Начало работы с LEGO® MINDSTORMS® Education EV3 на ПО MicroPython

BBBBBB *1.0.0*

СОДЕРЖАНИЕ

Данное руководство поможет научиться создавать программы для роботов LEGO® MINDSTORMS® EV3 в приложении MicroPython. Для этого нужно выполнить всего два шага.

- Установка. Сначала необходимо получить и установить все инструменты, необходимые для работы микрокомпьютера EV3 и вашего ПК в среде MicroPython. Вы также узнаете, как включать и выключать микрокомпьютер EV3, а также перемещаться по пунктам меню на экране.
- Создание и запуск программ. Вы узнаете, как создавать программы и загружать их в микрокомпьютер EV3. Вы также узнаете, как запускать программы на своём компьютере или микрокомпьютере EV3.

После запуска первой демонстрационной программы вы сможете попробовать другие примеры программ и начнёте создавать ваши собственные.

ГЛАВА
1

УСТАНОВКА

Этот раздел расскажет о том, какие шаги нужно выполнить, чтобы подготовить и установить все необходимые инструменты для начала программирования в среде MicroPython.

1.1 Что вам потребуется?

Для начала работы вам потребуется следующее оборудование.

- Компьютер под управлением ОС Windows 10 или MacOS
- Доступ в Интернет и права доступа администратора

  Они необходимы только в процессе установки. Для записи и воспроизведения программ специальных прав доступа не требуется.
- Карта памяти microSD

  Вам потребуется карта ёмкостью от 4 до 32 Гбайт. Карты памяти microSD этого типа также известны как microSDHC. Рекомендуется использовать карты с Application Performance Class A1.
- Слот для карт памяти microSD в вашем ПК или устройство чтения карт памяти microSD

  Если ваш компьютер не оснащён слотом (micro) SD, вы можете использовать внешнее USB-устройство чтения карт microSD.
- Кабель mini-USB, аналогичный тому, который входит в комплект поставки Базового набора LME EV3

Стандартная конфигурация данного оборудования изображена на рисунке 1.1.

Рисунок 1.1. Процесс настройки

1.2 Подготовка компьютера

Для создания программ на языке MicroPython вы будете использовать редактор кода Visual Studio Code. Выполните следующие шаги, чтобы загрузить, установить и настроить данное приложение.

1. Загрузите редактор Visual Studio Code.
2. Следуйте инструкциям на экране, чтобы установить приложение.
3. Запустите Visual Studio Code.
4. Откройте вкладку расширений (Extensions tab).
5. Установите расширение EV3 MicroPython, как показано на рисунке 1.2.

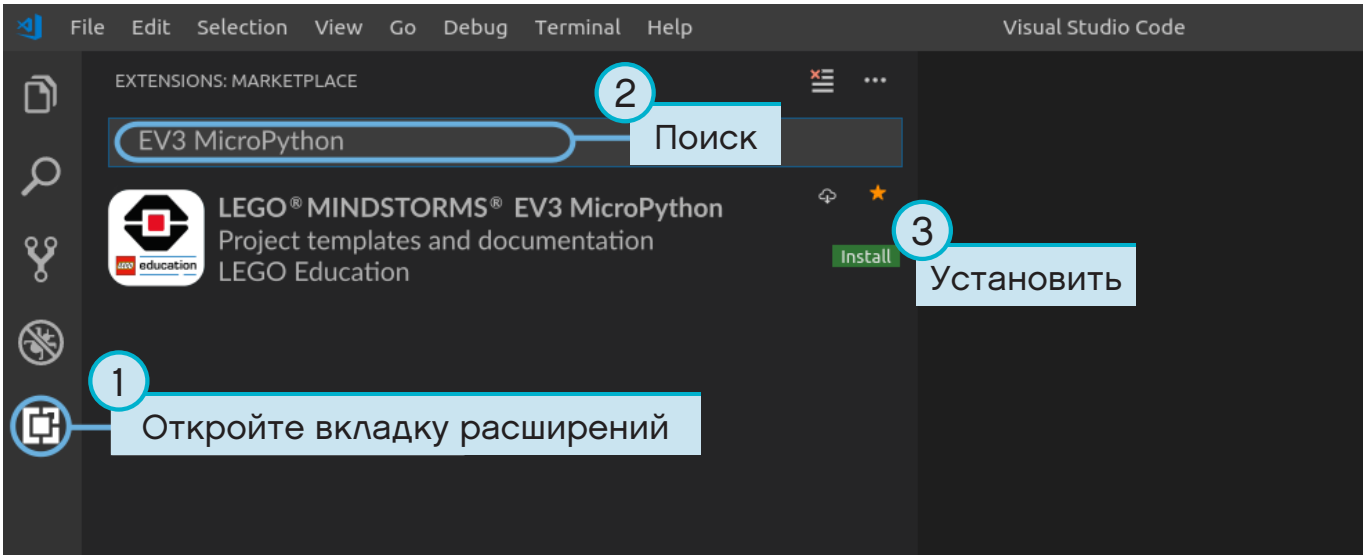


Рисунок 1.2. Установка расширения из каталога Marketplace редактора Visual Studio Code

1.3 Подготовка карты microSD

Для запуска программ MicroPython на микрокомпьютере EV3 необходимо устанавливать специальные инструменты на карту microSD.

Если карта microSD содержит файлы, которые вы хотите сохранить, сначала создайте резервную копию её содержимого. Изучите раздел «Управление файлами на микрокомпьютере EV3», чтобы узнать, как создавать резервные копии предыдущих программ MicroPython.

В процессе установки всё содержимое карты microSD, в том числе предыдущие программы MicroPython, будет удалено.

Для установки инструментов MicroPython на карту памяти microSD выполните следующие действия.

1. Загрузите изображение EV3 MicroPython на карту microSD и сохраните его в удобном месте. Размер файла — приблизительно 360 Мбайт. Распаковывать файл не требуется.
2. Скачайте и установите на карту microSD приложение для создания образов дисков, например Etcher.
3. Вставьте карту microSD в соответствующий слот компьютера или устройство чтения карт.
4. Запустите инструмент для создания образов дисков и следуйте инструкциям на экране, чтобы установить только что загруженный файл с изображением EV3 MicroPython. Если вы используете Etcher, вы можете следовать инструкциям, приведённым на рисунке 1.3.
    a. Выберите файл с изображением EV3 MicroPython, который вы только что загрузили на карту microSD.
    b. Выберите карту microSD. Убедитесь, что указанные тип и ёмкость устройства соответствуют типу и ёмкости вашей карты microSD.
    c. Запустите процесс записи. Это может занять несколько минут. Не извлекайте карту до завершения процесса записи.

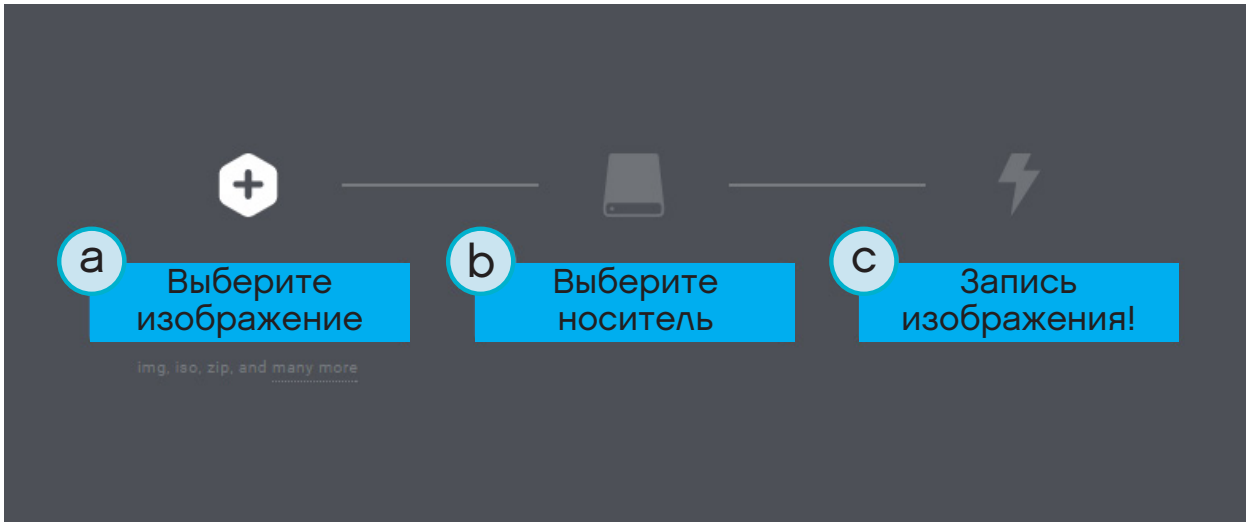


Рисунок 1.3. Использование приложения Etcher для записи изображения EV3 MicroPython на карту microSD

1.4 Обновление карты microSD

Чтобы обновить карту microSD, загрузите новый файл с изображением, перейдя по указанной выше ссылке, и запишите его на карту microSD, как описано выше. Не забудьте создать резервные копии всех программ MicroPython, которые вы хотите сохранить.

Не нужно предварительно удалять содержимое карты microSD. Оно будет удалено автоматически в процессе записи нового изображения.

1.5 Работа с микрокомпьютером EV3

Убедитесь, что микрокомпьютер EV3 выключен. Вставьте подготовленную карту microSD в соответствующий слот микрокомпьютера EV3, как показано на рисунке 1.4.

Рисунок 1.4. Установка карты microSD с записанным изображением в слот микрокомпьютера EV3

1.5.1 Включение и отключение микрокомпьютера EV3

Включите микрокомпьютер EV3, нажав тёмно-серую кнопку в центре.

Процесс загрузки может занять несколько минут. Во время загрузки подсветка микрокомпьютера EV3 будет мигать оранжевым цветом, а на экране будет отображаться текстовое сообщение. Микрокомпьютер EV3 будет готов к работе, когда подсветка его клавиш станет зелёной.

Чтобы отключить микрокомпьютер EV3, откройте раскрывающееся меню с помощью кнопки «Назад», нажав на центральную кнопку микрокомпьютера, затем выберите пункт Power Off, как показано на рисунке 1.5.

Рисунок 1.5. Выключение микрокомпьютера EV3

1.5.2 Просмотр параметров моторов и датчиков

Когда программа не выполняется, вы можете просмотреть статусы подключенных к микрокомпьютеру EV3 моторов и датчиков в браузере устройства, как показано на рисунке 1.6.

Рисунок 1.6. Просмотр параметров моторов и датчиков

1.5.3 Возврат к первоначальной прошивке микрокомпьютера EV3

Вы можете вернуться к встроенному ПО по умолчанию, восстановив предустановленные программы LEGO® в любое время. Для этого выполните следующие действия.

1. Выключите микрокомпьютер EV3, следуя инструкциям выше.
2. Подождите, пока экран и подсветка микрокомпьютера отключатся.
3. Извлеките карту microSD.
4. Включите микрокомпьютер EV3.

ГЛАВА

2

СОЗДАНИЕ И ЗАПУСК ПРОГРАММ

Теперь, когда компьютер и микрокомпьютер EV3 готовы, можно приступать к созданию программ.

Для упрощения процесса создания и управления программами давайте рассмотрим, каким образом организованы проекты и программы MicroPython для роботов EV3.

Программы хранятся в папках проекта, как показано на рисунке 2.1. Папка проекта — директория вашего компьютера, в которой содержится основная программа (**main.py**), а также дополнительные скрипты и файлы. Эта папка проекта, а также всё её содержимое, будет скопирована на микрокомпьютер EV3, на котором будет запущена основная программа.

На следующем рисунке показано, как создать такой проект и как перенести его в микрокомпьютер EV3.

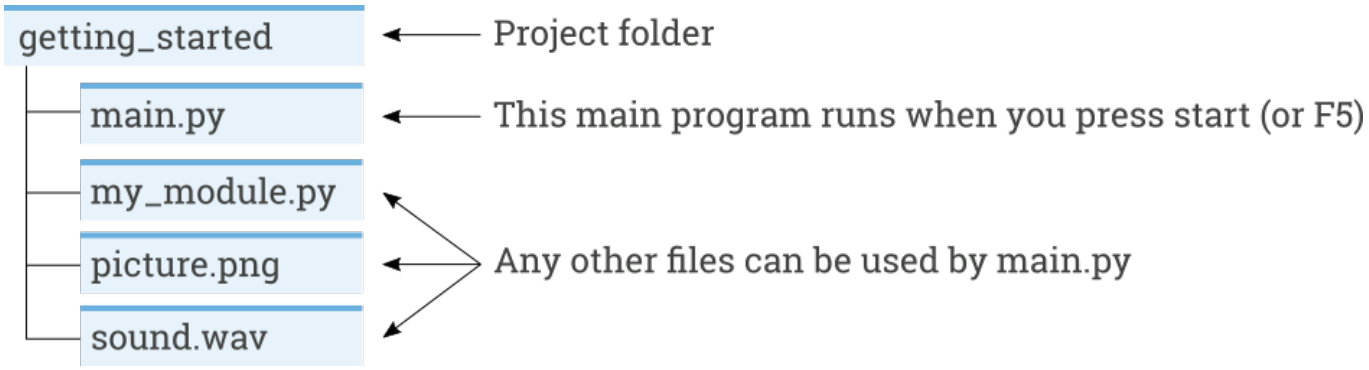


Рисунок 2.1. Папка проекта содержит программу **main.py** и дополнительные ресурсы, такие как файлы звуков или модули MicroPython.

