет функции, написанные для одной программы, использовать в других программах, где требуется схожая функциональность. Для объявления функции применяется оператор def, после которого следует сигнатура функции.
Сигнатура функции - это сочетание имени функции, и аргументов, которые она принимает. Если к сигнатуре функции приписать оператор вызова, то есть круглые скобки "()" сразу после имени

функции, то произойдет вызов функции, то есть она выполнится. При этом в круглых скобках передаются параметры функции, которые внутри тела функции доступны по именам аргументов, которые описаны в сигнатуре.

Для демонстрации необходимо открыть файл с исходным кодом C:\Program Files (x86)\RoboIntellect\ri_sdk\examples\python\example_2_1_cp2112.py. По ссылке доступно видео с описанием процесса запуска. В этом примере сначала объявляются необходимые для работы переменные, затем инициализируются RI SDK и ряд компонентов, после чего выполняется движение сервопривода башни из стороны в сторону. В завершение выполняется освобождение выделенной для компонентов из этого примера памяти. Для реализации движения из стороны в сторону используется функция RI_SDK_exec_ServoDrive_Turn(), которая принимает в качестве параметров дескриптор компонента сервопривода, угол поворота, скорость вращения, и признак асинхронного выполнения (см. рисунок ниже).

```
54      # Поворот сервопривода на 90 градусов.
55      errCode = lib.RI_SDK_exec_ServoDrive_Turn(servo_1, 90, 100, c_bool(False), errTextC)
56      if errCode != 0:
57          print(errCode, errTextC.raw.decode())
58          sys.exit(2)
59      # Поворот сервопривода на 90 градусов в противоположную сторону.
60      errCode = lib.RI_SDK_exec_ServoDrive_Turn(servo_1, -90, 100, c_bool(False), errTextC)
61      if errCode != 0:
62          print(errCode, errTextC.raw.decode())
63          sys.exit(2)
64      # Поворот сервопривода на 90 градусов.
65      errCode = lib.RI_SDK_exec_ServoDrive_Turn(servo_1, 90, 100, c_bool(False), errTextC)
66      if errCode != 0:
67          print(errCode, errTextC.raw.decode())
68          sys.exit(2)
69      # Поворот сервопривода на 90 градусов в противоположную сторону.
70      errCode = lib.RI_SDK_exec_ServoDrive_Turn(servo_1, -90, 100, c_bool(False), errTextC)
71      if errCode != 0:
72          print(errCode, errTextC.raw.decode())
73          sys.exit(2)
74      # Поворот сервопривода на 90 градусов.
75      errCode = lib.RI_SDK_exec_ServoDrive_Turn(servo_1, 90, 100, c_bool(False), errTextC)
76      if errCode != 0:
77          print(errCode, errTextC.raw.decode())
78          sys.exit(2)
```

После вызова функции необходимо выполнить проверку ошибки, чтобы удостовериться в корректном выполнении действия. Так как поворот выполняется несколько раз, программист вынужден дублировать код проверки ошибки.

Дублирования кода можно избежать, если перенести код вызова RI SDK и обработку ошибок в отдельную функцию. Рассмотрим пример C:\Program Files (x86)\RoboIntellect\ri_sdk\examples\python\example_2_2_cp2112.py (или C:\Program Files (x86)\RoboIntellect\ri_sdk\examples\python\example_2_2_ch341.py, если у вас RM001M2). В нём поворот сервопривода и обработка ошибки вынесены в отдельную функцию.

```
80  def turnServodrive(descriptor, angle, speed):
81      errCode = lib.RI_SDK_exec_ServoDrive_Turn(descriptor, angle, speed, c_bool(False), errTextC)
82      if errCode != 0:
83          print(errCode, errTextC.raw.decode())
84          sys.exit(2)
```

Это позволило сократить повороты башни из стороны в сторону всего до нескольких строчек кода:

```
90      # Поворот сервопривода на 90 градусов.
91      turnServodrive(servo_1, 90, 100)
92      # Поворот сервопривода на 90 градусов в противоположную сторону.
93      turnServodrive(servo_1, -90, 100)
94      # Поворот сервопривода на 90 градусов.
95      turnServodrive(servo_1, 90, 100)
96      # Поворот сервопривода на 90 градусов в противоположную сторону.
97      turnServodrive(servo_1, -90, 100)
98      # Поворот сервопривода на 90 градусов.
99      turnServodrive(servo_1, 90, 100)
```

Кроме того, теперь при внесении изменений в логику, например, обработки ошибок, изменения необходимо внести один раз, в многократно вызываемой функции, а не столько раз, сколько реально выполняется поворот сервопривода.

Запуск более сложной программы - пример №3.

Для запуска этой программы необходимо открыть исходный код готового примера через IDE Thonny, используя проводник ОС, или навигатор интерфейса самой IDE. Исходный код примера на Python - это файл C:\Program Files (x86)\RoboIntellect\ri_sdk\examples\python\example_3_cp2112.py. По ссылке доступно видео с описанием процесса запуска.

Третий пример программы для роборуки на Python является более сложным, так как является интерактивной программой, то есть подразумевается взаимодействие с пользователем для управления устройством. Кроме того, программа структурирована особым образом с помощью функций.

В программе №3 есть пример рекурсии. Рекурсия - это когда функция в процессе выполнения вызывает саму себя. Рекурсия применяется для решения некоторых задач, частый пример - вычисление факториала.

В примере, который приводится в этом разделе “сплошной” код работы с компонентами устройства разбит на несколько функций, что позволило сократить объем кода, и повторно использовать однажды написанный код. Например, в примере №1 много раз встречается вызов функции создания компонента (см. рисунок ниже).

```
102     errCode = lib.RI_SDK_CreateModelComponent("executor".encode(), "servodrive".encode(), "mg90s".encode(),
103                                               servo_2, errTextC)
104     if errCode != 0:
105         print(errCode, errTextC.raw.decode())
106         sys.exit(2)
```

В примере №3 этот код был вынесен в отдельную функцию, которая многократно вызывается для каждого компонента роборуки.

```
94  # createRISDKComponent() выполняет создание компонента заданных группы, типа устройства и модели.
95  # По-сути является "функцией-обёрткой" поверх функции RI SDK.
96  def createRISDKComponent(group, device, model, descriptor):
97      # Вызов функции RI SDK для создания компонента.
98      errCode = lib.RI_SDK_CreateModelComponent(group.encode(), device.encode(), model.encode(), descriptor, errTextC)
99      if errCode != 0:
100         print(errCode, errTextC.raw.decode())
101         sys.exit(2)
```

Это позволило, например, не писать проверку ошибки вызова функции RI SDK для каждого компонента в отдельности (см. рисунок ниже).

```
    # Инициализация компонента i2c адаптера ch341.
    createRISDKComponent("connector", "i2c_adapter", "ch341", i2c)
    # Инициализация компонента ШИМ pca9685 и связывание его с i2c адаптером.
    createRISDKComponent("connector", "pwm", "pca9685", pwm)
    linkPWMToController(pwm, i2c, c_uint8(0x40))
    # Инициализация компонента сервопривода mg90s (основание) и связывание его с ШИМ.
    createRISDKComponent("executor", "servodrive", "mg90s", servoBase)
    linkServoToController(servoBase, pwm, 0)
    # Инициализация компонента сервопривода mg90s (клешня) и связывание его с ШИМ.
    createRISDKComponent("executor", "servodrive", "mg90s", servoClaw)
    linkServoToController(servoClaw, pwm, 1)
    # Инициализация компонента сервопривода mg90s (первая стрела) и связывание его с ШИМ.
    createRISDKComponent("executor", "servodrive", "mg90s", servoHorizontal)
    linkServoToController(servoHorizontal, pwm, 2)
    # Инициализация компонента сервопривода mg90s (вторая стрела) и связывание его с ШИМ.
    createRISDKComponent("executor", "servodrive", "mg90s", servoVertical)
    linkServoToController(servoVertical, pwm, 3)
    # Инициализация компонента светодида ky016 и связывание его с ШИМ.
    createRISDKComponent("executor", "led", "ky016", led)
    linkLEDToController(led, pwm, 15, 14, 13)
```

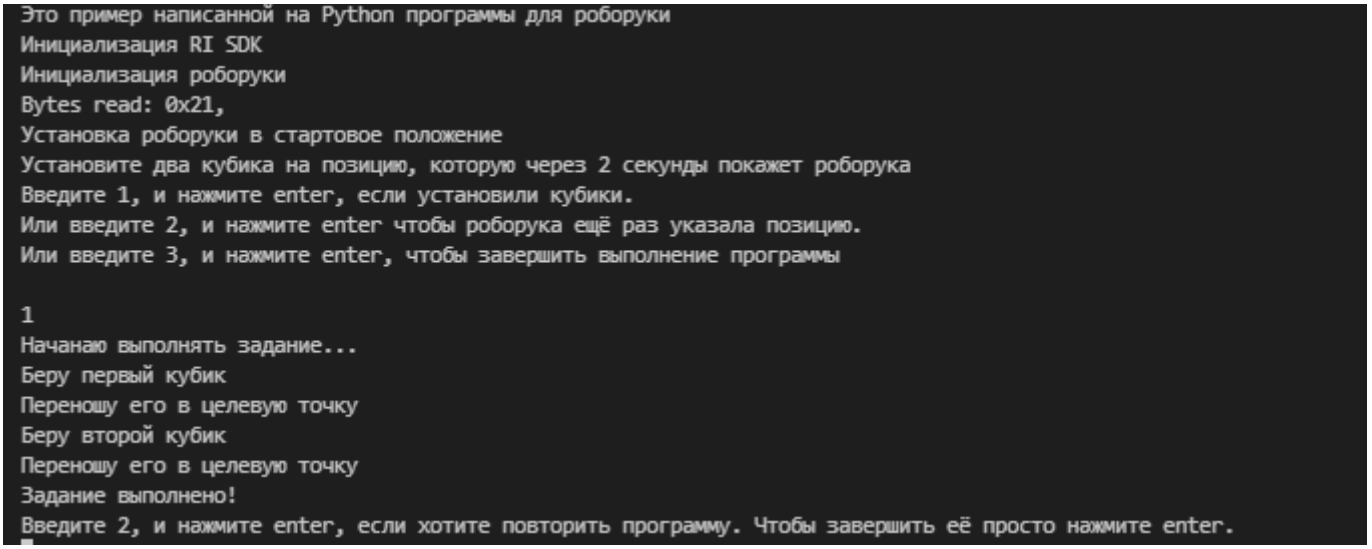
Программа из примера №3 состоит из трёх файлов исходного кода. В первом, example_3_cp2112.py хранится исходный код базового алгоритма программы, и содержит в себе точку входа в программу - функцию main().

```
import robohand
import constant

# main() - функция-"точка входа" в программах на Python и многих других языках.
def main():
    print("Это пример написанной на Python программы для роборуки")
    print("Инициализация RI SDK")
    # Инициализация RI SDK.
    robohand.initRISDK()
    print("Инициализация роборуки")
    # Инициализация компонентов роборуки.
    robohand.initRobohand()
    # Старт основного "рабочего цикла" роборуки для этого задания.
    workCycle()
    robohand.completeAndDestroy()

# workCycle() - функция "рабочего цикла" роборуки для этого задания.
def workCycle():
    # Запуск функции демонстрации исходной позиции кубиков.
    userChoise = robohand.showCubePosition(False)
    # Если пользователь выберет старт прорраммы, то...
    if userChoise == constant.USER_CHOISE_START:
        # ... то выполняется функция переноса кубиков в целевую позицию.
        robohand.runWork()
    # После завершения пользователю предлагается повторить упражнение целиком.
    userInstruction = 'Введите {0}, и нажмите enter, если хотите повторить программу. Чтобы завершить её просто нажмите enter.\n'.format(constant.USER_CHOISE_REPEAT)
    repeatWork = input(userInstruction)
    if repeatWork == constant.USER_CHOISE_REPEAT:
        # Рекурсивный вызов функции "рабочего цикла".
        return workCycle()
    else:
        # Или выход из функции "рабочего цикла".
        return

# Вызов функции-"точки входа" в программу.
main()
```

