Microbit-Русская документация

Выпуск 0.1

Maxim Bekurin

июл. 22, 2022

Эта документация включает уроки для учителей и документацию по работе с основными датчиками, с которыми вы можете столкнуться при обучении детей. Надеемся, Вам понравится работать с BBC micro:bit с использованием MicroPython и нашим вспомогательным материалом.

Глава 1

Содержание:

1.1 Инструкция

Мы ипользуем редактор кода Mu editor. Поддержка Microbit встроена в редактор и пользоваться им удобно.

После установки Mu подключите micro:bit к компьютеру через USB-кабель.

Напишите свой код в окне редактора и нажмите кнопку «Flash» для переноса его на Microbit. Если код не загрузился, убедитесь что он отображается как USB-накопитель в проводнике файлов системы (имя: Microbit)

Python — популярный язык программирования для самых разных компаний и организаций. Но знаете ли Вы, что можете использовать Python для написания кода для Микроконтроллеров. Использовать в Вашей повседневной жизни собирая умные устройства, которое состоит из датчиков, моторов, светодиодов, динамиков и других электронных устройств.

Один из таких микроконтроллеров Microbit. Он использует версию Python под названием MicroPython. Вы также сможете использовать свои программы и на других устройствах, например Raspberry Pi.

MicroPython не включает все стандартные библиотеки кода, поставляемые с «обычный» Python. Используется специальный модуль `microbit` в MicroPython, который позволяет вам управлять устройством.

Python и MicroPython — бесплатное программное обеспечение.Вы можете внести свой вклад в Python сообщество. Это может быть код, документация, отчеты об ошибках, группа сообщества или написание учебных пособий (например, это). На самом деле, все Python соответствующие ресурсы для BBC micro:bit были созданы международным команда волонтеров, работающих в свободное время.

Документация знакомят Вас с MicroPython и BBC. Не стесняйтесь принимать и адаптировать ее для своих уроках в классе и при создании умных устройств.

Вы добьетесь наибольшего успеха, если будете исследовать, экспериментировать и создавать игры. Если Вы написали неправильный код - это не сломает микроконтроллер ;)

Удачи в экспериментах!

1.2 Первая программа Hello, World!

Исторически принято написать первую программу, «Hello, World!».

Напишите код:

```
from microbit import *
display.scroll("Hello, World!")
```

Первая строка кода:

```
from microbit import *
```

Подключает библиотеку MicroPython для Microbit. Указав `*`, Вы подключаете все команды библиотеки. При первых программах так делать проще.

Вторая строка:

```
display.scroll("Hello, World!")
```

. . . Указывает MicroPython использовать дисплей как бегущую строку В скобках в кавычках указывается строка вывода «Hello, World!». К сожалению русские буквы так выводить нельзя. Давайте подробнее рассмотрим строку.

- `display` - объект, которому относится команда.
- `scroll` - команда, у команды необходимо указывать скобки `()`
- `"Hello, World!"` - строка- аргумент команды

Измените строку убрав двойные ковычки `Hello, World!` и нажмите кнопку `Flash`.

Предупреждение: Если Вы совершили ошибку MicroPython отобразит номер строки и название ошибки на Microbit. К сожалению это не очень удобно! Но номер строки Вам поможет ориентироваться в коде

Python ожидает, что вы наберете **ТОЧНО** правильный текст. Слова, `Microbit`, `microbit` и `microBit` в программирование - это разные вещи. Поэтому придерживайтесь правильного **регистра написания команд**

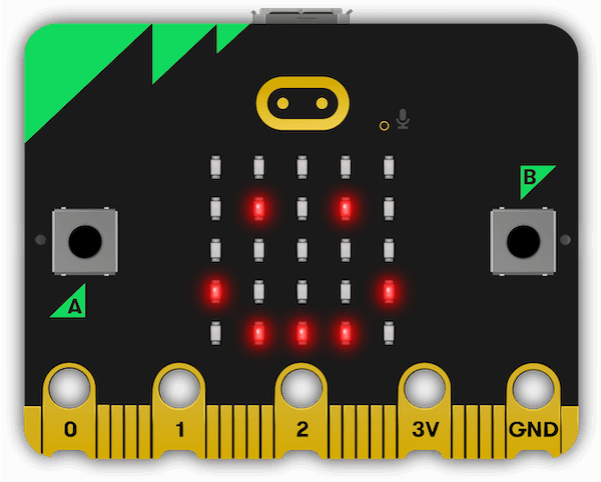
1.3 Изображения

Microbit имеет дисплей красных светодиодов 5x5. Вы можете выводить **изображение** на него и создавать интересную **анимацию**.

MicroPython имеет много встроенных изображений например «Счастливый смайлик»:

```
from microbit import *
display.show(Image.HAPPY)
```

Давайте вспомним что делает первая строка? Вторая строка использует команду объекта `display` под названием `show`. «Счастливый смайлик» является объектом типа (категории) `Image` и называется `HAPPY`. **Обратите внимание название указывается заглавными буквами**



Остальные встроенные изображения:

- `Image.HEART` - Большое сердце
- `Image.HEART_SMALL` - Малое сердце
- `Image.HAPPY` - Счастливый смайлик
- `Image.SMILE` - Улыбка
- `Image.SAD` - Грустный смайлик
- `Image.CONFUSED` - Смущенный смайлик
- `Image.ANGRY` - Злой смайлик
- `Image.ASLEEP` - Спящий смайлик
- `Image.SURPRISED` - Удивленный смайлик
- `Image.SILLY` - Глупый смайлик
- `Image.FABULOUS` - Воодушевленный смайлик
- `Image.MEH` -
- `Image.YES` - Да
- `Image.NO` - Нет
- `Image.CLOCK1` - ... - `Image.CLOCK12` - Стрелка часов
- `Image.ARROW_N`, `Image.ARROW_NE`, `Image.ARROW_E`, `Image.ARROW_SE`, `Image.ARROW_S`, `Image.ARROW_SW`, `Image.ARROW_W`, `Image.ARROW_NW` - Стороны Света
- `Image.TRIANGLE` - Треугольник смотрит вверх
- `Image.TRIANGLE_LEFT` - Треугольник смотрит влево
- `Image.CHESSBOARD` - Шахматная доска
- `Image.DIAMOND` - Алмаз
- `Image.DIAMOND_SMALL` - Маленький алмаз
- `Image.SQUARE` - Квадрат
- `Image.SQUARE_SMALL` - Маленький квадрат
- `Image.RABBIT` - Кролик

- `Image.COW` - Корова
- `Image.MUSIC_CROTCHET` - Знак Ноты
- `Image.MUSIC_QUAVER` - Дрожание
- `Image.MUSIC_QUAVERS` - Дрожание
- `Image.PITCHFORK` - Камертон (вилка)
- `Image.XMAS` - Рождество
- `Image.PACMAN` - Пакмен
- `Image.TARGET` - Цель
- `Image.TSHIRT` - Футболка
- `Image.ROLLERSKATE` - Коньки
- `Image.DUCK` - Утка
- `Image.HOUSE` - Дом
- `Image.TORTOISE` - Черепаха
- `Image.BUTTERFLY` - Бабочка
- `Image.STICKFIGURE` - Фигура (рисунок)
- `Image.GHOST` - Призрак
- `Image.SWORD` - Меч
- `Image.GIRAFFE` - Жираф
- `Image.SKULL` - Череп
- `Image.UMBRELLA` - Зонтик
- `Image.SNAKE` - Змея

Попробуйте различные изображения, указав команды друг в столбик. Измените аргумент с картинкой внутри команды:

```
display.show(Image.HAPPY)
display.show(Image.ANGRY)
display.show(Image.SKULL)
```

Далее Вы узнаете как эффективнее записать данный код

1.3.1 Собственные изображения

Давайте создадим собственное изображение

Создается таблица `5 на 5` из цифр 0-9. Если указан `0` светодиод выключен. Максимальная яркость `9`. Промежуточная яркость от `1` to `8`

Вот наш код:

```
from microbit import *

boat = Image("02469:"
             "02469:"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
              "02469:"
              "02469:"
              "02469")

display.show(boat)
```

На дисплее отобразится шкала яркости

Напишите код и попробуйте запрограммировать Microbit. Получилось !!! Давайте обсуждать дальше. Вы создали объект `boat` - типа `Image` и указали в скобках параметр, который состоит из 5 строк. Каждая строка заканчивается `:`. Итого у Вас 5 строк по 5 цифр.

Далее Вы подали объект в команду `show`. **Имя объекта пишется без «»**

Вы можете записать все и в одну строку, но так менее понятно:

```
boat = Image("05050:05050:05050:99999:09990")
display.show(boat)
```

1.3.2 Анимация

У Вас выше уже был опыт создания анимации (смена изображений). Теперь найчимся делать это эффективнее.

Для этого Вы можете использовать `список`, в который укажите перечень изображений. В Python список указвается квадратными скобками (`[` и `]`) и элементы разделены запятой (`,`):

```
face = [Image.HAPPY, Image.ANGRY, Image.SKULL ]
display.show(face)
```

В списках Вы можете хранить различные объекты: строки, числа, изображения и т. д.:

```
primes = [2, 3, 5, 7, 11, 13, 17, 19]
display.show(primes)
```

Примечание: Числа не нужно заключать в кавычки, так как они представляют значения. Если Вы напишите `"2"` число станет символом (строкой). Вы получите интересный эффект если напишите `"2"+"2"`

Напишите:

```
display.show(2+2)
display.show("2"+"2")
```

В списке можно даже хранить различные элементы:

```
mixed_up_list = ["hello!", 1.234, Image.HAPPY]
display.show(mixed_up_list)
```

Давайте вернемся к анимации готовых изображений и добавим параметры `loop` и `delay`:

```
face = [Image.HAPPY, Image.ANGRY, Image.SKULL ]
display.show(face , loop=True, delay=1000)
```

- `loop` - Зацикливание анимации - значение `True` (правда), проиграть 1 раз - значение `False` (ложь)
- `delay` - время задержки между сменой изображений (милисекунды)
- `wait` - блокирование выполнения пока не кончиться анимация - значение `True` (правда)

1.4 Кнопки

Любая программа чаще всего имеет элементы вывода информации и элементы ввода. В микроконтроллерах часто используются для этого кнопки

Microbit имеет две кнопки на лицевой части. Они называются `A` и `B`.

Напишите программу, которая выводит на дисплей информацию о кнопке `A`:

```
from microbit import *

sleep(10000)
display.scroll(str(button_a.get_presses()))
```

Микроконтроллер засыпает на 10 секунд (`sleep(10000)`). Вы нажимаете несколько раз на кнопку Следующая команда выводит количество нажатий `button_a.get_presses()`

- `button_a` - указывает на объект кнопка `A`. Кнопка `B` имеет указатель `button_b`
- `get_presses()` - функция (команда), которая возвращает количество нажатий
- `str()` - функция, которая переводит число в строку

Предупреждение: Функция `scroll()` принимает только строковые значения.

Функции можно вкладывать в друг друга. Они выполняются в порядке раскрывания (в начале самая вложенная) Примером может служить **Матрешка**. В таком случае выполнение по росту от маленькой до большой.

1.4.1 Отслеживание событий

Часто необходимо создать ожидание события. Вы создаете цикл, который опрашивает устройство и ждет от него выполнения условия.

Для создания такого цикла можно использовать `while`. Он проверяет, если условие события возвращает `True` он запускает *блок кода*, называемый *телом* цикла. Если возвращается `False`, он выходит из цикла и продолжает выполнять программу дальше:

```
from microbit import *

while not button_a.is_pressed():
    display.show(Image.ASLEEP)

display.show(Image.SURPRISED)
```

`not button_a.is_pressed()` **- кнопка** `A` **не нажата (**`not`**), выводится изображение** `Image.ASLEEP` иначе `Image.SURPRISED`

Вы можете использовать логические операторы для отслеживания нескольких событий:

```
while not button_a.is_pressed() and  not button_b.is_pressed():
```

- `and` - логическая И. кнопка `A` **и** ``B``**не нажата**

  while not button_a.is_pressed() or not button_b.is_pressed():

- `or` - логическая И. кнопка `A` **или** ``B``**не нажата**

1.4.2 Обработка событий

Если Вы хотите написать программу, которая реагирует на нажатие на кнопки, используйте Условие `if` и метод `is_pressed`.

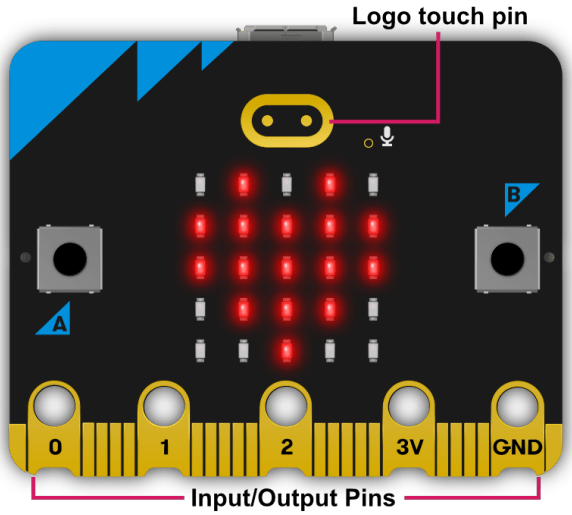
Зацикливание программы на постоянную проверку событий:

```
from microbit import *

while True:
    if button_a.is_pressed():
        display.show(Image.HAPPY)
    else:
        display.show(Image.SAD)
```

Если кнопка нажата отображается **счастливый смайлик**, иначе **грустный**. Метод `is_pressed` дает только два результата: `True` или `False`. Если вы нажимаете кнопку, она возвращает `True`, в противном случае возвращается `False`.

1.5 Контакты/Пины

Вдоль нижнего края располагаются контакты/пины. Каждый пин имеет свой номер. Пять больших площадок и 20 тонких. Большие площадки подписаны: **0,1,2,3V, GND**. К ним легко подключаться используя провода с зажимом типа **крокодил**



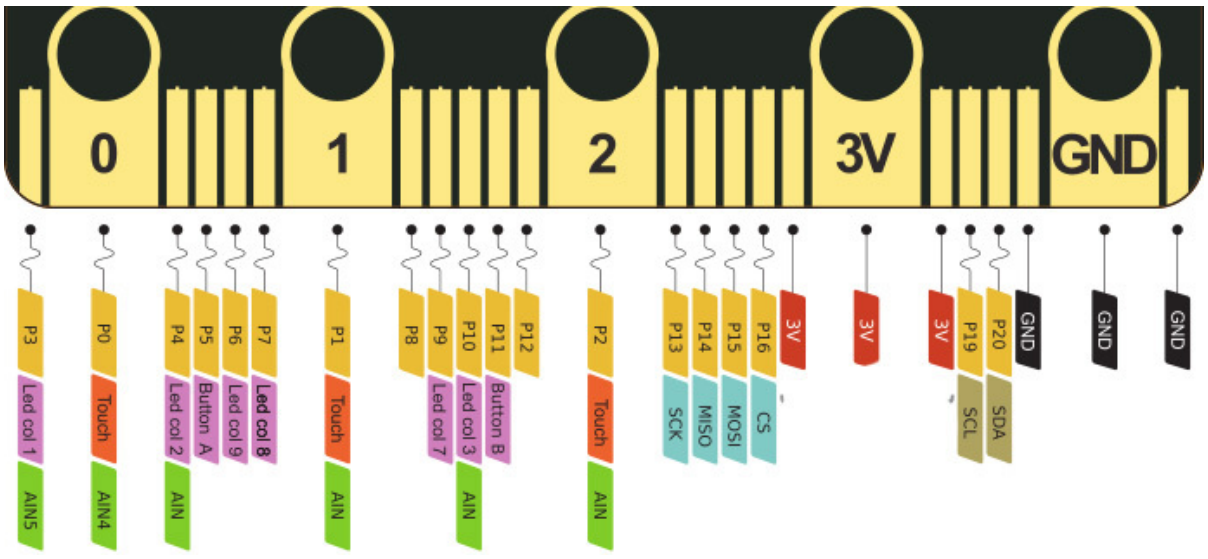
Другие пины удобно использовать с шилдами - платы расширения

В последней версии Microbit **V2** логотип micro:bit также можно использовать в качестве сенсорного экрана.

В MicroPython каждый пин представлен как объект типа `pinN`, где `N` это номер пина.

Пин с логотипом в Microbit **V2** называется `pin_logo`.

Не все пины имеют одинаковые методы (команды). Это зависит от его функционала, который заложили разработчики



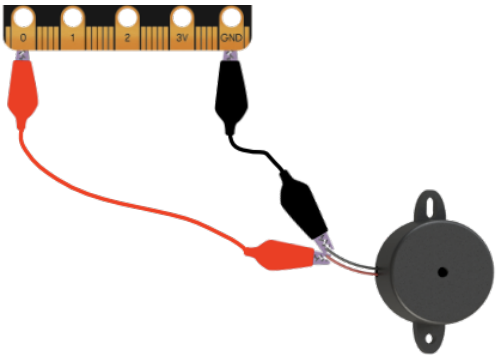
- `PN` - номер пина в линейке

- `Led colN` - пин дублируется в управлении дисплеем. Для его использования необходимо отключить дисплей
- `Button N` - пин дублируется в управлении кнопок.
- `3V` - пин питания 3 вольта (+). Используется для питания устройств и датчиков
- `GND` - пин земля (-)
- `AIN` - аналоговый пин.

К пинам подсоединяются различные устройтва и датчики. Конкретнее это прописанно в статьях про устройтва

1.6 Мелодии

MicroPython на Microbit содержит мощный музыкальный и звуковой инструмент. Вы можете легко генерировать звуковые и музыкальные сигналы. В Microbit `V1` необходимо подключить динамик к пину `0` и `GND` В Microbit `V2` динамик уже встроен и Вы можете сразу переходит к коду.



Примечание: Для проигрывания необходимо использовать пассивный зумер. Нельзя использовать активный

Давайте проиграем мелодию:

```
import music

music.play(music.NYAN)
```

Необходимо подключить библиотеку `music` и использовать команды указывая объект.

MicroPython имеет довольно много встроенных мелодий:

- `music.DADADADUM`
- `music.ENTERTAINER`
- `music.PRELUDE`
- `music.ODE`
- `music.NYAN`
- `music.RINGTONE`
- `music.FUNK`

- music.BLUES
- music.BIRTHDAY
- music.WEDDING
- music.FUNERAL
- music.PUNCHLINE
- music.PYTHON
- music.BADDY
- music.CHASE
- music.BA_DING
- music.WAWAWAWAA
- music.JUMP_UP
- music.JUMP_DOWN
- music.POWER_UP
- music.POWER_DOWN

Обратите внимание встроенные мелодии также пишутся через объект `music`

1.6.1 Собственные мелодии

У каждой ноты есть свое имя (например `C#` или `F`), также указывается продолжительность проигрывания. Октавы обозначаются цифрами ~ 0 самая низкая, 4 средняя, 8 высокая.

Каждая нота выражается в виде строки символов, подобных этому:

```
NOTE[Октава][:Длительность]
```

Например `"A1:4"`, указывается нота `A`, октава `1`, воспроизводится `4`.

Можно использовать список для создания мелодии, например:

```
import music

tune = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
        "E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
music.play(tune)
```

Примечание: MicroPython запоминает октаву и продолжительность, поэтому код можно переписать так:

```
import music

tune = ["C4:4", "D", "E", "C", "C", "D", "E", "C", "E", "F", "G:8",
        "E:4", "F", "G:8"]
music.play(tune)
```

1.6.2 Звуковые Эффекты

MicroPython позволяет создавать звуки, не используя ноты. Например, так можно создать эффект полицейской сирены

```
import music

while True:
    for freq in range(880, 1760, 16):
        music.pitch(freq, 6)
    for freq in range(1760, 880, -16):
        music.pitch(freq, 6)
```

Команда `music.pitch` ожидает частоту и продолжительность. В коде используется цикл `for` для прохода по генератору (range), который выдает числа от 880 до 1760 с шагом 16.

Сирена звучить вечно, она заключена в бесконечный цикл `while`.

1.7 Случайные числа

MicroPython имеет библиотеку `random`, которая может работать со случайными числами

Например вывод случайной буквы:

```
from microbit import *
import random

names = ["A", "B", "C", "D", "I", "F", "G" ]

display.scroll(random.choice(names))
```

Создаем список имен `names` и подаем его в функцию `random.choice`, которая выбирает случайную букву и выводит его на дисплей.

Попробуйте создать вывод случайной картинки из определенного списка.

1.7.1 Случайные числа

Случайные числа часто используются в играх. Например **игральные кости**

MicroPython имеет команду `random.randint`, которая выводит случайное число в указанном диапазоне:

```
from microbit import *
import random

display.show(random.randint(1, 6))
```

Функция `random.randint` выводит только целые числа.`

Также есть функция `random.randrange` она выводит случайное число не включая пограничные числа.

Функция `random.random` возвращает случайную дробную часть в диапазоне от `0.0` до `1.0`.

Например нам нужно случайное число, которое состоит из целой и дробкой части:

```
from microbit import *
import random

answer = random.randrange(100) + random.random()
display.scroll(str(answer))
```

1.7.2 Правда о случайности

На самом деле случайные числа **не случаны :)**. Это числа, которые тяжело предсказать заранее. В их расчетов используются множество псевдослучайнах параметров устройсва (время, температура микроконтроллера и т. д.)

Есть функция `random.seed`, которая приостанавливает случайность. Именно она используется в расчетах. Если Вы самостоятельно присвоете ей значение - она вернет всегда одно и тоже число.

Напишите программу, которая ломает случайность :)

```
from microbit import *
import random

random.seed(1337)
while True:
    if button_a.was_pressed():
        display.show(random.randint(1, 6))
```

1.8 Акселерометр

В микроконтроллере Microbit встроен датчик - акселерометр. Он измеряет движение вдоль осей:

- X - наклон влево и вправо.
- Y - наклоны вперед и назад.
- Z - движение вверх и вниз.

Для каждой оси есть метод, который возвращает положительное или отрицательное число. Единица измерения акселерометра мм/с. Акселерометр может фиксировать: вибрации, рывки, наклонное положение, столкновения и движение объекта.

Например измерение отклонений по оси **X** и отображения буквы `R` - отклонение вправо, `L` - отклонение на лево:

```
from microbit import *

while True:
    reading = accelerometer.get_x()
    if reading > 20:
        display.show("R")
    elif reading < -20:
        display.show("L")
    else:
        display.show("-")
```

В строке `reading = accelerometer.get_x()` записана переменная, которая хранит значение функции считывания отклонения по оси `X`.Далее прописано условие реагирование на события отклонения вправо и влено на 20.

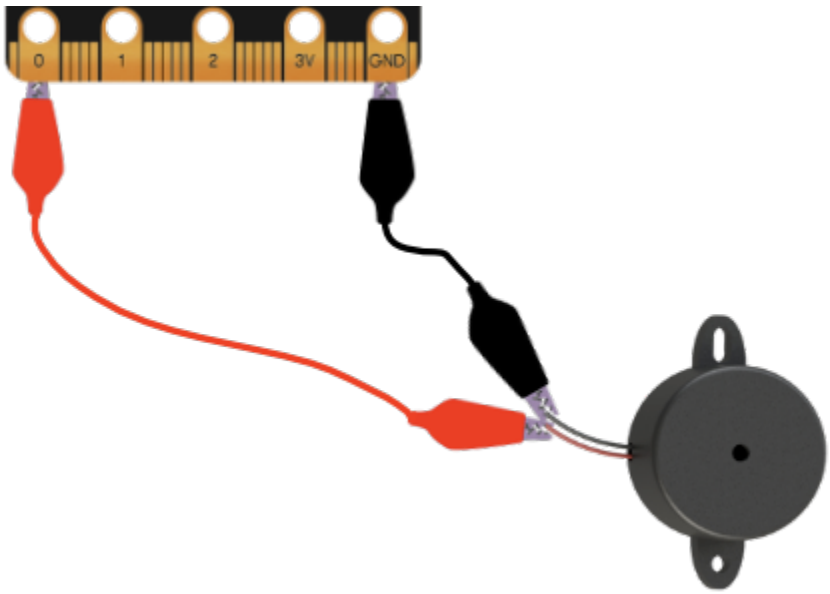
Для отслеживания отклонений по другим осям используйте функции``get_y`` по оси `Y` и `get_z` по оси `Z`.