2.1 Создание нового проекта

Чтобы создать новый проект, откройте вкладку EV3 MicroPython и нажмите *Create a new project*, как показано на Рисунке 2.2. Введите имя проекта в появившемся текстовом поле и нажмите *Enter*. При появлении соответствующего запроса укажите папку расположения данной программы и подтвердите свой выбор, нажав *Choose folder*.

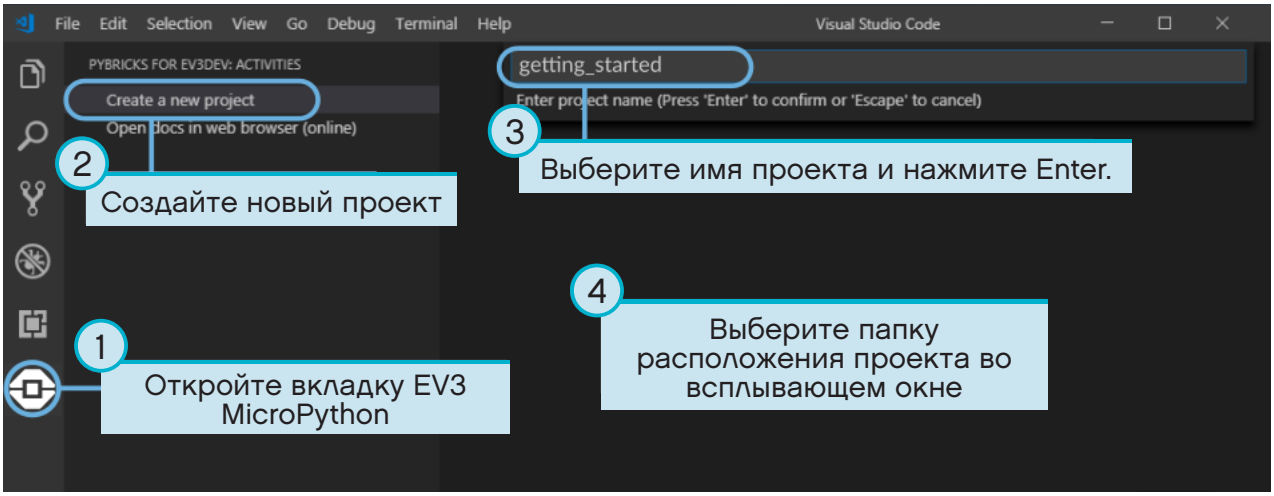


Рисунок 2.2. Создание нового проекта Название этого примера проекта — *getting_started*, но вы можете выбрать любое другое имя.

Новый проект уже включает в себя файл *main.py*. Чтобы посмотреть его содержимое и внести необходимые изменения, откройте этот файл в браузере файлов, как показано на рисунке 2.3. Здесь вы будете создавать свои программы.

Если вы новичок в программировании в среде MicroPython, мы рекомендуем сохранить имеющийся код, добавив в него (по необходимости) свой.

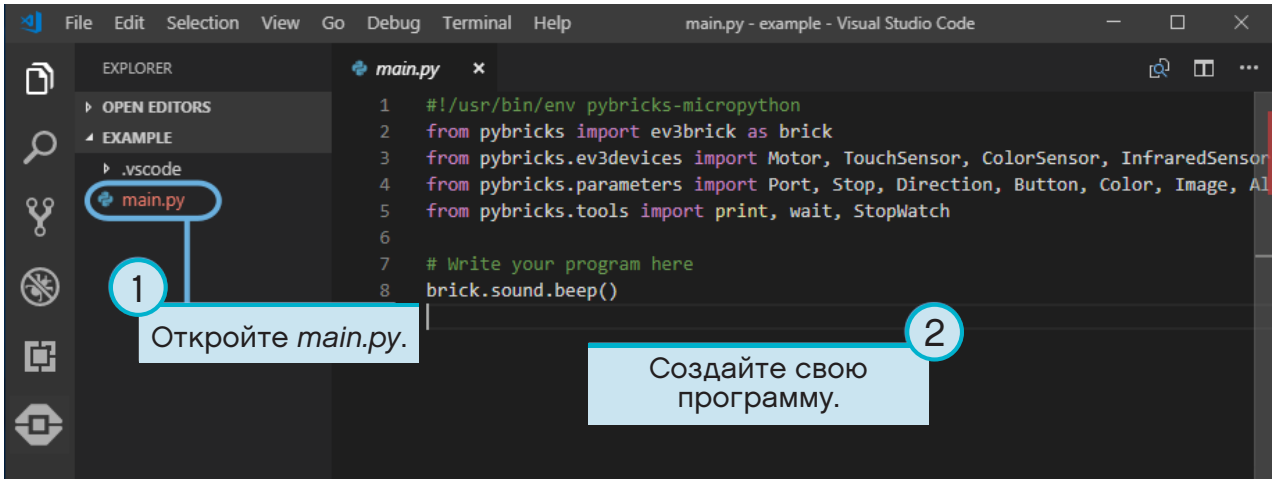


Рисунок 2.3. Открытие файла программы main.py, включённой в проект по умолчанию

2.2 Открытие существующего проекта

Чтобы открыть созданный ранее проект, нажмите *File* и выберите *Open Folder*, как показано на рисунке 2.4. Затем перейдите к папке ранее созданного проекта и нажмите *OK*. Ранее созданные проекты также можно открыть с помощью пункта меню *Open Recent*.

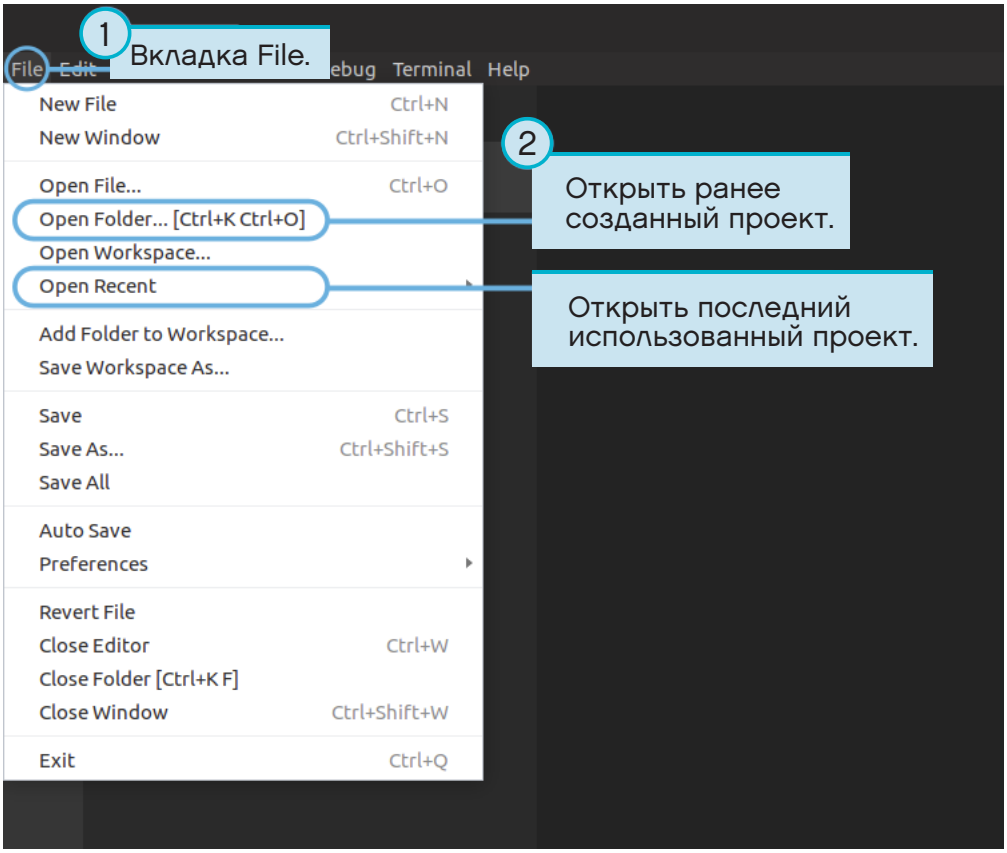


Рисунок 2.4. Открытие ранее созданного проекта

2.3 Подключение к микрокомпьютеру EV3 с помощью редактора кода Visual Studio Code

Чтобы перенести свою программу на модуль EV3, сначала нужно подключить его к компьютеру с помощью кабеля mini-USB и настроить подключение в редакторе Visual Studio Code. Для этого выполните следующие действия.

- Включите микрокомпьютер EV3.
- Подключите микрокомпьютер EV3 к своему компьютеру с помощью кабеля mini-USB.
- Настройте USB-подключение, как показано на рисунке 2.5.

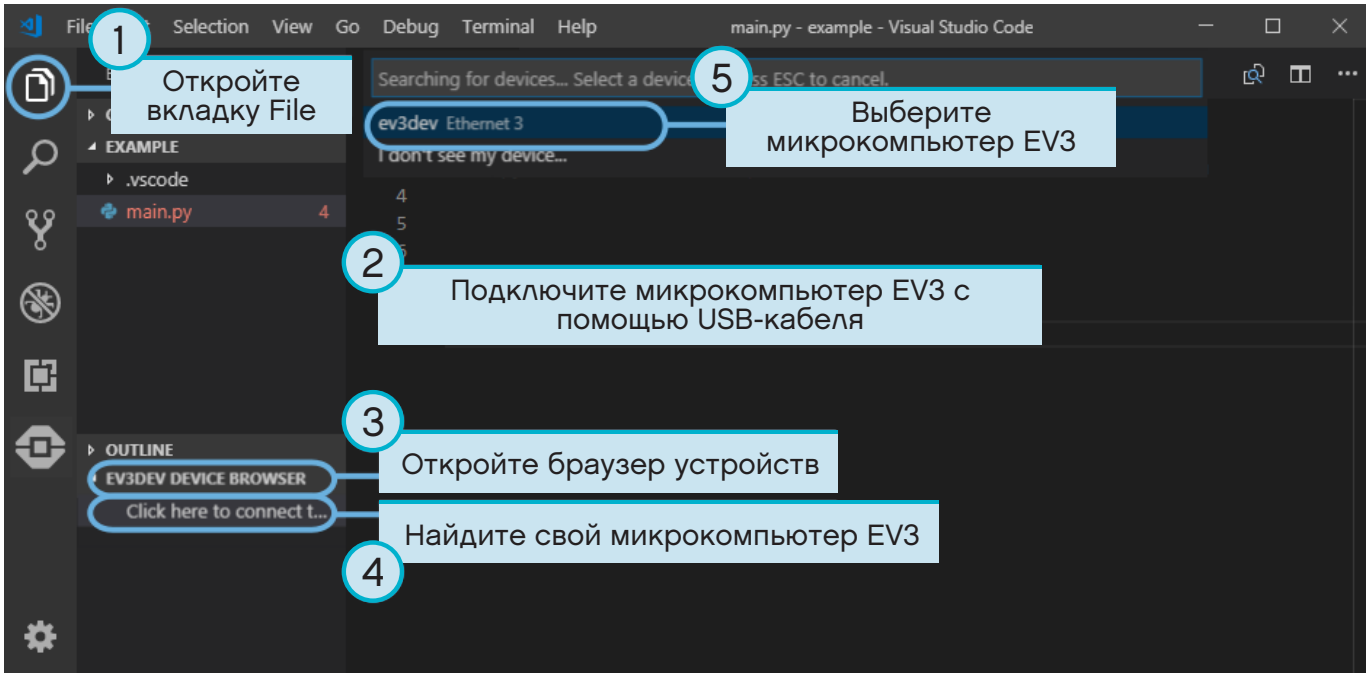


Рисунок 2.5. Настройка USB-подключения микрокомпьютера EV3 к компьютеру

2.4 Загрузка и запуск программы

Для запуска программы нажмите клавишу F5. Вы также можете запустить программу вручную. Для этого перейдите на вкладку отладки и нажмите на зелёную стрелку запуска, как показано на рисунке 2.6.