Все функции используемые в этом исходном коде объявлены в отдельном файле, robohand.py, который *импортируется в программу в качестве модуля*. *Модулем в Python является любой файл исходного кода, который подключается в программу с помощью оператора import. Модуль предоставляет все свои функции, переменные и объекты для повторного использования в программе.*

И, наконец, постоянные величины, используемые в example_3_cp2112.py и robohand.py, то есть так называемые *константы*, вынесены в третий файл - constants.py.

Важное отличие примера №3 - это наличие некоторой, пусть и примитивной, интерактивности. То есть программа взаимодействует с пользователем и ожидает принятия решения от него. Пользователь может сообщить программе свой выбор с помощью ввода управляющих символов в терминале (большинство IDE, в том числе и Thonny, интегрирует в себя терминал операционной системы).



Программа начинает свою работу с того, что задает роборуке исходное положение. Затем роборука демонстрирует точку, в которую необходимо установить кубики, поставляющиеся вместе с конструктором. Пользователь устанавливает два кубика друг на друга. Пользователь может ввести символ “1” с клавиатуры, и нажать enter, чтобы роборука начала перемещение кубиков в целевую точку, или ввести “2”, если хочет ещё раз увидеть, в какую позицию необходимо поставить кубики.
Программа выполняет перемещение кубиков в целевую точку, после чего “просит” пользователя выбрать, хочет ли он повторить упражнение, для чего нужно ввести символ “2”, или завершает свое выполнение, если пользователь просто нажал “enter”.
В процессе выполнения программа сигнализирует о выполняемых действиях с помощью светодиода. Он загорается красным, когда работают сервоприводы, и горит зелёным цветом, когда программа ожидает действия пользователя. Для того, чтобы свечение светодиода не препятствовало работе сервоприводов, функция свечения светодиода запускается в асинхронном режиме, то есть параметр async передаётся в функцию со значением True.

Краткое перечисление функций RI SDK

В таблице ниже перечислены функции внешнего API RI SDK. Более подробную информацию по функциям, параметрам вызова с примерами использования следует искать в документации на официальном сайте проекта - docs.robointellect.ru.

Базовые функции		
Сигнатура функции	**Описание работы функции**	**Описание параметров и возвращаемых значений**
RI_SDK_InitSDK(logLevel, errorText) : errorCode	Инициализация RI SDK. Инициализирует библиотеку RI SDK. Инициализацию необходимо выполнять в самом начале выполнения программы. Иначе остальные функции будут возвращать ошибку	***logLevel: c_int*** - Уровень глубины логирования (0 - только верхний уровень, 1, 2, 3 - более подробная трассировка) ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_CreateBasic(descriptor, errorText):errorCode	Создание базового компонента. Функция создает базовый компонент, который является абстракцией первого	***descriptor: c_int*** - Указатель на компонент, который будет создан ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит

	уровня над всеми остальными компонентами. Этот компонент не имеет своих собственных методов. Его можно расширить до компонента уровня группы.	ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_CreateGroupComponent(type, descriptor, errorText):errorCode	Создание компонента уровня группы. Создает и записывает в реестр библиотеки компонент уровня группы. Таких компонентов может быть три: 1. executor - исполнительный устройства (сервопривод, светодиод). 2. connector - устройства для связывания других устройств (i2c адаптер, ШИМ модулятор) 3. sensor - датчики (Датчик напряжения и силы тока) Такой компонент является абстракцией второго уровня над теми компонентами, которые входят в одну из 3-х групп. Такие компоненты реализуют общие методы компонентов группы. Компонент уровня группы может быть расширен до компонента устройства своей группы.	***type: c_char_p***- Тип компонента. Доступно: executor - исполнительный устройства (сервопривод, светодиод). connector - устройства для связывания других устройств (i2c адаптер, ШИМ модулятор) sensor - датчики (Датчик напряжения и силы тока) ***descriptor: c_int*** - Указатель на компонент, который будет создан ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_CreateDeviceComponent(entityName, deviceName, descriptor, errorText):errorCode	Создание компонента устройства. Создает и записывает в реестр библиотеки компонент уровня устройства. Это третий уровень абстракции. Такие компоненты реализуют методы управления своими устройствами. Компонент уровня устройства можно расширить до конкретной модели соответствующего устройства. Доступные устройства: 1. i2c_adapter - коннектор i2c 2. pwm - ШИМ 3. servodrive - сервопривод 4. led - светодиод Без этого расширения, использование методов управления устройством невозможно, так как именно модель содержит значения параметров, необходимых для управления.	***entityName: c_char_p***- Тип компонента. Доступно: executor - сервопривод. connector - rgb светодиод, ***deviceName: c_char_p***- Устройство компонента. Доступно: i2c_adapter - коннектор i2c pwm - ШИМ servodrive - сервопривод led - светодиод ***descriptor: c_int*** - Указатель на компонент, который будет создан ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_CreateModelComponent(entityName, deviceName, modelName, descriptor, errorTex):errorCode	Создание компонента устройства конкретной модели. Компонент четвертого уровня абстракции - модели устройства. Не реализует своих собственных методов, но содержит параметры данной модели, необходимые для расчетов при управлении устройством. Для управления устройством необходимо расширить компонент устройства до конкретной модели.	***entityName: c_char_p***- Тип компонента. Доступно: executor - сервопривод. connector - rgb светодиод ***deviceName: c_char_p***- Устройство компонента. Доступно: i2c_adapter - коннектор i2c pwm - ШИМ servodrive - сервопривод led - светодиод ***modelName: c_char_p***- Модель компонента. Доступно: ch341, cp2112, pca9685, mg90s, ky016 ***descriptor: c_int*** - Указатель на компонент, который будет создан

		errorText: Array[c_char, 1000]- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_DestroyComponent (descriptor, errorText):errorCode	Освобождение памяти, выделенной на компонент. Функция удаляет компонент с дескриптором равным параметру, переданному в функцию	***descriptor: c_int*** - Указатель на компонент ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_DestroySDK(isForce , errorText):errorCode	Освобождение памяти, выделенной на RI SDK. Освобождение памяти, выделенной на все компоненты и удаление реестра компонентов. Необходимо вызывать при завершении программы. После вызова данной функции дальнейшие вызовы других функций будут возвращать ошибки.	***isForce: c_bool*** - признак полного очищения реестра ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_LinkPWMToControlle r(descriptor, to, addr, errorText):errorCode	Связывание ШИМ с I2C адаптером. Сообщает библиотеки, что необходимо программно связать pwm с дескриптором ***descriptor*** с i2c адаптером с дескриптором ***to*** по адресу ***addr*** Эту функцию необходимо вызывать перед вызовом функций управления соответствующим ШИМ модулятором и перед вызовом функций управления устройством, который подключен к данному ШИМ модулятору (например сервопривод)	***descriptor: c_int*** - Указатель на компонент ***to: c_int*** - Указатель на адаптер ***addr: c_uint8*** - Адрес подключения ***errorText: c_cArray[c_char, 1000]*** - Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_LinkServodriveToC ontroller(descriptor, pwm, port, errorText):errorCode	Связывание сервопривода с ШИМ. Сообщает библиотеки, что необходимо на программно связать компонент сервопривода с дескриптором ***descriptor*** с ШИМ модулятором с дескриптором ***pwm*** по порту ***port***. Эту функцию необходимо вызывать перед вызовом функций управления соответствующим сервоприводом	***descriptor: c_int*** - Указатель на компонент ***pwm: c_int*** - Указатель на PWM ***port: c_int*** - Порт подключения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_LinkLedToControlle r(descriptor, pwm, rport , gport, bport, errorText):errorCode	Связывание светодиода с ШИМ. Сообщает библиотеки, что необходимо на программно связать компонент светодиода ***descriptor*** с ШИМ модулятором с дескриптором ***pwm*** по порту ***port***. Эту функцию необходимо вызывать перед вызовом функций управления соответствующим светодиодом	***descriptor: c_int*** - Указатель на компонент ***pwm: c_int*** - Указатель на PWM ***rport : c_int*** - Порт подключения красного цвета ***gport : c_int*** - Порт подключения зеленого цвета ***bport : c_int*** - Порт подключения синего цвета ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
Функции группы коннекторов		
RI_SDK_connector_Extend(basic, descriptor, errorText):errorCode	Расширение базового компонента до коннектора.	***basic : c_int*** - Указатель на базовый компонент ***descriptor: c_int*** - Указатель на коннектор