Современные устройства тоже содержат датчик акселерометр: телефоны, планшеты, электронные книги, игровые устройства и т. д.

1.8.1 Музыкальный инструмент

Давайте создадим устройство, которое проигрывает звук, который зависит от отклонения по оси `Y`. Отклонение меняет воспроизводимую частоту.

Подключите динамик к микроконтроллеру, если у Вас версия `V1`.



Напишите программный код:

```
from microbit import *
import music

while True:
    music.pitch(accelerometer.get_y(), 10)
```

Продолжительность проигрывание очень быстрая 10 милисекунд.

1.9 Жесты - Акселерометр

Акселерометр также предоставляет удобные функции для обнаружения жестов. Распознаваемые жесты:

- `up` - логотипом вверх
- `down` - логотипом вниз
- `left` - наклон влево
- `right` - наклон вправо
- `face up` - дисплеем верх
- `face down` - дисплеем вниз
- `freefall` - свободное падение
- `3g` - тряска с усилием 3g
- `6g` - тряска с усилием 6g
- `8g` - тряска с усилием 8g
- `shake` - тряска базовая

Используются метод `accelerometer.current_gesture( )` - Возвращает название текущего жеста. `accelerometer.is_gesture( имя )` - Возвратите значение `True` или `False`,если произошел указанный жест (вместо имя).

Пример кода **Волшебный шар**, который выводит случайное сообщение при тряске:

```
answers = [
    'Y',
    'N',
    '?',
    '%',
]
while True:
display.show('8')
if accelerometer.was_gesture('shake'):
    display.clear()
    sleep(1000)
    display.scroll(random.choice(answers))
sleep(10)
```

1.10 Компас

В микроконтроллер Microbit также встроен датчик компас.

Программа указывает на север:

```
from microbit import *

compass.calibrate()

while True:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    needle = ((15 - compass.heading()) // 30) % 12
    display.show(Image.ALL_CLOCKS[needle])
```

Примечание: Перед использованием компас необходимо откалибровать. Если этого не сделать итоговые результаты будут ошибочные. Функция `calibration` выводит пиксель и Вы должны вращать микроконтроллер, пока не загорятся все.

Функция``compass.heading`` принимает показание датчика и передает его в математическое выражение, которое преобразует его в число от 1 до 12. Далее число передается в массив `Image.ALL_CLOCKS` и преобразуется в стрелку.

1.11 Речь

Microbit умеет синтезировать речь по текстовому сообщению. Для этого используется синтезатор речи.

Подключите библиотеку `speech` и вызовите функцию `say`, передайте тектовое сообщение в нее:

```
import speech

speech.say("Hello, World")
```

В команде можно использовать параметры:

- `pitch` - высокий или низкий голос (0 = низкий, 255 = высокий)
- `speed` - скорость речи (0 = медленно, 255 = быстро)
- `mouth` - протяженность звуков (0 = закрытый звук, 255 = открытый звукы)
- `throat` - напряженность голоса (0 = расслабленный, 255 = напряженный)

Чтобы настроить параметры, вы передаете их в качестве аргументов функции `say`

Пример:

```
speech.say("I am a DALEK", speed=120, pitch=100, throat=100, mouth=200)
```

1.11.1 Создание длинной речи

Давайте напишем программу для речи:

```
import speech
import random
from microbit import sleep

# Генерируются случайные слова, которые вставляются в речь.
location = random.choice(["brent", "trent", "kent", "tashkent"])
action = random.choice(["wrapped up", "covered", "sang to", "played games with"])
obj = random.choice(["head", "hand", "dog", "foot"])
prop = random.choice(["in a tent", "with cement", "with some scent",
                      "that was bent"])
```

(continues on next page)

(продолжение с предыдущей страницы)

```
result = random.choice(["it ran off", "it glowed", "it blew up",
                        "it turned blue"])
attitude = random.choice(["in the park", "like a shark", "for a lark",
                          "with a bark"])
conclusion = random.choice(["where it went", "its intent", "why it went",
                            "what it meant"])

# формирование речи.
poem = [
    "there was a young man from {}".format(location),
    "who {} his {} {}".format(action, obj, prop),
    "one night after dark",
    "{} {}".format(result, attitude),
    "and he never worked out {}".format(conclusion),
    "EXTERMINATE",
]

# запуск фраз в цикле.
for line in poem:
    speech.say(line, speed=120, pitch=100, throat=100, mouth=200)
    sleep(500)
```

1.11.2 Фонемы

Часто генератор речи воспроизводит английское слово неправильно. Для точного воспроизведения речи используются фонемы.

Полный список фонем, которые понимает синтезатор речи, можно найти в английской документации по API для речи (https://microbit-micropython.readthedocs.io/en/v2-docs/speech.html). В качестве альтернативы можно использовать функцию `translate`.

Функция `pronounce` используется следующим образом:

```
speech.pronounce("/HEH5EH4EH3EH2EH2EH3EH4EH5EHLP.'')
```

1.12 Радиосигнал

Радиоканал является открытым каналом передачи сигнала и его прослушать может любое аналогичное учтройство. Если у Вас несколько устройств Microbit паралелльно передают сигналы, то они будут мешать друг другу.

Можно настроить радио на разные каналы (пронумерованные 0-83). Это работает точно так же, как детские рации: все настраиваются на один канал и слышат всех остальных. Как и в случае раций, при использовании соседних каналов возможны небольшие помехи.

Радиомодуль позволяет указать две части информации: адрес и группу. Адрес похож на почтовый адрес, тогда как группа похожа на конкретного получателя по адресу. Важно то, что радио отфильтровывает получаемые сообщения, которые не соответствуют *вашему* адресу и группе. Поэтому важно заранее определить адрес и группу, которые будет использовать ваше приложение.

Конечно, Microbit передает открытый сигнал. Главное, Вам не нужно беспокоиться об их фильтрации. Тем не менее, если кто-то захочет принять сигнал или подменить, они смогут это просто сделать.

В этом случае *важно* использовать зашифрованные средства связи. Криптография — увлекательная тема, но к сожалению, выходит за рамки этого руководства.

1.12.1 Программа «Светлячки»

Это «Светлячок»:

Использовать Microbit, чтобы создать что-то вроде роя светлячки сигнализирующих друг другу

Сначало подключим библиотеку `import radio` для работы с радиомодулем. Используем метод `radio.on()` чтобы включить радиомодуль. Радиомодуль тратит много энергии, поэтому Вы можете управлять его включением и выключением `radio.off()`.

Библиотека `radio` имеет много функций настройки сигнала, которая позволяет экономить энергию. Документация по API содержит всю информацию, необходимую для настройки радиостанции в соответствии с вашими потребностями.

Самый простой способ отправить сообщение:

```
radio.send("message")
```

Получить сообщение можно:

```
new_message = radio.receive()
```

По мере получения сообщений они помещаются в очеред. `receive` функция возвращает последнее сообщение из очереди в виде строки, освобождая место для нового входящего сообщения. Если очередь сообщений заполняется, то новые входящие сообщения игнорируются

Радиомодуль также может отправлять данные произвольного типа: числа, параметры со значениями

Программа светлячков:

```
# A micro:bit Firefly.
# By Nicholas H.Tollervey. Released to the public domain.
import radio
import random
from microbit import display, Image, button_a, sleep

# Анимация вспышка
flash = [Image().invert()*(i/9) for i in range(9, -1, -1)]

# Включить радиомодуль.
radio.on()

# Вечный цикл.
while True:
    # Если кнопка A нажата отправить сообщение.
    if button_a.was_pressed():
        radio.send('flash')
    # Читать входящие сообщения
    incoming = radio.receive()
    if incoming == 'flash':
        # Если пришло сообщение"flash" проиграть анимацию после случайно задержки
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        sleep(random.randint(50, 350))
        display.show(flash, delay=100, wait=False)
        # Передать ответное сообщение.
        if random.randint(0, 9) == 0:
            sleep(500)
            radio.send('flash')
```

Результат выглядит так:

1.13 Адресная светодиодная лента

Лента позволяет управлять каждым светодиодом отдельно. Можно произвольно изменять длину ленты и количество светодиодов. Адресные ленты отличаются плотностью — от 30 до 144 светодиодов на метр. Каждый светодиод в ленте состоит из обычного RGB светодиода и контроллера с тремя транзисторными выходами. Благодаря этому есть возможность управлять цветом любого светодиода и создавать потрясающие цветовые и световые эффекты.

Mocrobit имеет встроенный класс управления:

```
from neopixel import NeoPixel
```

1.13.1 Класс

`class NeoPixel`

Класс используется для определения объектов, имеющих поведение пассивного зуммера

Пример объявления объекта:

```
np = NeoPixel(pin0, 15)
```

Указывается номер контакта, куда подключена светодиодная лента и количество светоиодом в ленте.

Включить красный цвет на первом светодиоде:

```
np[0] = (255, 0, 0)
np.show()
```

Цифры определяют интенсивность трех светодиодов внутри каждого пикселя. Формат настройки интенсивности: (КРАСНЫЙ, ЗЕЛЕНЫЙ, СИНИЙ). В этой строке кода Вы устанавливаете красный цвет светодиода, потому что яркость КРАСНОГО светодиода является максимальной (255), а яркость двух других светодиодов минимальна (0). Команда **show** - применяет последние действия над светодиодами (включает их). Вы устанавливаете состояние нескольвим светодиодам и в конце применяете команду **show**

`np.show()`

Команда позволяет включить светодиоды

`np.clear()`

Команда позволяет выключить светодиоды

1.13.2 Пример программ

Включить все светодиоды:

```
from neopixel import NeoPixel
np = NeoPixel(pin0, 15)
for i in range(15):
    np[i] = (255, 0, 0)
np.show()
```

Включить все светодиоды с задержкой 1000 мс:

```
from neopixel import NeoPixel
np = NeoPixel(pin0, 15)
for i in range(15):
    np[i] = (255, 0, 0)
    np.show()
    sleep(1000)
```

Бегущий светодиод 100 мс:

```
from neopixel import NeoPixel
np = NeoPixel(pin0, 15)
for i in range(15):
    np[i] = (255, 0, 0)
    np.show()
    sleep(100)
    np.clear()
```

1.14 Справочник по API

Описание функция расположенных в объектах и классах в библиотеке Microbit.

1.14.1 Microbit Micropython API

Модуль

Весь функционал расположен в библиотеке:

```
from microbit import *
```

В дальнейщем в коде это строка указываться не будет. Вы ее подключаете самостоятельно

Функции доступные на прямую:

```
# Задержка, указывается в миллисекундах.
sleep(ms)
# возвращает количество миллисекунд с момента последнего включения Microbit.
running_time()
# Переводит Microbit в режим ошибки выполнения.
panic(error_code)
# Перезагрузка Microbit.
reset()
# Устанавливает громкость динамиков.
set_volume(128)    # V2
```

Остальная функциональность обеспечивается объектами и классами в модуле микробит, как будет описано ниже.

Обратите внимание, что API предоставляет только целые числа (т. е. числа с плавающей запятой не нужны, но они могут быть приняты). Таким образом, мы используем миллисекунды в качестве стандартной единицы времени.

Примечание: Вы можете увидеть список всех доступных модулей, написав `help('modules')` в REPL.

Кнопки

На микробите встроены 2 кнопки:

```
button_a
button_b
```

Объекккты имеют методы:

```
# возвращает True или False,обозначая была ли кнопка нажата во время вызов метода.
button.is_pressed()
# возвращает True или False, чтобы указать, была ли кнопка нажата после последнего␣
↪вызова этой метода
button.was_pressed()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# возвращает общее количество нажатий на кнопоку и сбрасывает этот счетчик на ноль
button.get_presses()
```

Дисплей

Светодиодный дисплей выводится через объект *display*:

```
# Возвращает значения яркости пикселя ( от 0 до 9) по координатам x, y.
display.get_pixel(x, y)
# Устанавливает значения яркости пикселя ( от 0 до 9) по координатам x, y.
display.set_pixel(x, y, val)
# Очищает дисплей.
display.clear()
# Выводит изображение на дисплей.
display.show(image, delay=0, wait=True, loop=False, clear=False)
# Может выводить итерируемый объект с задержкой.
display.show(iterable, delay=400, wait=True, loop=False, clear=False)
# Бегущая строка.
display.scroll(string, delay=400)
```

Звук V2

Команды описывающие работу со звуком:

```
# Переключение режима работы (тихо, громко).
SoundEvent.LOUD = SoundEvent('loud')
# Переключение в режим повышенных шумов.
SoundEvent.QUIET = SoundEvent('quiet')
```

Микрофон V2

Доступ к микрофону осуществляется через объект *microphone*:

```
# Возвращает имя последнего записанного звукового события

current_event()

# Отследить изменение событий `SoundEvent.LOUD`и `SoundEvent.QUIET`.
# Возвращает true, если звук был слышен хотя бы один раз с момента последнего
# вызов, иначе `false`

was_event(event)

# Кортеж историй звуковых событий

get_events()

#Установка порогового уровня реагирования на звук в диапазоне 0-255. Например,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# `set_threshold(SoundEvent.LOUD, 250)` сработает, только если
# звук очень громкий (>= 250).

set_threshold(128)

# Возвращает представление уровня звукового давления в диапазоне от 0 до 255.

sound_level()
```

Контакты

Контакты обеспечивают цифровые и аналоговые функции ввода/вывода сигналов. Запомните, что некоторые контакты используются во внутреннем функционале платы (дисплей, кнопки и т. д.). При их использовании необходимо отключать внутренний функционал.

Каждый контакт предоставляется как объект непосредственно в модуле `microbit`. Этот делает API относительно простым:

- pin0
- pin1
- ...
- pin15
- pin16
- *Предупреждение: P17-P18 (включительно) недоступны.*
- pin19
- pin20
- pin_logo **V2**
- pin_speaker **V2**

Каждый контакт является экземпляром класса `MicroBitPin`, который предлагает следующий API:

```
# Отправляет цифровой сигнал (может быть 0, 1, False, True)

pin.write_digital(value)

# Возвращает цифровой сигнал 1 или 0

pin.read_digital()

# Отправляет аналоговый сигнал от 0 до 1023

pin.write_analog(value)

# Возвращает аналоговый сигнал от 0 до 1023

pin.read_analog()
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# устанавливает период ШИМ-вывода/вывода в миллисекундах
# (подробнее https://en.wikipedia.org/wiki/Pulse-width_modulation)

pin.set_analog_period(int)

# устанавливает период ШИМ-вывода/вывода в микросекундах
# (подробнее https://en.wikipedia.org/wiki/Pulse-width_modulation)

pin.set_analog_period_microseconds(int)

# Доступно только для сенсорных контактов 0, 1 и 2. Возвращает логическое значение, если␣
↪контакт затронуть

pin.is_touched()

# Доступно только для сенсорных контактов 0, 1, 2, а также для логотипа micro:bit V2.
# Устанавливает сенсорный режим. Значение может быть либо RESISTIVE, либо CAPACITIVE.

pin.set_touch_mode(value)
```

За исключением контактов, отмеченных **V2**, которые предлагают следующий API:

```
pin_logo

# возвращает логическое значение если тронуть логотип

pin_logo.is_touched()

# Устанавливает сенсорный режим. Значение может быть либо RESISTIVE, либо CAPACITIVE

pin.set_touch_mode(value)
```

pin_speaker:

Принадлежит классу `MicroBitPin`, но не имеет функции `pin.is_touched()`.

Изображение

Примечание: Вам не всегда нужно создавать изображение самостоятельно — Вы можете получить доступ к изображениям, непосредственно с помощью *display.image*. *display.image* это экземпляр *Image*, поэтому Вы можете использовать одинаковые методы.

Изображение API:

```
# создает пустое изображение 5x5

image = Image()

# создать изображение из строки - каждый символ в строке представляет собой
# уровень яркости светодиода - 0 выключен, 9 - максимальная яркость. Двоеточие ":"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# указывает на конец строки.

image = Image('90009:09090:00900:09090:90009:')

# создать пустое изображение заданного размера

image = Image(width, height)

# Возвращает ширину изображения

image.width()

# Возвращает высоту изображения

image.height()

# Управляет яркостью светодиода в указанной позиции ( от 0 до 5).
# value от 0 до 9

image.set_pixel(x, y, value)

# Возвращает яркость пикселя в указанной позиции (от 0 до 9)

image.get_pixel(x, y)

# возвращает новое изображение, созданное сдвигом изображения влево на 'n' раз.

image.shift_left(n)

# возвращает новое изображение, созданное сдвигом изображения вправо на 'n' раз.

image.shift_right(n)

# возвращает новое изображение, созданное сдвигом изображения вверх на 'n' раз.

image.shift_up(n)

# возвращает новое изображение, созданное сдвигом изображения вниз на 'n' раз.

image.shift_down(n)

# получить компактное строковое представление изображения

repr(image)

# получить строковое представление изображения

str(image)

#operators
# возвращает новое изображение, созданное путем наложения двух изображений
```

(continues on next page)

(продолжение с предыдущей страницы)

```
image + image

# возвращает новое изображение, созданное путем умножения яркости каждого пикселя на n

image * n
```

Изображения из библиотеки

`Image.HEART` `Image.HEART_SMALL` `Image.HAPPY` `Image.SMILE` `Image.SAD` `Image.CONFUSED` `Image.ANGRY` `Image.ASLEEP` `Image.SURPRISED` `Image.SILLY` `Image.FABULOUS` `Image.MEH` `Image.YES` `Image.NO` `Image.TRIANGLE` `Image.TRIANGLE_LEFT` `Image.CHESSBOARD` `Image.DIAMOND` `Image.DIAMOND_SMALL` `Image.SQUARE` `Image.SQUARE_SMALL` `Image.RABBIT` `Image.COW` `Image.MUSIC_CROTCHET` `Image.MUSIC_QUAVER` `Image.MUSIC_QUAVERS` `Image.PITCHFORK` `Image.XMAS` `Image.PACMAN` `Image.TARGET` `Image.TSHIRT` `Image.ROLLERSKATE` `Image.DUCK` `Image.HOUSE` `Image.TORTOISE` `Image.BUTTERFLY` `Image.STICKFIGURE` `Image.GHOST` `Image.SWORD` `Image.GIRAFFE` `Image.SKULL` `Image.UMBRELLA` `Image.SNAKE`

Время:

`Image.CLOCK1` `Image.CLOCK2` `Image.CLOCK3` `Image.CLOCK4` `Image.CLOCK5` `Image.CLOCK6` `Image.CLOCK7` `Image.CLOCK8` `Image.CLOCK9` `Image.CLOCK10` `Image.CLOCK11` `Image.CLOCK12`

Стороны света:

`Image.ARROW_N` `Image.ARROW_NE` `Image.ARROW_E` `Image.ARROW_SE` `Image.ARROW_S` `Image.ARROW_SW` `Image.ARROW_W` `Image.ARROW_NW`

Ниже приведены название списков изображений Python. Вы можете их обрабатывать как списки.

`Image.ALL_CLOCKS` `Image.ALL_ARROWS`

Акселерометр

Доступ к акселерометру осуществляется через объект `accelerometer`:

```
# Ось X.
accelerometer.get_x()
# Ось Y.
accelerometer.get_y()
# Ось Z.
accelerometer.get_z()
# Получить показания трех осей X, Y, Z (в указанном порядке).
accelerometer.get_values()
# вернуть название текущего жеста.
accelerometer.current_gesture()
# вернуть True или False, чтобы указать произошел жест или нет.
accelerometer.is_gesture(name)
# вернуть True или False, чтобы указать произошел ли жест с последнего опроса
accelerometer.was_gesture(name)
# вернуть кортеж истории жестов.
accelerometer.get_gestures()
```

Объявленные жесты: `up`, `down`, `left`, `right`, `face up`, `face down`, `freefall`, `3g`, `6g`, `8g`, `shake`.

Компас

Доступ к компасу осуществляется через объект *compass*:

```
# калибровка компаса (это нужно для получения точных показаний).
compass.calibrate()
# вернуть числовое значение смещения градусов от «севера».
compass.heading()
# вернуть числовое значение силы магнитного поля micro:bit.
compass.get_field_strength()
# возвращает True или False, чтобы указать, откалиброван ли компас.
compass.is_calibrated()
# сбрасывает компас в состояние предварительной калибровки.
compass.clear_calibration()
```

Протокол I2C

На micro:bit есть шина I2C, доступная через объект *i2c*. Он имеет следующие методы:

```
# прочитать n байт с устройства с адресом; Repeat=True означает, что стоп-бит не будет␣
↪отправлен.
i2c.read(addr, n, repeat=False)
# записать buf на устройство с адресом; Repeat=True означает, что стоп-бит не будет␣
↪отправлен.
i2c.write(addr, buf, repeat=False)
```

Звук V2

Набор выразительных звуков доступен для micro:bit **V2**. Они могут быть доступ через модуль `microbit`.

Встроенные звуки

`Sound.GIGGLE` `Sound.HAPPY` `Sound.HELLO` `Sound.MYSTERIOUS` `Sound.SAD` `Sound.SLIDE` `Sound.SOARING` `Sound.SPRING` `Sound.TWINKLE` `Sound.YAWN`

Генератор речи V2

Динамик включен по умолчанию, и к нему можно получить доступ с помощью объекта `speaker`. Это можно отключить или включить:

```
# отключить встроенный динамик

speaker.off()

# включить встроенный динамик

speaker.on()

# возвращает True или False, чтобы указать, включен или выключен динамик
```

(continues on next page)

(продолжение с предыдущей страницы)

```
speaker.is_on()
```

Протокол UART

Используйте `uart` для связи с последовательным устройством, подключенным к контактам ввода-вывода устройства:

```
# Настроки связи  (используйте контакты 0 [TX] и 1 [RX]) со скоростью 9600.

uart.init()

# Возвращает True или False, проверяя есть ли входящие символы на чтение.

uart.any()

# Прочитать n входящих символов.

uart.read(n)

# Прочитать как можно больше входящих данных.

uart.read()

# Вернуть все символы до символа новой строки.

uart.readline()

# Читать байты в указанный буфер.

uart.readinto(buffer)

# Записать байты из буфера на подключенное устройство.

uart.write(buffer)
```

1.14.2 Модуль Microbit

Доступ до всех функций осуществляется через `microbit`.

Функции

`microbit.panic(`*n*`)`

Формирование ошибки и передача номера ошибки. Для продолжения работы требуется перезагрузка Microbit:

```
microbit.panic(255)
```

`microbit.reset()`

Перезагрузка платы:

```
microbit.reset()
```

`microbit.sleep(`*n*`)`

Задержка `n` миллисекунд. Приостановка работы Microbit:

```
microbit.sleep(1000)
```

`microbit.running_time()`

Возвращает количество миллисекунд с момента включения платы или перезапущен:

```
microbit.running_time()
```

`microbit.temperature()`

Возвращает температуру Microbit в градусах Цельсия:

```
microbit.temperature()
```

Атрибуты

Кнопки

На микроконтроллере есть две кнопки: `button_a` и `button_b`.