При необходимости после запуска программы её можно остановить с помощью всплывающей панели инструментов. Программу также можно в любое время остановить, нажав на кнопку «Назад» на модуле EV3.

Если в процессе выполнения программы какая-либо информация выводится на печать с помощью команды *print*, она будет отображаться в окне вывода.

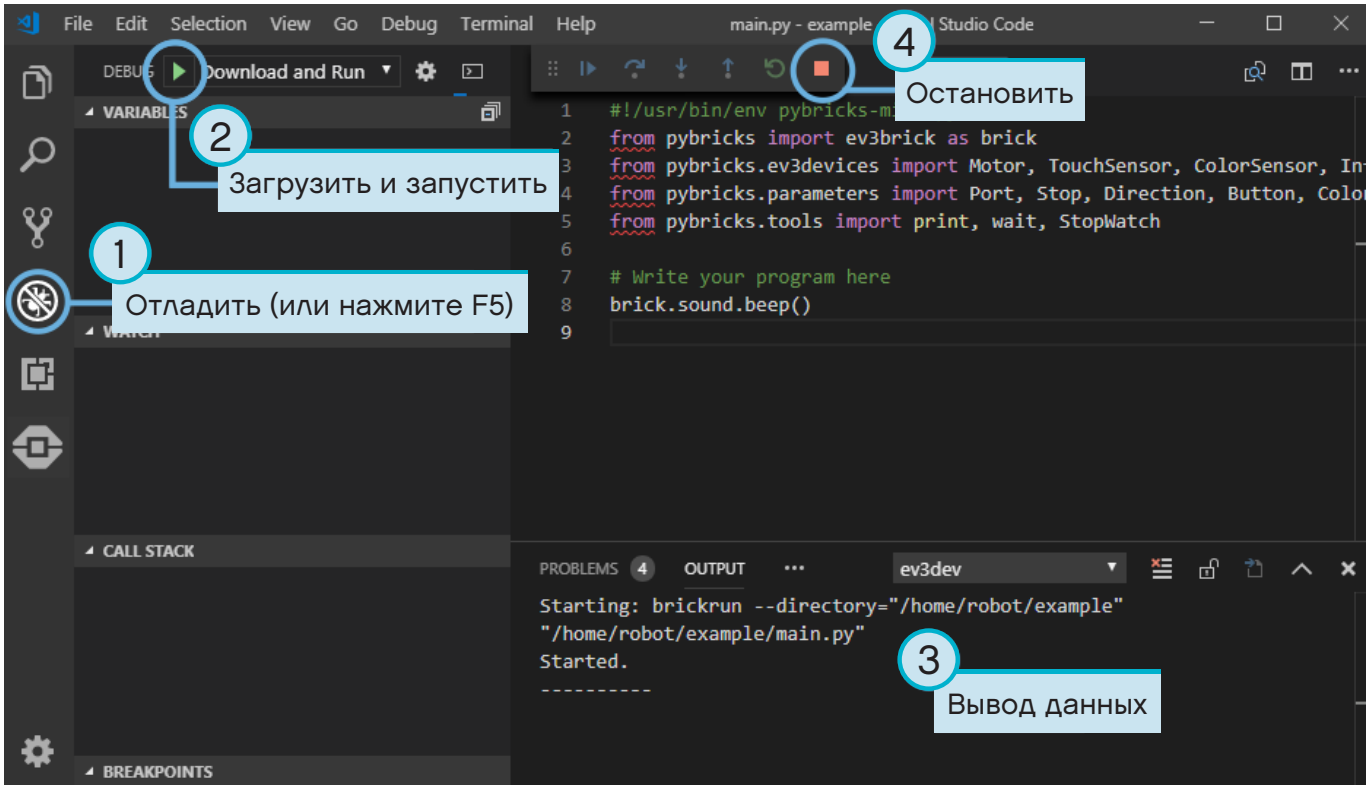


Рисунок 2.6. Запуск программы

2.5 Расширение примера программы

После того как вы запустили шаблон базового кода, вы сможете расширить программу, добавив в неё включение мотора. Сначала подключите Большой мотор к порту B на модуле EV3, как показано на рисунке 2.7.

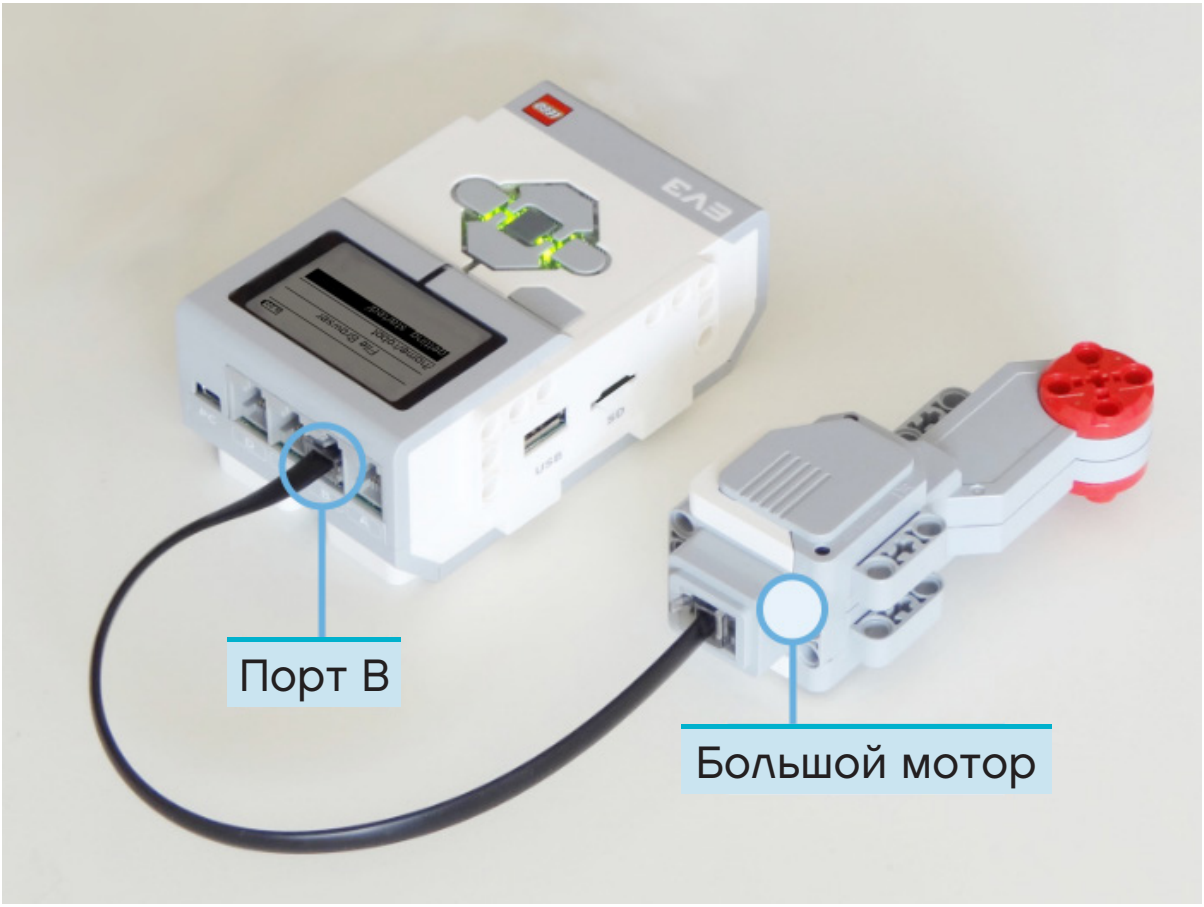


Рисунок 2.7. Микрокомпьютер EV3 с Большим мотором, подключённым к порту B.

Затем измените программу *main.py* так, чтобы она выглядела следующим образом.

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import Motor
from pybricks.parameters import Port

# Play a sound.
brick.sound.beep()

# Initialize a motor at port B.
test_motor = Motor(Port.B)

# Run the motor up to 500 degrees per second. To a target angle of 90 degrees.
test_motor.run_target(500, 90)

# Play another beep sound.
# This time with a higher pitch(1000 Hz) and longer duration(500 ms).
brick.sound.beep(1000, 500)
```

Эта программа заставит твоего робота издать звуковой сигнал, начать вращение мотора и затем снова издать звуковой сигнал более высокого тона. Запустите программу, чтобы убедиться, что она работает должным образом.

2.6 Управление файлами на микрокомпьютере EV3

После загрузки проектов на микрокомпьютер EV3 вы сможете запускать, удалять или копировать сохранённые на нём программы с помощью браузера устройств, как показано на рисунке 2.8.

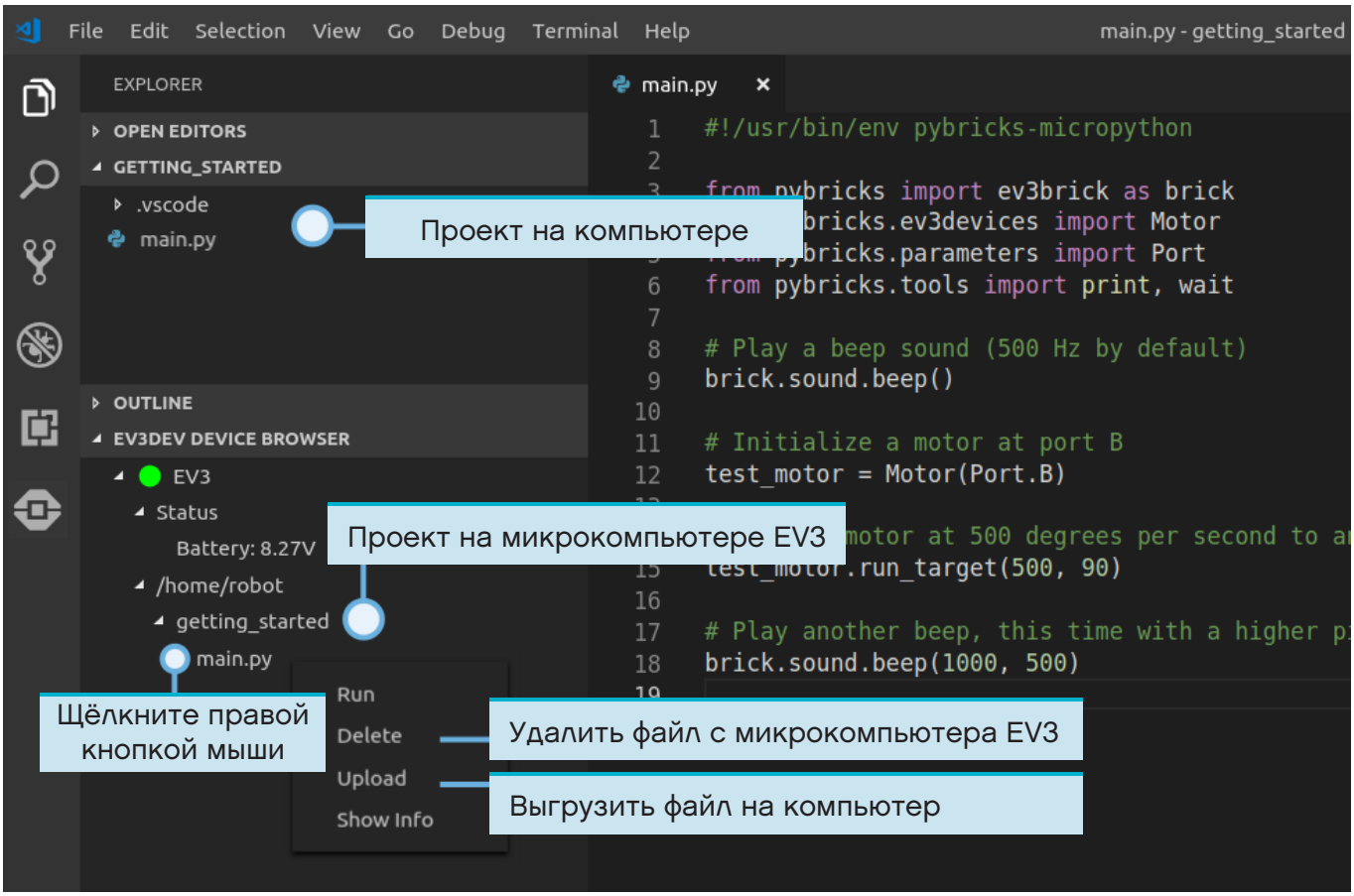


Рисунок 2.8. Использование браузера устройств EV3 для управления файлами на модуле EV3

2.7 Запуск программы без компьютера

Вы можете запустить ранее загруженные программы непосредственно с микрокомпьютера EV3.

Для этого найдите нужную программу в окне *File browser* на экране микрокомпьютера EV3 и нажмите на центральную кнопку микрокомпьютера, чтобы запустить её, как показано на рисунке 2.9.

Рисунок 2.9. Запуск программы с помощью кнопок микрокомпьютера EV3

ГЛАВА

3

МОДУЛЬ EV3 — ПРОГРАММИРУЕМЫЙ МОДУЛЬ EV3

Микрокомпьютер LEGO® MINDSTORMS® EV3

3.1 Кнопки

buttons ()

Проверьте, какие кнопки нажаты на микрокомпьютере EV3.

Returns Список нажатых кнопок.

Return type Список *Button*

Примеры