	Расширяет базовый компонент с дескриптором ***basic.*** Записывает в параметр ***descriptor*** дескриптор нового компонента	***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
Функции группы исполнителей		
RI_SDK_executor_Extend(basic, descriptor, errorText):errorCode	Расширение базового компонента до исполнителя. Расширяет базовый компонент с дескриптором ***basic.*** Записывает в параметр ***descriptor*** дескриптор нового компонента	***basic : c_int*** - Указатель на базовый компонент ***descriptor: c_int*** - Указатель на исполнитель ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_executor_State(descriptor ,state, errorText):errorCode	Чтение состояния исполнителя. Записывает в параметр ***state*** константу состояния исполнительного устройства с дескриптором равным ***descriptor***	***descriptor: c_int*** - Указатель на исполнитель ***state: c_int*** - Код состояния исполнителя Состояния для всех компонентов 0 - Компонент ожидает вызова действий. 1 - Компонент выполняет действие. Состояния светодиода 2 - простое свечение 3 - мигание 4 - мерцание ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
Функции I2C адаптера		
RI_SDK_connector_i2c_Open(descriptor, addr, errorText):errorCode	Открытие соединения по указанному адресу. Сообщает i2c адаптеру с дескриптором ***descriptor,*** что необходимо открыть соединение по адресу ***addr*** Создание нового подключения по какому-либо адресу необходимо делать перед тем, как производить чтение/запись по этому адресу	***descriptor: c_int*** - Указатель на i2c адаптер ***addr: c_uint8*** - Адрес открытия соединения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_connector_i2c_Extend(connectorDescriptor, descriptor, errorText):errorCode	Расширение компонента группы. Расширяет компонент группы коннекторов с дескриптором ***connectorDescriptor,*** записывает в параметр ***descriptor*** дескриптор нового компонента (i2c адаптера)	***connectorDescriptor: c_int*** - Указатель на группу ***descriptor: c_int*** - Указатель на i2c адаптер ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_connector_i2c_ExtendToModel(baseDescriptor, descriptor, modelName, errorTex):errorCode	Расширение I2C адаптера до модели. Расширяет компонент устройства i2c адаптер с дескриптором ***baseDescriptor,*** Записывает в параметр ***descriptor*** дескриптор нового компонента(компонент конкретной модели i2c адаптера)	***baseDescriptor : c_int*** - Указатель на устройство ***descriptor: c_int*** - Указатель на i2c модель ***modelName: c_char_p*** - Модель компонента. Доступно: ch341, cp2112 ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки

RI_SDK_connector_i2c_State(descriptor, state, errorText):errorCode	Чтение состояния адаптера. Записывает константу состояния i2c адаптера с дескриптором ***descriptor*** в параметр ***state*** Состояние I2C соединения: 0 - Линия свободна - означает, что в данный момент времени не выполняется никакая функция чтения/записи с данного i2c адаптера. 1 - Линия занята. Означает, что уже выполняется какая-либо функция чтения/записи для данного i2c адаптера. Если линия у какого-либо i2c адаптера занята, то при выполнении функций чтения/записи для данного i2c адаптера программа будет ожидать, пока линия освободится.	***descriptor: c_int*** - Указатель на i2c адаптер ***state: c_int*** - Указатель на i2c адаптер Состояние I2C соединения: 0 - Линия свободна, 1 - Линия занята. ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_connector_i2c_CloseAll(descriptor, errorText):errorCode	Закрытие всех открытых соединений. Закрывает все открытые соединения для i2c адаптера с дескриптором ***descriptor*** После закрытия соединения по данному адресу, функции чтения/записи для всех адресов будут возвращать ошибку. Для того, чтобы продолжить использовать функции чтения/записи необходимо вновь открыть соединение по нужному адресу	***descriptor: c_int*** - Указатель на i2c адаптер ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_connector_i2c_Close(descriptor, addr, errorText):errorCode	Закрытие соединения. Закрывает соединение по адресу ***addr*** для i2c адаптера с дескриптором ***descriptor*** После закрытия соединения по данному адресу, функции чтения/записи для данного адреса будут возвращать ошибку. Для того, чтобы продолжить использовать функции чтения/записи необходимо вновь открыть соединение по этому адресу	***descriptor: c_int*** - Указатель на i2c адаптер ***addr: c_uint8*** - Адрес соединения, которое необходимо закрыть ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_connector_i2c_WriteBytes(descriptor, addr, buf, len, errorText):errorCode	Запись байт по указанному адресу. Записывает ***len*** байтов из ***buf*** через i2c адаптера по адресу ***addr*** Длина массива байт ***buf*** должна быть раной ***len***	***descriptor: c_int*** - Указатель на i2c адаптер ***addr: c_uint8*** - Адрес соединения ***buf: Array[c_uint8, len]***- Буфер, массив байт для записи ***len: c_int*** - Размер буфера ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_connector_i2c_ReadBytes(descriptor, addr, buf, len, errorText):errorCode	Чтение байт по указанному адресу. Читает с i2c адаптера ***len*** байтов по адресу ***addr*** и записывает их в ***buf***	***descriptor: c_int*** - Указатель на i2c адаптер ***addr: c_uint8*** - Адрес соединения ***buf: Array[c_uint8, len]***- Буфер, массив байт для чтения ***len: c_int*** - Размер буфера ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки

RI_SDK_connector_i2c_WriteByte(descriptor, addr, value, errorText):errorCode	Запись байта по указанному адресу. Записывает байт из ***value*** через i2c адаптера по адресу ***addr***	***descriptor: c_int*** - Указатель на i2c адаптер ***addr: c_uint8*** - Адрес соединения ***value: c_uint8*** - Байт для записи ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_connector_i2c_ReadByte(descriptor, addr, value, errorText):errorCode	Чтение байта по указанному адресу. Читает с i2c адаптера один байт по адресу ***addr*** и записывает их в ***value***	***descriptor: c_int*** - Указатель на i2c адаптер ***addr: c_uint8*** - Адрес соединения ***value: c_uint8*** - Байт для чтения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
Функции ШИМ		
RI_SDK_sigmod_PWM_Extend(connectorDescriptor, descriptor, errorText):errorCode	Расширение компонента группы. Расширяет компонент группы коннекторов с дескриптором ***connectorDescriptor,*** Записывает в параметр ***descriptor*** дескриптор нового компонента (ШИМ модулятора)	***connectorDescriptor: c_int*** - Указатель на группу ***descriptor: c_int*** - Указатель на pwm ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_ExtendToModel(baseDescriptor, descriptor, modelName, errorTex):errorCode	Расширение ШИМ до модели. Расширяет компонент устройства ШИМ модулятор с дескриптором ***baseDescriptor,*** Записывает в параметр ***descriptor*** дескриптор нового компонента (компонент конкретной модели ШИМ модулятора)	***baseDescriptor : c_int*** - Указатель на устройство ***descriptor: c_int*** - Указатель на pwm модель ***modelName: c_char_p*** - Модель компонента. Доступно: pca9685 ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_GetResolution(descriptor, resolution, errorText):errorCode	Чтение разрешения ШИМ. Записывает в параметр ***resolution*** значение разрешения для ШИМ модулятора с дескриптором ***descriptor***	***descriptor: c_int*** - Указатель на устройство ***resolution : c_int*** - Разрешение ШИМ ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_GetFreq(descriptor, freq, errorText):errorCode	Чтение частоты ШИМ. Записывает в параметр ***freq*** значение частоты для ШИМ модулятора с дескриптором ***descriptor***	***descriptor: c_int*** - Указатель на устройство ***freq: c_int*** - Частота для контроллера ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_SetFreq(descriptor, freq, errorText):errorCode	Установка частоты ШИМ. Устанавливает новое значение частоты равное ***freq*** для ШИМ модулятора с дескриптором ***descriptor.***	***descriptor: c_int*** - Указатель на устройство ***freq: c_int*** - Частота для контроллера ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки

		Для модели ***pca9685*** тактовая частота (***referenceClockSpeed***) равна 25000000 Гц (25 МГц), а разрешение (***resolution***) равно 4096. При этом значение частоты (***freq***) должно быть больше отношения эталонной частоты к разрешению, умноженного на 2,5 ***freq > referenceClockSpeed / (2,5 * resolution)***
RI_SDK_sigmod_PWM_WriteRegBytes(descriptor, reg, buf, len, errorText):errorCode	Запись байт в регистр. Производит запись ***len*** байт из массива ***buf*** по регистру ***reg*** для ШИМ модулятора с дескриптором ***descriptor***	***descriptor: c_int*** - Указатель на устройство ***reg: c_uint8*** - Регистр ***buf: Array[c_uint8, len]***- Буфер, массив байт для записи ***len: c_int*** - Размер буфера ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_ReadRegBytes(descriptor, reg, buf, len, errorText):errorCode	Чтение байты из регистра. Читает ***len*** байт по регистру ***reg*** у ШИМ модулятора с дескриптором ***descriptor***. Записывает прочитанные байты в буфер ***buf***.	***descriptor: c_int*** - Указатель на устройство ***reg: c_uint8*** - Регистр ***buf: Array[c_uint8, len]***- Буфер, массив байт для записи ***len: c_int*** - Размер буфера ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_WriteByte(descriptor, reg, value, errorTex):errorCode	Запись байта в регистр. Производит запись одного байта со значением ***value*** по регистру ***reg*** для ШИМ модулятора с дескриптором ***descriptor***	***descriptor: c_int*** - Указатель на устройство ***reg: c_uint8*** - Регистр ***value: c_uint8*** - Байт для записи ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_ReadByte(descriptor, reg, value, errorTex):errorCode	Чтение байта из регистра. Читает один байт по регистру ***reg*** у ШИМ модулятора с дескриптором ***descriptor***. Записывает прочитанный байт в ***value***.	***descriptor: c_int*** - Указатель на устройство ***reg: c_uint8*** - Регистр ***value: c_uint8*** - Байт для записи ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_GetPortFreq(descriptor, port, freq, errorTex):errorCode	Чтение частоты порта. Записывает в параметр ***freq*** значение частоты на порту ***port*** для ШИМ модулятора с дескриптором ***descriptor*** *Не поддерживается для модели ***pca9685***	***descriptor: c_int*** - Указатель на устройство ***port : c_int*** - Номер порта на шим плате ***freq: c_int*** - Частота для контроллера ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_SetPortFreq(descriptor, port, freq, errorTex):errorCode	Установка частоты порта. Устанавливает новое значение частоты равное ***freq*** на порту ***port*** для ШИМ модулятора с дескриптором ***descriptor.***	***descriptor: c_int*** - Указатель на устройство ***port : c_int*** - Номер порта на шим плате ***freq: c_int*** - Частота для контроллера ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит

	*Не поддерживается для модели ***pca9685***	ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_ResetPort(descriptor, port, errorTex):errorCode	Сброс порта. Сбрасывает соединение, установленное на порту ***port*** у ШИМ модулятора с дескриптором ***descriptor.***	***descriptor: c_int*** - Указатель на устройство ***port : c_int*** - Номер порта на шим плате ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_ResetAll(descriptor, errorTex):errorCode	Сброс всех портов. Сбрасывает все соединения, установленные на всех портах у ШИМ модулятора с дескриптором ***descriptor.***	***descriptor: c_int*** - Указатель на устройство ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_sigmod_PWM_GetPortDutyCycle(descriptor, port, on, off, errorTex):errorCode	Чтение скважности. Получает значение скважности на порту ***port*** для ШИМ модулятора с дескриптором ***descriptor***. Записывает значения количества тактов до перевода выхода в состояние логической «1» в ***on*** и количество тактов до перевода выхода в состояние логического «0» в ***off***	***descriptor: c_int*** - Указатель на устройство ***port : c_int*** - Номер порта на шим плате ***on: c_int*** - Количество тактов до перевода выхода в состояние логической «1» ***off: c_int*** - Количество тактов до перевода выхода в состояние логического «0» ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки Значения количества тактов до перевода выхода в состояние логической «1» и логического «0» должны находится в диапазоне от 0 до установленного значения разрешения ШИМ (для модели ***pca9685*** - 4096)
RI_SDK_sigmod_PWM_SetPortDutyCycle(descriptor, port, on, off, errorTex):errorCode	Установка скважности. Устанавливает новое значение скважности на порту ***port*** для ШИМ модулятора с дескриптором ***descriptor***. Скважность задается с помощью параметров ***on*** и ***off***, в которых передаются соответственно количества тактов до перевода выхода в состояние логической «1» и логического «0». Значение ***on*** и ***off*** зависит от спецификаций конкретного ШИМ контроллера, например для PCA9685 разрешение ШИМ: 12 бит = 4096 тактов (рабочий цикл от 0 до 100%). Этот параметр показывает, с какой точностью мы можем менять коэффициент заполнения. Чем больше разрешение,тем плавнее будет меняться мощность на управляемом устройстве.	***descriptor: c_int*** - Указатель на устройство ***port : c_int*** - Номер порта на шим плате ***on: c_int*** - Количество тактов до перевода выхода в состояние логической «1» ***off: c_int*** - Количество тактов до перевода выхода в состояние логического «0» ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки Значения количества тактов до перевода выхода в состояние логической «1» и логического «0» должны находится в диапазоне от 0 до установленного значения разрешения ШИМ (для модели ***pca9685*** - 4096)
RI_SDK_sigmod_PWM_Close(descriptor, errorText):errorCode	Закрытие соединения с i2c адаптером Закрывает текущее соединение с i2c адаптером у ШИМ модулятора с дескриптором ***descriptor***	***descriptor: c_int*** - Указатель на устройство ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
Функции сервопривода		

RI_SDK_exec_ServoDrive_Extend(exec, descriptor, errorText):errorCode	Расширение исполнителя до сервопривода. Расширяет компонент группы исполнителей с дескриптором ***exec,*** Записывает в параметр ***descriptor*** дескриптор нового компонента (Сервопривода)	***exec : c_int*** - Указатель на группу ***descriptor: c_int*** - Указатель на сервопривод ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_ServoDrive_ExtendToModel(base, descriptor, modelName, errorText):errorCode	Расширение сервопривода до модели. Расширяет компонент устройства сервопривода с дескриптором ***base,*** Записывает в параметр ***descriptor*** дескриптор нового компонента (компонент конкретной модели сервопривода)	***base: c_int*** - Указатель на сервопривод ***descriptor: c_int*** - Указатель на модель ***modelName: c_char_p*** - Модель компонента. Доступно: mg90s ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_ServoDrive_CustomDeviceInit(descriptor, maxPulse, maxSpeed, pulseRange, errorText):errorCode	Инициализация собственного компонента. Инициализирует собственный компонент сервопривода. Альтернатива функции **RI_SDK_exec_ServoDrive_ExtendToModel.** Устанавливает значения максимального импульса (мкс), максимальной скорости вращения (градусов в секунду), размера рабочего диапазона в мкс. Эти значение устанавливаются для сервопривода с дескриптором ***descriptor***. Данная функция используется для получения возможности работы с сервоприводом, модель которого не поддерживается библиотекой.	***descriptor: c_int*** - Указатель на сервопривод ***maxPulse: c_int*** - Максимальное значение импульса (мкс) ***maxSpeed: c_int*** - Максимальная угловая скорость (градус / секунда) ***pulseRange: c_int*** - Диапазон вращения сервопривода (мкс) ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_ServoDrive_TurnByDutyCycle(descriptor, step, errorText):errorCode	Абсолютный поворот. Угол задается через кол-во шагов сервопривода. Дает команду сервоприводу дескриптором ***descriptor*** встать в положение, соответствующее переданному значению скважности. Значение скважности не может быть больше значения разрешения ШИМ, к которому подключен сервопривод. Также значение скважности должно быть больше или равным ((***minPulse*********freq***)/1000000*0xFFFF)+1) >> 4 и меньше или равным ((***maxPulse*********freq***)/1000000*0xFFFF)+1) >> 4 Где: ***minPulse*** - минимальное значение импульса для данного сервопривода (***maxPulse - pulseRange***); ***maxPulse***- максимальное значение импульса для данного сервопривода; ***freq -*** значение частоты ШИМ модулятора, к которому подключен сервопривод. Для модели mg90s, у которой размер рабочего диапазона равен 2444 мкс, максимальное значение импульса равно 2771 мкс, при подключении к ШИМ модулятору pca9586, у которого частота равна 50 Гц, значение скважности должно	***descriptor: c_int*** - Указатель на сервопривод ***step: c_int*** - Количество шагов для шим преобразователя ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальный импульс - 2711 мкс Для частоты ШИМ модулятора в 50Гц минимальное количество шагов - 0, максимальное - 555

	попадать в промежуток от 55 до 554 шагов включительно.	
RI_SDK_exec_ServoDrive_TurnByPulse(descriptor, pulse, errorText):errorCode	Абсолютный поворот к углу, определенному значением импульса. Дает команду сервоприводу дескриптором ***descriptor*** встать в положение, соответствующее переданному значению управляющего импульса. Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальный импульс - 2711 мкс	***descriptor: c_int*** - Указатель на сервопривод ***pulse: c_int*** - Значение импульса (мкс) ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальный импульс - 2711 мкс
RI_SDK_exec_ServoDrive_GetCurrentAngle(descriptor, angle, errorText):errorCode	Чтение текущего угла. Получает значение угла поворота у сервопривода с дескриптором ***descriptor*** и записывает его в ***angle***	***descriptor: c_int*** - Указатель на сервопривод ***angle: c_int*** - Текущий угол сервопривода ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_ServoDrive_GetState(descriptor, state, errorText):errorCode	Чтение состояния сервопривода. Получает значение константы состояния сервопривода с дескриптором ***descriptor*** и записывает его в ***state*** Возможные значения: 0 - ожидает вызова действий. Сервопривод ничего не делает и готов к работе. 1 - сервопривод в данный момент осуществляет движение.	***descriptor: c_int*** - Указатель на сервопривод ***state: c_int*** - Получение состояния: 0 - ожидает вызова действий 1 - действие выполняется в данный момент ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_ServoDrive_MinStepRotate(descriptor, direction, speed, async, errorText):errorCode	Поворот на минимальный шаг. Дает команду сервоприводу с дескриптором ***descriptor*** повернуть на 1 градус. Направление поворота задается параметром ***direction***, а скорость поворота параметром ***speed***. Параметр ***async*** устанавливает режим выполнения команды - асинхронный или синхронный. При синхронном режиме, программа, которая вызвала данную функцию, ожидает завершение выполнения команды. При асинхронном режиме этого ожидания не происходит, и выполнение команды осуществляется параллельно дальнейшей работе программы, вызвавшей данную функцию.	***descriptor: c_int*** - Указатель на сервопривод ***direction: c_int*** - Направление движения: 0 - положительное 1 - отрицательное ***speed c_int*** - Угловая скорость (градус / секунда) ***async: c_bool***- Флаг асинхронного режима выполнения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальная скорость для модели ***mg90s*** - 400 градусов в секунду
RI_SDK_exec_ServoDrive_Turn(descriptor, angle, speed, async, errorText):errorCode	Поворот на заданный угол с заданной угловой скоростью. Дает команду сервоприводу с дескриптором ***descriptor*** повернуть на ***angle*** градусов. Направление поворота задается параметром ***direction***, а скорость поворота параметром ***speed***. Параметр ***async*** устанавливает режим выполнения команды - асинхронный или синхронный.	***descriptor: c_int*** - Указатель на сервопривод ***angle: c_int*** - Угол поворота в градусах. Если угол положительный, поворот осуществляется по часовой стрелке, если отрицательный, то против часовой. ***speed c_int*** - Угловая скорость (градус / секунда) ***async: c_bool***- Флаг асинхронного режима выполнения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит

	При синхронном режиме, программа, которая вызвала данную функцию, ожидает завершение выполнения команды. При асинхронном режиме этого ожидания не происходит, и выполнение команды осуществляется параллельно дальнейшей работе программы, вызвавшей данную функцию.	ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальная скорость для модели ***mg90s*** - 400 градусов в секунду
RI_SDK_exec_ServoDrive_Stop(descriptor, errorText):errorCode	Остановка выполняемого действия. Дает команду сервоприводу с дескриптором ***descriptor*** остановить движение, если оно производится в данный момент.	***descriptor: c_int*** - Указатель на сервопривод ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_ServoDrive_Rotate(descriptor, direction, speed, async, errorText):errorCode	Вращение с заданной угловой скоростью. Дает команду сервоприводу с дескриптором ***descriptor*** вращаться до тех пор, пока не будет достигнуто крайнее значение угла поворота (либо максимальный угол поворота либо минимальный, в зависимости от направления движения). Направление поворота задается параметром ***direction***, а скорость поворота параметром ***speed***. Параметр ***async*** устанавливает режим выполнения команды - асинхронный или синхронный. При синхронном режиме, программа, которая вызвала данную функцию, ожидает завершение выполнения команды. При асинхронном режиме этого ожидания не происходит, и выполнение команды осуществляется параллельно дальнейшей работе программы, вызвавшей данную функцию.	***descriptor: c_int*** - Указатель на сервопривод ***direction: c_int*** - Направление движения: 0 - положительное 1 - отрицательное ***speed c_int*** - Угловая скорость (градус / секунда) ***async: c_bool***- Флаг асинхронного режима выполнения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальная скорость для модели ***mg90s*** - 400 градусов в секунду
RI_SDK_exec_ServoDrive_RotateWithRelativeSpeed(descriptor, direction, speed, async, errorText):errorCode	Вращение в заданном направлении с заданной относительной скоростью. Дает команду сервоприводу с дескриптором ***descriptor*** вращаться до тех пор, пока не будет достигнуто крайнее значение угла поворота (либо максимальный угол поворота либо минимальный, в зависимости от направления движения). Направление поворота задается параметром ***direction***, а скорость поворота параметром ***speed***. При этом скорость поворота задается в процентах от максимальной скорости Параметр ***async*** устанавливает режим выполнения команды - асинхронный или синхронный. При синхронном режиме, программа, которая вызвала данную функцию, ожидает завершение выполнения команды. При асинхронном режиме этого ожидания не происходит, и выполнение команды осуществляется параллельно дальнейшей работе программы, вызвавшей данную функцию.	***descriptor: c_int*** - Указатель на сервопривод ***direction: c_int*** - Направление движения: 0 - положительное 1 - отрицательное ***speed c_int*** - Процент от максимальной скорости (0 - 100) ***async: c_bool***- Флаг асинхронного режима выполнения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальная скорость для модели ***mg90s*** - 400 градусов в секунду
RI_SDK_exec_ServoDrive_TurnWithRelativeSpeed(descri	Поворот на заданный угол с заданной относительной скоростью.	***descriptor: c_int*** - Указатель на сервопривод ***angle: c_int*** - Угол поворота в градусах. Если угол положительный,

ptor, angle, speed, async, errorText):errorCode	Поворот на заданный угол с заданной угловой скоростью. Дает команду сервоприводу с дескриптором ***descriptor*** повернуть на ***angle*** градусов. Направление поворота задается параметром ***direction***, а скорость поворота параметром ***speed***. При этом скорость поворота задается в процентах от максимальной скорости. Параметр ***async*** устанавливает режим выполнения команды - асинхронный или синхронный. При синхронном режиме, программа, которая вызвала данную функцию, ожидает завершение выполнения команды. При асинхронном режиме этого ожидания не происходит, и выполнение команды осуществляется параллельно дальнейшей работе программы, вызвавшей данную функцию	поворот осуществляется по часовой стрелке, если отрицательный, то против часовой. ***speed c_int*** - Процент от максимальной скорости (0 - 100) ***async: c_bool***- Флаг асинхронного режима выполнения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальная скорость для модели ***mg90s*** - 400 градусов в секунду
Функции светодиода		
RI_SDK_exec_RGB_LED_Extend(exec, descriptor, errorText):errorCode	Расширение исполнителя до светодиода. Расширяет компонент группы исполнителей с дескриптором ***exec,*** Записывает в параметр ***descriptor*** дескриптор нового компонента (светодиод)	***exec : c_int*** - Указатель на группу ***descriptor: c_int*** - Указатель на светодиод ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_RGB_LED_ExtendToModel(base, descriptor, modelName, errorText):errorCode	Расширение светодиода до модели. Расширяет компонент устройства светодиода с дескриптором ***base,*** Записывает в параметр ***descriptor*** дескриптор нового компонента (компонент конкретной модели светодиода)	***base: c_int*** - Указатель на группу ***descriptor: c_int*** - Указатель на светодиод ***modelName: c_char_p*** - Модель компонента. Доступно: ky016 ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_RGB_LED_SinglePulse(descriptor, r, g, b,duration, async, errorText):errorCode	Непрерывное свечение. Дает команду непрерывного свечения для светодиода с дескриптором ***descriptor***. Параметры ***r, g, b*** устанавливают яркость свечения для красного, зеленого и синего цвета соответственно. Параметр ***duration*** задает продолжительность свечения.Параметр ***async*** устанавливает режим выполнения команды - асинхронный или синхронный. При синхронном режиме, программа, которая вызвала данную функцию, ожидает завершение выполнения команды. При асинхронном режиме этого ожидания не происходит, и выполнение команды осуществляется параллельно дальнейшей работе программы, вызвавшей данную функцию.	***descriptor: c_int*** - Указатель на светодиод ***r: c_int*** - Значение яркости для красного цвета (0 -255) ***g: c_int*** - Значение яркости для зеленого цвета (0 -255) ***b: c_int*** - Значение яркости для синего цвета (0 -255) ***duration c_int*** - Продолжительность импульса (мс) При значении 0 операция выполняется до тех пор, пока не придет другая команда или команда остановки ***async: c_bool***- Флаг асинхронного режима выполнения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_RGB_LED_Stop(descriptor, errorText):errorCode	Прекращение подачи сигнала на светодиод.	***descriptor: c_int*** - Указатель на светодиод ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит

	Дает команду светодиоду с дескриптором ***descriptor*** остановить выполнение выполняемой в данный момент команды	ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_RGB_LED_GetState(descriptor, state, errorTex):errorCode	Чтение состояния светодиода. Получает значение константы состояния светодиода с дескриптором ***descriptor*** и записывает его в ***state*** Возможные значения: 0 - Компонент ожидает вызова действий. - Светодиод не выполняет никакую команду 2 - Простое свечение. Светодиод выполняет команду простого свечения. 3 - Мигание. Светодиод выполняет одну из 2-х команд мигания. 4 - Мерцание. Светодиод выполняет команду мерцания.	***descriptor: c_int*** - Указатель на светодиод ***state: c_int*** - Получение состояния: 0 - Компонент ожидает вызова действий. 2 - простое свечение 3 - мигание 4 - мерцание ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_RGB_LED_GetColor(descriptor, r, g, b, errorText):errorCode	Чтение цвета светодиода. Получает значения текущих яркостей красного, зеленого и синего цветов у сервопривода с дескриптором ***descriptor*** и записывает эти значения в ***r, g, b.***	***descriptor: c_int*** - Указатель на светодиод ***r: c_int*** - Значение яркости для красного цвета (0 -255) ***g: c_int*** - Значение яркости для зеленого цвета (0 -255) ***b: c_int*** - Значение яркости для синего цвета (0 -255) ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_RGB_LED_FlashingWithFrequency(descriptor, r, g, b, frequency, qty, async, errorText):errorCode	Мигание с заданной частотой. Дает команду светодиоду с дескриптором ***descriptor*** мигать с частотой ***frequency***. Количество миганий задается параметром ***qty***. Параметры ***r, g, b*** устанавливают яркость свечения для красного, зеленого и синего цвета соответственно. Параметр ***async*** устанавливает режим выполнения команды - асинхронный или синхронный. При синхронном режиме, программа, которая вызвала данную функцию, ожидает завершение выполнения команды. При асинхронном режиме этого ожидания не происходит, и выполнение команды осуществляется параллельно дальнейшей работе программы, вызвавшей данную функцию.	***descriptor: c_int*** - Указатель на светодиод ***r: c_int*** - Значение яркости для красного цвета (0 -255) ***g: c_int*** - Значение яркости для зеленого цвета (0 -255) ***b: c_int*** - Значение яркости для синего цвета (0 -255) ***frequency: c_int*** - Частота импульса (гц) ***qty: c_int*** - Количество итераций При значении 0 операция выполняется до тех пор, пока не придет другая команда или команда остановки ***async: c_bool***- Флаг асинхронного режима выполнения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки
RI_SDK_exec_RGB_LED_FlashingWithPause(descriptor, r, g, b, duration, pause, qty, async, errorText):errorCode	Мигание с заданной продолжительностью и паузой. Дает команду светодиоду с дескриптором ***descriptor*** мигать с паузой между миганиями равной ***pause***. Параметр ***duration*** устанавливает время одного светового импульса . Количество миганий задается параметром ***qty***. Параметры ***r, g, b*** устанавливают яркость свечения для красного, зеленого и синего цвета соответственно. Параметр ***async*** устанавливает режим выполнения команды - асинхронный или синхронный. При синхронном режиме, программа, которая вызвала данную функцию, ожидает завершение выполнения команды. При асинхронном режиме этого ожидания не происходит, и выполнение команды	***descriptor: c_int*** - Указатель на светодиод ***r: c_int*** - Значение яркости для красного цвета (0 -255) ***g: c_int*** - Значение яркости для зеленого цвета (0 -255) ***b: c_int*** - Значение яркости для синего цвета (0 -255) ***duration: c_int*** - Продолжительность одного светового импульса(мс) ***pause: c_int*** - Пауза между импульсами (мс) ***qty: c_int*** - Количество итераций При значении 0 операция выполняется до тех пор, пока не придет другая команда или команда остановки ***async: c_bool***- Флаг асинхронного режима выполнения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки)