Атрибуты

`button_a`

`Button` объект, представляющий левую кнопку.

`button_b`

Представляет правую кнопку.

Класс

class Button

Шаблон кнопки.

Примечание: Этот класс фактически недоступен пользователю, он используется только два экземпляра кнопок, которые уже инициализированы.

---

is_pressed()

Возвращает True если кнопка button удерживается, и False.

was_pressed()

Возвращает True если была нажата кнопка с последнего момента вызова этого метода.

get_presses()

Возвращает общее количество нажатий кнопок и сбрасывает это общее количество. до нуля перед возвратом.

Примеры

```
import microbit

while True:
    if microbit.button_a.is_pressed() and microbit.button_b.is_pressed():
        microbit.display.scroll("AB")
        break
    elif microbit.button_a.is_pressed():
        microbit.display.scroll("A")
    elif microbit.button_b.is_pressed():
        microbit.display.scroll("B")
    microbit.sleep(100)
```

Контакты

Контакты — это способ связи вашей платы с внешними устройствами. В вашем распоряжении 19 контактов, пронумерованных 0-16 и 19-20. Контакты 17 и 18 нет в наличии. Существует также pin_logo и pin_speaker в версии **Microbit V2**.

Приведенный ниже скрипт изменит отображение на micro:bit если показания на контакте 0 поменяется:

```
from microbit import *


while True:
    if pin0.read_digital():
        display.show(Image.HAPPY)
    else:
        display.show(Image.SAD)
```

Функции

Расположение контактов на микроконтроллере **Microbit V1**

Расположение контактов на микроконтроллере **Microbit V2**

В таблице приведены доступные выводы, их типы и назначения.

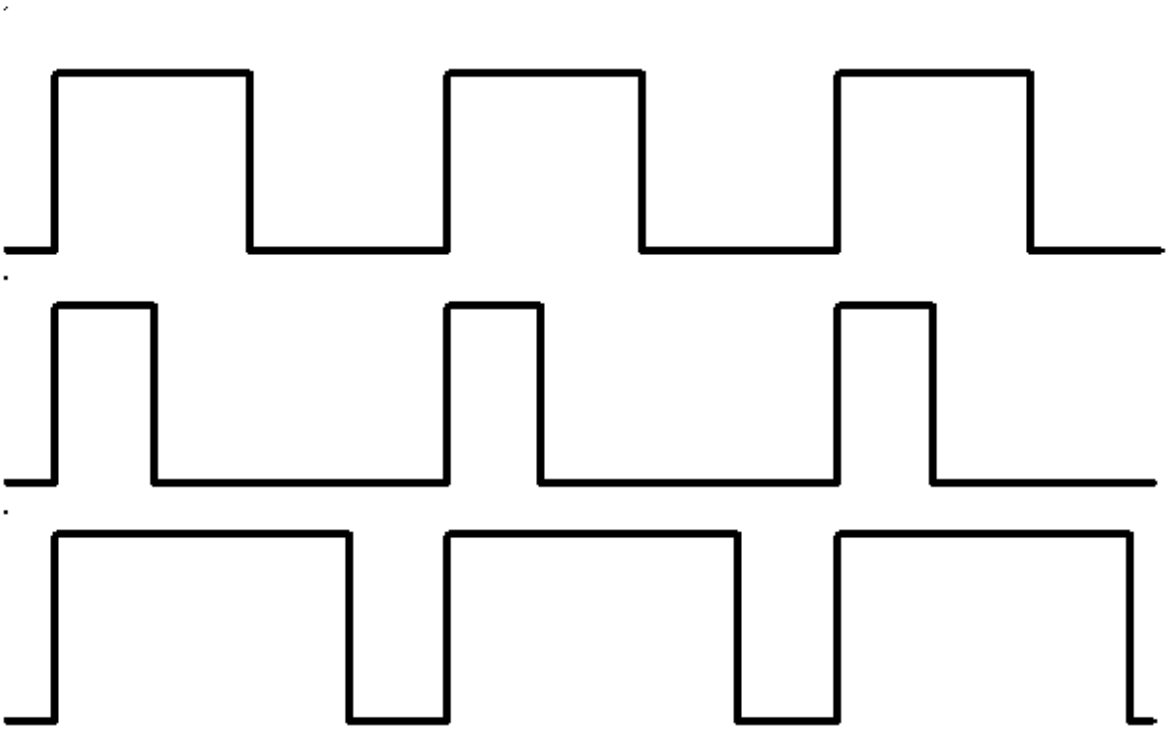
Pin	Type	Function	Function **V2**
0	Touch	Pad 0	Pad 0
1	Touch	Pad 1	Pad 1
2	Touch	Pad 2	Pad 2
3	Analog	Column 1	Column 3
4	Analog	Column 2	Column 1
5	Digital	Button A	Button A
6	Digital	Column 9	Column 4
7	Digital	Column 8	Column 2
8	Digital		
9	Digital	Column 7	
10	Analog	Column 3	Column 5
11	Digital	Button B	Button B
12	Digital		
13	Digital	SPI SCK	SPI SCK
14	Digital	SPI MISO	SPI MISO
15	Digital	SPI MOSI	SPI MOSI
16	Digital		
19	Digital	I2C SCL	I2C SCL
20	Digital	I2C SDA	I2C SDA

Последнее устройство micro:bit **V2** имеет два дополнительных контакта, к которым вы можете получить доступ. в MicroPython, но недоступны через соединитель:

- `pin_logo` - Чувствительный к прикосновениям логотип на передней панели micro:bit, который по умолчанию настроен на емкостный сенсорный режим.
- `pin_speaker` - контакт для обращения к динамику micro:bit. Этот API предназначен только для использования в операциях с выводами широтно-импульсной модуляции, например. `pin_speaker.write_analog(128)`.

Широтно-импульсная модуляция

Контакты вашей платы не могут выводить аналоговый сигнал, как аудиоусилитель. Но можно сделать это - путем модуляции напряжения на выводе. Контакты могут либо включить полный выход 3,3 В или выключить его 0 В. Но можно управлять яркостью светодиодов или скоростью электродвигателя используя ШИМ (очень быстро включать и выключать напряжение и контролировать ее продолжительность). Функция `write_analog` использует ШИМ.



Выше вы можете увидеть диаграммы трех разных ШИМ-сигналов. Все они имеют одинаковый период (частоту), но они имеют разные рабочие циклы.

Первый будет сгенерирован командой `write_analog(511)`, от составляет половину нагрузки - питание включено ровно половину времени. Результатом этого будет 1,65В вместо 3,3В.

Второй сигнал имеет рабочий цикл 25% и может быть сгенерирован командой `write_analog(255)`. Имеет такой же эффект, если выводить 0,825В на контакт.

Это хорошо работает с устройствами, как двигатели, которые имеют огромную инерцию. Или светодиоды, которые мигают слишком быстро, чтобы человеческий глаз мог увидеть разницу. Но не будет хорошо работать с генерацией звуковых волн. ШИМ может генерировать прямоугольные звуки.

Класс

Есть три вида контактов. Они представлены классами, перечисленными ниже. Обратите внимание, что они образуют иерархию, так что каждый класс имеет все функциональные возможности предыдущего класса, и добавляет свой собственный функционал к нему.

Примечание: Эти классы недоступны для пользователя, Вы не можете создать новые их экземпляры. Можете использовать только уже предоставленные экземпляры.

```
class MicroBitDigitalPin
```

```
    read_digital()
```

Возвращает 1, если сигнал высокий, и 0, если низкий.

`write_digital(value)`

Установите контакт на высокий уровень сигнала, если `value` равна 1. `value` равна 0 - низкий уровань.

`set_pull(value)`

Установить состояние контакта.Одно из трех возможных значений: `pin.PULL_UP`, `pin.PULL_DOWN` или `pin.NO_PULL`. См. ниже обсуждение состояний извлечения по умолчанию.

`get_pull()`

Извлечь состояние контакта, одно из трех возможных значений: `NO_PULL`, `PULL_DOWN`, или `PULL_UP`.

`get_mode()`

Возвращает режим вывода. Когда контакт используется для определенной функции, например запись цифрового значения или чтение аналогового значения, режим вывода изменztncz. Пины могут иметь один из следующих режимов: `"unused"`, `"analog"`, `"read_digital"`, `"write_digital"`, `"display"`, `"button"`, `"music"`, `"audio"`, `"touch"`, `"i2c"`, `"spi"`.

`class MicroBitAnalogDigitalPin`

`read_analog()`

Вернуть напряжение, приложенное к контакту, целое число между 0 (что означает 0 В) и 1023 (что означает 3,3 В).

`write_analog(value)`

Установите ШИМ-сигнал на контакт с рабочим циклом, пропорциональным предоставленный в `value`. `value` может быть либо целым числом, либо число с плавающей запятой от 0 до 1023.

`set_analog_period(period)`

Установите период выводимого ШИМ-сигнала на `period` в миллисекундах. Минимальное допустимое значение составляет 1 мс

`set_analog_period_microseconds(period)`

Установите период выводимого ШИМ-сигнала на `period` в микросекунды. Минимальное допустимое значение составляет 256 мкс.

`class MicroBitTouchPin`

`is_touched()`

Вернет `True` если к контакту прикасаются пальцем, иныче вернет `False`.

Примечание: Сенсорный режим по умолчанию для контактов на краевом разъеме: *resistive*. В версии **V2** есть контакт логотип в состоянии *capacitive*.

Resistive touch Этот тест проводится путем измерения сопротивления между контактом и землей. Низкое сопротивление дает значение `True`. Для хорошего сигнала прикоснитесь второй рукой к заземлению.

Capacitive touch Этот тест проводится путем взаимодействия с электрическим полем конденсатора. использование пальца в качестве проводника. Capacitive touch не требует заземления как части цепи.

set_touch_mode(*value*)

Примечание: Сенсорный режим по умолчанию для контактов имеет состояние *resistive*. Пин логотип в версии **V2** состояние *capacitive*.

Установите сенсорный режим для данного вывода. Значение может быть либо `CAPACITIVE`, либо `RESISTIVE`. Например, `pin0.set_touch_mode(pin0.CAPACITIVE)`.

Режим вытягивания контакта настраивается автоматически, когда контакт меняется. Режимы ввода - когда вы используете `read_analog` / `read_digital` / `is_touched`. Режим извлечения по умолчанию для них, соответственно, `NO_PULL`, `PULL_DOWN`, `PULL_UP`. Вызов `set_pull` настроит контакт так, чтобы он был в `read_digital`.

Примечание: Micro:bit имеет внешние слабые (10M) подтягивающие устройства, установленные на штифтах. 0, 1 и 2 только для того, чтобы сенсорное распознавание работало.

Также на контакты 5 и 11 установлены внешние (10k) подтягивающие устройства, чтобы чтобы кнопки A и B работали.

Выводы GPIO также используются для отображения, как описано в таблице выше. Если вы хотите использовать эти штифты для другой цели, вам может потребоваться повернуть display off.

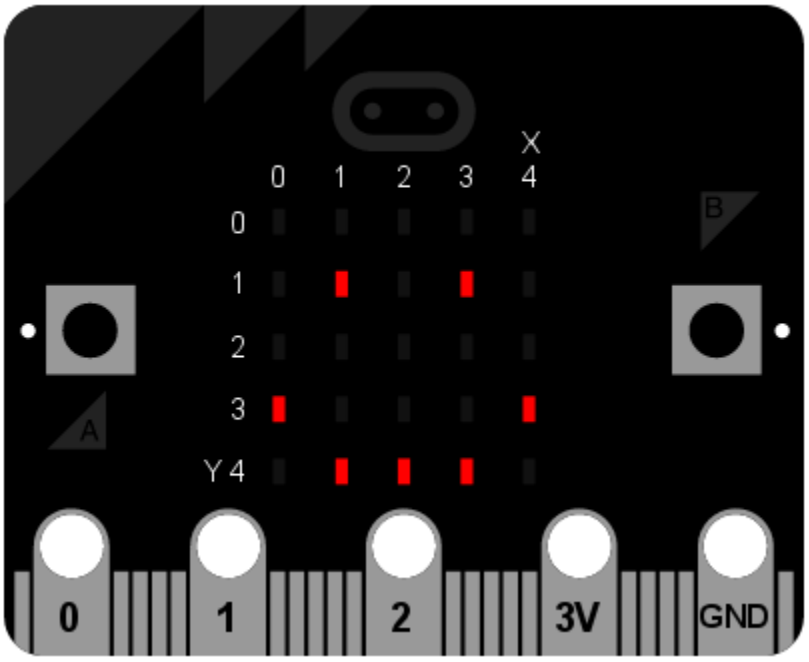
Смотрите edge connector data sheet.

Класс

Изображение

Класс `Image` используется для создания изображений, которые можно легко отобразить на светодиодная матрица устройства. Учитывая объект изображения, его можно отобразить через API отображения:

```
display.show(Image.HAPPY)
```



Существует четыре способа создания образа:

- `Image()` - Создает пустое изображение 5x5
- `Image(string)` - Создает изображение, проанализировав строку.
- `Image(width, height)` - Создать пустое изображение заданного размера
- `Image(width, height, buffer)` - Создать изображение из заданного буфера

Класс

`class Image(`*string*`)`

`class Image(`*width=None*, *height=None*, *buffer=None*`)`

Если используется `string`, она должна состоять из цифр 0-9, расположенных в строки, описывающие изображение, например:

```
image = Image("90009:"
              "09090:"
              "00900:"
              "09090:"
              "90009")
```

создаст изображение размером 5×5. Конец строки обозначается двоеточием. Также можно использовать новую строку (n) для обозначения конца строки. Пример:

```
image = Image("90009\n"
              "09090\n"
              "00900\n"
              "09090\n"
              "90009")
```

Другая форма создает пустое изображение со столбцами `width` и `height` строк. Опционально `buffer` может быть массивом `width` × `height` целые числа в диапазоне 0-9 для инициализации изображения:

```
Image(2, 2, b'\x08\x08\x08\x08')
```

или:

```
Image(2, 2, bytearray([9,9,9,9]))
```

Создаст изображение размером 2 x 2 пикселя с максимальной яркостью..

Примечание: Аргументы ключевого слова не могут быть переданы `buffer`.

`width()`

Вернуть количество столбцов в изображении.

`height()`

Вернуть количество строк в изображении.

`set_pixel(`*x*, *y*, *value*`)`

Установите яркость пикселя в столбце «x» и строке «y» на `value`, которое должно быть между 0 (темный) и 9 (яркий).

Этот метод вызовет исключение при вызове любого из встроенных изображения только для чтения, такие как `Image.HEART`.

`get_pixel(`*x*, *y*`)`

Возвращает яркость пикселя в столбце `x` и строке `y` как целое число от 0 до 9.

`shift_left(`*n*`)`

Вернуть новое изображение, созданное путем сдвига изображения влево на `n` столбцы.

`shift_right(`*n*`)`

Здвиг вправо `image.shift_left(-n)`.

`shift_up(`*n*`)`

вернуть новое изображение, созданное путем сдвига изображения вверх на `n` строк.

`shift_down(`*n*`)`

Сдвиг вниз `image.shift_up(-n)`.

`crop(`*x*, *y*, *w*, *h*`)`

Верните новое изображение, обрезав изображение до ширины `w` и а высота `h`, начиная с пикселя в столбце `x` и строке `y`.

`copy()`

Вернуть точную копию изображения.

`invert()`

Верните новое изображение, инвертировав яркость пикселей в исходное изображение.

`fill(`*value*`)`

Установите яркость всех пикселей изображения на `value`, которое должно быть между 0 (темный) и 9 (яркий).

Этот метод вызовет исключение при вызове любого из встроенных изображения только для чтения, такие как `Image.HEART`.

`blit`(*src*, *x*, *y*, *w*, *h*, *xdest=0*, *ydest=0*)

Скопируйте прямоугольник, определенный `x`, `y`, `w`, `h` из изображения `src` в это изображение в `xdest`, `ydest`. Области в исходном прямоугольнике, но за пределами исходного изображения, обрабатываются как имеющие значение 0. `shift_left()`, `shift_right()`, `shift_up()`, `shift_down()` и `crop()` все они могут быть реализованы с помощью `blit()`.

Например, img.crop(x, y, w, h) можно реализовать как:

```
def crop(self, x, y, w, h):
    res = Image(w, h)
    res.blit(self, x, y, w, h)
    return res
```

Атрибуты

Класс `Image` также имеет следующие встроенные экземпляры: включены в качестве его атрибутов (имена атрибутов указывают, что изображение представляет собой):

- `Image.HEART`
- `Image.HEART_SMALL`
- `Image.HAPPY`
- `Image.SMILE`
- `Image.SAD`
- `Image.CONFUSED`
- `Image.ANGRY`
- `Image.ASLEEP`
- `Image.SURPRISED`
- `Image.SILLY`
- `Image.FABULOUS`
- `Image.MEH`
- `Image.YES`
- `Image.NO`
- `Image.CLOCK12`, `Image.CLOCK11`, `Image.CLOCK10`, `Image.CLOCK9`, `Image.CLOCK8`, `Image.CLOCK7`, `Image.CLOCK6`, `Image.CLOCK5`, `Image.CLOCK4`, `Image.CLOCK3`, `Image.CLOCK2`, `Image.CLOCK1`
- `Image.ARROW_N`, `Image.ARROW_NE`, `Image.ARROW_E`, `Image.ARROW_SE`, `Image.ARROW_S`, `Image.ARROW_SW`, `Image.ARROW_W`, `Image.ARROW_NW`
- `Image.TRIANGLE`
- `Image.TRIANGLE_LEFT`
- `Image.CHESSBOARD`
- `Image.DIAMOND`
- `Image.DIAMOND_SMALL`
- `Image.SQUARE`

- Image.SQUARE_SMALL
- Image.RABBIT
- Image.COW
- Image.MUSIC_CROTCHET
- Image.MUSIC_QUAVER
- Image.MUSIC_QUAVERS
- Image.PITCHFORK
- Image.XMAS
- Image.PACMAN
- Image.TARGET
- Image.TSHIRT
- Image.ROLLERSKATE
- Image.DUCK
- Image.HOUSE
- Image.TORTOISE
- Image.BUTTERFLY
- Image.STICKFIGURE
- Image.GHOST
- Image.SWORD
- Image.GIRAFFE
- Image.SKULL
- Image.UMBRELLA
- Image.SNAKE

Наконец, связанные коллекции изображений были сгруппированы вместе:

```
* ``Image.ALL_CLOCKS``
* ``Image.ALL_ARROWS``
```

Команды

```
repr(image)
```

Получить компактное строковое представление изображения.

```
str(image)
```

Получить удобочитаемое строковое представление изображения.

```
image1 + image2
```

Создайте новое изображение, добавив значения яркости из двух изображений для каждый пиксель.

```
image * n
```

Создайте новое изображение, умножив яркость каждого пикселя на `n`.

Аудио

Этот модуль позволяет воспроизводить собственные звуки. Если вы используете micro:bit **V2**.

По умолчанию вывод звука подключено к контакту 0 *built-in speaker* **V2**. Вы можете подключить проводные наушники или динамик к контакту 0 и GND, чтобы слышать звуки.

Функции

`audio.play`(*source*, *wait=True*, *pin=pin0*, *return_pin=None*)

Проиграть звук.

- **source**: `Sound` - The `microbit` модуль содержит список встроенных звуков, которые вы можете передать в функцию `audio.play()`.
- **source**: `AudioFrame` - ТИсходный аргумент также может быть итерируемым элементов. Подробнее в модуле `AudioFrame`.
- **wait**: Если `wait` присвоено `True`, microbit будет заблокирован до конца проигрывания звука.
- **pin**: Необязательный аргумент для указания другого выходного контакта, Вы можете поменять значение (по умолчанию `pin0`).
- **return_pin**: задает контакт вместо земли. Это игнорируется для версии **V2**.

`audio.is_playing()`

Возвращает `True` если звук воспроизводится, иначе возврат `False`.

`audio.stop()`

Останавливает воспроизведение звука.

Класс

`class audio.AudioFrame`

An `AudioFrame` объект представляет собой список из 32 мелодий.

Воспроизведение одного кадра занимает чуть более 4 мс.

Использование аудио

Вам понадобится источник звука, для функции `play`. Вы можете использовать встроенные звуки **V2** из модуля `microbit`, `microbit.Sound`, создать свой собственный, как в `examples/waveforms.py`.

Встроенные звуки V2

Встроенные звуки можно вызвать с помощью `audio.play(Sound.NAME)`.

- `Sound.GIGGLE`
- `Sound.HAPPY`
- `Sound.HELLO`
- `Sound.MYSTERIOUS`
- `Sound.SAD`
- `Sound.SLIDE`
- `Sound.SOARING`
- `Sound.SPRING`
- `Sound.TWINKLE`
- `Sound.YAWN`

Модули

Акселерометр

Объект дает вам доступ к акселерометру. Акселерометр также предоставляет удобные функции для обнаружения жестов: `up`, `down`, `left`, `right`, `face up`, `face down`, `freefall`, `3g`, `6g`, `8g`, `shake`.

Функции

`microbit.accelerometer.get_x()`

Возвращение параметра ускорения по оси `x` - положительное или отрицательное целое, в зависимости от направления. Измерение приведено в милли-g. По умолчанию акселерометр настроен на диапазон +/- 2g, и поэтому этот метод будет возвращаться в диапазоне +/- 2000 мг.

`microbit.accelerometer.get_y()`

Возвращение параметра ускорения по оси `y` - положительное или отрицательное целое, в зависимости от направления. Измерение приведено в милли-g. По умолчанию акселерометр настроен на диапазон +/- 2g, и поэтому этот метод будет возвращаться в диапазоне +/- 2000 мг.

`microbit.accelerometer.get_z()`

Возвращение параметра ускорения по оси `z` - положительное или отрицательное целое, в зависимости от направления. Измерение приведено в милли-g. По умолчанию акселерометр настроен на диапазон +/- 2g, и поэтому этот метод будет возвращаться в диапазоне +/- 2000 мг.

`microbit.accelerometer.get_values()`

Возвращение параметра ускорения по трем осям - набор чисел в порядке X, Y, Z.

`microbit.accelerometer.current_gesture()`

Вернуть название текущего жеста.

Примечание: MicroPython понимает следующие названия жестов: `"up"`, `"down"`, `"left"`, `"right"`, `"face up"`, `"face down"`, `"freefall"`, `"3g"`, `"6g"`, `"8g"`, `"shake"`. Данные представлены в виде строк

`microbit.accelerometer.is_gesture(name)`

Возвращайте True или False, чтобы проверить, активен ли в данный момент определенный жест.

`microbit.accelerometer.was_gesture(name)`

Возвращайте True или False, чтобы проверить, был ли указанный жест активен с момента последнего опроса.

`microbit.accelerometer.get_gestures()`

Возвращает кортеж истории жестов. Также очищает историю жестов перед возвратом.

Примечание: Жесты не обновляются в фоновом режиме, поэтому должны быть постоянными. Вы вызываете какой-либо метод акселерометра для обнаружения жестов. Как правило жесты могут быть обнаружены с помощью цикла с небольшой задержкой *microbit.sleep()*.