```
# Do something if the left button is pressed
if Button.LEFT in brick.buttons():
    print("The left button is pressed.")
```

```
# Wait until any of the buttons are pressed
while not any(brick.buttons()):
    wait(10)

# Wait until all buttons are released
while any(brick.buttons()):
    wait(10)
```

3.2 Подсветка

light(*color*)

Выберите цвет подсветки микрокомпьютера.

Parameters **color**(Color) — цвет подсветки. Выберите Color.BLACK или None, чтобы отключить подсветку.

Пример

```
# Make the light red
brick.light(Color.RED)

# Turn the light off
brick.light(None)
```

3.3 Звук

classmethod sound.**beep**(*frequency=500, duration=100, volume=30*)

Воспроизведение сигнала/звука.

Parameters

- **frequency** (*frequency: Hz*) — частота сигнала (*по умолчанию*: 500).
- **duration** (*time: ms*) — длительность сигнала (*по умолчанию*: 100).
- **volume** (*percentage: %*) — громкость сигнала (*по умолчанию*: 30).

Пример

```
# A simple beep
brick.sound.beep()

# A high pitch(1500 Hz) for one second(1000 ms) at 50% volume
brick.sound.beep(1500, 1000, 50)
```

classmethod sound.**beeps**(*number*)

Воспроизведение ряда стандартных звуковых сигналов с короткой паузой между ними.

Parameters number(*int*) — количество сигналов.

Пример

```
# Make 5 simple beeps
brick.sound.beeps(5)
```

classmethod sound.**file**(*file_name, volume=100*)

Воспроизведение звукового файла.

Parameters

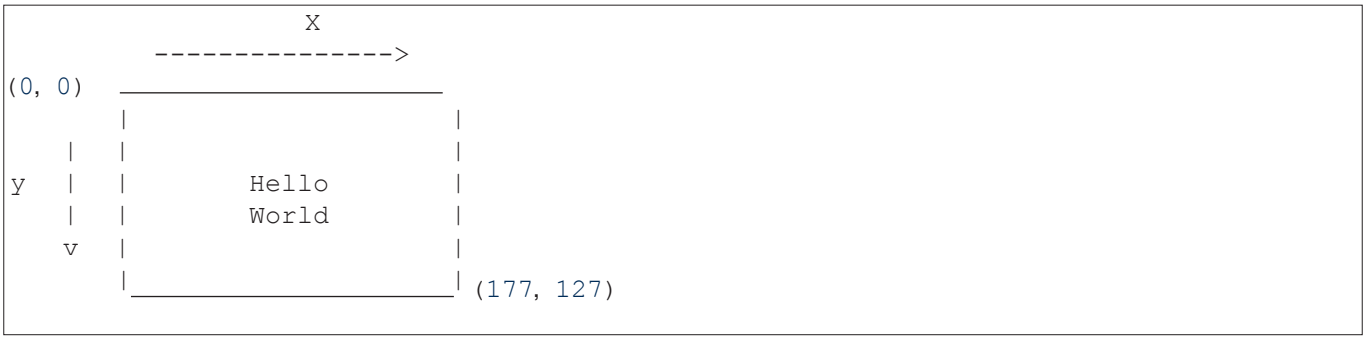
- **file_name** (*str*) — путь к звуковому файлу, включая его расширение.
- **volume** (*percentage: %*) — громкость звука (*по умолчанию*: 100).

Пример

```
# Play one of the built-in sounds
brick.sound.file(SoundFile.HELLO)

# Play a sound file from your project folder
brick.sound.file('mysound.wav')
```

3.4 Экран



classmethod display.**clear**()
Очистка дисплея.

classmethod display.**text**(*text, coordinate=None*)
Отображение текста.

Parameters

- **text**(*str*) — текст, выводимый на экран.
- **coordinate**(*tuple*) — координаты (x, y). Они отображаются в верхнем левом углу первого символа слова. Если координаты не указаны, слово будет напечатано на следующей строке.

Пример

```
# Clear the display
brick.display.clear()

# Print ``Hello`` near the middle of the screen
brick.display.text("Hello",(60, 50))

# Print ``World`` directly underneath it
brick.display.text("World")
```

classmethod display.**image**(*file_name, alignment=Align.CENTER, coordinate=None, clear=True)*
Отображение изображения.

Место размещения изображения на экране можно задать с помощью функции alignment или указав координаты coordinate, но оба способа одновременно использовать нельзя.

Parameters

- **file_name**(*str*) — путь к файлу изображения. В папке проекта вы можете указать полный или относительный путь к файлу изображения.
- **alignment**(Align) — место размещения изображения на экране (*по умолчанию*: Align.CENTER).
- **coordinate**(*tuple*) — координаты (x, y). Они отображаются в верхнем левом углу изображения (*по умолчанию*: None).
- **clear**(*bool*) — очистка экрана перед отображением изображения (*по умолчанию*: True).

Пример

```
# Show a built-in image of two eyes looking upward
brick.display.image(ImageFile.UP)

# Display a custom image from your project folder
brick.display.image('pybricks.png')

# Display a custom image at the top right of the screen, without clearing
# the screen first
brick.display.image('arrow.png', Align.TOP_RIGHT, clear=False)
```

3.5 Батарея

classmethod battery.**voltage**()

Получение данных о напряжении батареи.

Returns Напряжение батареи.

Return type *voltage: mV*

Примеры

```
# Play a warning sound when the battery voltage
# is below 7 Volt(7000 mV = 7V)
if brick.battery.voltage() < 7000:
    brick.sound.beep()
```

classmethod battery.**current**()

Получение данных о силе тока, генерируемого батареей.

Returns Сила тока батареи.

Return type *current: mA*

ГЛАВА
4

УСТРОЙСТВА EV3 — МОТОРЫ И ДАТЧИКИ EV3

Моторы и датчики LEGO® MINDSTORMS® EV3.

4.1 Моторы

class Motor(*port, direction=Direction.CLOCKWISE, gears=None*)

Средний или Большой мотор LEGO® MINDSTORMS® EV3

Устройства под номером 99455/6148292 или 95658/6148278, входящие в наборы

- 31313: LEGO MINDSTORMS EV3 (2013)
- 45544: Базовый набор LEGO MINDSTORMS Education EV3 (2013)
- 45503 или 45502: Поставляемые отдельно устройства (2013)

Parameters

- **port** (`Port`) — порт, к которому подключён мотор.
- **direction** (`Direction`) — положительное направление скорости (*по умолчанию*: Direction.CLOCKWISE).
- **gears** (*`list`*) — список передач, подключённых к мотору (*по умолчанию*: `None`).

  Пример: [`12`, `36`] — зубчатая передача, состоящая из колеса с 12 зубцами и колеса с 36 зубцами. См. передаточное отношение для приведённых примеров.

  Вы можете использовать список различных вариантов зубчатых передач, например: `[[12, 36], [20, 16, 40]]`.

  После выбора зубчатой передачи все команды и настройки моторов будут автоматически скорректированы в соответствии с итоговым передаточным отношением. Направление вращения мотора не зависит от количества добавленных зубчатых передач.

  Например, если передаточное отношение зубчатой передачи `gears=[12, 36]` равно трём, то скорость вращения на выходе будет сокращена в три раза. Для компенсации при выполнении заданной команды мотор автоматически повернётся на угол, значение которого в три раза больше, и в три раза увеличит скорость. Таким образом, если вы выбрали `run_angle(200, 90)`, ваш механизм повернётся на 90 градусов со скоростью 200 град./с.

  Это необходимо учитывать и в следующих главах данного документа. Под словами «угол поворота мотора» или «скорость мотора» следует понимать «угол поворота механизма» и «скорость поворота механизма» и т. д., поскольку передаточное отношение вычисляется автоматически.

  Настройка `gears` доступна только для моторов с датчиками оборотов.

Пример

```
# Initialize a motor(by default this means clockwise, without any gears).
example_motor = Motor(Port.A)

# Initialize a motor where positive speed values should go counterclockwise
right_motor = Motor(Port.B, Direction.COUNTERCLOCKWISE)

# Initialize a motor with a gear train
robot_arm = Motor(Port.C, Direction.CLOCKWISE, [12, 36])
```

Методы управления моторами без датчиков оборотов

dc(*duty*)
Установка цикла работы мотора

Parameters **duty**(*percentage: %*) — цикл работы (от –100,0 до 100).

Пример

```
# Set the motor duty cycle to 75%.
example_motor.duty(75)
```

Методы управления моторами с датчиками оборотов

angle()
Получение данных об угле поворота мотора.

Returns Угол поворота мотора.

Return type *angle: deg*

reset_angle(*angle*)
Сброс накопленного угла поворота мотора.

Parameters **angle**(*angle: deg*) — значение угла поворота, которое должно быть установлено после сброса.

speed()
Получение данных о скорости (угловой скорости) мотора.

Returns Скорость мотора.

Return type *rotational speed: deg/s*

stop(*stop_type=Stop.COAST*)
Остановка мотора.

Parameters **stop_type**(`Stop`) — движение накатом, торможение или фиксация после остановки (*по умолчанию*: `Stop.COAST`).

run(*speed*)
Вращение мотора с постоянной скоростью (угловой скоростью).

Скорость мотора будет увеличиваться до достижения требуемого значения, а цикл работы мотора будет скорректирован таким образом, чтобы скорость оставалась постоянной даже при некоторой нагрузке. Мотор продолжит вращаться в фоновом режиме, пока не будет дана новая команда или программа не будет остановлена.

Parameters **speed**(*rotational speed: deg/s*) — скорость вращения мотора.

run_time(*speed, time, stop_type=Stop.COAST, wait=True*)

Запуск и вращение мотора с постоянной скоростью (угловой скоростью) в течение заданного времени.

Скорость мотора будет увеличиваться до достижения требуемого значения, а цикл работы мотора будет скорректирован таким образом, чтобы скорость оставалась постоянной даже при некоторой нагрузке. Мотор начнёт замедляться за такое время, которое позволит ему остановиться по истечении указанного времени.

Parameters

- **speed** (*rotational speed: deg/s*) — скорость мотора.
- **time** (*time: ms*) — продолжительность манёвра.
- **stop_type** (`Stop`) — движение накатом, торможение или фиксация мотора после его остановки (*по умолчанию*: *`Stop.COAST`*).
- **wait** (*`bool`*) — ожидание завершения манёвра до продолжения выполнения оставшейся части программы (*по умолчанию*: `True`). Это означает, что выполнение программы будет приостановлено на указанное время `time`.

run_angle (*speed, rotation_angle, stop_type=Stop.COAST, wait=True*)
Поворот мотора на заданный угол с постоянной скоростью (угловая скорость).

Скорость мотора будет увеличиваться до достижения требуемого значения, а цикл работы мотора будет скорректирован таким образом, чтобы скорость оставалась постоянной даже при некоторой нагрузке. Мотор начнёт замедляться за такое время, которое позволит ему остановиться после выполнения поворота на определённый угол.

Parameters

- **speed** (*rotational speed: deg/s*) — скорость мотора.
- **rotation_angle** (*angle: deg*) — угол, на который должен повернуться мотор.
- **stop_type** (`Stop`) — движение накатом, торможение или фиксация мотора после его остановки (*по умолчанию*: *`Stop.COAST`*).
- **wait** (*`bool`*) — ожидание завершения манёвра до продолжения выполнения оставшейся части программы (*по умолчанию*: `True`). Это означает, выполнение программы будет приостановлено, пока мотор не выполнит поворот точно на заданный угол.

run_target (*speed, target_angle, stop_type=Stop.COAST, wait=True*)
Поворот мотора на заданный угол с постоянной скоростью (угловая скорость)

Скорость мотора будет увеличиваться до достижения требуемого значения, а цикл работы мотора будет скорректирован таким образом, чтобы скорость оставалась постоянной даже при некоторой нагрузке. Мотор начнёт замедляться за такое время, которое позволит ему остановиться под указанным целевым углом.

Направление вращения выбирается автоматически в зависимости от целевого угла.

Parameters

- **speed** (*rotational speed: deg/s*) — абсолютная скорость мотора. Направление вращения будет выбрано автоматически в зависимости от целевого угла поворота, на него не влияет знак скорости вращения (отрицательная или положительная).
- **target_angle** (*angle: deg*) — целевой угол поворота мотора независимо от его текущего положения.
- **stop_type** (`Stop`) — движение накатом, торможение или фиксация мотора после его остановки (*по умолчанию*: *`Stop.COAST`*).
- **wait** (*`bool`*) — ожидание завершения манёвра до продолжения выполнения оставшейся части программы (*по умолчанию*: `True`). Это означает, что выполнение программы будет приостановлено, пока мотор не выполнит поворот на целевой угол.

Усовершенствованные методы управления моторами с датчиками оборотов

track_target (*target_angle*)

Отслеживание целевого угла, изменяющегося с течением времени

Данная функция похожа на функцию *run_target()*, однако она не требует ввода значений скорости и ускорения: мотор выполнит поворот на целевой угол с максимально допустимой скоростью. Скорость и ускорение корректируются с помощью скорости изменения целевого угла target_angle.

Данный метод удобен при выполнении быстрых циклов, в которых целевой угол поворота мотора постоянно меняется.

Parameters **target_angle** (*angle: deg*) — целевой угол поворота мотора.

Пример