	осуществляется параллельно дальнейшей работе программы, вызвавшей данную функцию.	***errorCode: c_int*** - код ошибки
RI_SDK_exec_RGB_LED_Flicker(descriptor, r, g, b, duration, qty, async, errorText):errorCode	Мерцание. Дает команду светодиоду с дескриптором ***descriptor*** мерцать (постепенно изменять яркость цвета от 0 до значений ***r, g, b*** для красного, зеленого и синего цвета соответственно и обратно). Параметр ***duration*** устанавливает время, за которое яркость света изменяется от 0 до значения установленного параметрами ***r, g, b*** и за которое яркость уменьшается обратно до 0. Количество мерцаний задается параметром ***qty***. Параметр ***async*** устанавливает режим выполнения команды - асинхронный или синхронный. При синхронном режиме, программа, которая вызвала данную функцию, ожидает завершение выполнения команды. При асинхронном режиме этого ожидания не происходит, и выполнение команды осуществляется параллельно дальнейшей работе программы, вызвавшей данную функцию.	***descriptor: c_int*** - Указатель на светодиод ***r: c_int*** - Значение максимальной яркости для красного цвета (0 -255) ***g: c_int*** - Значение максимальной яркости для зеленого цвета (0 -255) ***b: c_int*** - Значение максимальной яркости для синего цвета (0 -255) ***duration: c_int*** - Продолжительность изменения яркости от 0 до максимальной (мс) ***qty: c_int*** - Количество итераций При значении 0 операция выполняется до тех пор, пока не придет другая команда или команда остановки ***async: c_bool***- Флаг асинхронного режима выполнения ***errorText: Array[c_char, 1000]***- Текст ошибки (передается как параметр, если происходит ошибка метод записывает в этот параметр текст ошибки) ***errorCode: c_int*** - код ошибки

Расшифровка кодов ошибок

Код ошибки	Краткое описание	Подробное описание и пути решения
1. Ошибки уровня ядра		
100001	Неопределенный режим логирования	Возникает при вызове функции ***RI_SDK_InitSDK*** с недействительным значением уровня логирования ***logLevel*** Необходимо использовать доступные значения данного аргумента, указанные в описании функции ***RI_SDK_InitSDK*** в документации и таблице “**Краткое перечисление функций RI SDK”**
100002	Недопустимый тип компонента	Данная ошибка возникает при передаче в функцию, дескриптор компонента недопустимого типа для данной функции. Например при вызове функции ***RI_SDK_sigmod_PWM_ResetPort*** требуется передать дескриптор ШИМ модулятора. Если же передать в нее дескриптор сервопривода, то возникнет данная ошибка. Убедитесь, что Вы используете дескриптор нужного компонента.
100003	Параметр длины буфера больше размера буфера	Данная ошибка возникает в случае вызова функций чтения/записи массивов байт (Например ***RI_SDK_connector_i2c_WriteBytes***) в случае, если

		переданный массив имеет длину большую, чем указано в параметре len данной функции Необходимо перед вызовом данной функции узнать длину массива передаваемого в одну из функций чтения/записи. Это можно узнать с помощью встроенного метода len(array), в которой в качестве аргумента принимает и массив и возвращает его длину.
110001	Реестр компонентов не инициализирован	Данная ошибка возникает в том случае, если перед использованием функций библиотеки не была вызвана функция инициализации ***RI_SDK_InitSDK***. Или в случае, когда перед вызовом функций была вызвана функция освобождения памяти ***RI_SDK_DestroySDK.*** Перед вызовом функций необходимо вызывать ***RI_SDK_InitSDK***. При этом функцию ***RI_SDK_DestroySDK*** нужно вызывать в конце выполнения программы.
110002	Компонент не найден в реестре	Эта ошибка может возникнуть если в функцию, требующую указатель на дескриптор компонента, было передано значение дескриптора несуществующего компонента. (Кроме функций создания компонентов ***RI_SDK_CreateBasic,*** ***RI_SDK_CreateGroupComponent,*** ***RI_SDK_CreateDeviceComponent,*** ***RI_SDK_CreateModelComponent***) Это может возникать, если компонент не был создан или уже был удален. Также это может возникнуть в том случае, если переменной, в которой хранится значение дескриптора, было присвоено новое значение. Убедитесь, что в передаваемой Вами переменной действительно хранится дескриптор на существующий компонент.
110005	Реестр компонентов не пуст	Возникает при вызове функции ***RI_SDK_DestroySDK*** со значением флага ***isForce=false*** в том случае, если созданные ранее компоненты не были удалены. В данном случае необходимо либо до вызова ***RI_SDK_DestroySDK*** удалить все компоненты с помощью метода ***RI_SDK_DestroyComponent*** либо вызвать ***RI_SDK_DestroySDK*** со значением флага ***isForce=true***
120001	Группа не существует	Возникает при вызове функций создания компонентов всех уровней (***RI_SDK_CreateGroupComponent***, ***RI_SDK_CreateDeviceComponent***, ***RI_SDK_CreateModelComponent***) при передаче в качестве аргумента название несуществующей группы. Доступны следующие значения: **executor** - исполнитель. **connector** - коннектор **sensor** - датчик
120002	Не удалось создать компонент уровня типа устройства	Возникает при попытке создания компонента уровня устройства ***RI_SDK_CreateDeviceComponent*** и уровня модели ***RI_SDK_CreateModelComponent***. Причиной данной ошибки является указание несуществующего типа устройства для указанной в аргументе функции группы устройств Необходимо указать одно из доступных значений типа устройств, согласно описанию функции в документации и в таблице “**Краткое перечисление функций RI SDK”**
120003	Не удалось создать компонент уровня модели устройства	Возникает при попытке создания компонента уровня устройства ***RI_SDK_CreateModelComponent***. Причиной

		данной ошибки является указание несуществующего типа устройства для указанной в аргументе функции группы устройств либо несуществующей модели для указанного в аргументе функции устройства. Необходимо указать одно из доступных значений модели, согласно описанию функции в документации и в таблице “**Краткое перечисление функций RI SDK”**
130001	Ошибка связывания pwm с адаптером	Возникает при вызове функции связывания Шим модулятора с i2c адаптером ***RI_SDK_LinkPWMToController***. Ошибка может возникать если: 1) Если переданы дескрипторы несуществующих i2c адаптера и ШИМ модулятора. Убедитесь, что перед вызовом функции связывания были созданы i2c адаптер и ШИМ модулятор и, что в их дескрипторы были переданы в функцию связывания. 2) Если произошла ошибка создания подключения по i2c шине. Может возникнуть если устройство не подключено к компьютеру, либо при отсутствии в корне проекта библиотек по работе с i2c ***CH341DLL.DLL/CH341DLLA64.dll, SLABHIDDevice.dll, SLABHIDtoSMBus.dll***. Проверьте их наличие рядом с файлом библиотеки RI_SDK.
130002	Указанный компонент не обладает функционалом связывания	Возникает при вызове функций связывания ***RI_SDK_LinkPWMToController, RI_SDK_LinkServodriveToController, RI_SDK_LinkLedToController*** Происходит,если компонент к котором осуществляется подключение, имеет имеет не тот тип, который необходим. Например при подключении сервопривода к ШИМ необходимо передать дескриптор ШИМ. Если передать дескриптор другого типа то возникнет данная ошибка Проверьте корректность передаваемых параметров согласно документации и таблицы “**Краткое перечисление функций RI SDK”**
130003	Указанный компонент не поддерживает протокол связывания	Возникает при вызове функций связывания ***RI_SDK_LinkPWMToController, RI_SDK_LinkServodriveToController, RI_SDK_LinkLedToController*** Происходит,если компонент который осуществляет подключение, имеет имеет не тот тип, который необходим. Например при подключении сервопривода к ШИМ необходимо передать дескриптор сервопривода. Если передать дескриптор другого типа то возникнет данная ошибка Проверьте корректность передаваемых параметров согласно документации и таблицы “**Краткое перечисление функций RI SDK”**
130004	Ошибка связывания компонента с ШИМ	Возникает при вызове функций связывания с ШИМ модулятором сервопривода ***RI_SDK_LinkServodriveToController*** либо светодиода ***RI_SDK_LinkLedToController***.

		Ошибка может возникать если, переданы дескрипторы несуществующих i ШИМ модулятора и, либо сервопривода (в ***RI_SDK_LinkServodriveToController***), либо светодиода (в ***RI_SDK_LinkLedToController***) Необходимо убедиться, что в аргументах указаны корректные значение устройства, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
130005	Ошибка связывания компонента с ШИМ на порту	Возникает при вызове функций связывания с ШИМ модулятором сервопривода ***RI_SDK_LinkServodriveToController*** либо светодиода ***RI_SDK_LinkLedToController***. Ошибка может возникать в следующих случаях: 1) если на указанном порту уже существует подключение. В этом случае необходимо указать свободной порт, по которому еще не осуществлено подключение 2) если указан номер порта, выходящий за рамки от 0 до 15. В этом случае. убедитесь, что указано корректное значение порта согласно документации и таблицы “Краткое перечисление функций RI SDK”
2. Ошибки компонентов		
2.1. Коннекторы		
2.1.1. I2C адаптер		
211001	Адрес уже занят	Возникает при вызове метода ***RI_SDK_connector_i2c_Open***. Означает, что уже существует соединение по указанному адресу. В данном случае необходимо указать другой адрес, по которому в программе еще не осуществлялось подключение.
211002	Ошибка создания соединения	Возникает при вызове методов ***RI_SDK_connector_i2c_Open***. Означает, что драйверу не удалось осуществить подключение. Это может возникнуть если устройство не подключено к компьютеру, либо при отсутствии в корне проекта библиотек по работе с i2c ***CH341DLL.DLL/CH341DLLA64.dll, SLABHIDDevice.dll, SLABHIDtoSMBus.dll***. Проверьте их наличие рядом с файлом библиотеки RI_SDK и подключение робота к компьютеру
211003	Ошибка закрытия соединения по адресу	Возникает при вызове методов ***RI_SDK_connector_i2c_Close***, ***RI_SDK_connector_i2c_CloseAll***. Означает, что драйверу не удалось закрыть соединение i2c адаптера. Это может возникнуть если устройство не подключено к компьютеру, либо при отсутствии в корне проекта библиотек по работе с i2c ***CH341DLL.DLL/CH341DLLA64.dll, SLABHIDDevice.dll, SLABHIDtoSMBus.dll***. Проверьте их наличие рядом с