Примеры

Программа волшебный шар.

```
'''Программа волшебный шар.'''


from random import choice

from microbit import accelerometer, display, sleep

answers = ['Yes', 'No', 'Ask more']

while True:
    display.show('8')
    if accelerometer.was_gesture('shake'):
        display.clear()
        sleep(1000)
        display.scroll(choice(answers))
    sleep(10)
```

Перемещайте устройство, чтобы избежать препятствий.

```
'''Движение пикселя'''

import random

from microbit import (Image, accelerometer, button_a, display, running_time,
                      sleep)

MIN_X = -1024
MAX_X = 1024
WALL_MIN_SPEED = 400
PLAYER_MIN_SPEED = 200
WAALL_MAX_SPEED = 100
PLAYER_MAX_SPEED = 50
SPEED_MAX = 12
```

(continues on next page)

(продолжение с предыдущей страницы)

```
range_x = MAX_X - MIN_X

while True:

    template = Image('00000:'*5)
    s = template.set_pixel
    player_x = 2
    wall_y = -1
    hole = 0
    score = 0
    handled_this_wall = False
    wall_speed = WALL_MIN_SPEED
    player_speed = PLAYER_MIN_SPEED
    wall_next = 0
    player_next = 0

    while True:
        t = running_time()
        player_update = t >= player_next
        wall_update = t >= wall_next
        if not (player_update or wall_update):
            next_event = min(wall_next, player_next)
            delta = next_event - t
            sleep(delta)
            continue

        if wall_update:
            # расчет новой скорости
            speed = min(score, SPEED_MAX)
            wall_speed = WALL_MIN_SPEED + int((WAALL_MAX_SPEED - WALL_MIN_SPEED) * speed␣
↪/ SPEED_MAX)
            player_speed = PLAYER_MIN_SPEED + int((PLAYER_MAX_SPEED - PLAYER_MIN_SPEED)␣
↪* speed / SPEED_MAX)

            wall_next = t + wall_speed
            if wall_y < 5:
                # замена препядствия
                use_wall_y = max(wall_y, 0)
                for wall_x in range(5):
                    if wall_x != hole:
                        s(wall_x, use_wall_y, 0)

        wall_reached_player = (wall_y == 4)
        if player_update:
            player_next = t + player_speed
            # найти новую координату x
            x = accelerometer.get_x()
            x = min(max(MIN_X, x), MAX_X)
            # отключить старую позицию (пиксель)
            s(player_x, 4, 0)
            x = ((x - MIN_X) / range_x) * 5
            x = min(max(0, x), 4)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            x = int(x + 0.5)

            if not handled_this_wall:
                if player_x < x:
                    player_x += 1
                elif player_x > x:
                    player_x -= 1
        if wall_update:
            # обновить положение стены
            wall_y += 1
            if wall_y == 7:
                wall_y = -1
                hole = random.randrange(5)
                handled_this_wall = False

            if wall_y < 5:
                # нарисовать новую стену
                use_wall_y = max(wall_y, 0)
                for wall_x in range(5):
                    if wall_x != hole:
                        s(wall_x, use_wall_y, 6)

        if wall_reached_player and not handled_this_wall:
            handled_this_wall = True
            if (player_x != hole):
                # столкновение! игра закончена!
                break
            score += 1

        if player_update:
            # включить новый пиксель
            s(player_x, 4, 9)

        display.show(template)

    display.show(template.SAD)
    sleep(1000)
    display.scroll("Score:" + str(score))

    while True:
        if (button_a.is_pressed() and button_a.is_pressed()):
            break
        sleep(100)
```

Компас

Этот модуль позволяет получить доступ к встроенному электронному компасу. Перед использованием, компас должен быть откалиброван, иначе показания могут быть неправильными

Предупреждение: Калибровка состоит из небольшой игры, в которой нужно нарисовать круг на Светодиодный дисплей при вращении устройства

Функции

`microbit.compass.calibrate()`

апускает процесс калибровки. Инструктивное сообщение будет прокручиваться пользователю, после чего ему нужно будет повернуть устройство, чтобы нарисовать круг на светодиодном дисплее.

`microbit.compass.is_calibrated()`

Возвращает `True` если компас был успешно откалиброван, или возвращается `False`.

`microbit.compass.clear_calibration()`

Отменяет калибровку.

`microbit.compass.get_x()`

Дает показания магнитного поля по оси «x», как положительное или отрицательное целое число, в зависимости от направления поля.

`microbit.compass.get_y()`

Дает показания магнитного поля по оси «y», как положительное или отрицательное целое число, в зависимости от направления поля.

`microbit.compass.get_z()`

Дает показания магнитного поля по оси «z», как положительное или отрицательное целое число, в зависимости от направления поля..

`microbit.compass.heading()`

Дает направление по компасу, рассчитанное на основе приведенных выше показаний, в качестве целого числа в диапазоне от 0 до 360, представляющее угол в градусах, по часовой стрелке, с севером как 0.

`microbit.compass.get_field_strength()`

Возвращает целочисленное значение величины магнитного поля вокруг устройства.

Примеры

```
"""
    compass.py
    Создает компас.
    Сначала пользователю необходимо откалибровать компас. В компасе используется
    встроенные изображения часов для отображения положения стрелки
"""
from microbit import Image, compass, display, sleep
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Начало калибровки
compass.calibrate()

while True:
    sleep(100)
    needle = ((15 - compass.heading()) // 30) % 12
    display.show(Image.ALL_CLOCKS[needle])
```

Дисплей

Этот модуль управляет светодиодным дисплеем. Класс используется для отображения изображений, анимации и текста.

Функции

`microbit.display.get_pixel(`*x*, *y*`)`

Возвращает яркость светодиода в столбце `x` и строке `y»`` в виде целого число от 0 до 9.

`microbit.display.set_pixel(`*x*, *y*, *value*`)`

Устанавливает яркость светодиода в столбце `x` и строке `y»`` в виде целого число от 0 до 9.

`microbit.display.clear()`

Очищает дисплей.

`microbit.display.show(`*image*`)`

Отображает изображение типа `image`.

`microbit.display.show(`*value*, *delay=400*, *wait=True*, *loop=False*, *clear=False*`)`

- `loop` - Зацикливание анимации - значение `True` (правда), проиграть 1 раз - значение `False` (ложь)
- `delay` - время задержки между сменой изображений (милисекунды)
- `wait` - блокирование выполнения пока не кончиться анимация - значение `True` (правда)
- `clear` - очистить дисплей после отображения изображения - значение `True` (правда)

Примечание: Если вы используете генератор как `итерируемый`, позаботьтесь о том, чтобы не выделять память в генераторе, так как выделение памяти в прерывании запрещено и вызовет `MemoryError`.

`microbit.display.scroll(`*value*, *delay=150*, ***, *wait=True*, *loop=False*, *monospace=False*`)`

Прокручивает `value` как бегущую строку. Если `value` является целым числом или числом с плавающей запятой, сначала преобразуйте ее в строку с помощью `str()`. Параметр `delay` контролирует, насколько быстро текст будет прокручиваться.

Если `wait` равно `True`, эта микроконтроллер будет заблокирован до тех пор, пока анимация не завершится.

Если `loop` равно `True`, Анимация повторяется.

Если `monospace` равно `True`, все символы будут занимать 5 пиксельных столбцов по ширине, иначе будет ровно один пустой пиксель-столбец между каждым символ по мере их прокрутки.

Обратите внимание `wait`, `loop` и `monospace` указываются явно.

`microbit.display.on()`

Используйте функцию `on()` для включения дисплея.

`microbit.display.off()`

Используйте функцию `off()` для отключения дисплея.

`microbit.display.is_on()`

Возвращает `True``если дисплей включен, иначе ``False`.

`microbit.display.read_light_level()`

Используйте светодиоды дисплея в режиме обратного смещения, чтобы определить количество света которое падает на дисплей. Возвращает целое число от 0 до 255, представляющее уровень освещенности, с большим значением света.

Примеры

Чтобы непрерывно прокручивать строку по дисплею в фоновом режиме, можно использовать:

```
import microbit

microbit.display.scroll('Hello!', wait=False, loop=True)
```

Протокол I²C

Модуль `i2c` позволяет вам общаться с устройствами, подключенными к вашей плате. с использованием протокола шины I²C. Может быть несколько устройств, подключенных одновременно, и каждый из них имеет свой уникальный адрес. Фиксированный для устройства или настроен на нем. Ваша плата действует как мастер I²C.

Используется 7-битная адресация для устройства. http://www.totalphase.com/support/articles/200349176-7-bit-8-bit-and-10-bit-I2C-Slave-Addressing.

Это может отличаться от других решений, связанных с micro:bit.

Как Вы должны общаться с устройствами (какие байты отправлять и как интерпретировать ответы) зависит от рассматриваемого устройства и описаны отдельно в документации к этому устройству.

Функции

`microbit.i2c.init(`*freq=100000*, *sda=pin20*, *scl=pin19*`)`

Повторно инициализируйте периферийное устройство с указанной тактовой частотой `freq` на указанные контакты `sda` и `scl`.

> **Предупреждение:** На плате micro:bit V1 изменение выводов I²C по умолчанию приведет к отключению акселерометра и компаса, так как они подключены внутри этих контактов. Это предупреждение не относится к **V2** пересмотр микро: бита, так как он имеет отдельные линии I²C <https://tech.microbit.org/hardware/i2c/>`

`microbit.i2c.scan()`

Просканируйте шину на наличие устройств. Возвращает список 7-битных адресов, соответствующих на те устройства, которые ответили на сканирование.

`microbit.i2c.read(`*addr*, *n*, *repeat=False*`)`

Прочитать `n` байт с устройства с 7-битным адресом `addr`. Если `repeat` равно `True`, стоповый бит посылаться не будет

`microbit.i2c.write(`*addr*, *buf*, *repeat=False*`)`

Записать байты из `buf` в устройство с 7-битным адресом `addr`. Если `repeat` имеет значение `True`, стоп-бит не отправляется.

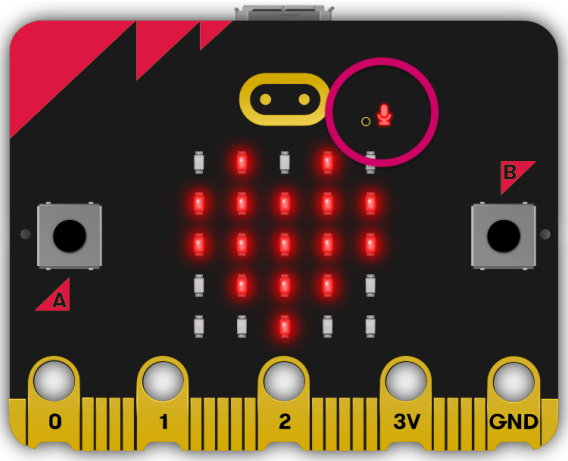
Подключение

Вы должны соединить вывод `SCL` устройства с выводом 19 micro:bit, а контакт `SDA` устройства к контакту micro:bit 20. Вы также должны подключить заземление на землю micro:bit (контакт «GND»). Возможно, вам потребуется питание устройства с помощью внешнего источника питания или micro:bit.

На линиях I^2C платы есть внутренние подтягивающие резисторы, но если Вы используете длинные провода или большое количество устройств, придется добавить дополнительные подтягивающие резисторы для обеспечения бесшумной связи.

Микрофон V2

Этот объект позволяет вам получить доступ к встроенному микрофону, доступному на микро:бит **V2**. Его можно использовать для реакции на звук. Вход микрофона расположен на передней панели рядом со светодиодом активности микрофона, который горит, когда микрофон используется.



События микрофона

Микрофон может реагировать на предопределенный набор звуковых событий, которые в зависимости от амплитуды и длины волны звука.

Эти звуковые события представлены экземплярами класса SoundEvent, доступны через переменные в `microbit.SoundEvent`:

- `microbit.SoundEvent.QUIET`: Представляет переход звуковых событий, от `loud` до `quiet` как разговор или фоновая музыка.
- `microbit.SoundEvent.LOUD`: Представляет переход звуковых событий, от `quiet` до `loud` например, аплодисментов или криков.

Функции

`microbit.microphone.current_event()`

- **return**: название последнего записанного звукового события, `SoundEvent('loud')` или `SoundEvent('quiet')`.

`microbit.microphone.was_event(`*event*`)`

- **event**: звуковое событие, например `SoundEvent.LOUD` или `SoundEvent.QUIET`.
- **return**: `true` если звук был слышен хотя бы один раз с момента последнего позвони, иначе `false`. `was_event()` также очищает звук история событий перед возвратом.

`microbit.microphone.is_event(`*event*`)`

- **event**: звуковое событие, например `SoundEvent.LOUD` или `SoundEvent.QUIET`.
- **return**: `true` если звуковое событие является последним с момента последнего позвони, иначе `false`. Не очищает историю звуковых событий.

`microbit.microphone.get_events()`

- **return**: кортеж истории событий. Самый последний указан последним. `get_events()` также очищает историю звуковых событий перед возвратом.

`microbit.microphone.set_threshold(`*event*, *value*`)`

- **event**: звуковое событие, например `SoundEvent.LOUD` или `SoundEvent.QUIET`.
- **value**: Пороговый уровень в диапазоне 0-255. Например, `set_threshold(SoundEvent.LOUD, 250)` будет срабатывать только в том случае, если звук очень громко ($>= 250$).

`microbit.microphone.sound_level()`

- **return**: представление уровня звукового давления в диапазоне от 0 до 255.

Примеры

Пример, запускающий некоторые функции микрофона:

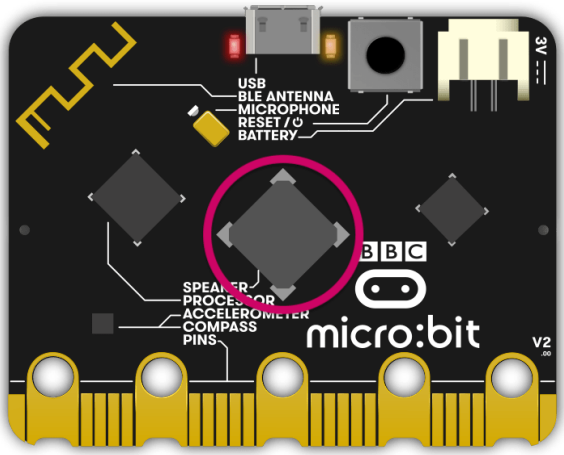
> # Базовый тест для микрофона. Этот тест должен обновлять дисплей, когда # Нажата кнопка A и *pressed* громкий или тихий звук, выводя # Результаты. На кнопке B этот тест должен обновлять дисплей при громком или # тихий звук *is* слышен, вывод результатов. При встряхивании это должно печатать # последние слышимые звуки, вы должны попробовать этот тест, издавая громкий звук # и тихий, прежде чем встряхнуть.

Код:: .. include:: ../examples/microphoneV2.py

code
python

Генератор речи V2

Micro:bit **V2** имеет встроенный динамик, расположенный на задней стороне платы.



По умолчанию звук выводится через крайний контакт на контакте 0 и встроенный динамик **V2**. Вы можете подключить проводные наушники или динамик к контакт 0 и GND на краевом разъеме, чтобы слышать звуки.

Динамик можно включить или выключить, используя перечисленные здесь функции.

Функции

`microbit.speaker.off()`

Используйте `off()` чтобы выключить динамик. Это не отключает вывод звука к контакту.

`microbit.speaker.on()`

Используйте `on()` включить динамик.

Примеры

Пример, который запускает некоторые функции API динамика.

```
from microbit import *

# Убедитесь, что динамик включен
print(speaker.is_on())
# Воспроизвести звук
audio.play(Sound.HELLO)
sleep(2000)
# Отключить динамик
speaker.off()
```

Протокол SPI

Модуль `spi` позволяет вам общаться с устройством, подключенным к плате, используя шину последовательного периферийного интерфейса (SPI). SPI использует master-slave - архитектура с одним мастером. Вам нужно будет указать соединения для трех сигналов:

- SCLK : тактовый сигнал .
- MOSI : Master Output, Slave Input (Ведущий передает, Ведомый принимает).
- MISO : Master Input, Slave Output (Ведущий принимает, Ведомый передает).

Функции

`microbit.spi.init`(*baudrate=1000000*, *bits=8*, *mode=0*, *sclk=pin13*, *mosi=pin15*, *miso=pin14*)

Инициализировать связь SPI с указанными параметрами на указанные `pin`. Обратите внимание, что для правильной связи параметры должны быть одинаковыми на обоих взаимодействующих устройствах.

`baudrate` определяет скорость общения.

`bits` определяет размер передаваемых байтов. В настоящее время только `bits=8` поддерживается. Однако это может измениться в будущем.

`mode` определяет комбинацию тактовых часов и фазы согласно следующему соглашению, с тактом в качестве старшего бита и фаза как младший бит:

SPI Mode	Polarity (CPOL)	Phase (CPHA)
0	0	0
1	0	1
2	1	0
3	1	1

Полярность (также известная как CPOL) 0 означает, что часы находятся на логическом значении 0 в режиме ожидания. и становится высоким (логическое значение 1), когда активен; полярность 1 означает, что часы на логическом значении 1 в режиме ожидания и становится низким (логический 0) в активном состоянии. Фаза (он же CPHA) 0 означает, что выборка данных производится по переднему фронту тактового сигнала, и 1 означает на задней кромке (viz. https://en.wikipedia.org/wiki/Signal_edge).

Аргументы `sclk`, `mosi` и `miso` определяют контакты, которые будут использоваться для каждый тип сигнала.

`spi.read(`*nbytes*`)`

Читать не более `nbytes`. Возвращает прочитанное.

`spi.write(`*buffer*`)`

Запишите `буфер` байтов на шину.

`spi.write_readinto(`*out*, *in*`)`

Запишите буфер `out` на шину и прочитайте любой ответ в `in` буфер. Длина буфера должна быть одинаковой.

Протокол UART

Модуль `uart` позволяет вам общаться с устройством, подключенным к вашей плате, используя последовательный интерфейс.

Функции

`microbit.uart.init(`*baudrate=9600*, *bits=8*, *parity=None*, *stop=1*, ***, *tx=None*, *rx=None*`)`

Инициализировать последовательную связь с указанными параметрами на указанные выводы `tx` и `rx`. Обратите внимание, что для правильной связи параметры должны быть одинаковыми на обоих взаимодействующих устройствах.

> **Предупреждение:** Инициализация UART на внешних контактах вызовет включение консоли Python. USB становится недоступным, так как использует то же оборудование. Чтобы настроить консоли обратно, Вы должны повторно инициализировать UART, ничего не передавая для `tx` или `rx` (или передача `None` этим аргументам). Это означает что вызова uart.init(115200) достаточно для восстановления консоли Python.

`baudrate` определяет скорость передачи данных:

- 9600
- 14400
- 19200
- 28800
- 38400
- 57600
- 115200

`bits` определяет размер передаваемых байтов, плата поддерживает только 8. Параметр `parity` определяет, как проверяется четность, Параметр `stop` указывает количество стоповых битов и должен быть равен 1 для эта доска.

Если `tx` и `rx` не указаны, то внутренние контакты USB-UART TX/RX используются, они подключаются к последовательному преобразователю USB на micro:bit. Вы можете указать любые другие контакты, которые вы хотите, передача желаемых объектов вывода в параметры `tx` и `rx`.

Примечание: При подключении устройства убедитесь, что вы «перекрестили» провода — TX вывод на вашей плате должен быть соединен с выводом RX на устройстве, а контакт RX – с контактом TX на устройстве. Также убедитесь, что заземление обоих устройств соединено.

`uart.any()`

Возвращает `True` если какие-либо данные ожидаются, иначе `False`.

`uart.read(`[*nbytes*]`)`

Читатет байты. Если указано `nbytes`, тогда можно прочитать не больше этого количества байт, в противном случае прочитайте как можно больше байтов.

Возвращаемое значение: объект bytes или `None` по тайм-ауту.

Объект bytes содержит последовательность байтов. Так как ASCII персонажи могут вписать одиночные байты, этот тип объекта часто используется для представления простого текста и предлагает методы манипулирования им как таковым, например. вы можете отобразить текст используя функцию `print()`.

Вы также можете преобразовать этот объект в строковый объект, и если есть символы, отличные от ASCII, могут быть указаны в кодировке:

```
msg_bytes = uart.read()
msg_str = str(msg, 'UTF-8')
```

Примечание: Тайм-аут для всех операций чтения UART зависит от скорости передачи данных и в противном случае нельзя изменить через Python. Тайм-аут в миллисекундах определяется выражением: `microbit_uart_timeout_char = 13000 / baudrate + 1`

Примечание: Внутренний буфер UART RX составляет 64 байта, поэтому убедитесь, что данные считываются до того, как буфер заполнится или некоторые данные могут быть потеряны.

Предупреждение: Получение `0x03` остановит вашу программу, подняв клавиатуру. Прерывать. Вы можете включить или отключить это, используя `micropython.kbd_intr()`.

`uart.readinto(`*buf*[, *nbytes*]`)`

Считайте байты в `buf`. Если указано `nbytes`, то можно прочитать не более столько байтов. В противном случае читать не более `len(buf)` байт.

Возвращаемое значение: количество байтов, прочитанных и сохраненных в `buf` или `None` на тайм-аут.

`uart.readline()`

Прочитать строку, заканчивающуюся символом новой строки.

Возвращаемое значение: прочитанная строка или `None` по тайм-ауту. Символ новой строки включены в возвращаемые байты.

`uart.write(`*buf*`)`

Запишите буфер на шину, это может быть байтовый объект или строка:

```
uart.write('hello world')
uart.write(b'hello world')
uart.write(bytes([1, 2, 3]))
```

Возвращаемое значение: количество записанных байтов или `None` по тайм-ауту..

1.14.3 Bluetooth

Microbit V1

В то время как у BBC Microbit есть аппаратное обеспечение, позволяющее устройству работать как устройство Bluetooth с низким энергопотреблением (BLE), у него всего 16 КБ ОЗУ. Стек BLE один занимает 12 КБ ОЗУ, что означает, что для MicroPython недостаточно памяти для поддержки Bluetooth на micro:bit V1.

Примечание: MicroPython использует радиооборудование с модулем *radio*. Это позволяет пользователям создавать простые, но эффективные беспроводные сети micro:bit устройства.

Кроме того, протокол, используемый в модуле *radio*, намного проще. чем BLE, что значительно упрощает использование в образовательном контексте

Microbit V2

nRF52833, используемый micro:bit V2, имеет 128 КБ ОЗУ, что позволяет Micropython использовать BLE. В настоящее время единственной реализованной функцией является прошивка BLE, позволяющая пользователям обновить прошивку на micro:bit через Bluetooth.

1.14.4 Файловая система

Полезно хранить данные так, чтобы они оставались неповрежденными между перезапусками устройства. На традиционных компьютерах это достигается файловой системой, состоящей из именованных файлов, содержащих необработанные данные, и именованные каталоги, содержащие файлы. Python поддерживает различные операции, необходимые для работы с такими файловыми системами.

Поскольку micro:bit является микроконтроллером, емкость хранилища MicroPython предоставляет небольшой набор функций, необходимых для сохранения данных на устройство. Из-за ограничений памяти **есть приблизительно 30 КБ доступно** в файловой системе.

Предупреждение: Перепрошивка устройства УНИЧТОЖИТ ВАШИ ДАННЫЕ.

Поскольку файловая система хранится во флэш-памяти micro:bit и перепрошивка устройства перезаписывает всю доступную флэш-память поэтому все ваши данные будут потеряны.

Однако, если вы выключите устройство, данные останутся нетронутыми до тех пор, пока Вы либо удаляете их, либо перепрошиваете устройство.