```
# Initialize motor and timer
from math import sin
motor = Motor(Port.A)
watch = StopWatch()
amplitude = 90

# In a fast loop, compute a reference angle
# and make the motor track it.
while True:
    # Get the time in seconds
    seconds = watch.time()/1000
    # Compute a reference angle. This produces
    # a sine wave that makes the motor move
    # smoothly between -90 and +90 degrees.
    angle_now = sin(seconds)*amplitude
    # Make the motor track the given angle
    motor.track_target(angle_now)
```

stalled()

Проверка пробуксовки мотора.

Под пробуксовкой мотора следует понимать ситуацию, в которой мотор не удаётся повернуть даже при максимальном крутящем моменте. Пример такой ситуации — физическая блокировка мотора или механизма, препятствующая их дальнейшему вращению.

Пробуксовка мотора может произойти, например, если цикл работы, вычисленный ПИД-контроллерами, достиг максимального значения (тогда duty = duty_limit), а мотор не может набрать минимальную скорость (тогда speed < stall_speed) по меньшей мере в течение времени stall_time.

Для того чтобы снизить уязвимость мотора перед пробуксовкой, вы можете изменить значения duty_limit, stall_speed и stall_time с помощью настроек *set_dc_settings()* и *set_pid_settings()*.

Returns True, если обнаружена пробуксовка мотора. False, если пробуксовка отсутствует.

Return type bool

run_until_stalled (*speed, stop_type=Stop.COAST, duty_limit=default*)

Вращение мотора с постоянной скоростью (угловой скоростью) до пробуксовки. Под пробуксовкой мотора следует понимать ситуацию, в которой мотор не удаётся повернуть даже при максимальном крутящем моменте. Более точное определение пробуксовки см. в *stalled()*.

Аргумент `duty_limit` позволяет установить максимальное значение крутящего момента мотора на время выполнения данного манёвра. Данная функция позволяет избежать применения полного крутящего момента мотора к зубчатому или рычажному механизму.

Parameters

- **speed** (*rotational speed: deg/s*) — скорость мотора.
- **stop_type** (`Stop`) — движение накатом, торможение или фиксация мотора после его остановки (*по умолчанию*: *`Stop.COAST`*).
- **duty_limit** (*percentage: %*) — относительное максимальное значение крутящего момента. Данная функция напоминает *`set_dc_settings()`*, однако ограничение крутящего момента устанавливается лишь на определённое время: программа вернётся к предыдущему значению крутящего момента после выполнения команды.

set_dc_settings (*duty_limit, duty_offset*)

Выберите соответствующие настройки, чтобы скорректировать выполнение команды *`dc()`*. Выбранные настройки влияют на выполнение других команд `run`, которые используют метод *`dc()`* в фоновом режиме.

Parameters

- **duty_limit** (*percentage: %*) — относительное максимальное значение крутящего момента во время выполнения мотором последовательности команд. Эта функция позволяет выбрать максимальное значение цикла работы мотора, которое будет применено при выполнении всех последующих команд. Она уменьшает максимальный крутящий момент пропорционально указанному абсолютному максимальному значению крутящего момента пробуксовки. Эта функция помогает избежать применения полного крутящего момента мотора к зубчатым или рычажным механизмам LEGO® и их непреднамеренного запуска на максимальной скорости (*по умолчанию*: 100).
- **duty_offset** (*percentage: %*) — минимальный цикл работы при использовании функции *`dc()`*. Эта функция немного увеличивает крутящий момент так, что мотор будет вращаться даже при очень маленьких значениях цикла, что особенно удобно при конструировании собственных управляющих устройств с обратной связью (*по умолчанию*: 0).

set_run_settings (*max_speed, acceleration*)

Конфигурация максимальной скорости и ускорения/замедления мотора для всех команд run.

Указанные параметры применяются к командам мотора `run`, `run_time`, `run_angle`, `run_target` или `run_until_stalled`. См. параметры по умолчанию для каждого мотора.

Parameters

- **max_speed** (*rotational speed: deg/s*) — максимальная скорость мотора в процессе выполнения команды.
- **acceleration** (*rotational acceleration: deg/s/s*) — ускорение до достижения целевой скорости и замедление до остановки. Значение должно быть положительным числом. При необходимости мотор автоматически изменит знак скорости для замедления.

Пример

```
# Set the maximum speed to 200 deg/s and acceleration to 400 deg/s/s.
example_motor.set_run_settings(200, 400)