		файлом библиотеки RI_SDK и подключение робота к компьютеру
211004	Подключение к устройству по адресу уже существует	Возникает при вызове методов ***RI_SDK_LinkPWMToController***. Означает, что уже существует соединение по указанному адресу.В данном случае необходимо указать другой адрес, по которому в программе еще не осуществлялось подключение.
211005	I2C адаптер еще не расширен до конкретной модели	Возникает при вызове методов ***RI_SDK_connector_i2c_Open*, *RI_SDK_LinkPWMToController*.** Означает, что i2c адаптер еще не расширен до конкретной модели и библиотека еще не знает, какой драйвер необходимо использовать для создания подключения. Перед вызовом данных методов, необходимо воспользоваться ***RI_SDK_connector_i2c_ExtendToModel*** для расширения i2c адаптер до конкретной модели
211006	Ошибка при создании подключения по I2C	Возникает при вызове методов ***RI_SDK_connector_i2c_Open*, *RI_SDK_LinkPWMToController*.** Означает, что драйверу не удалось осуществить подключение. Это может возникнуть если устройство не подключено к компьютеру, либо при отсутствии в корне проекта библиотек по работе с i2c ***CH341DLL.DLL/CH341DLLA64.dll, SLABHIDDevice.dll, SLABHIDtoSMBus.dll***. Проверьте их наличие рядом с файлом библиотеки RI_SDK и подключение робота к компьютеру
211007	Ошибка записи одного байта	Возникает при вызове метода ***RI_SDK_connector_i2c_WriteByte***. Означает, что драйверу не удалось считать байт. Это может возникнуть если устройство не подключено к компьютеру, либо при отсутствии в корне проекта библиотек по работе с i2c ***CH341DLL.DLL/CH341DLLA64.dll, SLABHIDDevice.dll, SLABHIDtoSMBus.dll*.** Проверьте их наличие рядом с файлом библиотеки RI_SDK и подключение робота к компьютеру.
211008	Ошибка чтения первого байта	Возникает при вызове метода ***RI_SDK_connector_i2c_ReadByte*.** Означает, что драйверу не удалось записать байт. Это может возникнуть если устройство не подключено к компьютеру, либо при отсутствии в корне проекта библиотек по работе с i2c ***CH341DLL.DLL/CH341DLLA64.dll, SLABHIDDevice.dll, SLABHIDtoSMBus.dl*l**. Проверьте их наличие рядом с файлом библиотеки RI_SDK и подключение робота к компьютеру.
	2.1.1.1. CH341	
211101	Адрес уже занят	Возникает при вызове метода ***RI_SDK_connector_i2c_Open*.** Означает, что уже существует соединение по указанному адресу. В данном случае необходимо указать другой адрес, по которому в программе еще не осуществлялось подключение.

211102	Ошибка создания подключения	Возникает при вызове метода ***RI_SDK_connector_i2c_Open*.** Означает, что драйверу не удалось осуществить подключение. Это может возникнуть если устройство не подключено к компьютеру, либо при отсутствии в корне проекта библиотек по работе с i2c ***CH341DLL.DLL/CH341DLLA64.dll, SLABHIDDevice.dll, SLABHIDtoSMBus.dll*.** Проверьте их наличие рядом с файлом библиотеки RI_SDK и подключение робота к компьютеру.
211103	Подключение к адресу устройства уже существует	Возникает при вызове метода ***RI_SDK_LinkPWMToController*.** Означает, что уже существует соединение по указанному адресу. В данном случае необходимо указать другой адрес, по которому в программе еще не осуществлялось подключение.
	2.1.1.2. CP2112	
211201	Адрес уже занят	Возникает при вызове метода ***RI_SDK_connector_i2c_Open*.** Означает, что уже существует соединение по указанному адресу. В данном случае необходимо указать другой адрес, по которому в программе еще не осуществлялось подключение.
211202	Ошибка создания подключения	Возникает при вызове методов ***RI_SDK_connector_i2c_Open*.** Означает, что драйверу не удалось осуществить подключение. Это может возникнуть если устройство не подключено к компьютеру, либо при отсутствии в корне проекта библиотек по работе с i2c ***CH341DLL.DLL/CH341DLLA64.dll, SLABHIDDevice.dll, SLABHIDtoSMBus.dl*l**. Проверьте их наличие рядом с файлом библиотеки RI_SDK и подключение робота к компьютеру.
211203	Подключение к адресу устройства уже существует	Возникает при вызове методов ***RI_SDK_LinkPWMToController*.** Означает, что уже существует соединение по указанному адресу. В данном случае необходимо указать другой адрес, по которому в программе еще не осуществлялось подключение.
	2.1.2. PWM модулятор	
212001	Не поддерживается такое соотношение разрешения и частоты	Возникает при вызове метода установки частоты ШИМ ***RI_SDK_sigmod_PWM_SetFreq.*** Означает, что устанавливаемое значение частоты ШИМ не поддерживается для разрешения и тактовой частоты данной модели ШИМ модулятора. Необходимо передавать в функцию ***RI_SDK_sigmod_PWM_SetFreq*** значение удовлетворяющее требованиям, указанным в документации и таблице “**Краткое перечисление функций RI SDK”**
212002	Не задан контроллер соединения	Возникает при вызове метода ***RI_SDK_sigmod_PWM_SetPortDutyCycle***, а также при вызове методов компонентов, подключенных к ШИМ модулятору. Эта ошибка возникает в тех случаях, когда ШИМ модулятор не подключен к контроллеру, например к i2c

		адаптеру, а значит никак не связан с вашим компьютером. Во избежания данной ошибки необходимо вызывать метод линковки ШИМ ***RI_SDK_LinkPWMToController*** перед вызовом метода ***RI_SDK_sigmod_PWM_SetPortDutyCycle*** и методов управления компонентов, подключенных к ШИМ
212003	Неверное значение On	Возникает при вызове метода ***RI_SDK_sigmod_PWM_SetPortDutyCycle***. Означает, что при вызове этого метода было передано некорректное значение количества тактов до перевода выхода в состояние логической «1». Оно должно находится в диапазоне от 0 до установленного значения разрешения ШИМ (для модели ***pca9685*** - 4096). Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
212004	Неверное значение Off	Возникает при вызове метода ***RI_SDK_sigmod_PWM_SetPortDutyCycle***. Означает, что при вызове этого метода было передано некорректное значение количества тактов до перевода выхода в состояние логического «0». Оно должно находится в диапазоне от 0 до установленного значения разрешения ШИМ (для модели ***pca9685*** - 4096) Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
212005	Порт уже занят	Возникает при вызове методов ***RI_SDK_LinkServodriveToController***, ***RI_SDK_LinkLedToController***. Означает, что по данному порту уже существует подключение. В данном случае необходимо указать другой порт, по которому в программе еще не осуществлялось подключение.
212006	Нет порта с данным индексом	Возникает при вызове методов ***RI_SDK_LinkServodriveToController***, ***RI_SDK_LinkLedToController***. Означает, что значение порта, переданное в одну из этих 2-х функций либо меньше 0, либо больше максимально возможного для выбранной модели ШИМ Для модели ***pca9685*** наибольший индекс порта равен 15 (0 -15, 16 портов) Необходимо указать индекс порта от 0 до максимального значения для выбранной модели ШИМ, согласно документации и таблицы “**Краткое перечисление функций RI SDK”**
212007	Частота не установлена	Возникает при вызове методов ***RI_SDK_connector_i2c_WriteBytes, RI_SDK_connector_i2c_ReadBytes, RI_SDK_connector_i2c_WriteByte, RI_SDK_connector_i2c_ReadByte***, а также в методах управления компонентами, подключенных к ШИМ модулятору (сервопривод, свтодиод). Означает, что ШИМ модулятор не расширен до конкретной модели, а значит библиотека не знает, какое значение частоты использовать при расчетах.

		Необходимо расширить компонент до конкретной модели, воспользовавшись функцией ***RI_SDK_sigmod_PWM_ExtendToModel*** или сразу создать компонент ШИМ модулятора конкретной модели с помощью функции ***RI_SDK_CreateModelComponent***
212008	Разрешение не установлено	Возникает при вызове методов ***RI_SDK_connector_i2c_WriteBytes, RI_SDK_connector_i2c_ReadBytes, RI_SDK_connector_i2c_WriteByte, RI_SDK_connector_i2c_ReadByte,*** а также в методах управления компонентами, подключенных к ШИМ модулятору (сервопривод, свтодиод). Означает, что ШИМ модулятор не расширен до конкретной модели, а значит библиотека не знает, какое значение разрешения использовать при расчетах. Необходимо расширить компонент до конкретной модели, воспользовавшись функцией ***RI_SDK_sigmod_PWM_ExtendToModel*** или сразу создать компонент ШИМ модулятора конкретной модели с помощью функции ***RI_SDK_CreateModelComponent***
212009	Количество портов не установлено	Возникает при вызове методов ***RI_SDK_connector_i2c_WriteBytes, RI_SDK_connector_i2c_ReadBytes, RI_SDK_connector_i2c_WriteByte, RI_SDK_connector_i2c_ReadByte***, а также в методах управления компонентами, подключенных к ШИМ модулятору (сервопривод, свтодиод). Означает, что ШИМ модулятор не расширен до конкретной модели, а значит библиотека не знает, максимальное количество портов можно использовать. Необходимо расширить компонент до конкретной модели, воспользовавшись функцией ***RI_SDK_sigmod_PWM_ExtendToModel*** или сразу создать компонент ШИМ модулятора конкретной модели с помощью функции ***RI_SDK_CreateModelComponent***
212010	Тактовая частота не установлена	Возникает при вызове методов ***RI_SDK_connector_i2c_WriteBytes, RI_SDK_connector_i2c_ReadBytes, RI_SDK_connector_i2c_WriteByte, RI_SDK_connector_i2c_ReadByte,*** а также в методах управления компонентами, подключенных к ШИМ модулятору (сервопривод, свтодиод). Означает, что ШИМ модулятор не расширен до конкретной модели, а значит библиотека не знает, какое значение тактовой частоты использовать при расчетах. Необходимо расширить компонент до конкретной модели, воспользовавшись функцией ***RI_SDK_sigmod_PWM_ExtendToModel*** или сразу создать компонент ШИМ модулятора конкретной модели с помощью функции ***RI_SDK_CreateModelComponent***
212011	Не установлено соединение с контроллером	Возникает при вызове методов ***RI_SDK_connector_i2c_WriteBytes, RI_SDK_connector_i2c_ReadBytes, RI_SDK_connector_i2c_WriteByte, RI_SDK_connector_i2c_ReadByte***, а также в методах управления компонентами, подключенных к ШИМ модулятору (сервопривод, свтодиод). Означает, что ШИМ модулятор не подключен к i2c адаптеру. Необходимо перед использованием указанных