MicroPython на micro:bit обеспечивает простую файловую систему; то есть нет понятие иерархии каталогов, файловая система — это просто список именованных файлов. Чтение и запись файла осуществляется с помощью стандартной Python команды``open``. и результирующий файловый объект типом `TextIO` или `BytesIO`. Операции по работе с файлами в файловой системе содержатся в модуле

os module.

Если файл заканчивается расширением `.py`, его можно импортировать. Например, файл с именем `hello.py` можно импортировать следующим образом: `import hello`.

Пример кода в MicroPython REPL может выглядеть примерно так:

```
>>> with open('hello.py', 'w') as hello:
...     hello.write("print('Hello')")
...
>>> import hello
Hello
>>> with open('hello.py') as hello:
...   print(hello.read())
...
print('Hello')
>>> import os
>>> os.listdir()
['hello.py']
>>> os.remove('hello.py')
>>> os.listdir()
[]
```

`open`(*filename*, *mode='r'*)

Возвращает файловый объект, указанный в аргументе `filename`. По умолчанию используется режим `'r'` - открытие для чтения в текстовый режим. Другим распространенным режимом является `'w'` для записи (перезапись содержимое файла, если он уже существует). Доступны два других режима для использования в сочетании с описанными выше: `'t'` означает текстовый режим (для чтения и записи строк), а `'b'` означает двоичный режим (для чтения и записи байтов). В текстовом режиме файловый объект будет экземпляр `TextIO`. В двоичном режиме файловый объект будет экземпляр `BytesIO`.

`class TextIO`

`class BytesIO`

Экземпляры этих классов представляют файлы micro:bit. Класс TextIO используется для представления текстовых файлов. BytesIO класс используется для представления двоичных файлов. Они работают точно так же за исключением того, что TextIO работает со строками, а BytesIO работает с байтами.

Вы не создаете экземпляры этих классов напрямую. Скорее, надлежащим образом сконфигурированный экземпляр класса возвращается функцией `open` описано выше

`close()`

Закроет файл. После закрытия файла любая операция над файлом вызовет исключение.

`name()`

Возвращает имя файла, который представляет объект. Это будет то же, что и аргумент `filename`, переданный в вызов `open`, создавшая экземпляр объекта.

`read`(*size*)

Читает и возвращать символов `size` в виде одной строки или `size` байтов из файла. Для удобства, если `size` не указано или -1, возвращаются все данные, содержащиеся в файле.

Если возвращается 0 символов или байтов, а `size` не равен 0, это указывает на конец файла.

Исключение `MemoryError` произойдет, если `size` больше, чем доступная оперативная память.

`readinto`(*buf*, *n=- 1*)

Прочитать символы или байты в буфер `buf`. Если указано `n`, прочитать `n` байтов или символов в буфер `buf`.

`readline`(*size*)

Прочитать и вернуть одну строку из файла. Если указан `size`, будут прочтено столько строк.

Ограничитель строки всегда `'\n'` для строк или `b'\n'` для строк. байт.

`writable`()

Возвратите `True`, если файл поддерживает запись.

`write`(*buf*)

Запишите строку или байты `buf` в файл и верните количество символов или байтов, записанных.

1.14.5 Микроконтроллер

Модуль машины содержит специальные функции, связанные с micro:bit аппаратное обеспечение. Большинство функций этого модуля позволяют добиться прямого и неограниченный доступ и управление аппаратными блоками в системе (такими как ЦП, таймеры, автобусы и т.д.). При неправильном использовании это может привести к неисправности, зависаниям, сбои вашей платы и, в крайних случаях, повреждение оборудования.

Функции

`machine.unique_id`()

Возвращает байтовую строку с уникальным идентификатором доски. Это будет варьироваться от одного экземпляра платы к другому.

`machine.reset`()

Сбрасывает устройство аналогично нажатию внешней кнопки RESET.

`machine.freq`()

Возвращает частоту процессора в герцах.

`machine.disable_irq`()

Отключите запросы на прерывание. Возвращает предыдущее состояние IRQ, которое должно быть считается непрозрачным значением. Это возвращаемое значение должно быть передано в *machine.enable_irq()* действия по восстановлению прерываний до их исходное состояние, до *machine.disable_irq()* was called.

`machine.enable_irq`()

Повторно включите запросы на прерывание. Параметр *state* должен быть значением которое было возвращено с самого последнего звонка в *machine.disable_irq()*.

`machine.time_pulse_us`(*pin*, *pulse_level*, *timeout_us=1000000*)

Время импульса на заданном *pin* и возвращение длительности импульса в микросекунды. Аргумент *pulse_level* должен быть равен 0, чтобы отсчитывать низкий импульс или 1 для определения времени высокого пульса.

Если текущее входное значение вывода отличается от *pulse_level*, функция сначала (*) ждет, пока ввод вывода не станет равным *pulse_level*, затем (**) умножить на продолжительность, на которую контакт равен *pulse_level*. Если вывод уже равен *pulse_level*, тогда синхронизация начинается сразу.

Функция вернет -2, если время ожидания условия истекло. отмеченный (*) выше, и -1, если во время основного измерения был тайм-аут, отмечены (**) выше. Тайм-аут одинаков для обоих случаев и задается *timeout_ us* (в микросекундах).

Чтение памяти

Модуль `machine` позволяет читать из памяти устройства, получая 1 байт (8 бит; `mem8`), 2 байта (16 бит; `mem16`) или 4 байта (32 бита; `mem32`) слов с физических адресов. Например: `mem8[0x00]` читает 1 байт по физическому адресу `0x00`. Это имеет ряд применений, например, если вы хотите прочитать данные из регистров nRF51.

1.14.6 MicroPython

Доступ и управление внутренними компонентами MicroPython.

Функции

`micropython.const(`*expr*`)`

Используется для объявления того, что выражение является константой, чтобы компилятор мог оптимизировать его. Использование этой функции должно быть следующим:

```
from micropython import const
CONST_X = const(123)
CONST_Y = const(2 * CONST_X + 1)
```

Константы, объявленные таким образом, по-прежнему доступны как глобальные переменные из вне модуля, в котором они объявлены. С другой стороны, если константа начинается с подчеркивания, она скрывается. Она недоступна как глобальная переменная и не занимает памяти во время выполнения.

`micropython.opt_level(`$\left[$*level*$\right]$`)`

Если задан уровень, эта функция устанавливает уровень оптимизации для последующей компиляции скриптов и возвращает None. В противном случае возвращается текущий уровень оптимизации.

Уровень оптимизации управляет следующими функциями компиляции:

- Утверждения: на уровне 0 операторы утверждений включены и скомпилированы. в байт-код; на уровнях 1 и выше утверждения не компилируются.
- **Встроенная переменная `__debug__`: на уровне 0 эта переменная расширяется до** Истинный; на уровнях 1 и выше расширяется до False.
- **Номера строк исходного кода: на уровнях 0, 1 и 2 номер строки исходного кода** хранятся вместе с байт-кодом, чтобы исключения могли сообщать о номер строки, в которой они произошли; на уровнях 3 и выше номера строк не хранится.

Уровень оптимизации по умолчанию обычно равен 0

`micropython.mem_info(`$\left[$*verbose*$\right]$`)`

Печать информации об используемой в данный момент памяти. Если подробный аргумент задано, то печатается дополнительная информация.

`micropython.qstr_info(`[*verbose*]`)`

Вывести информацию о интернированных в данный момент строках. Если подробный аргумент дается, то печатается дополнительная информация.

Это включает в себя количество интернированных строк и объем оперативной памяти, которую они используют. В подробном режиме он выводит имена всех строк, хранящихся в оперативной памяти..

`micropython.stack_use()`

Возвращает целое число, представляющее текущий объем стека, который использовался. Абсолютное значение this не особенно полезно, скорее оно следует использовать для вычисления различий в использовании стека в разных точках.

`micropython.heap_lock()`

`micropython.heap_unlock()`

Блокировка или разблокировка кучи. Когда заблокировано, выделение памяти не может быть выполнено, и `MemoryError` будет вызвана, если будет предпринята попытка выделения кучи..

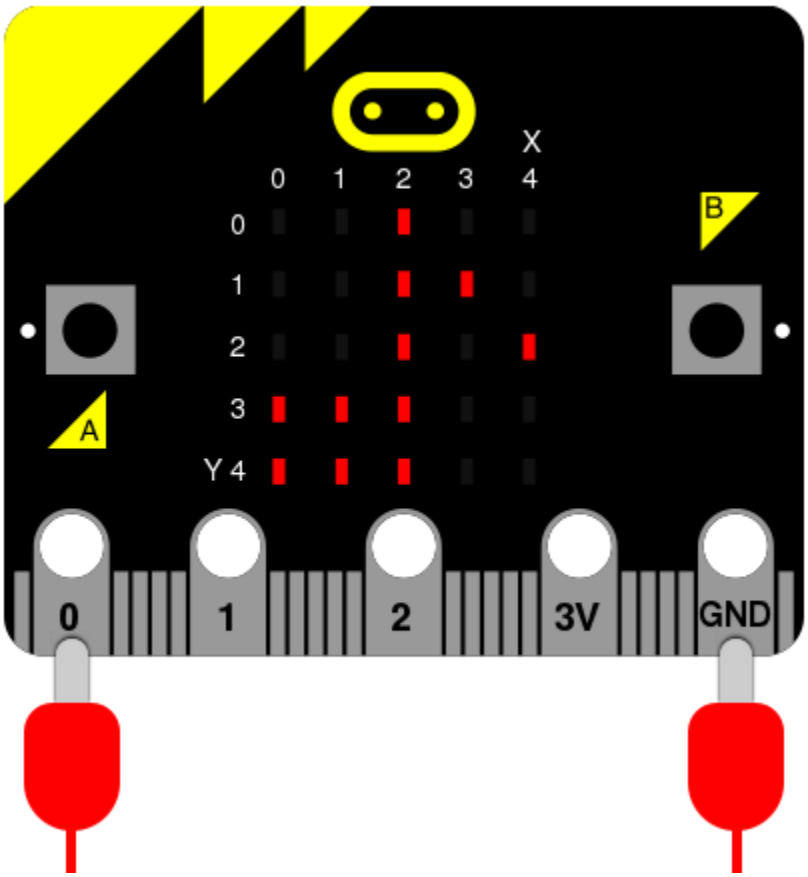
`micropython.kbd_intr(`*chr*`)`

Установите символ, который вызовет исключение KeyboardInterrupt. По умолчанию во время выполнения скрипта это значение равно 3, что соответствует Ctrl-C. Прохождение -1 для этой функции отключит захват Ctrl-C, а передача 3 восстановить его.

Эту функцию можно использовать для предотвращения захвата Ctrl-C на входящий поток символов, который обычно используется для REPL, в случае этот поток используется для других целей

1.14.7 Музыка

Это `music`, и вы можете использовать его для создания и воспроизведения мелодий. По умолчанию звук выводится через краевой разъем на контакте 0 и встроенный динамик **V2**. Вы можете подключить проводные наушники или динамик к контакту 0 и GND на краевом разъеме, чтобы слышать звук:



Вы также можете отключить/включить встроенный динамик или вывести звук на другой pin.

Для доступа к этому модулю вам необходимо:

```
import music
```

Музыкальная нотация

Индивидуальная нота определяется таким образом:

```
NOTE[octave][:duration]
```

Например, `A1:4` относится к ноте «A» в октаве 1, которая длится четыре тика (тик - это произвольный отрезок времени, определяемый настройкой темпа функции - см. ниже). Если используется имя ноты `R`, оно рассматривается как нота молчания.

Знаки бемоли и диезы обозначаются буквой `b` (бемоль - строчная буква b) и `#` (диез - символ решетки). Например, «Ab» — это ля-бемоль, а «C#» — это до-диез.

Имена заметок нечувствительны к регистру

Параметры «октава» и «длительность» — это состояния, которые переносятся на последующие примечания до повторного указания. Состояния по умолчанию: «октава = 4». (содержит среднее C) и `duration = 4` (привязка, учитывая темп по умолчанию настройки - см. ниже).

Пример:

```
['c1:4', 'e:2', 'g', 'c2:4']
```

Таким образом будет закодировано начало 5-й симфонии Бетховена:

```
['r4:2', 'g', 'g', 'g', 'eb:8', 'r:2', 'f', 'f', 'f', 'd:8']
```

Функции

`music.set_tempo`(*ticks=4*, *bpm=120*)

Устанавливает приблизительный темп воспроизведения.

Количество тиков (выраженное целым числом) составляет долю. Каждый удар должен воспроизводиться с определенной частотой в минуту (выражаемой более привычным BPM — также целым числом).

Предлагаемые значения по умолчанию допускают следующее полезное поведение:

- `music.set_tempo()` - сбросить темп на значение по умолчанию тиков = 4, ударов в минуту = 120
- `music.set_tempo(ticks=8)` - изменить «определение» бита
- `music.set_tempo(bpm=180)` - просто изменить темп

Вычислить длину тика в миллисекундах очень просто с помощью арифметики.: `60000/bpm/ticks_per_beat` . Для значений по умолчанию, которые `60000/120/4 = 125 milliseconds` или `1 beat = 500 milliseconds`.

`music.get_tempo`()

Возвращает текущий темп в виде кортежа целых чисел: `(ticks, bpm)`.

`music.play`(*music*, *pin=pin0*, *wait=True*, *loop=False*)

Играет объект `music`.

Если `music` это строка, то будет одна нота, например, `'c1:4'`.

Если `music` указывается в виде списка (как определено в разделе, посвященном музыкальный DSL выше), они воспроизводятся один за другим.

В обоих случаях `duration` и `octave` значения сбрасываются по умолчанию перед воспроизведением музыки.

Необязательный аргумент для указания выходного контакта, может использоваться для переопределения по умолчанию `microbit.pin0`.

Если `wait` установлен в `True`, эта функция блокирует Microbit.

Если `loop` установлен в `True`, мелодия повторяется до команды `stop`

`music.pitch`(*frequency*, *duration=- 1*, *pin=pin0*, *wait=True*)

Воспроизведение высоты тона с частотой, заданной для указанного количества миллисекунды. Например, если частота установлена на 440, а длина на 1000, то мы слышим стандартный концерт A в течение одной секунды.

Обратите внимание, что вы можете воспроизводить только однин канал в любой момент времени..

Необязательный аргумент для указания выходного контакта может использоваться для переопределения по умолчанию `microbit.pin0`.

Если `wait` установлен в `True`, эта функция блокирует Microbit.

Если `duration` является отрицательным, высота тона воспроизводится непрерывно до тех пор, пока блокирующий вызов не прервется или не будет установлена новая частота. `stop` также обрывает.

`music.stop(`*pin=pin0*`)`

Останавливает воспроизведение музыки науказанном контакте. Необязательный аргумент может быть предоставлен для указания контакта, например: `music.stop(pin1)`.

`music.reset()`

Сбрасывает состояние атрибутов следующим образом:

- `ticks = 4`
- `bpm = 120`
- `duration = 4`
- `octave = 4`

Встроенные модули

В образовательных и развлекательных целях модуль содержит несколько примеров мелодий, выраженных в виде списков Python. Вы можете использовать их так:

```
>>> import music
>>> music.play(music.NYAN)
```

Все мелодии либо не защищены авторскими правами, сочинены Николасом Х.Толлерви и выпущены в общественное достояние или имеют неизвестного композитора и покрыты положение о добросовестном (образовательном) использовании.

Мелодии:

- `DADADADUM` - открытие к 5-й симфонии до минор Бетховена.
- `ENTERTAINER` - вступительный фрагмент классического регтайма Скотта Джоплина «The Entertainer».
- `PRELUDE` - открытие первой прелюдии до мажор из 48 прелюдий и фуг И. С. Баха.
- `ODE` - тема «Ода к радости» из 9-й симфонии ре минор Бетховена.
- `NYAN` - тема Nyan Cat (http://www.nyan.cat/). Композитор неизвестен.
- `RINGTONE` - что-то похожее на рингтон мобильного телефона. Используется для обозначения входящего сообщения.
- `FUNK` - фанковая басовая партия для секретных агентов и криминальных авторитетов.
- `BLUES` - 12-тактный блюзовый бас в стиле буги-вуги.
- `BIRTHDAY` - «С Днем Рождения тебя. . . », чтобы узнать о статусе авторских прав, см.: http://www.bbc.co.uk/news/world-us-canada-34332853
- `WEDDING` - свадебный хор из оперы Вагнера «Лоэнгрин».
- `FUNERAL` - «похоронный марш», также известный как Соната для фортепиано № 2 Фредерика Шопена си-бемоль минор, соч. 35.
- `PUNCHLINE` - забавный фрагмент, который означает, что шутка была сделана.
- `PYTHON` - Марш Джона Филипа Соузы «Колокол Свободы», также известный как тема «Летающего цирка Монти Пайтона» (в честь которого назван язык программирования Python).

- `BADDY` - эра немого кино появление злодея.
- `CHASE` - сцена погони эпохи немого кино.
- `BA_DING` - короткий сигнал, указывающий на то, что что-то произошло.
- `WAWAWAWAA` - очень грустный тромбон.
- `JUMP_UP` - для использования в игре, указывающей на движение вверх.
- `JUMP_DOWN` - для использования в игре, указывающей на движение вниз.
- `POWER_UP` - фанфары, чтобы указать, что достижение разблокировано.
- `POWER_DOWN` - грустные фанфары, чтобы указать на потерянное достижение.

Примеры

```
"""
    Воспроизведение простой мелодии с помощью музыкального модуля Micropython.
    В этом примере требуется динамик, подключенные к P0 и GND,
    или micro:bit V2 со встроенным динамиком.
"""
from microbit import *

import music

# play Prelude in C.
notes = [
    'c4:1', 'e', 'g', 'c5', 'e5', 'g4', 'c5', 'e5', 'c4', 'e', 'g', 'c5', 'e5', 'g4', 'c5
↪', 'e5',
    'c4', 'd', 'a', 'd5', 'f5', 'a4', 'd5', 'f5', 'c4', 'd', 'a', 'd5', 'f5', 'a4', 'd5',
↪ 'f5',
    'b3', 'd4', 'g', 'd5', 'f5', 'g4', 'd5', 'f5', 'b3', 'd4', 'g', 'd5', 'f5', 'g4', 'd5
↪', 'f5',
    'c4', 'e', 'g', 'c5', 'e5', 'g4', 'c5', 'e5', 'c4', 'e', 'g', 'c5', 'e5', 'g4', 'c5',
↪ 'e5',
    'c4', 'e', 'a', 'e5', 'a5', 'a4', 'e5', 'a5', 'c4', 'e', 'a', 'e5', 'a5', 'a4', 'e5',
↪ 'a5',
    'c4', 'd', 'f#', 'a', 'd5', 'f#4', 'a', 'd5', 'c4', 'd', 'f#', 'a', 'd5', 'f#4', 'a',
↪ 'd5',
    'b3', 'd4', 'g', 'd5', 'g5', 'g4', 'd5', 'g5', 'b3', 'd4', 'g', 'd5', 'g5', 'g4', 'd5
↪', 'g5',
    'b3', 'c4', 'e', 'g', 'c5', 'e4', 'g', 'c5', 'b3', 'c4', 'e', 'g', 'c5', 'e4', 'g',
↪'c5',
    'a3', 'c4', 'e', 'g', 'c5', 'e4', 'g', 'c5', 'a3', 'c4', 'e', 'g', 'c5', 'e4', 'g',
↪'c5',
    'd3', 'a', 'd4', 'f#', 'c5', 'd4', 'f#', 'c5', 'd3', 'a', 'd4', 'f#', 'c5', 'd4', 'f#
↪', 'c5',
    'g3', 'b', 'd4', 'g', 'b', 'd', 'g', 'b', 'g3', 'b3', 'd4', 'g', 'b', 'd', 'g', 'b'
]

music.play(notes)
```

1.14.8 NeoPixel (Адресная светодиодная лента)

Модуль «neopixel» позволяет использовать NeoPixel (WS2812) с индивидуальной адресацией. Обратите внимание, чтобы использовать модуль `neopixel`, Вам нужно импортировать его отдельно:

```
import neopixel
```

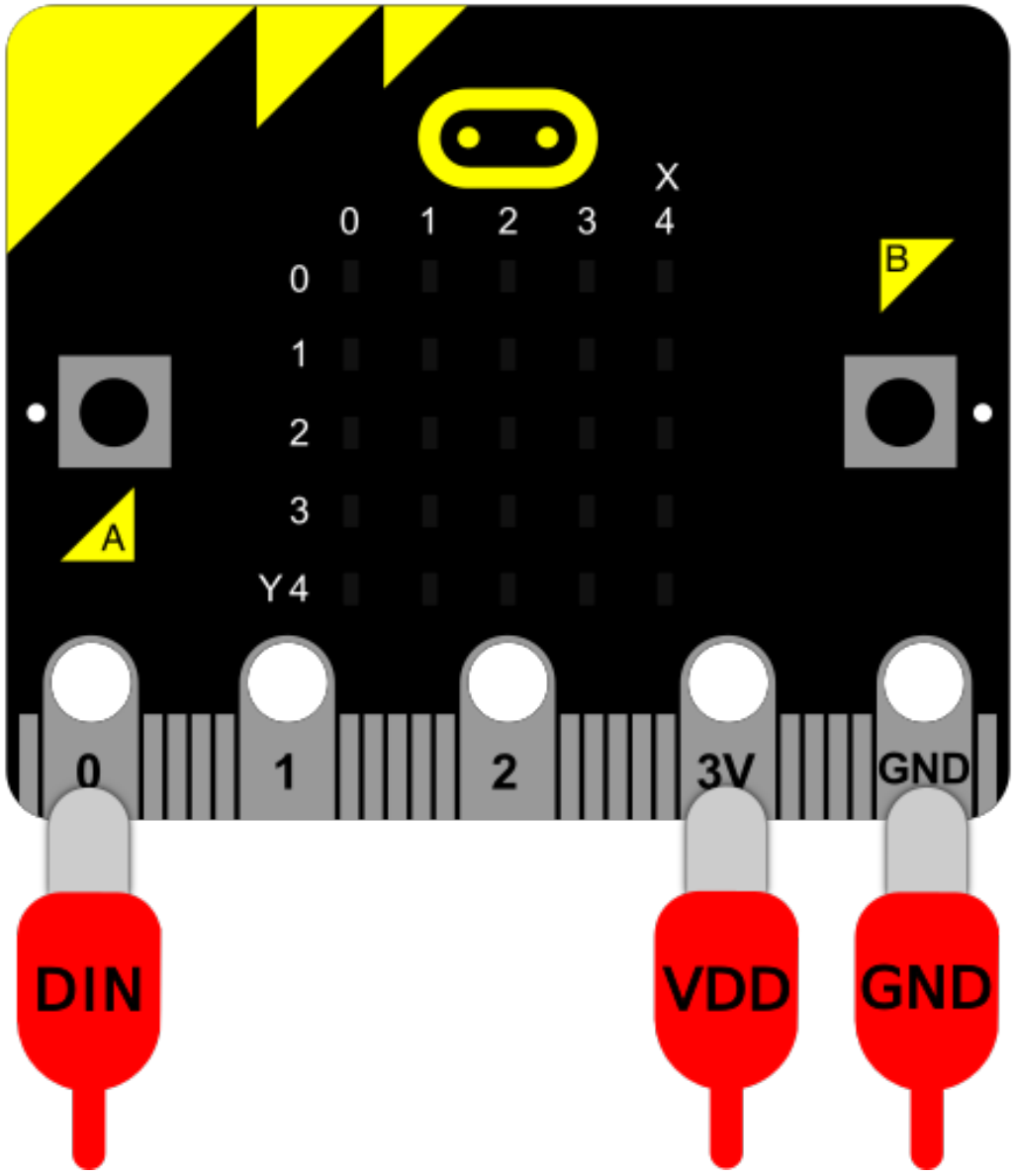
Примечание: Судя по нашим тестам, модуль Microbit NeoPixel поддерживает до 256 светодиодов. Micro:bit может подавать только 90 мА **V1** или 190 мА **V2** на внешние устройства, большое количество светодиодов требует внешнего источника питания.