# Make the motor run for 5 seconds. Even though the speed argument is 300
# deg/s in this example, the motor will move at only 200 deg/s because of
# the settings above.
example_motor.run_time(300, 5000)
```

set_pid_settings (*kp, ki, kd, tight_loop_limit, angle_tolerance, speed_tolerance, stall_speed, stall_time*)

Выбери настройки контроллеров положения и скорости. См. также параметры ПИД и параметры по умолчанию для каждого мотора.

Parameters

- **kp** (*int*) — пропорциональная постоянная функции управления положением (и интегральной скоростью).
- **ki** (*int*) — интегральная постоянная функции управления положением.
- **kd** (*int*) — производная постоянная функции управления положением (и пропорциональной скоростью).
- **tight_loop_limit** (*time: ms*) — если в течение указанного времени после начала выполнения какой-либо команды будет запущена любая команда run, контроллеры воспринимают это как попытку прямого управления скоростью. В таком случае настройки ускорения будут проигнорированы и программа незамедлительно перейдёт к отслеживанию значений скорости, заданных для команды run. Данная функция будет полезна при выполнении быстрых циклов, в которых более предпочтителен быстрый отклик мотора, нежели плавное ускорение (например, при движении робота вдоль линии).
- **angle_tolerance** (*angle: deg*) — допустимое отклонение от целевого угла поворота для признания движения завершённым.
- **speed_tolerance** (*rotational speed: deg/s*) — допустимое отклонение от нулевой скорости для признания движения завершённым.
- **stall_speed** (*rotational speed: deg/s*) — см. *stalled()*.
- **stall_time** (*time: ms*) — см. *stalled()*.

4.2 Датчики

4.2.1 Датчик касания

class TouchSensor (*port*)

Датчик касания LEGO® MINDSTORMS® EV3.

Элемент 95648/6138404, входящий в наборы

- 31313: LEGO MINDSTORMS EV3 (2013)
- 45544: Базовый набор LEGO MINDSTORMS Education EV3 (2013)
- 45507: Поставляемые отдельно устройства (2013)

Parameters **port** (Port) — порт для подключения датчика.

pressed ()

Проверка нажатия на датчик.

Returns True при наличии нажатия. False, если нажатие отсутствует.

Return type bool

4.2.2 Датчик цвета

class ColorSensor (*port*)

Датчик цвета LEGO® MINDSTORMS® EV3.

Элемент 95650/6128869, входящий в наборы

- 31313: LEGO MINDSTORMS EV3 (2013)
- 45544: Базовый набор LEGO MINDSTORMS Education EV3 (2013)
- 45506: Поставляемые отдельно устройства (2013)

Parameters **port** (`Port`) — порт для подключения датчика.

color ()

Определение цвета поверхности.

Returns `Color.BLACK`, `Color.BLUE`, `Color.GREEN`, `Color.YELLOW`, `Color.RED`, `Color.WHITE`, `Color.BROWN` или `None`.

Return type `Color` или `None`, если цвет не определён.

ambient ()

Измерение интенсивности внешнего освещения.

Returns Интенсивность внешнего освещения в диапазоне от 0 (самое тёмное) до 100 (самое яркое).

Return type *percentage: %*

reflection ()

Измерение интенсивности красного света, отражаемого поверхностью.

Returns Интенсивность отражённого света в диапазоне от 0 (нет отражения) до 100 (высокая интенсивность отражённого света).

Return type *percentage: %*

rgb ()

Измерение интенсивности красного, зелёного и синего света, отражаемого поверхностью.

Returns Интенсивность отражённого красного, зелёного и синего света в диапазоне от 0,0 (нет отражения) до 100,0 (высокая интенсивность отраженного света).

Return type набор трёх значений *percentages*

4.2.3 Инфракрасный датчик и маяк

class InfraredSensor (*port*)

Инфракрасный датчик и маяк LEGO® MINDSTORMS® EV3.

Элементы 95654/6132629 и 72156/6127283, входящие в наборы

- 31313: LEGO MINDSTORMS EV3 (2013)
- 45509 и 45508: отдельные части (2013)

Parameters **port** (`Port`) — порт для подключения датчика.

distance ()

Измерение относительного расстояния между датчиком и объектом с помощью инфракрасного излучения.

Returns Относительное расстояние в диапазоне от 0 (минимальное) до 100 (максимальное).

Return type *relative distance: %*

beacon (*channel*)
Измерение относительного расстояния и угла между пультом дистанционного управления и инфракрасным датчиком.

Parameters **channel** (*int*) — номер канала пульта дистанционного управления.

Returns Набор значений относительного расстояния (от 0 до 100) и приблизительного угла (от –75 до 75 градусов) между пультом дистанционного управления и инфракрасным датчиком.

Return type (*relative distance: %, angle: deg*) или (`None`, `None`), если пульт дистанционного управления не обнаружен.

buttons (*channel*)
Проверка нажатия кнопок на пульте дистанционного управления.

Parameters **channel** (*int*) — номер канала пульта дистанционного управления.

Returns Список кнопок, нажатых на пульте дистанционного управления, подключённого к указанному каналу.

Return type Список *Button*

4.2.4 Ультразвуковой датчик

class UltrasonicSensor (*port*)
Ультразвуковой датчик LEGO® MINDSTORMS® EV3.

Элемент 95652/6138403, входящий в наборы

- 45544: Базовый набор LEGO MINDSTORMS Education EV3 (2013)
- 45504: Поставляемые отдельно устройства (2013)

Parameters **port** (`Port`) — порт для подключения датчика.

distance (*silent=False*)
Измерение расстояния между датчиком и объектом с помощью ультразвуковых волн.

Parameters **silent** (*bool*) — выбери `True`, чтобы отключить датчик после измерения расстояния.

Выбери `False`, чтобы оставить датчик включённым (*по умолчанию*).

Если выбрано значение `silent=True`, датчик будет испускать звуковые волны только во время измерения расстояния. Отключение датчика позволяет уменьшить помехи, создаваемые для других ультразвуковых датчиков, однако занимает около 300 мс.

Returns Расстояние (миллиметры).

Return type *distance: mm*

presence ()
Проверка наличия других ультразвуковых датчиков путём обнаружения ультразвуковых волн.

Если другой ультразвуковой датчик работает в беззвучном режиме, его наличие можно обнаружить только тогда, когда он производит измерения.

Returns `True`, если ультразвуковые датчики обнаружены, `False`, если ультразвуковые датчики не обнаружены.

Return type `bool`

4.2.5 Гироскопический датчик

class GyroSensor (*port, direction=Direction.CLOCKWISE*)

Гироскопический датчик LEGO® MINDSTORMS® EV3.

Элемент 99380/6138411, входящий в наборы

- 45544: Базовый набор LEGO MINDSTORMS Education EV3 (2013)
- 45505: Поставляемые отдельно устройства (2013)

Parameters

- **port** (`Port`) — порт для подключения датчика.
- **direction** (`Direction`) — положительное направление вращения относительно красной точки в верхней части датчика (*по умолчанию*: Direction.CLOCKWISE).

speed ()

Определение скорости (угловой скорости) датчика.

Returns Угловая скорость датчика.

Return type *rotational speed: deg/s*

angle ()

Получение суммарного значения угла поворота датчика.

Returns Угол вращения.

Return type *angle: deg*

reset_angle (*angle*)

Выбор значения угла поворота датчика.

Parameters **angle** (*angle: deg*) — значение угла поворота, устанавливаемое после сброса настроек.

ГЛАВА 5

`ПАРАМЕТРЫ` — ПАРАМЕТРЫ И ПОСТОЯННЫЕ

Постоянные параметры/аргументы для Pybricks API.

`class Port`

Порт программируемого модуля EV3.

Порты для подключения моторов

`A`

`B`

`C`

`D`

Порты для подключения датчиков

`S1`

`S2`

`S3`

`S4`

`class Direction`

Направления вращения для положительных значений скорости: по часовой стрелке или против часовой стрелки.

`ПО ЧАСОВОЙ СТРЕЛКЕ`

При положительном значении скорости мотор будет вращаться по часовой стрелке.

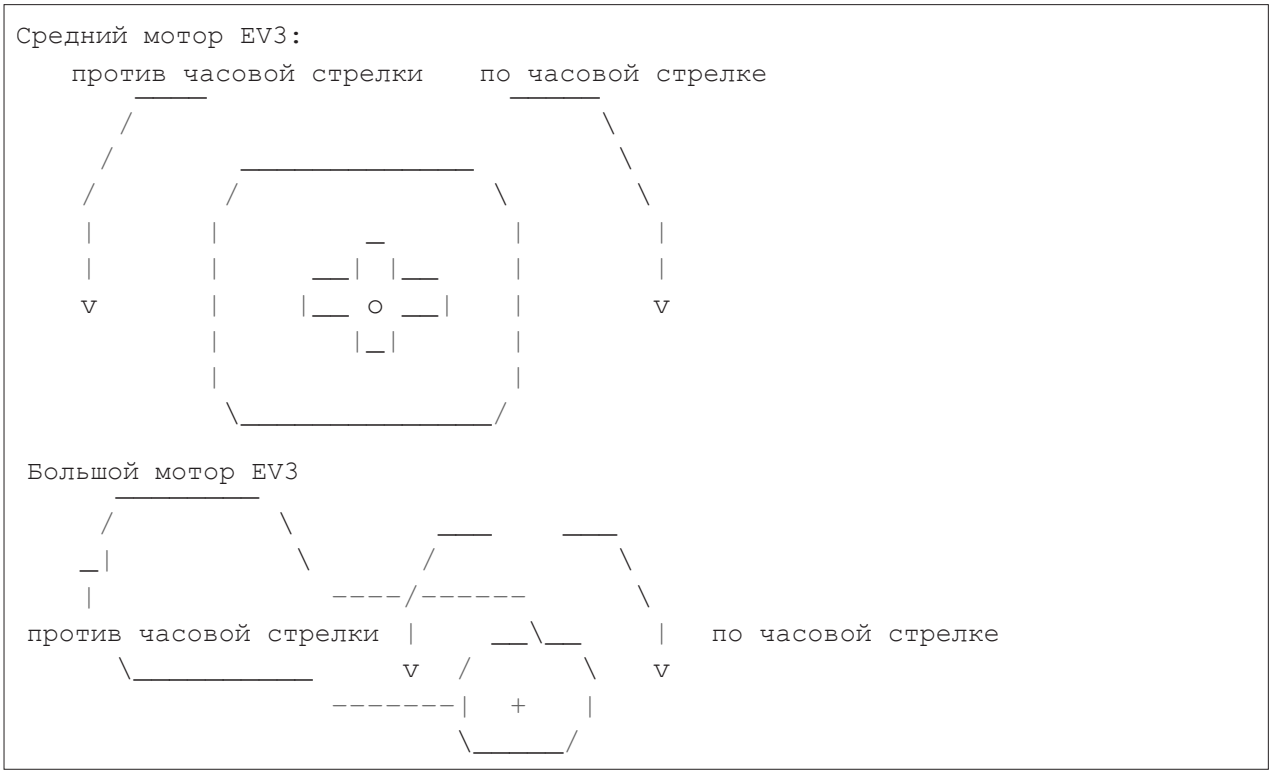
`ПРОТИВ ЧАСОВОЙ СТРЕЛКИ`

При положительном значении скорости мотор будет вращаться против часовой стрелки.

Для всех моторов под направлением вращения подразумевается направление вращения вала, если смотреть на него прямо.

Для моторов NXT или EV3 под направлением вращения подразумевается направление вращения красного/оранжевого вала, расположенного в правой нижней части мотора, если смотреть на него прямо.

Параметр	Положительная скорость	Отрицательная скорость
Direction.CLOCKWISE	clockwise	counterclockwise
Direction.COUNTERCLOCKWISE	counterclockwise	clockwise



class Stop

Действие после остановки мотора: движение накатом, торможение или фиксация.

ДВИЖЕНИЕ НАКАТОМ

Свободное вращение мотора.

ТОРМОЖЕНИЕ

Пассивное сопротивление небольшим внешним воздействиям.

ФИКСАЦИЯ

Управление мотором и фиксация угла поворота, указанного в команде. Данная функция доступна только на моторах с датчиками угла поворота.

Тип действия после остановки определяет характер сопротивления дальнейшему движению.

Параметр	Сопротивление	Физическое значение
Stop.COAST	Низкое	Трение
Stop.BRAKE	Среднее	Трение + крутящий момент, противоположный направлению движения
Stop.HOLD	Высокое	Трение + крутящий момент для фиксации заданного угла поворота мотора

class Color

Цвет света или поверхности.

BLACK

BLUE

GREEN

YELLOW

RED

WHITE

BROWN

ORANGE

`PURPLE`

`class Button`
Кнопки на программируемом модуле или пульте дистанционного управления.

`LEFT_DOWN`

`DOWN`

`RIGHT_DOWN`

`LEFT`

`CENTER`

`RIGHT`

`LEFT_UP`

`UP`

`BEACON`

`RIGHT_UP`

LEFT__UP	UP/BEACON	RIGHT__UP
LEFT	CENTER	RIGHT
LEFT__DOWN	DOWN	RIGHT__DOWN

`class Align`
Выравнивание изображения на экране.

`BOTTOM_LEFT`

`BOTTOM`

`BOTTOM_RIGHT`

`LEFT`

`CENTER`

`RIGHT`

`TOP_LEFT`

`TOP`

`TOP_RIGHT`

`class ImageFile`
Пути к стандартным изображениям EV3.

`Information`

`RIGHT`

`FORWARD`

`ACCEPT`

`QUESTION_MARK`

`STOP_1`

`LEFT`

`DECLINE`

THUMBS_DOWN

BACKWARD

NO_GO

WARNING

STOP_2

THUMBS_UP

LEGO

EV3

EV3_ICON

Предметы

TARGET

Глаза

BOTTOM_RIGHT

BOTTOM_LEFT

EVIL

CRAZY_2

KNOCKED_OUT

PINCHED_RIGHT

WINKING

DIZZY

DOWN

TIRED_MIDDLE

MIDDLE_RIGHT

SLEEPING

MIDDLE_LEFT

TIRED_RIGHT

PINCHED_LEFT

PINCHED_MIDDLE

CRAZY_1

NEUTRAL

AWAKE

UP

TIRED_LEFT

ANGRY

class SoundFile

Пути к стандартным звукам EV3.

Выражения

SHOUTING

CHEERING

CRYING

OUCH

LAUGHING_2

SNEEZING

SMACK

BOING

BOO

UH_OH

SNORING

KUNG_FU

FANFARE

CRUNCHING

MAGIC_WAND

LAUGHING_1

Информация

LEFT

BACKWARDS

RIGHT

OBJECT

COLOR

FLASHING

ERROR

ERROR_ALARM

DOWN

FORWARD

ACTIVATE

SEARCHING

TOUCH

UP

ANALYZE

STOP

DETECTED

TURN

START

Общение

MORNING

EV3

GO

GOOD_JOB

OKEY_DOKEY

GOOD

NO

THANK_YOU

YES

GAME_OVER

OKAY

SORRY

BRAVO

GOODBYE

HI

HELLO

MINDSTORMS

LEGO

FANTASTIC

Движения

SPEED_IDLE

SPEED_DOWN

SPEED_UP

Цвет

BROWN

GREEN

BLACK

WHITE

RED

BLUE

YELLOW

Механические звуки

TICK_TACK

HORN_1

BACKING_ALERT

MOTOR_IDLE

AIR_RELEASE

AIRBRAKE

RATCHET

MOTOR_STOP

HORN_2

LASER

SONAR

MOTOR_START

Животные

INSECT_BUZZ_2

ELEPHANT_CALL

SNAKE_HISS

DOG_BARK_2

DOG_WHINE

INSECT_BUZZ_1

DOG_SNIFF

T_REX_ROAR

INSECT_CHIRP

DOG_GROWL

SNAKE_RATTLE

DOG_BARK_1

CAT_PURR

Числа

ZERO

ONE

TWO

THREE

FOUR

FIVE

SIX

SEVEN

EIGHT

NINE

TEN

Системные

READY

CONFIRM

GENERAL_ALERT

CLICK

OVERPOWER

ГЛАВА
6

ИНСТРУМЕНТЫ — СИНХРОНИЗАЦИЯ И РЕГИСТРАЦИЯ ДАННЫХ

Общие инструменты для синхронизации и регистрации данных.

print (*value, ..., sep, end, file, flush*)

Вывод значений в процессе или после остановки программы.

Пример

```
# Print some text
print("Hello, world")

# Print some text and a number
print("Value:", 5)
```

wait (*time*)

Приостановка выполнения пользовательской программы на указанное время.

Parameters **time** (*time: ms*) — время ожидания.

class StopWatch

Секундомер для измерения временных интервалов. Данная функция аналогична функции «Секундомер» на вашем телефоне.

time ()

Получение значения текущего времени секундомера.

Returns Значение прошедшего времени.

Return type *time: ms*

pause ()

Остановка секундомера.

resume ()

Возобновление отсчёта времени.

reset ()

Сбросить текущее значение секундомера до 0.

Сброс значения не повлияет на работу секундомера:

- если секундомер был остановлен, он не возобновит работу (но значение времени будет равно нулю);
- если секундомер включён, он продолжит работать (снова начнёт отсчёт от нуля).

ГЛАВА 7

РОБОТОТЕХНИКА — МОДУЛЬ РОБОТОТЕХНИКИ

Модуль робототехники для Pybricks API.

class DriveBase (*left_motor, right_motor, wheel_diameter, axle_track*)

Класс конструкторов, представляющий собой роботизированное транспортное средство с двумя колёсами, приводимыми в движение мотором, и дополнительными роликами.

Parameters

- **left_motor** (Motor) — мотор, который приводит в движение левое колесо.
- **right_motor** (Motor) — мотор, который приводит в движение правое колесо.
- **wheel_diameter** (*dimension: mm*) — диаметр колёс.
- **axle_track** (*dimension: mm*) — расстояние между двумя серединными точками двух колёс.

Пример

```
# Initialize two motors and a drive base
left = Motor(Port.B)
right = Motor(Port.C)
robot = DriveBase(left, right, 56, 114)
```

drive (*speed, steering*)

Начало движения с заданными горизонтальной и угловой скоростью, измеренными в центральной точке между двумя колёсами робота.

Parameters

- **speed** (*speed: mm/s*) — горизонтальная скорость робота.
- **steering** (*rotational speed: deg/s*) — угловая скорость робота.

Пример

```
# Initialize two motors and a drive base
left = Motor(Port.B)
right = Motor(Port.C)
robot = DriveBase(left, right, 56, 114)

# Initialize a sensor
sensor = UltrasonicSensor(Port.S4)

# Drive forward until an object is detected
robot.drive(100, 0)
while sensor.distance() > 500:
    wait(10)
robot.stop()
```

drive_time(*speed, steering, time*)

Движение с заданными горизонтальной и угловой скоростью в течение заданного интервала времени и остановка.

Parameters

- **speed**(*speed: mm/s*) — горизонтальная скорость робота.
- **steering**(*rotational speed: deg/s*) — угловая скорость робота.
- **time**(*time: ms*) — продолжительность манёвра.

Пример

```
# Drive forward at 100 mm/s for two seconds
robot.drive(100, 0, 2000)