		выше методов воспользоваться функцией линковки ***RI_SDK_LinkPWMToController***
212012	Соединение с i2c шиной не существует	Возникает при вызове метода ***RI_SDK_sigmod_PWM_Close***. Означает, что соединение с i2c шиной не существует, и следовательно его нельзя закрыть. Рекомендуется не вызывать эту функцию, если ШИМ модулятор не был подключен к i2c адаптеру с помощью функции ***RI_SDK_LinkPWMToController***
212015	Настройка частоты для определенного порта не поддерживается на этом устройстве	Возникает при вызове метода ***RI_SDK_sigmod_PWM_SetPortFreq*** Означает, что для ШИМ модулятора, выбранной Вами модели, не поддерживается установка значения частоты для каждого порта в отдельности Необходимо уточнить в документации и таблице “**Краткое перечисление функций RI SDK”** для каких моделей поддерживается функция ***RI_SDK_sigmod_PWM_SetPortFreq***
212016	Получение частоты для определенного порта, не поддерживаемого на этом устройстве	Возникает при вызове метода ***RI_SDK_sigmod_PWM_GetPortFreq*** Означает, что для ШИМ модулятора, выбранной Вами модели, не поддерживается получение значения частоты для каждого порт в отдельности Необходимо уточнить в документации и таблице “**Краткое перечисление функций RI SDK”** для каких моделей поддерживается функция ***RI_SDK_sigmod_PWM_SetPortFreq***
2.1.2.1. PCA9685		
212101	Нет модели для модулятора PWM	Возникает при вызове метода ***RI_SDK_sigmod_PWM_ExtendToModel*** Означает, что для ШИМ модулятора указанной Вами модели не существует. Необходимо выбрать ту модель, которая поддерживается библиотекой. На данный момент библиотека поддерживает работу только с **pca9685**. Необходимо уточнить в документации и таблице “**Краткое перечисление функций RI SDK”** какие модели ШИМ модулятора поддерживает библиотека
2.2. Исполнители		
2.2.1. Сервоприводы		
221001	Выход за пределы рабочего диапазона	Возникает при вызове методов сервоприводом, управляющим его вращением. Означает, что в такую функцию было передано значение угла, либо управляющего импульса, выходящее за рамки рабочего диапазона данной модели сервопривода либо за рамки установленного рабочего диапазона через функцию инициализации собственного сервопривода ***RI_SDK_exec_ServoDrive_CustomDeviceInit*** Необходимо, либо устанавливать значения углов вращения и управляющего импульса, не выходящие за рамки рабочего диапазона. Либо воспользоваться функцией ***RI_SDK_exec_ServoDrive_CustomDeviceInit*** и установить подходящий для Вас диапазон через

		задание максимального значения импульса и размера рабочего диапазона. Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальный импульс - 2711 мкс Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
221002	Тип полученного направления вращения не определен	Возникает при вызове методов вращения, где необходимо указать направление движения: ***RI_SDK_exec_ServoDrive_MinStepRotate, RI_SDK_exec_ServoDrive_Rotate, RI_SDK_exec_ServoDrive_RotateWithRelativeSpeed*** Означает, что при вызове этих методов было указано неопределенное значение направления движения Поддерживаются следующие значения: 0 - по часовой стрелке 1 - против часовой стрелки Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
221003	Рабочий диапазон не может быть равен нулю или быть отрицательным	Возникает при вызове функции инициализации собственного компонента ***RI_SDK_exec_ServoDrive_CustomDeviceInit*** Означает, что в эту функцию было передано отрицательное значение размера рабочего диапазона или значение равное 0. Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
221004	ШИМ-модулятор не установлен	Возникает при вызове методов сервоприводом, управляющим его вращением. Означает, что сервопривод не был подключен к ШИМ модулятору. Необходимо перед вызовом таких функций воспользоваться функцией линковки ***RI_SDK_LinkServodriveToController***.
221005	Значение максимальной ширины управляющего импульса не установлено	Возникает при вызове методов сервоприводом, управляющим его вращением. Означает, что сервопривод не был расширен до конкретной модели и библиотека не знает, какое значение управляющего импульса необходимо использовать. Необходимо расширить сервопривод до конкретной модели с помощью ***RI_SDK_exec_ServoDrive_ExtendToModel***, либо сразу создавать компонент сервопривода конкретной модели через функцию ***RI_SDK_CreateModelComponent***. Также можно воспользоваться функции инициализации собственного компонента ***RI_SDK_exec_ServoDrive_CustomDeviceInit***. Тогда необходимости расширения до конкретной модели не будет.
221006	Максимальная скорость не установлена	Возникает при вызове методов сервоприводом, управляющим его вращением.

		Означает, что сервопривод не был расширен до конкретной модели и библиотека не знает, какое значение максимальной скорости вращения необходимо использовать. Необходимо расширить сервопривод до конкретной модели с помощью ***RI_SDK_exec_ServoDrive_ExtendToModel***, либо сразу создавать компонент сервопривода конкретной модели через функцию ***RI_SDK_CreateModelComponent***. Также можно воспользоваться функции инициализации собственного компонента ***RI_SDK_exec_ServoDrive_CustomDeviceInit***. Тогда необходимости расширения до конкретной модели не будет.
221009	Скорость вращения больше максимальной	Возникает при вызове методов сервоприводом, управляющим его вращением. Означает, что в эти функции передается значение скорости вращения больше максимальной скорости для данной модели. Необходимо либо передавать скорость не больше максимальной скорости, либо воспользоваться функцией инициализации собственного сервопривода ***RI_SDK_exec_ServoDrive_CustomDeviceInit*** для установки нового значения максимальной скорости. Максимальная скорость для модели ***mg90s*** - 400 градусов в секунду Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
221010	Скорость вращения меньше или равна нулю	Возникает при вызове методов сервоприводом, управляющим его вращением Означает, что в эти функции передается значение скорости вращения меньшее или равное 0. Проверьте, какое значение скорости вы передаете в функции вращения. Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
221011	Выход за пределы рабочего диапазона вращения	Возникает при вызове методов сервоприводом, управляющим его вращением ***RI_SDK_exec_ServoDrive_Turn,*** ***RI_SDK_exec_ServoDrive_MinStepRotate,*** ***RI_SDK_exec_ServoDrive_TurnWithRelativeSpeed*** Означает, что в такую функцию было передано значение угла, которое приводит к выходу за рамки рабочего диапазона вращения (целевой угол становиться больше максимально возможного) для выбранной Вами модели сервопривода RI_SDK_exec_ServoDrive_CustomDeviceInit Необходимо, либо устанавливать значения углов вращения и управляющего импульса, которое не приводит к выходу за рамки рабочего диапазона. Либо воспользоваться функцией ***RI_SDK_exec_ServoDrive_CustomDeviceInit*** и установить подходящий для Вас диапазон через задание максимального значения импульса и размера рабочего диапазона.

		Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальный импульс - 2711 мкс Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
221012	Целевой угол поворота сервопривода меньше нуля	Возникает при вызове методов сервоприводом, управляющим его вращением ***RI_SDK_exec_ServoDrive_Turn,*** ***RI_SDK_exec_ServoDrive_MinStepRotate,*** ***RI_SDK_exec_ServoDrive_TurnWithRelativeSpeed*** Означает, что в такую функцию было передано значение угла, которое приводит к выходу за рамки рабочего диапазона вращения для выбранной Вами модели сервопривода RI_SDK_exec_ServoDrive_CustomDeviceInit Необходимо, либо устанавливать значения углов вращения и управляющего импульса, которое не приводит к выходу за рамки рабочего диапазона. Либо воспользоваться функцией ***RI_SDK_exec_ServoDrive_CustomDeviceInit*** и установить подходящий для Вас диапазон через задание максимального значения импульса и размера рабочего диапазона. Размер рабочего диапазона для ***mg90s*** - 2444 мкс () ~ 219 градусов (от 0 до 219) Максимальный импульс - 2711 мкс Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
221013	Максимальное значение ширины управляющего импульса не может быть меньше размера рабочего диапазона	Возникает при вызове функции инициализации собственного компонента ***RI_SDK_exec_ServoDrive_CustomDeviceInit*** Означает, что в эту функцию было передано значение максимального управляющего импульса меньше размера рабочего диапазона, что означает, что минимальное значение управляющего импульса будет меньше 0. Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
	2.2.1.1. MG90S	
221101	Модель для сервопривода не существует	Возникает при вызове метода ***RI_SDK_exec_ServoDrive_Extend*** Означает, что для сервопривода указанной Вами модели не существует. Необходимо выбрать ту модель, которая поддерживается библиотекой. На данный момент библиотека поддерживает работу только с ***mg90s***. Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
	2.2.2. Светодиод	
222001	Неправильное значение цвета	Возникает при вызове методов управления светодиодом.

		Означает, что значение яркости как минимум одного из трех цветов (красный, зеленый, синий) некорректно. Значение яркости цвета может принимать от 0 до 255. Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**
222002	Не заданы контроллеры для одного из портов подключения светодиода	Возникает при вызове методов управления светодиодом. Означает, что светодиод не был подключен к ШИМ модулятору. Необходимо перед вызовом таких функций воспользоваться функцией линковки ***RI_SDK_LinkLedToController***.
	2.2.2.1. KY016	
222101	Модель для светодиода не существует	Возникает при вызове метода ***RI_SDK_exec_RGB_LED_Extend*** Означает, что для светодиода указанной Вами модели не существует. Необходимо выбрать ту модель, которая поддерживается библиотекой. На данный момент библиотека поддерживает работу только с ***ky016***. Необходимо убедиться, что в аргументах указаны корректные значения, соответствующие документации и таблице “**Краткое перечисление функций RI SDK”**

Список изменений документа

Дата	Изменение	Автор
14.02.22	Создан этот документ.	Юдаев А.А.
25.02.22	Описан раздел “Запуск более сложной программы - пример №3”.	Юдаев А.А.
24.02.22	Создана и заполнена таблица “Краткое перечисление функций RI SDK”	Юдаев А.А. Пчелинцев С.К.
02.03.22	Создана и заполнена таблица “Расшифровка кодов ошибок”	Пчелинцев С.К.
03.03.22	Улучшено описание кодов ошибок и путей их решения	Пчелинцев С.К.
06.09.22	Описаны различия моделей роботов в описаниях к примерам кода.	Юдаев А.А.