NeoPixel рассчитаны на работу при напряжении 5 В, но, к счастью, они по-прежнему работают при источник питания 3В от BBC micro:bit.

Image attribution: *adafruit flexible Neopixel matrix <https://www.adafruit.com/product/2547>*

Чтобы соединить ленту светодиодов, вам нужно присоеденить micro:bit, как показано на рисунке. ниже (при условии, что Вы хотите управлять пикселями с контакта 0). Вывод VDD может быть помечен как-то еще на некоторых вариантах «V+». В некоторых случаях он может называться «+5V», и его безопасно использовать только в том случае, если у вас нет других устройств 5V подключеных к Microbit.

Предупреждение: Не используйте разъем 3 В на Microbit для питания более 8 светодиодов за раз. Если вы хотите использовать более 8, вы должны использовать отдельный разъем 3v-5v. Желательно иметь блок питания для светодиодной ленты.



Класс

`class neopixel.NeoPixel(`*pin*, *n*`)`

`class neopixel.NeoPixel(`*pin*, *n*, *bpp*`)`

Создать новый обект ленты с `n` светодиодами, подключенную к контакту `pin`. Microbit **V2** также поддерживает светодиоды RGBW, поэтому третий аргумент может передавать `NeoPixel` количество байтов на пиксель (bpp). Для RGBW это `4` а не по умолчанию `3` за RGB и GRB.

Каждый пиксель адресуется по положению (начиная с 0). Неопиксели RGB (красный, зеленый, синий) / RGBW (красный, зеленый, синий, белый) **V2** значения от 0 до 255 в виде кортежа. Например, в RGB, `(255,255,255)`. Вариант RGBW, `(255,255,255,0)` или `(0,0,0,255)` - белый цвет.

`clear()`

Очистить все пиксели.

`show()`

`write()`

Включить светодиоды. Вызывается, чтобы любые обновления стали видимыми. Для micro:bit **V2** есть дополнительная команда``write()`` метод доступен и эквивалентен `show()`

`fill(`*colour*`)`

V2 Включает светодиоды заданным значением RGB/RGBW. Параметр *colour* должен быть кортеж той же длины, что и количество байтов на пиксель (бпп). Например `fill((0,0, 255))`. Используйте вместе с `show()`, чтобы обновить состояние.

Команды

Запись цвета не обновляет дисплей (используйте `show()` для этого).

```
np[0] = (255, 0, 128)  # первый светодиод
np[-1] = (0, 255, 0)  # последний светодиод
np.show()  # только после этой команды они поменяют цвет
```

Чтобы проверить цвет светодиода (созданного как список) Вызовите команду `print`.

```
print(np[0])
```

Использование Neopixels

Взаимодействуйте с лентой и библиотекой Neopixels, как со списком кортежей. Каждый кортеж представляет смесь цветов RGB (красный, зеленый и синий) / RGBW (красный, зеленый, синий и белый) для конкретного светодиода. Значения RGBW могут варьироваться от 0 до 255..

Например, инициализируйте полосу из 8 светодиодов RGB на полосе, подключенной к контакту 0:

```
import neopixel
np = neopixel.NeoPixel(pin0, 8)
```

Установите светодиоды, индексируя их (как со списком Python). Например, чтобы установить первый светодиод на полную яркость красного цвета:

```
np[0] = (255, 0, 0)
```

Или последний светодиод в фиолетовый:

```
np[-1] = (255, 0, 255)
```

Наконец, чтобы передать новые данные о цвете, используйте метод .show():

```
np.show()
```

Примечание: Если вы не видите никаких изменений на ленте, убедитесь, что у вас есть `show()`

Примеры

```
"""
    Повторно отображает случайные цвета на светодиодной ленте.
"""
from random import randint

import neopixel
from microbit import pin0, sleep

# Установите длину ленты
np = neopixel.NeoPixel(pin0, 8)

while True:
    # Включение ленты
    for pixel_id in range(0, len(np)):
        red = randint(0, 60)
        green = randint(0, 60)
        blue = randint(0, 60)
        # Назначьте текущему светодиоду случайное значение красного, зеленого и синего␣
↪цветов от 0 до 60
        np[pixel_id] = (red, green, blue)
        # Отображение светодиодов
        np.show()
        sleep(100)
```

1.14.9 Библиотека OS

MicroPython содержит модуль «os», основанный на модуле «os» в Стандартной библиотеке Python. Он используется для доступа к «операционной системе». Так как на Microbit нет операционной системы в MicroPython модуль обеспечивает функции, относящиеся к управлению ростой постоянной файловой системы на устройстве и выводит информацию о текущей системе.

Для доступа к этому модулю вам необходимо:

```
import os
```

Мы предполагаем, что вы сделали это для приведенных ниже примеров.

Команды

`os.listdir()`

Возвращает список имен всех файлов, содержащихся в локальном постоянная файловая система на устройстве.

`os.remove(`*filename*`)`

Удаляет (удаляет) файл, указанный в аргументе `filename`. Если файл не существует `OSError` произойдет исключение.

`os.size(`*filename*`)`

Возвращает размер в байтах файла, указанного в аргументе `filename`. Если файл не существует `OSError` произойдет исключение.

`os.uname()`

Возвращает информацию, идентифицирующую текущую операционную систему. Возврат значения — это объект с пятью атрибутами:

- `sysname` - название операционной системы
- `nodename` - имя машины в сети (определяется реализацией)
- `release` - выпуск операционной системы
- `version` - версия операционной системы
- `machine` - аппаратный идентификатор

Примечание: В MicroPython нет базовой операционной системы. В результате информация, возвращенная `uname` в основном полезна для сведения о версиях.

1.14.10 Радио

`radio` Модуль позволяет устройствам работать вместе через беспроводную сеть.

Радиомодуль концептуально очень прост:

- Широковещательные сообщения имеют определенную настраиваемую длину (до 251 байта).
- Полученные сообщения считываются из очереди настраиваемого размера. Если очередь заполнена, новые сообщения игнорируются. Чтение сообщения удаляет его из очереди.
- Сообщения передаются и принимаются по предварительно выбранному каналу (с номерами 0-83).
- Трансляции имеют определенный уровень мощности — чем больше мощность, тем больше радиус действия.
- Сообщения фильтруются по адресу и группе.
- Скорость пропускной способности может быть одной из трех предустановленных настроек.
- Отправка и получение байтов для работы с произвольными данными.
- Используйте *receive_full*, чтобы получить полную информацию о входящем сообщении: данные, уровень принимаемого сигнала и отметку времени в микросекундах, когда сообщение было получено.
- Для удобства детей можно легко отправлять и получать сообщения в виде строк.
- Конфигурация по умолчанию разумна и совместима с другими платформами, предназначенными для BBC micro:bit..

Для доступа к этому модулю вам необходимо:

```
import radio
```

Мы предполагаем, что Вы сделали это для приведенных ниже примеров.

Константы

`radio.RATE_250KBIT`

Константа, используемая для обозначения пропускной способности 256 Кбит в секунду.

`radio.RATE_1MBIT`

Константа, используемая для обозначения пропускной способности 1 Мбит в секунду.

`radio.RATE_2MBIT`

Константа, используемая для обозначения пропускной способности 2 Мбит в секунду.

Функции

`radio.on()`

Включает радио. Это должно быть вызвано явно, так как радио потребляет энергию и занимает память, которая в противном случае может понадобиться.

`radio.off()`

Выключает радио, тем самым экономя энергию и память.

`radio.config(`***kwargs*`)`

Конфигурирует различные настройки на основе ключевых слов, относящиеся к радио.

`length` (по умолчанию=32) определяет максимальную длину сообщения в байтах, переданное по радио. Он может иметь длину до 251 байта (254–3 байта).

`queue` (по умолчанию=3) указывает количество сообщений, которые могут хранится в очереди входящих сообщений. Если не осталось свободных мест в очереди для входящих сообщений, входящее сообщения отбрасываются.

`channel` (по умолчанию=7) может быть целым числом от 0 до 83 (включительно), которое определяет произвольный «канал» радио. Сообщения будут отправляться через этот канал, и только полученные сообщения через этот канал будут помещены в очередь входящих сообщений. Каждый шаг Ширина 1 МГц, частота 2400 МГц.

`power` (по умолчанию=6) — целочисленное значение от 0 до 7 (включительно) указывает мощность сигнала, используемого при передаче сообщения. Чем выше значение, тем сильнее сигнал, но тем больше энергии потребляется устройством. Нумерация переводится на позиции в следующем списке значений дБм: -30, -20, -16, -12, -8, -4, 0, 4.

`address` (по умолчанию = 0x75626974) — произвольное имя, выраженное в виде 32-битного адреса, который используется для фильтрации входящих пакетов на оборудовании. Сохраняет только те, которые соответствуют заданному вами адресу.

`group` (по умолчанию = 0) представляет собой 8-битное значение (0-255), используемое с `address` при фильтрации сообщений.

`data_rate` (по умолчанию = радио.RATE_1MBIT) указывает скорость, с которой происходит передача данных. Может быть одной из следующих констант, определенных в `radio`: `RATE_250KBIT`, `RATE_1MBIT` или `RATE_2MBIT`.

Если `config` не вызывается, то принимаются значения по умолчанию.

`radio.reset()`

Сбросьте настройки до их значений по умолчанию (как указано в документации для `config`.

Примечание: Ни один из следующих методов отправки или приема не будет работать, пока радио не будет включенно.

`radio.send_bytes(`*message*`)`

Отправляет сообщение, содержащее байты.

`radio.receive_bytes()`

Получите следующее входящее сообщение в очереди сообщений. Возвращает `None`, если нет ожидающих сообщений. Сообщения возвращаются в виде байтов.

`radio.receive_bytes_into(`*buffer*`)`

Получите следующее входящее сообщение в очереди сообщений. Копирует сообщение в `buffer`, обрезая конец сообщения, если это необходимо. Возвращает `None`, если нет ожидающих сообщений, в противном случае возвращает длину сообщения (которое может быть больше, чем длина буфера).

`radio.send(`*message*`)`

Отправляет строку сообщения. Это эквивалент `send_bytes(bytes(message, 'utf8'))` но с `b'\x01\x00\x01'` в начале (чтобы сделать его совместимым с другими платформами, которые поддерживают Microbit).

`radio.receive()`

Работает точно так же, как `receive_bytes`, но возвращает сообщение если не было отправленно.

В настоящее время он эквивалентен `str(receive_bytes(), 'utf8')`, но проверяет, что первые три байта равны `b'\x01\x00\x01'` (чтобы сделать его совместим с другими платформами, которые поддерживают micro:bit).

`ValueError` исключение возникает, если преобразование в строке не удается.

`radio.receive_full()`

Возвращает кортеж, содержащий три значения. Если нет ожидающих сообщений, то возвращает `None`.

Три значения в кортеже представляют:

```
* следующее входящее сообщение в очереди сообщений в виде байтов.
* RSSI (сила сигнала): значение от 0 (самый сильный) до -255 (самый слабый),␣
↪измеренное в дБм.
* отметка времени в микросекундах: значение, возвращаемое ``time.ticks_us()``когда␣
↪сообщение было получено.
```

Например:

```
details = radio.receive_full()
if details:
    msg, rssi, timestamp = details
```

эта функция полезна для предоставления информации, необходимой для триангуляции и/или трилитерация с другими устройствами micro:bit.

Примеры

```
# A micro:bit Firefly.
# By Nicholas H.Tollervey. Released to the public domain.
import radio
import random
from microbit import display, Image, button_a, sleep

# Анимация вспышка
flash = [Image().invert()*(i/9) for i in range(9, -1, -1)]

# Включить радиомодуль.
radio.on()

# Вечный цикл.
while True:
    # Если кнопка A нажата отправить сообщение.
    if button_a.was_pressed():
        radio.send('flash')
    # Читать входящие сообщения
    incoming = radio.receive()
    if incoming == 'flash':
        # Если пришло сообщение"flash" проиграть анимацию после случайно задержки
        sleep(random.randint(50, 350))
        display.show(flash, delay=100, wait=False)
        # Передать ответное сообщение.
        if random.randint(0, 9) == 0:
            sleep(500)
            radio.send('flash')
```

1.14.11 Random (Генератор случайных чисел)

Этот модуль основан на `random` в стандартной библиотеке Python. Он содержит функции для генерации случайного поведения.

Для доступа к этому модулю вам необходимо:

```
import random
```

Мы предполагаем, что Вы сделали это для приведенных ниже примеров.

Функции

`random.getrandbits(`*n*`)`

Возвращает целое случайное число `n`.

Предупреждение: Поскольку базовая функция генератора возвращает не более 30 бит, `n` может быть только значением от 1 до 30 (включительно).

`random.seed(`*n*`)`

Используется для инициализации случайных чисел. По умолчанию генератор случайных чисел использует текущее системное время. Если вы дважды используете одно и то же начальное значение, вы получите один и тот же результат, что означает случайное число дважды.

`random.randint(`*a*, *b*`)`

Возвращает случайное целое число `N` в диапазоне `a <= N <= b`.

`random.randrange(`*stop*`)`

Возвращает случайно выбранное целое число от нуля до `stop`.

`random.randrange(`*start*, *stop*`)`

Возвращает случайно выбранное целое число из последовательности `range(start, stop)` .

`random.randrange(`*start*, *stop*, *step*`)`

Возвращает случайно выбранное целое число из последовательности с шагом `step`.

`random.choice(`*seq*`)`

Возвращает случайный элемент из непустой последовательности `seq`. Если `seq` является пустой, возвращает `IndexError`.

`random.random()`

Возвращает следующее случайное число с плавающей запятой в диапазоне [0.0, 1.0)

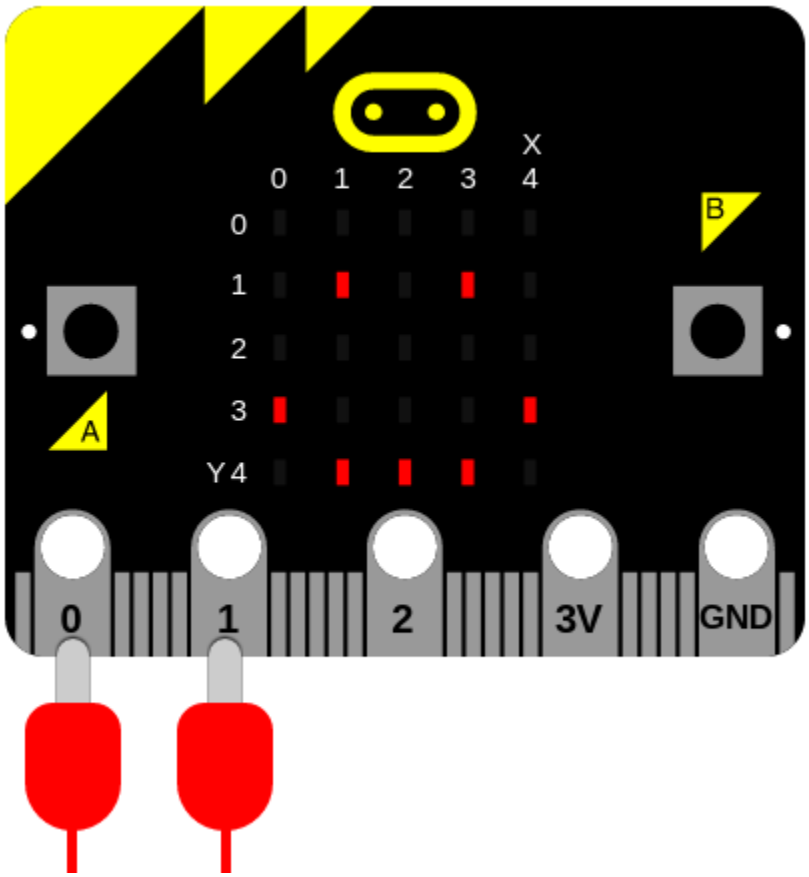
`random.uniform(`*a*, *b*`)`

Возвращает случайное число с плавающей запятой `N` такой, что `a <= N <= b` для `a <= b` и `b <= N <= a` для `b < a`.

1.14.12 Генератор речи

Этот модуль заставляет micro:bit говорить, петь и издавать другие звуки, похожие на речь.

built-in speaker **V2**. Вы можете подключить проводные наушники или динамик к контакту 0 и GND на краевом разъеме, чтобы слышать звук:



Примечание: Эта работа основана на удивительных усилиях по обратному инжинирингу Себастьян Маке на основе старой программы преобразования текста в речь (TTS) под названием SAM. (программное обеспечение Automated Mouth), первоначально выпущенное в 1982 г. Commodore 64. Результатом стала небольшая библиотека C, которую разработчики приняли и адаптирован для micro:bit. Вы можете узнать больше из his homepage. Много информации в этом документе была получено от первоначального руководства, которое можно найти here.

Синтезатор речи может воспроизводить звук продолжительностью около 2,5 секунд от 0 до 255 символов текстового ввода.

Для доступа к этому модулю вам необходимо:

```
import speech
```

Функции

`speech.translate(`*words*`)`

Данные английские слова в строке `words`, вернуть строку, содержащую лучшее предположение о подходящих фонемах для произнесения. Выход генерируется с использованием библиотеки https://github.com/s-macke/SAM/wiki/Text-to-phoneme-translation-table.

Эту функцию следует использовать для генерации первого приближения фонем, которые могут быть дополнительно отредактированы вручную для улучшения точности, интонации и акцент.

`speech.pronounce(`*phonemes*, ***, *pitch=64*, *speed=72*, *mouth=128*, *throat=128*`)`

`speech.pronounce(`*phonemes*, ***, *pitch=64*, *speed=72*, *mouth=128*, *throat=128*, *pin=pin0*`)`

Произнесите фонемы в строке `phonemes`. Подробнее смотреть ниже как использовать фонемы для точного управления выходным сигналом синтезатора речи. Переопределите дополнительные настройки высоты тона, скорости, рта и горла, чтобы изменить тембр (качество) голоса.

`speech.say(`*words*, ***, *pitch=64*, *speed=72*, *mouth=128*, *throat=128*`)`

`speech.say(`*words*, ***, *pitch=64*, *speed=72*, *mouth=128*, *throat=128*, *pin=pin0*`)`

Произнесите английские слова в строке `words`.

`speech.sing(`*phonemes*, ***, *pitch=64*, *speed=72*, *mouth=128*, *throat=128*`)`

`speech.sing(`*phonemes*, ***, *pitch=64*, *speed=72*, *mouth=128*, *throat=128*, *pin=pin0*`)`

Пропойте фонемы, содержащиеся в строке `phonemes`. Изменение высоты тона и продолжительность ноты описана далее. Переопределите необязательный шаг, настройки скорости, рта и горла, чтобы изменить тембр (качество) голос.

Пунктуация

Пунктуация используется для изменения подачи речи. Синтезатор понимает четыре знака препинания: дефис, запятую, точку и вопросительный знак.

Дефис (`-`) вставляет короткую паузу в речи.

Запятая (`,`) отмечает границу фразы и вставляет паузу примерно вдвое больше дефиса.

Точка (`.`) и вопросительный знак (`?`) обозначает конец предложения.

Точка вставляет паузу и заставляет высоту тона падать.

Тембр

Тембр звука – это качество звука. Это разница между голос робота и голосом человека. Чтобы контролировать тембр меняйте числовые настройки `pitch`, `speed`, `mouth` и `throat`.

Высота тона и скорость настройки достаточно очевидны и в основном сводятся к следующемим категориям:

Pitch:

- 0-20 непрактичный
- 20-30 очень высоко
- 30-40 высокий
- 40-50 высокий нормальный
- 50-70 нормальный

- 70-80 низкий нормальный
- 80-90 низкий
- 90-255 очень низко

(По умолчанию 64)

Скорость:

- 0-20 непрактичный
- 20-40 очень быстро
- 40-60 быстрый
- 60-70 быстрый разговорный
- 70-75 нормальный разговорный
- 75-90 повествование
- 90-100 медленный
- 100-225 очень медленный

(По умолчанию 72)

Значения рта и горла немного сложнее объяснить, и следующие описания основаны на наших слуховых впечатлениях от речи, значение каждой настройки можно изменять.

Для начала вот несколько примеров:

```
speech.say("I am a little robot",  speed=92, pitch=60, throat=190, mouth=190)
speech.say("I am an elf", speed=72, pitch=64, throat=110, mouth=160)
speech.say("I am a news presenter", speed=82, pitch=72, throat=110, mouth=105)
speech.say("I am an old lady", speed=82, pitch=32, throat=145, mouth=145)
speech.say("I am E.T.", speed=100, pitch=64, throat=150, mouth=200)
speech.say("I am a DALEK - EXTERMINATE", speed=120, pitch=100, throat=100, mouth=200)
```

Фонемы

`say` функция позволяет легко произносить речь, но часто это не точное. Чтобы убедиться, что синтезатор речи произносит вещи *правильно* как хотите, нужно использовать фонемы: самые маленькие воспринимаемые единицы звука, которые можно использовать для различения различных слова.

`pronounce` функция принимает строку, содержащую упрощенную и читаемую версию International Phonetic Alphabet и необязательные аннотации для указания интонация и ударение.

Преимущество использования фонем в том, что вам не нужно знать, как правильно писать! Скорее, вам нужно только знать, как сказать слово, чтобы написать его по буквам. фонетически.

В таблице ниже перечислены фонемы, понятные синтезатору.

Примечание: Таблица содержит фонему в виде символов и пример слова. То слова-примеры имеют звук фонемы (в скобках), но не обязательно одинаковые буквы

ПРОСТЫЕ ГЛАСНЫЕ ЗВОНКОВЫЕ СОГЛАСНЫЕ IY f(ee)t R (r)ed IH p(i)n L a(ll)ow EH b(e)g W a(w)ay AE S(a)m W (wh)ale AA p(o)t Y (y)ou AH b(u)dget M Sa(m) AO t(al)k N ma(n) OH c(o)ne NX so(ng) UH b(oo)k B (b)ad UX l(oo)t D (d)og ER b(ir)d G a(g)ain AX gall(o)n J (j)u(dg)e IX dig(i)t Z (z)oo ZH plea(s)ure ДИФТОНГИ V se(v)en EY m(a)de DH (th)en AY h(igh) OY b(oy) AW h(ow) OW sl(ow) UW cr(ew)

ГЛУХИЕ СОГЛАСНЫЕ S (S)am SH fi(sh) F (f)ish TH (th)in P (p)oke T (t)alk K (c)ake CH spee(ch) /H a(h)ead

СПЕЦИАЛЬНЫЕ ФОНЕМЫ UL sett(le) (=AXL) UM astron(om)y (=AXM) UN functi(on) (=AXN) Q kitt-en (glottal stop)

Пользователю также доступны следующие нестандартные символы:

```
YX          окончание дифтонга (weaker version of Y)
WX          diphthong ending (weaker version of W)
RX          R после гласной (smooth version of R)
LX          L после гласной (smooth version of L)
/X          H перед гласной или согласной не переднего ряда - как в (wh)o
DX          T как в pi(t)y (weaker version of T)
```

Вот некоторые редко используемые комбинации фонем (и предлагаемые альтернативы):

```
ФОНЕМА        НАВЕРНОЕ ВЫ ХОТИТЕ:     ЕСЛИ ЭТО НЕ РАЗДЕЛЯЕТ СЛОГИ, КАК:
КОМБИНАЦИЯ
GS             GZ e.g. ba(gs)         bu(gs)pray
BS             BZ e.g. slo(bz)        o(bsc)ene
DS             DZ e.g. su(ds)         Hu(ds)son
PZ             PS e.g. sla(ps)        -----
TZ             TS e.g. cur(ts)y       -----
KZ             KS e.g. fi(x)          -----
NG             NXG e.g. singing       i(ng)rate
NK             NXK e.g. bank          Su(nk)ist
```

Если Вы используете что-либо кроме фонем, описанных выше, `ValueError` будет возбуждено исключение. Передайте фонемы в виде такой строки:

```
speech.pronounce("/HEHLOW")  # "Hello"
```

Фонемы делятся на две большие группы: гласные и согласные.