# Turn at 45 deg/s for three seconds
robot.drive(0, 45, 3000)
```

stop(*stop_type=Stop.COAST*)

Остановка робота.

Parameters **stop_type**(`Stop`) — движение накатом, торможение или фиксация после остановки (*по умолчанию*: `Stop.COAST`).

ГЛАВА

8

СИГНАЛЫ И ЕДИНИЦЫ ИЗМЕРЕНИЯ

Для многих команд вы можете указать аргументы, используя хорошо знакомые физические величины. На этой странице приведён обзор каждой такой величины и единицы её измерения.

8.1 Время: ms

Все значения времени и продолжительности измеряются в миллисекундах (ms).

Например, продолжительность движения, *run_time*, продолжительность *wait* или значения времени, возвращаемые *StopWatch*, указываются в миллисекундах.

8.2 Угол: deg

Все углы измеряются в градусах (deg). Один полный поворот колеса соответствует 360 градусам.

Например, значения угла поворота *Motor* или *GyroSensor* измеряются в градусах.

8.3 Скорость вращения: deg/s

Скорость вращения или угловая скорость показывают, насколько быстро вращается какой-либо предмет. Эта скорость измеряется в градусах в секунду (deg/s).

Например, скорости вращения *Motor* или *GyroSensor* измеряются в градусах в секунду.

Хотя в программах рекомендуется использовать значения в градусах в секунду, их можно перевести в другие широко применяемые единицы измерения с помощью следующей таблицы.

	deg/s	rpm
1 deg/s =	1	1/6=0,167
1 rpm =	6	1

8.4 Расстояние: mm

Если допустимо, расстояния измеряются в миллиметрах (mm).

Например, значение расстояния `UltrasonicSensor` измеряется в миллиметрах.

Хотя в программах рекомендуется использовать значения в миллиметрах, их можно перевести в другие широко применяемые единицы измерения с помощью следующей таблицы.

	mm	cm	inch
1 mm =	1	0,1	0,0394
1 cm =	10	1	0,394
1 inch =	25,4	2,54	1

8.5 Размер: mm

Если допустимо, размеры, как и расстояния, измеряются в миллиметрах (mm).

Например, диаметр колеса измеряется в миллиметрах.

8.6 Относительное расстояние: %

Некоторые расстояния нельзя точно измерить в определённых единицах измерения, тем не менее они определяются в диапазоне от очень маленьких (0 %) до очень больших (100 %). Это также применимо ко всем относительным расстояниям.

Например, расстояние, измеряемое с помощью `InfraredSensor`, является относительным расстоянием.

8.7 Скорость: mm/s

Линейные скорости измеряются в миллиметрах в секунду (mm/s).

Например, скорость роботизированного транспортного средства измеряется в mm/s.

8.8 Угловое ускорение: deg/s/s

Ускорение вращения или *угловое ускорение* показывает, насколько быстро изменяется угловая скорость. Она выражается в виде изменения скорости в градусах в секунду в течение одной секунды (deg/s/s). Эта единица измерения также обычно записывается как deg/s^2.

Например, с помощью значения углового ускорения можно указать, как плавно или как быстро скорость `Мотора` достигает заданного значения постоянной скорости.

8.9 Проценты: %

Для некоторых сигналов не предусмотрены какие-либо определённые единицы измерения, но они определяются в диапазоне от минимальных (0 %) до максимальных (100 %). Значения в процентах определённого типа являются *относительными расстояниями*.

Например, *громкость* звука измеряется в диапазоне от 0 % до 100 %.

8.10 Частота: Hz

Частота звука измеряется в герцах (Hz).

Например, вы можете выбрать частоту *beep*, чтобы изменить высоту звука.

8.11 Напряжение: mV

Напряжение измеряется в милливольтах (mV).

Например, вы можете проверить напряжение *battery.*

8.12 Сила тока: mA

Сила электрического тока измеряется в миллиамперах (mA).

Например, вы можете проверить силу тока, направляемого *battery*.

ГЛАВА

9

РОБОТ-УЧИТЕЛЬ

Выполняя следующий пример программы, робот-учитель (Рисунок 9.1) будет двигаться до тех пор, пока не обнаружит препятствие. Затем он копирует все значения, разворачивается и снова начинает движение.

Инструкции по сборке робота-учителя см. на веб-сайте LEGO Education.

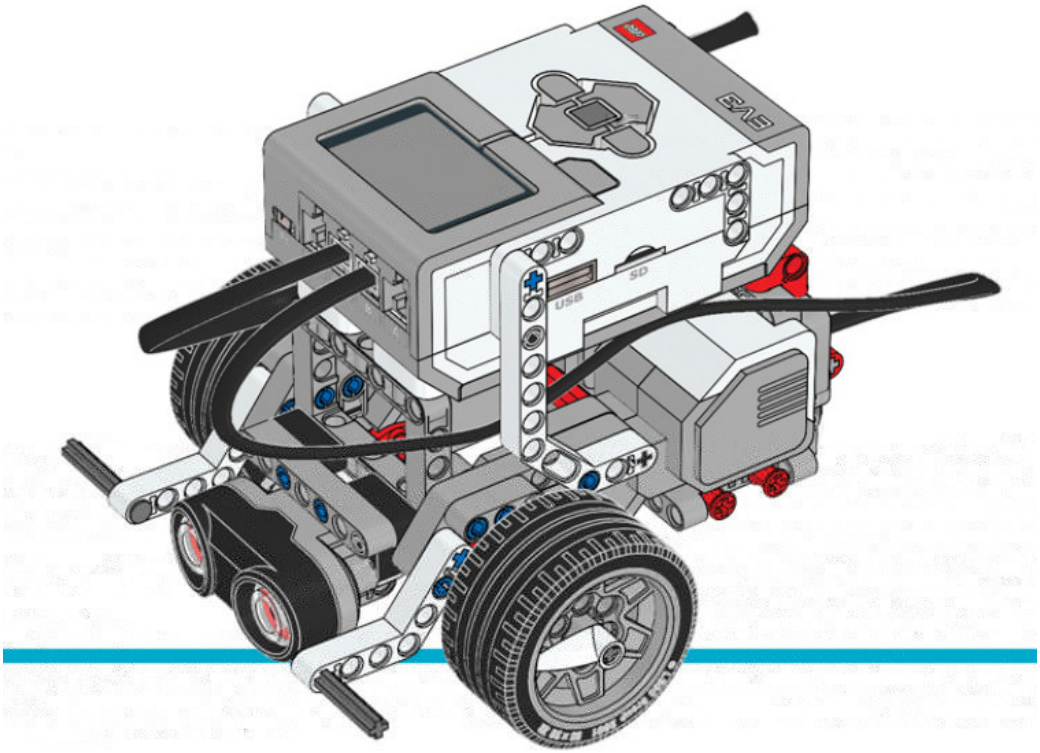


Рисунок 9.1. Робот-учитель с ультразвуковым датчиком.

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import Motor, TouchSensor, ColorSensor
from pybricks.parameters import Port, Button, Color, ImageFile, SoundFile
from pybricks.tools import wait
from pybricks.robotics import DriveBase

# Play a sound.
brick.sound.beep()

# Initialize the Ultrasonic Sensor. It is used to detect
# obstacles as the robot drives around.
obstacle_sensor = UltrasonicSensor(Port.S4)

# Initialize two motors with default settings on Port B and Port C.
# These will be the left and right motors of the drive base.
left_motor = Motor(Port.B)
right_motor = Motor(Port.C)

# The wheel diameter of the Robot Educator is 56 millimeters.
wheel_diameter = 56

# The axle track is the distance between the centers of each of the wheels.
# For the Robot Educator this is 114 millimeters.
axle_track = 114

# The DriveBase is composed of two motors, with a wheel on each motor.
# The wheel_diameter and axle_track values are used to make the motors
# move at the correct speed when you give a motor command.
robot = DriveBase(left_motor, right_motor, wheel_diameter, axle_track)

# The following loop makes the robot drive forward until it detects an
# obstacle. Then it backs up and turns around. It keeps on doing this
# until you stop the program.
while True:
    # Begin driving forward at 200 millimeters per second.
    robot.drive(200, 0)

    # Wait until an obstacle is detected. This is done by repeatedly
    # doing nothing(waiting for 10 milliseconds) while the measured
    # distance is still greater than 300 mm.
    while obstacle_sensor.distance() > 300:
        wait(10)

    # Drive backward at 100 millimeters per second. Keep going for 2 seconds.
    robot.drive_time(-100, 0, 2000)

    # Turn around at 60 degrees per second, around the midpoint between
    # the wheels. Keep going for 2 seconds.
    robot.drive_time(0, 60, 2000)
```

ГЛАВА

10

СОРТИРОВЩИК ЦВЕТОВ

Следующий пример программы для сортировщика цветов (рисунок 10.1) позволяет определять цвет балок Technic с помощью датчика цвета.

Робот по очереди сканирует разноцветные балки и складывает их в лоток. Звуковой сигнал свидетельствует о том, что робот обнаружил заданный цвет. Когда лоток будет полон или пользователь нажмёт на центральную кнопку на модуле EV3, робот начнёт сортировать кубики Technic по цветам.

Инструкции по сборке сортировщика цветов см. на веб-сайте LEGO® Education.

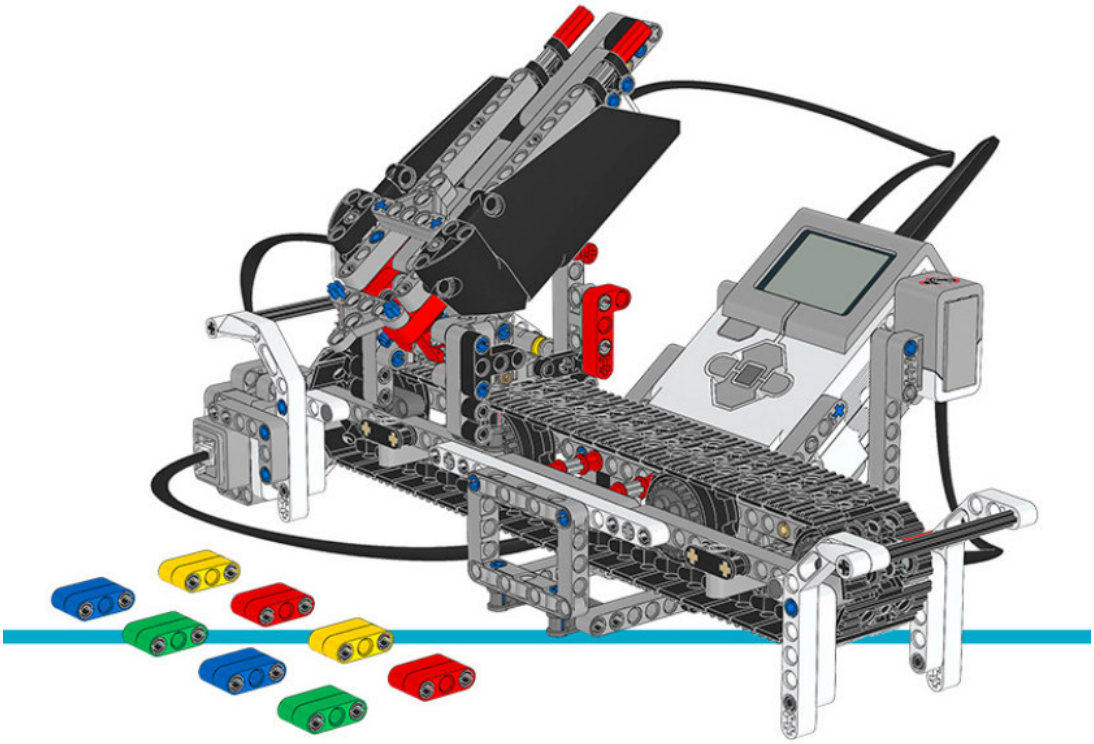


Рисунок 10.1. Сортировщик цветов

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import Motor, TouchSensor, ColorSensor
from pybricks.parameters import Port, Button, Color, ImageFile, SoundFile
from pybricks.tools import wait

# The colored objects are either red, green, blue, or yellow.
POSSIBLE_COLORS =(Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW)

# Initialize the motors that drive the conveyor belt and eject the objects.
belt_motor = Motor(Port.D)
feed_motor = Motor(Port.A)

# Initialize the Touch Sensor. It is used to detect when the belt motor
# has moved the sorter module all the way to the left.
touch_sensor = TouchSensor(Port.S1)