Ниже описнные правила относятся к английскому языку

Гласные далее подразделяются на простые гласные и дифтонги. Простые гласные не меняют звучание, когда вы их произносите, тогда как дифтонги начинаются с единицы звучат и заканчиваются другим. Например, когда вы произносите слово «масло», «ой» гласная начинается со звука «о», но меняется на звук «и».

Согласные также подразделяются на две группы: звонкие и глухие. Озвученный согласные требуют, чтобы говорящий использовал свои голосовые связки для воспроизведения звука. Например, такие согласные, как «Л», «Н» и «З», звонкие. Глухие согласные производятся потоком воздуха, например «Р», «Т» и «SH».

Как только вы привыкнете к этому, система фонем станет легкой. Для начала немного написание может показаться сложным (например, в слове «приключение» есть буква «СН»), но Правило состоит в том, чтобы писать то, что вы говорите, а не то, что пишете. Эксперименты лучше всего способ разрешения проблемных слов.

Также важно, чтобы речь звучала естественно и понятно. Помогать с улучшением качества разговорного вывода часто полезно использовать встроенный система ударения для добавления интонации или ударения.

Имеется восемь маркеров стресса, обозначенных цифрами от 1 до 8. Просто вставьте необходимое число после ударной гласной. Например, отсутствие выражения «/HEHLOW» значительно улучшается (и

дружелюбнее), когда написано «/HEH3LOW».

Также возможно изменить значение слов через то, как они подчеркнул. Рассмотрим фразу «Почему я должен идти в магазин пешком?». Возможно произносится по-разному:

```
# Вам нужна причина, чтобы сделать это.
speech.pronounce("WAY2 SHUH7D AY WAO5K TUX DHAH STOH5R.")
# Вы не хотите идти.
speech.pronounce("WAY7 SHUH2D AY WAO7K TUX DHAH STOH5R.")
# Вы хотите, чтобы это сделал кто-то другой.
speech.pronounce("WAY5 SHUH7D AY2 WAO7K TUX DHAH STOHR.")
# Вы бы предпочли водить.
speech.pronounce("WAY5 SHUHD AY7 WAO2K TUX7 DHAH STOHR.")
# Вы хотите пойти куда-нибудь еще.
speech.pronounce("WAY5 SHUHD AY WAO5K TUX DHAH STOH2OH7R.")
```

Проще говоря, разные ударения в речи создают более выразительный тон речи. голос.

Они работают, повышая или понижая высоту тона и удлиняя соответствующую гласную. звук в зависимости от числа, которое вы даете:

1. очень эмоциональный стресс
2. очень сильное ударение
3. довольно сильный стресс
4. обычный стресс
5. жесткий стресс
6. нейтральное (без изменения высоты тона) напряжение
7. сбрасывающее напряжение
8. экстремальное напряжение сбрасывания высоты тона

Чем меньше число, тем более экстремальным будет акцент. Однако такие маркеры ударения помогут правильно произносить сложные слова. Например, если слог произносится недостаточно, поставьте маркер нейтрального ударения.

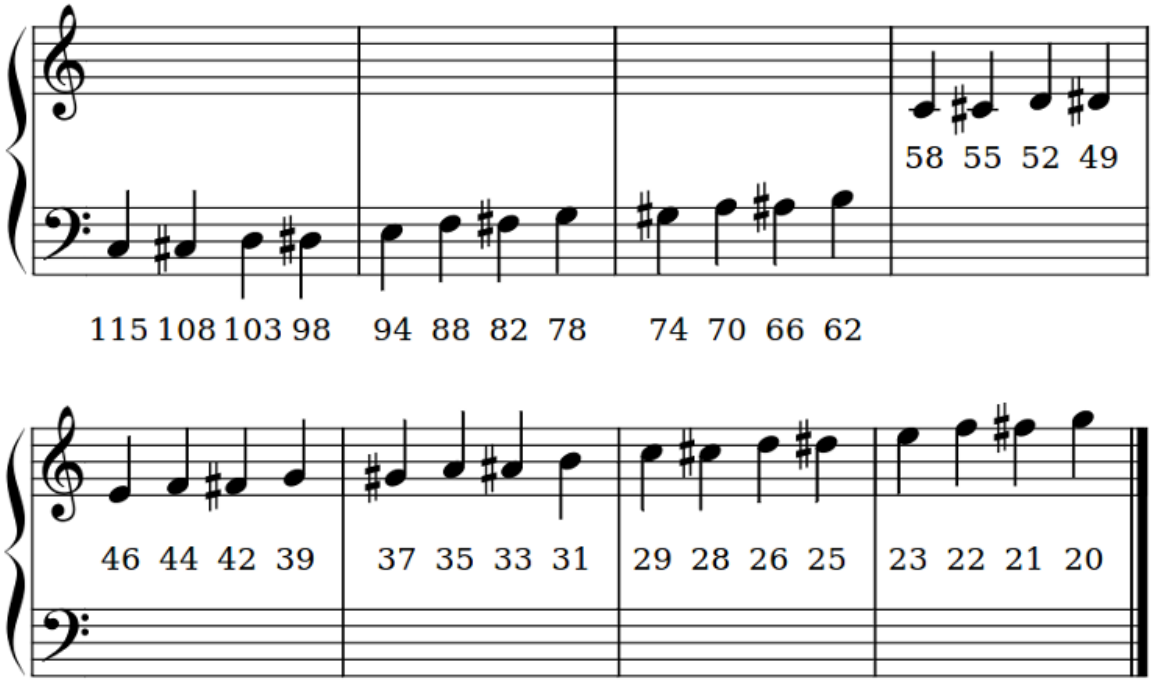
Также можно удлинять слова с помощью маркеров ударения.:

```
speech.pronounce("/HEH5EH4EH3EH2EH2EH3EH4EH5EHLP.'')
```

Пение

Можно заставить MicroPython петь фонемы.

Это делается путем аннотирования числа, связанного с высотой тона, к фонеме. Чем ниже число, тем выше тон. Числа примерно переводятся в музыкальные ноты как показано на схеме ниже:



Аннотации работают, предваряя знак решетки (#) и номер шага перед фонемой. Шаг останется прежним, пока не появится новая аннотация. Например, заставьте MicroPython петь вот такую гамму:

```
solfa = [
    "#115DOWWWWWW",   # Doh
    "#103REYYYYYY",   # Re
    "#94MIYYYYYY",    # Mi
    "#88FAOAOAOAOR",  # Fa
    "#78SOHWWWWW",    # Soh
    "#70LAOAOAOAOR",  # La
    "#62TIYYYYYY",    # Ti
    "#58DOWWWWWW",    # Doh
]
song = ''.join(solfa)
speech.sing(song, speed=100)
```

Для того, чтобы спеть ноту на определенную продолжительность, удлините повторяя гласные или звонкие согласные фонемы (как показано в пример выше).

Экспериментирование, внимательное слушание и корректировка — единственный верный способ работать.

Как это работает?

В оригинальном мануале все понятно:

> Во-первых, вместо того, чтобы записывать фактическую форму речевого сигнала, мы сохраняем только частотные спектры. Делая это, мы экономим память и подбираем другие преимущества. Во-вторых, мы [...] храним некоторые данные о времени. Эти числа, относящиеся к длительности каждой фонемы при разных обстоятельствах, а также некоторые данные о времени перехода, чтобы мы могли знать, как смешать фонему с ее соседями. В-третьих, мы разрабатываем систему правил иметь дело со всеми этими данными, и, к нашему большому удивлению, наш компьютер болтать в кратчайшие сроки.
>
> —S.A.M. owner's manual.

Выходные данные передаются через функции, предоставляемые модулем «аудио» и, привет, вуаля, у нас есть говорящий микро:бит.

Примеры

```
"""
    speech.py
    ~~~~~~~~

    Simple speech example to make the micro:bit say, pronounce and sing
    something. This example requires a speaker/buzzer/headphones connected
    to P0 and GND,or the latest micro:bit device with built-in speaker.
"""
import speech
from microbit import sleep

# The say method attempts to convert English into phonemes.
speech.say("I can sing!")
sleep(1000)
speech.say("Listen to me!")
sleep(1000)

# Clearing the throat requires the use of phonemes. Changing
# the pitch and speed also helps create the right effect.
speech.pronounce("AEAE/HAEMM", pitch=200, speed=100)  # Ahem
sleep(1000)

# Singing requires a phoneme with an annotated pitch for each syllable.
solfa = [
    "#115DOWWWWWW",   # Doh
    "#103REYYYYYY",   # Re
    "#94MIYYYYYY",    # Mi
    "#88FAOAOAOAOR",  # Fa
    "#78SOHWWWWW",    # Soh
    "#70LAOAOAOAOR",  # La
    "#62TIYYYYYY",    # Ti
    "#58DOWWWWWW",    # Doh
]

# Sing the scale ascending in pitch.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
song = ''.join(solfa)
speech.sing(song, speed=100)
# Reverse the list of syllables.
solfa.reverse()
song = ''.join(solfa)
# Sing the scale descending in pitch.
speech.sing(song, speed=100)
```

1.14.13 utime (модуль работы со временем)

Модуль `utime` предоставляет функции для получения текущего времени и даты, измерения временных интервалов и задержек.

Примечание: Модуль `utime` представляет собой реализацию MicroPython стандартного модуля Python. модуль `time`. Его можно импортировать как с помощью `import utime`, так и с помощью `import time`, но модуль тот же.

Функции

`utime.sleep(`*seconds*`)`

Задержка в течение заданного количества секунд. Вы можете использовать число с плавающей запятой заснуть на долю секунды или использовать

`utime.sleep_ms()` и *`utime.sleep_us()`*.

`utime.sleep_ms(`*ms*`)`

Задержка для заданного количества миллисекунд должна быть положительной или равной 0.

`utime.sleep_us(`*us*`)`

Задержка для заданного количества микросекунд должна быть положительной или равной 0.

`utime.ticks_ms()`

Возвращает увеличивающийся счетчик миллисекунд с произвольной контрольной точкой.

`utime.ticks_us()`

Так же, как *`utime.ticks_ms()`* выше, но в микросекундах.

`utime.ticks_add(`*ticks*, *delta*`)`

Смещение значения тиков на заданное число, которое может быть как положительным, так и отрицательный. Учитывая значение тиков, эта функция позволяет вычислить тики значение дельта тикает до или после него, следуя модульной арифметике определение тиковых значений.

Example:

```
# Узнать какое значение тиков было 100 мс назад
print(ticks_add(time.ticks_ms(), -100))

# Рассчитать срок эксплуатации и протестировать его
deadline = ticks_add(time.ticks_ms(), 200)
while ticks_diff(deadline, time.ticks_ms()) > 0:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    do_a_little_of_something()

# Узнайте TICKS_MAX, используемый этим портом
print(ticks_add(0, -1))
```

utime.ticks_diff(*ticks1*, *ticks2*)

Измерьте разницу в отметках между значениями, возвращенными из *utime.ticks_ms()* или *ticks_us()*.

Порядок аргументов такой же, как и для вычитания пользователем, `ticks_diff(ticks1, ticks2)` имеет то же значение, что и `ticks1 - ticks2`.

utime.ticks_diff() предназначен для различных видов использования разницы, среди них:

Опрос с тайм-аутом. В этом случае порядок событий известен, и вы будете иметь дело только с положительными результатами *utime.ticks_diff()*:

```
# Дождитесь подтверждения вывода GPIO, но не более 500 мкс.
start = time.ticks_us()
while pin.value() == 0:
    if time.ticks_diff(time.ticks_us(), start) > 500:
        raise TimeoutError
```

Планирование событий. В этом случае результат *utime.ticks_diff()* может быть отрицательный, если событие просрочено:

```
# Этот фрагмент кода не оптимизирован
now = time.ticks_ms()
scheduled_time = task.scheduled_time()
if ticks_diff(scheduled_time, now) > 0:
    print("Too early, let's nap")
    sleep_ms(ticks_diff(scheduled_time, now))
    task.run()
elif ticks_diff(scheduled_time, now) == 0:
    print("Right at time!")
    task.run()
elif ticks_diff(scheduled_time, now) < 0:
    print("Oops, running late, tell task to run faster!")
    task.run(run_faster=true)
```

1.15 Датчики

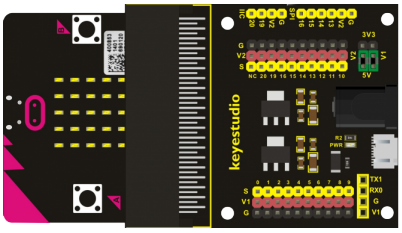
Порядок подключения файлов с классами для программы MU

1) Сохраните файл на свой компьютер.
2) Скопируйте файл в в корневой каталог программы MU (. . . mu_code).
3) Напишите и прошейте вашу программу в Mu.
4) На экране будет прокручиваться сообщение об ошибке и отсутствии библиотеки.
5) После завершения щелкните значок «Files» в Mu и загрузите файл на свой микробит (перетащите в левую колонку).

6) Нажмите Перезагрузку на Microbit.

1.15.1 Шилд для датчиков V2

Скачать файл с классов



Класс

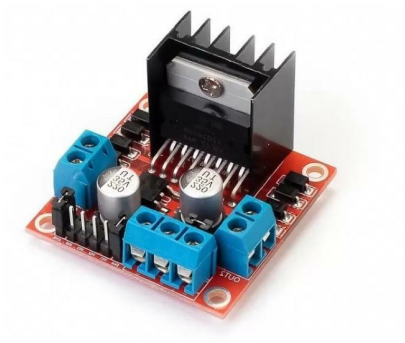
```
class Sensor_Shield
```

??()

Пример программы

1.15.2 Драйвер моторов L298

Скачать файл с классов



Класс

```
class Driver_Motor
```

Класс используется для определения объектов, имеющих поведение **трехколесного робота** с пассивным колесом. Класс следует применять при подключении моторов к драйверу двигателей. Драйвер должен иметь 6 управляющих пинов. 4 пина указывают направления и 2 пина указывают мощность мотора.

Пример объявления объекта:

```
l_motor=Motor(pin13,pin14,pin0)
r_motor=Motor(pin15,pin16,pin1)

car = Driver_Motor(l_motor,r_motor)
```

Вы объявляете объекты левого и правого мотора, которые подключены к драйверу. Передаете их параметрами в объект класса **Driver_Motor**.

`car.forward(`*speed: int*`)`

Команда указывает роботу ехать вперед. **speed** - задает скорость вращения.

`car.backward(`*speed: int*`)`

Команда указывает роботу ехать назад. **speed** - задает скорость вращения

`car.stop()`

Команда останавливает робота.

`car.left(`*speed: int*`)`

Команда указывает роботу повернуть налево. **Левое колесо-отключено**.

`car.right(`*speed: int*`)`

Команда указывает роботу повернуть направо. **правое колесо-отключено**.

Танковый разворот

Розворот робота на месте. Моторы вращаются в различных направлениях

`car.left_tank(`*speed: int*`)`

Команда указывает роботу повернуть налево.

`car.right_tank(`*speed: int*`)`

Команда указывает роботу повернуть направо.

Пример программы

```
from microbit import pin0, pin1, pin13, pin14, pin15, pin16, sleep

from Driver_Motor import Driver_Motor, Motor

l_motor = Motor(pin13, pin14, pin0)
r_motor = Motor(pin15, pin16, pin1)

car = Driver_Motor(l_motor, r_motor)

speed = 500

car.forward(speed)
sleep(1000)

car.backward(speed)
sleep(1000)
```

(continues on next page)

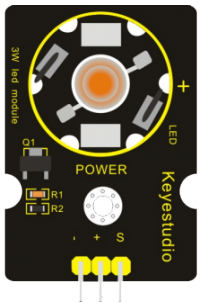
(продолжение с предыдущей страницы)

```
car.left_tank(speed)
sleep(500)
```

1.15.3 Сверхяркий светодиод

Скачать файл с классов

Светодиод - это полупроводниковый прибор. Когда через полупроводники проходит электрический ток, отрицательный заряд электронов соединяются с ионами положительно заряженных дырок. В этот момент выделяется энергия, и мы видим излучение света.



Класс

class LED_3W_Module

Класс используется для определения объектов, имеющих поведение светодиода

Пример объявления объекта:

```
led=LED_3W_Module(pin1)
```

led.on()
: Команда позволяет включить светодиод

led.off()
: Команда позволяет выключить светодиод

led.bright(*arg: int*)
: Команда включает светодиод с указанной яркостью (0-1023)

Пример программы

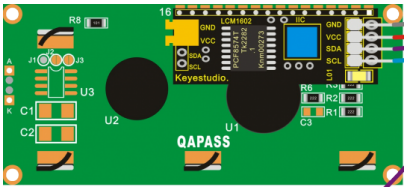
```
from microbit import *

from LED_3W_Module import LED_3W_Module

led1 = LED_3W_Module(pin0)
for i in range(1023):
    led1.bright(i)
    sleep(10)
```

1.15.4 Двухстрочный монитор 1602_I2C

`Скачать файл с классов`

I2C / IIC (Inter-Integrated Circuit) – это протокол, изначально создававшийся для связи интегральных микросхем внутри электронного устройства. Разработка принадлежит фирме Philips.

В основе i2c протокола является использование 8-битной шины, которая нужна для связи блоков в управляющей электронике, и системе адресации, благодаря которой можно общаться по одним и тем же проводам с несколькими устройствами.



- 16 знаков шириной, 2 ряда;
- Белый текст на синем фоне;
- Рабочее напряжение чипа: 4,5-5,5
- Оптимальное рабочее напряжение модуля 5,0 В.
- Включенная одиночная светодиодная подсветка легко регулируется с помощью резистора.
- Встроенный набор символов поддерживает английский текст
- Настройка потенциометром контраста

Класс

```
class 1602_I2C

??()
```

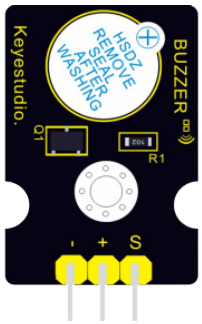
Пример программы

1.15.5 Активный зуммер

`Скачать файл с классов`

Пьезоэлектрический звуковой излучатель, относящийся к электроакустическим устройствам, и производящий слышимый звук или ультразвук с помощью обратного пьезоэлектрического эффекта.

Пьезоэлектрические зуммеры применяются в будильниках, игрушках, бытовой технике, телефонных аппаратах. Ультразвук получаемый с их помощью нередко используют в отпугивателях против грызунов, в увлажнителях воздуха, в ультразвуковой очистке.



Активный пьезоизлучатель генерирует звуковые сигналы при подачи на него напряжения. В него встроен источник колебаний.

Класс

```
class Active_Buzzer
```

```
??()
```

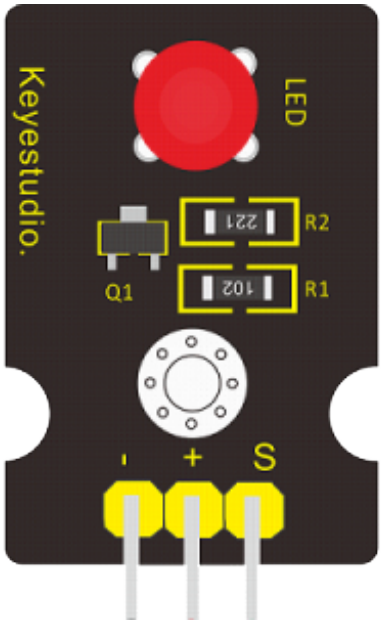
Пример программы

```
import music
from microbit import pin0, sleep

from Active_Buzzer import Active_Buzzer

sound = Active_Buzzer(pin0)

sound.beep(2000)
sleep(1000)

sound.siren(2000)
sleep(1000)

sound.play(music.BLUES)
sleep(1000)

sound.play_time(music.BIRTHDAY, 3000)
```

1.15.6 Модуль светодиода

Скачать файл с классов

Светодиод - это полупроводниковый прибор. Когда через полупроводники проходит электрический ток, отрицательный заряд электронов соединяются с ионами положительно заряженных дырок. В этот момент выделяется энергия, и мы видим излучение света.



Класс

class LED_Module

Класс используется для определения объектов, имеющих поведение светодиода

Пример объявления объекта:

```
led=LED_Module(pin1)
```

led.on()

Команда позволяет включить светодиод

led.off()

Команда позволяет выключить светодиод

led.bright(*arg: int*)

Команда включает светодиод с указанной яркостью (0-1023)

Пример программы

```
from LED_Module import LED_Module
from microbit import *

led1 = LED_Module(pin0)

led1.on()
sleep(1000)
led1.off()
```

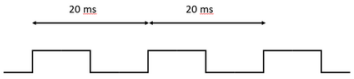
1.15.7 Сервопривод

`Скачать файл с классов`

Для контроля положения вала, на сервоприводе установлен датчик обратной связи, например потенциометр или энкодер. Позиционер преобразует угол поворота вала обратно в электрический сигнал.

Для работы сервопривода мы должны послать ему импульсный сигнал частотой 50 Гц (Гц). Это стандарт почти для всех серводвигателей постоянного тока. Импульсный сигнал выглядит так:

50 Гц означает, что один импульс происходит 50 раз в секунду. Если разбить его по-другому, один импульс отправляется каждые 1/50 секунды , что равно 20 миллисекундам (сокращенно мс).



Класс

`class Micro_Servo`

Класс используется для определения объектов, имеющих поведение серво-моторов.

Пример объявления объекта:

```
servo=Micro_Servo(pin0)
```

`servo.angle(`*arg: int*`)`

Команда перемещает сервомотор в указанный угол.

Пример программы

```
from microbit import button_a, pin0, sleep

from Micro_Servo import Micro_Servo

servo = Micro_Servo(pin0)

while True:
```

(continues on next page)

(продолжение с предыдущей страницы)

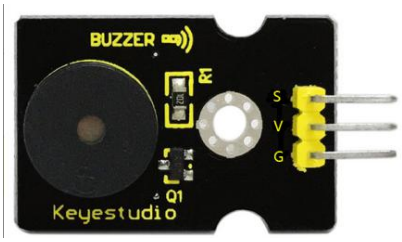
```
    if button_a.was_pressed():
        for i in range(180):
            servo.angle(i)
            sleep(10)
```

1.15.8 Пассивный зуммер

Скачать файл с классов

Пьезоэлектрический звуковой излучатель, относящийся к электроакустическим устройствам, и производящий слышимый звук или ультразвук с помощью обратного пьезоэлектрического эффекта.