# Initialize the Color Sensor. It is used to detect the color of the objects.
color_sensor = ColorSensor(Port.S3)

# This is the main loop. It waits for you to scan and insert 8 colored objects.
# Then it sorts them by color. Then the process starts over and you can scan
# and insert the next set of colored objects.
while True:
    # Get the feed motor in the correct starting position.
    # This is done by running the motor forward until it stalls. This
    # means that it cannot move any further. From this end point, the motor
    # rotates backward by 180 degrees. Then it is in the starting position.
    feed_motor.run_until_stalled(120)
    feed_motor.run_angle(450, -180)

    # Get the conveyor belt motor in the correct starting position.
    # This is done by first running the belt motor backward until the
    # touch sensor becomes pressed. Then the motor stops, and the the angle is
    # reset to zero. This means that when it rotates backward to zero later
    # on, it returns to this starting position.
    belt_motor.run(-500)
    while not touch_sensor.pressed():
        pass
    belt_motor.stop()
    wait(1000)
    belt_motor.reset_angle(0)

    # Clear all the contents from the display.
    brick.display.clear()

    # When we scan the objects, we store all the color numbers in a list.
    # We start with an empty list. It will grow as we add colors to it.
    color_list = []

    # This loop scans the colors of the objects. It repeats until 8 objects
    # are scanned and placed in the chute. This is done by repeating the loop
    # while the length of the list is still less than 8.
    while len(color_list) < 8:
        # Show an arrow that points to the color sensor.
        brick.display.image(ImageFile.RIGHT)
```

(продолжение на следующей странице)

(начало на предыдущей странице)

```
    # Show how many colored objects we have already scanned.
    brick.display.text(len(color_list))

    # Wait for the center button to be pressed or a color to be scanned.
    while True:
        # Store True if the center button is pressed or False if not.
        pressed = Button.CENTER in brick.buttons()
        # Store the color measured by the Color Sensor.
        color = color_sensor.color()
        # If the center button is pressed or a color is detected,
        # break out of the loop.
        if pressed or color in POSSIBLE_COLORS:
            break

    if pressed:
        # If the button was pressed, end the loop early.
        # We will no longer wait for any remaining objects
        # to be scanned and added to the chute.
        break
    else:
        # Otherwise, a color was scanned.
        # So we add(append) it to the list.
        brick.sound.beep(1000, 100, 100)
        color_list.append(color)

        # We don't want to register the same color once more if we're
        # still looking at the same object. So before we continue, we
        # wait until the sensor no longer sees the object.
        while color_sensor.color() in POSSIBLE_COLORS:
            pass
        brick.sound.beep(2000, 100, 100)

        # Show an arrow pointing to the center button,
        # to ask if we are done.
        brick.display.image(ImageFile.BACKWARD)
        wait(2000)

# Play a sound and show an image to indicate that we are done scanning.
brick.sound.file(SoundFile.READY)
brick.display.image(ImageFile.EV3)

# Now sort the bricks according the list of colors that we stored.
# We do this by going over each color in the list in a loop.
for color in color_list:

    # Wait for one second between each sorting action.
    wait(1000)

    # Run the conveyor belt motor to the right position based on the color.
    if color == Color.BLUE:
        brick.sound.file(SoundFile.BLUE)
        belt_motor.run_target(500, 10)
    elif color == Color.GREEN:
    brick.sound.file(SoundFile.GREEN)
        belt_motor.run_target(500, 132)
```

(продолжение на следующей странице)

(начало на предыдущей странице)

```
    elif color == Color.YELLOW:
        brick.sound.file(SoundFile.YELLOW)
        belt_motor.run_target(500, 360)
    elif color == Color.RED:
        brick.sound.file(SoundFile.RED)
        belt_motor.run_target(500, 530)

    # Now that the conveyor belt is in the correct position,
    # eject the colored object.
    feed_motor.run_angle(1500, 90)
    feed_motor.run_angle(1500, -90)
```

ГЛАВА

11

РОБОТИЗИРОВАННАЯ РУКА H25

Следующий пример программы запускает бесконечный цикл перемещения штабелированных чёрных ступиц колёс. Роботизированная рука включается, а затем начинает перемещение втулок колёс.

Инструкции по сборке этого робота см. на веб-сайте LEGO Education.

Совет. При сборке робота поверните модуль EV3 так, чтобы обеспечить свободный доступ к microSD-карте.

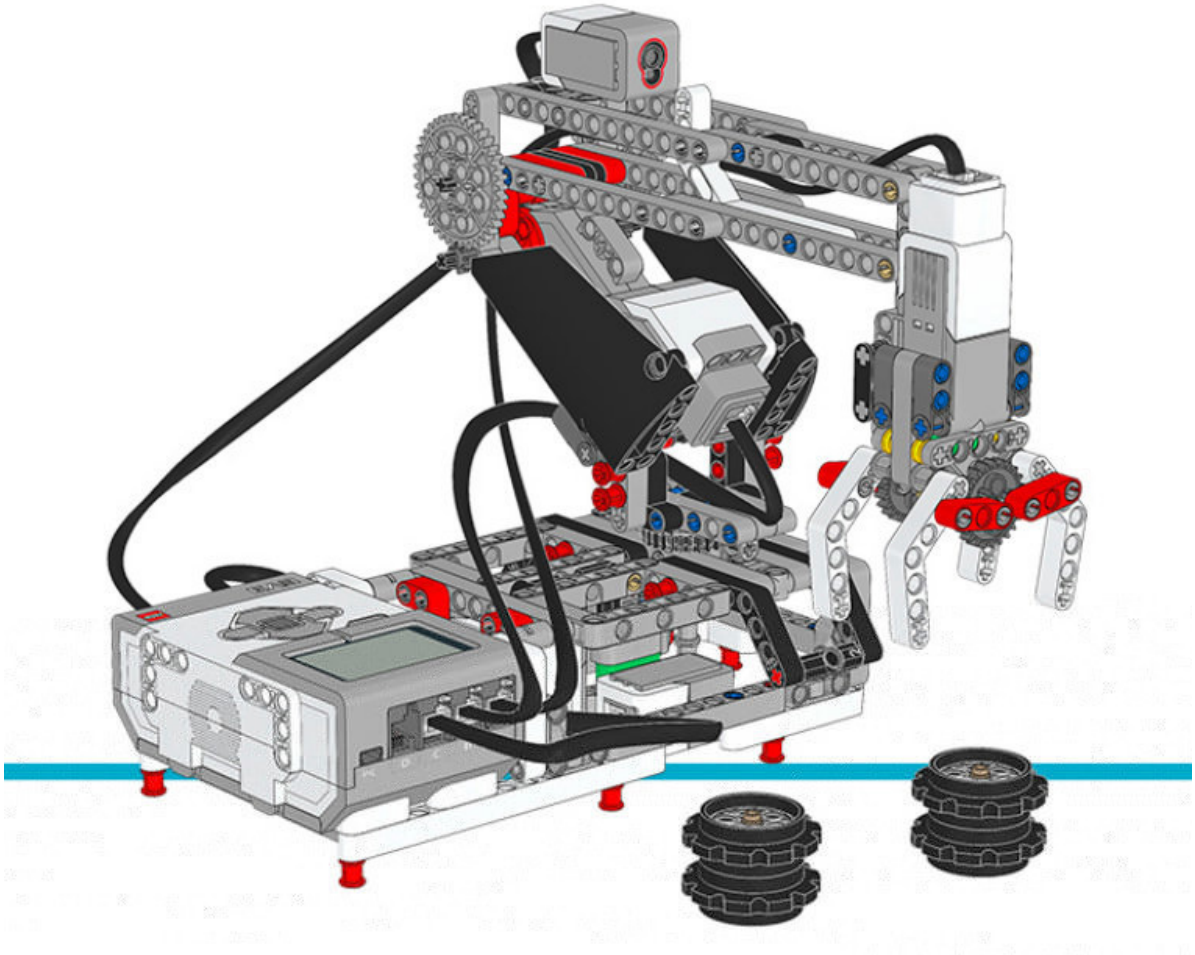


Рисунок 11.1. Роботизированная рука H25

(начало на предыдущей странице)

```
#!/usr/bin/env pybricks-micropython

from pybricks import ev3brick as brick
from pybricks.ev3devices import Motor, TouchSensor, ColorSensor
from pybricks.parameters import Port, Stop, Direction
from pybricks.tools import wait

# Configure the gripper motor on Port A with default settings.
gripper_motor = Motor(Port.A)

# Configure the elbow motor. It has an 8-teeth and a 40-teeth gear
# connected to it. We would like positive speed values to make the
# arm go upward. This corresponds to counterclockwise rotation
# of the motor.
elbow_motor = Motor(Port.B, Direction.COUNTERCLOCKWISE, [8, 40])

# Configure the motor that rotates the base. It has a 12-teeth and a
# 36-teeth gear connected to it. We would like positive speed values
# to make the arm go away from the Touch Sensor. This corresponds
# to counterclockwise rotation of the motor.
base_motor = Motor(Port.C, Direction.COUNTERCLOCKWISE, [12, 36])

# Limit the elbow and base accelerations. This results in
# very smooth motion. Like an industrial robot.
elbow_motor.set_run_settings(60, 120)
base_motor.set_run_settings(60, 120)

# Set up the Touch Sensor. It acts as an end-switch in the base
# of the robot arm. It defines the starting point of the base.
base_switch = TouchSensor(Port.S1)

# Set up the Color Sensor. This sensor detects when the elbow
# is in the starting position. This is when the sensor sees the
# white beam up close.
elbow_sensor = ColorSensor(Port.S3)

# Initialize the elbow. First make it go down for one second.
# Then make it go upwards slowly(15 degrees per second) until
# the Color Sensor detects the white beam. Then reset the motor
# angle to make this the zero point. Finally, hold the motor
# in place so it does not move.
elbow_motor.run_time(-30, 1000)
elbow_motor.run(15)
while elbow_sensor.reflection() < 32:
    wait(10)
elbow_motor.reset_angle(0)
elbow_motor.stop(Stop.HOLD)

# Initialize the base. First rotate it until the Touch Sensor
# in the base is pressed. Reset the motor angle to make this
# the zero point. Then hold the motor in place so it does not move.
base_motor.run(-60)
while not base_switch.pressed():
    wait(10)
base_motor.reset_angle(0)
base_motor.stop(Stop.HOLD)
```

(продолжение на следующей странице)

(начало на предыдущей странице)

```
# Initialize the gripper. First rotate the motor until it stalls.
# Stalling means that it cannot move any further. This position
# corresponds to the closed position. Then rotate the motor
# by 90 degrees such that the gripper is open.
gripper_motor.run_until_stalled(200, Stop.COAST, 50)
gripper_motor.reset_angle(0)
gripper_motor.run_target(200, -90)

def robot_pick(position):
    # This function makes the robot base rotate to the indicated
    # position. There it lowers the elbow, closes the gripper, and
    # raises the elbow to pick up the object.

    # Rotate to the pick-up position.
    base_motor.run_target(60, position, Stop.HOLD)
    # Lower the arm.
    elbow_motor.run_target(60, -40)
    # Close the gripper to grab the wheel stack.
    gripper_motor.run_until_stalled(200, Stop.HOLD, 50)
    # Raise the arm to lift the wheel stack.
    elbow_motor.run_target(60, 0, Stop.HOLD)

def robot_release(position):
    # This function makes the robot base rotate to the indicated
    # position. There it lowers the elbow, opens the gripper to
    # release the object. Then it raises its arm again.

    # Rotate to the drop-off position.
    base_motor.run_target(60, position, Stop.HOLD)
    # Lower the arm to put the wheel stack on the ground.
    elbow_motor.run_target(60, -40)
    # Open the gripper to release the wheel stack.
    gripper_motor.run_target(200, -90)
    # Raise the arm.
    elbow_motor.run_target(60, 0, Stop.HOLD)

# Play three beeps to indicate that the initialization is complete.
brick.sound.beeps(3)

# Define the three destinations for picking up and moving the wheel stacks.
LEFT = 160
MIDDLE = 100
RIGHT = 40

# This is the main part of the program. It is a loop that repeats endlessly.
#
# First, the robot moves the object on the left towards the middle.
# Second, the robot moves the object on the right towards the left.
# Finally, the robot moves the object that is now in the middle, to the right.
#
# Now we have a wheel stack on the left and on the right as before, but they
# have switched places. Then the loop repeats to do this over and over.
while True:
```

(продолжение на следующей странице)

(начало на предыдущей странице)

```
    # Move a wheel stack from the left to the middle.
    robot_pick(LEFT)
    robot_release(MIDDLE)

    # Move a wheel stack from the right to the left.
    robot_pick(RIGHT)
    robot_release(LEFT)

    # Move a wheel stack from the middle to the right.
    robot_pick(MIDDLE)
    robot_release(RIGHT)
```

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ ПО МОДУЛЮ PYTHON

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

T

U

V

W