Пьезоэлектрические зуммеры применяются в будильниках, игрушках, бытовой технике, телефонных аппаратах. Ультразвук получаемый с их помощью нередко используют в отпугивателях против грызунов, в увлажнителях воздуха, в ультразвуковой очистке.



Пассивный пьезоизлучатель не генерирует звуковой сигнал при подачи напряжения, генерация происходит за счет внешнего источника колебаний.

Класс

class Passive_Buzzer

Класс используется для определения объектов, имеющих поведение пассивного зуммера

Пример объявления объекта:

```
sound=Passive_Buzzer(pin0)
```

sound.sound.play(*music.BLUES*)

Команда позволяет проиграть мелодию. Аналогична команде **music.play(music.BLUES, pin0)**

sound.play_time(*music.BIRTHDAY*, *3000*)

Команда позволяет проиграть мелодию определенное время. Блокирует Microbit

sound.stop()

Команда позволяет остановить проигрывание времени.

Пример программы

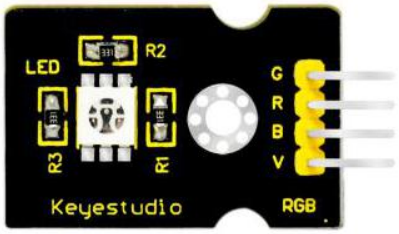
```
import music
from microbit import pin0

from Passive_Buzzer import Passive_Buzzer

sound = Passive_Buzzer(pin0)
sound.play(music.BLUES)
sound.play_time(music.BIRTHDAY, 3000)
```

1.15.9 Модуль светодиода RGB

Скачать файл с классов

RGB-светодиод — это совмещённые в одном корпусе светодиоды красного, зелёного и синего цветов.

Светодиод имеет 4 ноги. 3 ноги — аноды, соответствующие отдельным цветам и одна — общий катод. Подавая сигнал на один из анодов, можно добиться свечения одним из цветов. Используя широтно-импульсную модуляцию (PWM-сигнал) для всех анодов одновременно, можно получить свечение произвольным цветом.



Класс

class RGB_LED_Module

Класс используется для определения объектов, имеющих поведение светодиода-RGB

Пример объявления объекта:

```
led=RGB_LED_Module(red=pin0, green=pin1, blue=pin2)
```

led.on_red()

Команда позволяет включить красный свет светодиода. Аналогичные команды **on_green()** и **on_blue()**.

led.off_red()

Команда позволяет выключить красный свет светодиода. Аналогичные команды **off_green()** и **off_blue()**.

led.bright_red(*arg: int*)

Команда позволяет включить красный свет с указанной яркостьюсветодиода. Аналогичные команды **bright_green(arg:int)** и **bright_blue(arg:int)**.

`led.off()`

Команда позволяет выключить светодиод полностью

`led.conversion`(*start=led.red*, *finish=led.green*, *time=3000*)

Команда позволяет сделать планый переход от одного цвета к другому (от красного к зеленому) за указанное время (мм).

`random_rgb`(*time=3000*, *step=10*)

Команда выводит случайные цвета светодиода с шагом step в течении указанного времени time

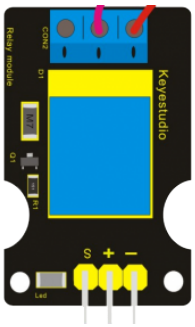
Пример программы

```
from microbit import *
from RGB_LED_Module import RGB_LED_Module

led=RGB_LED_Module(pin0,pin1,pin2)


led.random_rgb(3000,100)
sleep(2000)

led.conversion(led.red, led.green, 3000)
sleep(2000)

led.conversion(start=led.red, finish=led.green, time=3000) # можно использовать явную␣
↪передачу параметров
```

1.15.10 Реле

`Скачать файл с классов`

Реле позволяет подключать устройства более высокого напряжения и управлять ими через микроконтроллер. Вы передаете сигнал на Реле и он замыкает цепь с более высоким напряжением, которая подключена к другим контактам



Класс

```
class Single_Relay
```

Класс используется для определения объектов, имеющих поведение реле

Пример объявления объекта:

```
relay=LED_Module(pin1)
```

`relay.on()`
: Команда позволяет включить светодиод

`relay.off()`
: Команда позволяет выключить светодиод

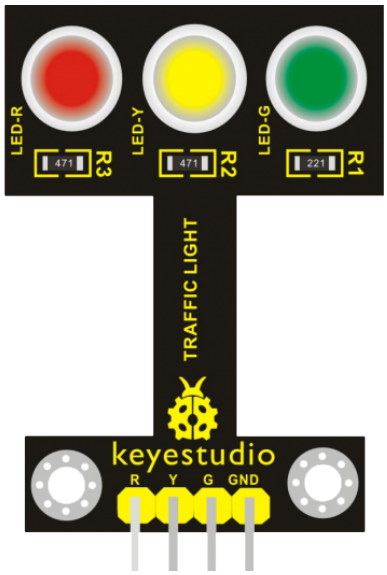
Пример программы

```
from microbit import pin1, sleep

from Single_Relay import Single_Relay

relay = Single_Relay(pin1)

relay.on()
sleep(1000)
relay.off()
```

1.15.11 Модуль светофор

`Скачать файл с классов`

Модель имеет три светодиода - отражает работу светофора. Имеет 4 пина. Три для управления напряжением на светодиодах и один GND



Класс

`class Traffic_Light`

Класс используется для определения объектов, имеющих поведение светодиодного светофора

Пример объявления объекта:

```
led=Traffic_Light(red=pin0, yellow=pin1, blue=pin2)
```

`led.on_red()`

Команда позволяет включить красный светодиод. Аналогичные команды **on_yellow()** и **on_blue()**.

`led.off_red()`

Команда позволяет выключить красный светодиод. Аналогичные команды **off_yellow()** и **off_blue()**.

`led.off()`

Команда позволяет выключить все светодиоды полностью

Пример программы

```
from microbit import pin0, pin1, pin2, sleep

from Traffic_Light import Traffic_Light

led1 = Traffic_Light(pin0, pin1, pin2)


while True:
    led1.on_red()
    sleep(1000)
    led1.off_red()
    led1.on_yellow()
    sleep(1000)
    led1.off_yellow()
    led1.on_green()
    sleep(1000)
    led1.off_green()
```

1.15.12 Мотор

`Скачать файл с классов`

Коллекторный моторчик может быть рассчитан на разное напряжения питания. Если двигатель работает от 3-5 Вольт, то можно моторчик подключать напрямую к плате . Если моторы рассчитаны на большее напряжение ими следует управлять через полевой (биполярный) транзистор или через драйвер L298N.

Класс

`class Motor`

Класс используется для определения объектов, имеющих поведение моторов. Класс следует применять при подключении моторов к драйверу двигателей. Драйвер должен иметь 6 управляющих пинов. 4 пина указывают направления и 2 пина указывают мощность мотора.

Пример объявления объекта:

```
l_motor=Motor(pin13,pin14,pin0)
```

`button0.forward(`*speed: int*`)`

Команда вращает мотор в направлении вперед (при ошибке - поменяйте первые пины местами) **speed** - задает скорость вращения

`button0.backward(`*speed: int*`)`

Команда вращает мотор в направлении назад (при ошибке - поменяйте первые пины местами). **speed** - задает скорость вращения

`button0.stop()`

Команда останавливает вращения мотора

Пример программы

```
from microbit import pin0, pin1, pin13, pin14, pin15, pin16

from Motor import Motor

l_motor = Motor(pin13, pin14, pin0)
r_motor = Motor(pin15, pin16, pin1)

while True:
    l_motor.forward(500)
    r_motor.forward(500)
```

1.15.13 Датчик температуры 18B20

Скачать файл с классов

Датчик температуры позволяет определять температуру окружающего воздуха в диапазоне от -55 до 150 °C.

Класс

class B20_Temperature

Класс используется для определения объектов, имеющих поведение аналогового датчика температуры

Пример объявления объекта:

```
temp=Analog_Temperature(pin0)
```

temp.get_signal()

Команда позволяет получить аналоговый сигнал с датчика

temp.temperature()

Команда позволяет получить сигнал с датчика и переводит его в градусы

Пример программы

```
from microbit import *

from B20_Temperature import B20_Temperature

temp = B20_Temperature(pin0)

while True:

    display.scroll(temperature())
    sleep(1000)
    display.scroll(temp.temperature())
    sleep(1000)
```

1.15.14 Датчик испарения спиртовых паров

`Скачать файл с классов`

Датчик построен на базе полупроводникового газоанализатора MQ-3. На логический выход датчик выдаёт аналоговый сигнал, пропорциональный содержанию паров спирта в окружающей среде. Определяет концентрацию алкоголя в выдыхаемом воздухе или обнаружить утечку технического спирта.

Класс

`class Analog_Alcohol`

Класс используется для определения объектов, имеющих поведение датчика испарения спиртов

Пример объявления объекта:

```
alcohol=Analog_Alcohol(pin0)
```

`alcohol.calibrate()`

Команда позволяет провести каллибровку датчика. Прогревает датчик в течении 20 секунд. Далее Вы вращаете подстроечный резистор, пока не погаснет красный светодиод. Пробуете поднести датчик к испарению спирта - диод должен загореться. Убираете датчик от испарения - диод гаснет. Подтверждаете калибровку, удерживая **кнопку «A»**. Выводится показание датчика.

`alcohol.get_signal()`

Команда позволяет получить аналоговый сигнал с датчика

Пример программы

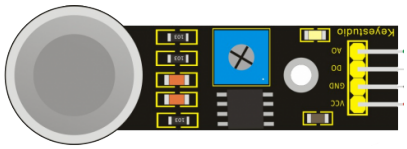
```
from microbit import display, pin0

from Analog_Alcohol import Analog_Alcohol

alcohol = Analog_Alcohol(pin0)

alcohol.calibrate()
while True:
    display.scroll(alcohol.get_signal())
```

1.15.15 Датчик углекислого газа

`Скачать файл с классов`

Сенсор построен на базе полупроводникового сенсора качества воздуха MQ-135. На логический выход датчик выдаёт аналоговый сигнал, пропорциональный концентрации углекислого газа. В газоанализатор встроен нагревательный элемент, который необходим для химической реакции: во время работы сенсор будет горячим — это нормально.

Перед первым использованием рекомендуем «прожечь» датчик. Подайте на него напряжение и оставьте в работающем состоянии на сутки. Эта процедура увеличит точность показаний сенсора.



Класс

`class Analog_Gas`

Класс используется для определения объектов, имеющих поведение датчика углекислого газа.

Пример объявления объекта:

```
gas=Analog_Gas(pin0)
```

`gas.calibrate()`

Команда позволяет провести каллибровку датчика. Прогревает датчик в течении 20 секунд. Далее Вы вращаете подстроечный резистор, пока не погаснет красный светодиод. Пробуете поднести датчик к углекислому газу (дышать на него) - диод должен загореться. Убираете датчик - диод гаснет. Подтверждаете калибровку, удерживая **кнопку «A»**. Выводится показание датчика.

`gas.get_signal()`

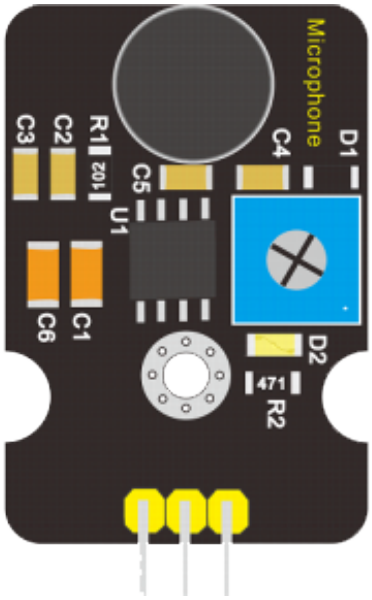
Команда позволяет получить аналоговый сигнал с датчика

Пример программы

```
from microbit import display, pin0

from Analog_Gas import Analog_Gas

gas = Analog_Gas(pin0)

gas.calibrate()
while True:
    display.scroll(gas.get_signal())
```

1.15.16 Аналоговый микрофон

Скачать файл с классов



Датчик звука используется для отслежения уровня шума или обнаружения громких сигналов: хлопков, стуков или свиста. Регулятором чувствительности можно выбирать, от какого звука будет срабатывать датчик - от слабого, громкого или очень громкого звука.

Библиотека помогает откалибровать звуковой датчик KY038 с помощью micro:bit. Он также обеспечивает функцию счетчика хлопков. Предполагается, что цифровой выход звукового датчика подключен к контакту 0 микробита. Он использует аналоговый вход, так как цифровой вход кажется не очень надежным для питания датчика 3В. Питание и земля датчика также подключены к micro:bit.

Класс

class Analog_Sound

Класс используется для определения объектов, имеющих поведение датчика звука

Пример объявления объекта:

```
mic=Analog_Sound(pin0)
```

mic.calibrate()

Калибровка звукового датчика. Стрелка показывает на вращение подстроечного резистора. Если на экране стрелка, то вращайте подстроечный резистор и делайте хлопки. Ряды красных светодиодов показывают приемлемый звуковой сигнал.

mic.count_claps(*self*, *sleep_time=100*)

Команда возвращает количество хлопков, которые датчик звука зафиксировал. Параметр **sleep_time** отражает паузу между хлопками. При медленных хлопках показатель необходимо увеличить.

mic.level_sound()

Команда возвращает аналоговое значение датчика

Пример программы

```
from microbit import display, pin0

from Analog_Sound import Analog_Sound

mic = Analog_Sound(pin0)
mic.calibrate()
while True:
    display.show(mic.count_claps())
```

1.15.17 Датчик температуры (аналоговый)

Скачать файл с классов

Датчик температуры служить прекрасным датчиком для робота, благодаря которому он сможет определять расстояния до объектов, объезжать препятствия, или строить карту помещения. Его можно также использовать в качестве датчика для сигнализации, срабатывающего при приближении объектов.

Класс

class Analog_Temperature

Класс используется для определения объектов, имеющих поведение аналогового датчика температуры

Пример объявления объекта:

```
temp=Analog_Temperature(pin0)
```

temp.get_signal()

Команда позволяет получить аналоговый сигнал с датчика

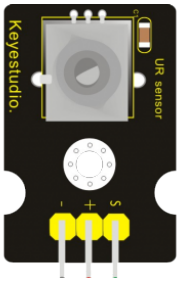
Пример программы

```
from microbit import display, pin0, sleep

from Analog_Temperature import Analog_Temperature

temp = Analog_Temperature(pin0)

display.scroll(str(temp.get_signal()))
sleep(1000)
```

1.15.18 Переменный резистор

Скачать файл с классов

Потенциометр используется для слежения угла поворота или для управления, путям изменения сопротивления на потенциометре. Потенциометр очень удобно использовать для ручного задания параметров вроде времени таймера, яркости или контрастности дисплея, скорости вращения двигателя, высоты звука.



Класс

class Analog_Rotation

Класс используется для определения объектов, имеющих поведение поворотного переменного резистора

Пример объявления объекта:

```
rotation=Analog_Rotation(pin0)
```

rotation.get_signal()

Команда позволяет получить аналоговый сигнал

rotation.get_angle(*max_signal: int*)

Команда позволяет получить значение угла поворота в диапазоне 180. **max_signal** - максимальное аналоговое значение сигнала переменного резистора

rotation.get_scale(*max_signal: int*, *scale: int*)

Команда позволяет получить значение угла поворота в диапазоне 180. **max_signal** - максимальное аналоговое значение сигнала переменного резистора **scale** - размер диапазона.

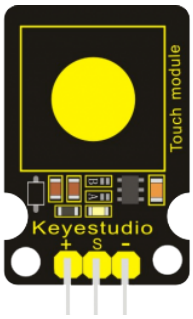
Пример программы

```
from microbit import display, pin0, sleep

from Analog_Rotation import Analog_Rotation

rotation = Analog_Rotation(pin0)

display.scroll(str(rotation.get_signal()))
sleep(1000)

display.scroll(str(rotation.get_angle(800)))
sleep(1000)

display.scroll(str(rotation.get_scale(1000)))
sleep(1000)
```

1.15.19 Емкостной датчик

Работает как обычная кнопка, но в нём нет подвижных частей. Сенсор работает через неметаллические материалы — пластмассу, картон, фанеру или стекло. Эту особенность можно использовать для создания скрытых или защищённых элементов управления.

Поместите модуль в герметичный корпус или спрячьте под лицевую панель устройства — кнопка почувствует приближение пальца.



Класс

Скачать файл с классов

class Capacitive_Touch

Класс используется для определения объектов, имеющих поведение емкостного датчика

Пример объявления объекта:

```
touch=Push_Button(pin0)
```

touch.is_pressed()

Команда возвращает **True**, если прилодижить палец к емкостному датчику

touch.click(*time: int*)

Команда возвращает **True**, если было прикосновение к емкостному датчику. **time** - указывает паузу между состояниями прикосновния и базовым

touch.count_pressed()

Команда возвращает количество прошедших ее опросов методом **is_pressed()**, при которых было прикосновение к емкостному датчику

touch.reset_pressed()

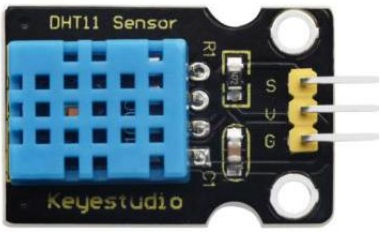
Команда сбрасывает счетчик прикосновений

Пример программы

```
from microbit import Image, display, pin0, sleep

from Capacitive_Touch import Capacitive_Touch

touch = Capacitive_Touch(pin0)

while True:
    if touch.is_pressed():
        display.show(Image.YES)
        break
    sleep(200)
```

1.15.20 Датчик температуры (DHT11)

Скачать файл с классов



Класс

class DHT11_Temperature

??()

Пример программы

1.15.21 Приемник ИК-сигнала

Скачать файл с классов

Класс

class Digital_IR_Receiver

??()

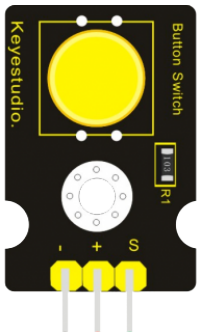
Пример программы

```
from microbit import *

while  True :
     display.scroll('Test')
     sleep ( 1000 )
```

1.15.22 Кнопка

Скачать файл с классов

Тактовая кнопка — простой, всем известный механизм, замыкающий цепь пока есть давление на толкатель



Класс

`class Push_Button`

Класс используется для определения объектов, имеющих поведение кнопки

Пример объявления объекта:

```
button0=Push_Button(pin0)
```

`button0.is_pressed()`

Команда возвращает **True**, если кнопка нажата

`button0.click(`*time: int*`)`

Команда возвращает **True**, если кнопка была нажата-отжата («клик») **time** - указывает паузу между состояниями нажата-отжата

`button0.count_pressed()`

Команда возвращает количество прошедших ее опросов методом **is_pressed()**, при которых кнопка была нажата

`button0.reset_pressed()`

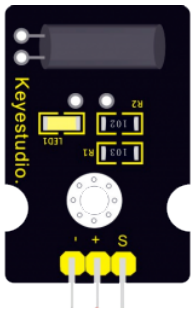
Команда сбрасывает счетчик нажатий

Пример программы

```
from microbit import display, pin0, sleep

from Push_Button import Push_Button

button0 = Push_Button(pin0)

while True:
    if button0.click(200):
        display.show(button0.count_pressed())
    if button0.count_pressed() > 3:
        button0.reset_pressed()
        break
    sleep(200)
display.show(button0.count_pressed())
```

1.15.23 Датчик наклона

`Скачать файл с классов`



Класс

```
class Digital_Tilt
```

Класс используется для определения объектов, имеющих поведение датчика вибрации.

Пример объявления объекта:

```
tilt= Digital_Tilt(pin0)
```

```
tilt.vibration()
```

Команда возвращает **True**, если произошла вибрация датчика.

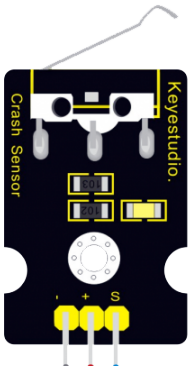
Пример программы

```
from microbit import Image, display, pin0

from Digital_Tilt import Digital_Tilt

tilt = Digital_Tilt(pin0)

while True:
    if tilt.vibration():
        display.show(Image.YES)
```

1.15.24 Кнопка (концевик)

```
Скачать файл с классов
```

Концевик — используется для фиксирования события нажатия рычажка. Помещается в бампер робота для фиксирования удара об препятствие. события закрывания или открывания дверей и окон.



Класс

class Crash_Sensor

Класс используется для определения объектов, имеющих поведение кнопки-концевик

Пример объявления объекта:

```
crash=Crash_Sensor(pin0)
```

crash.is_pressed()

Команда возвращает **True**, если кнопки-концевик нажата

crash.click(*time: int*)

Команда возвращает **True**, если кнопки-концевик была нажата-отжата («клик») **time** - указывает паузу между состояниями нажата-отжата

crash.count_pressed()

Команда возвращает количество прошедших ее опросов методом **is_pressed()**, при которых кнопки-концевик была нажата

crash.reset_pressed()

Команда сбрасывает счетчик нажатий

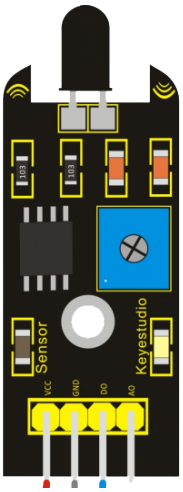
Пример программы

```
from microbit import display, pin0, sleep

from Crash_Sensor import Crash_Sensor

crash = Crash_Sensor(pin0)

while True:
    if crash.is_pressed():
        display.show(crash.count_pressed())
    if crash.count_pressed() > 3:
        crash.reset_pressed()
        break
    sleep(200)
display.show(crash.count_pressed())
```

1.15.25 Датчик огня

`Скачать файл с классов`

Датчик позволяет обнаружить открытое пламя. Он улавливает излучение в диапазоне 760 — 1100 нм, свойственное пламени. Датчик имеет подстроечный резистор, для отсеивания внешних факторов.



Класс

`class Flame_Sensor`

Класс используется для определения объектов, имеющих поведение оптического датчика огня.

Пример объявления объекта:

```
fire= Flame(pin0)
```

`fire.calibrate()`

Команда позволяет провести каллибровку датчика. Вы вращаете подстроечный резистор, пока не погаснет светодиод. Пробуете поднести датчик к огню, светодиод должен загореться. Подтвердите калибровку, удерживая **кнопку «A»**. Выводится показание датчика.

`fire.get_signal()`

Команда позволяет получить аналоговый сигнал с датчика

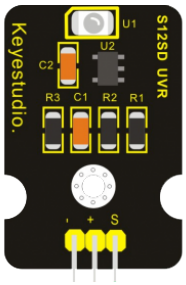
Пример программы

```
from microbit import display, pin0

from Flame import Flame

fire = Flame(pin0)

fire.calibrate()
while True:
    display.scroll(fire.get_signal())
```

1.15.26 Датчик ультрафиолета

`Скачать файл с классов`

Датчик можно использовать для контроля интенсивности ультрафиолетового света или использовать в качестве УФ-детектора пламени.



Класс

`GUVA-Ultraviolet`

`??()`

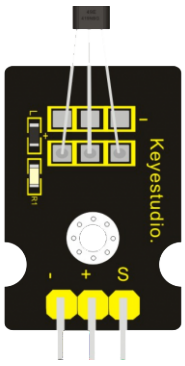
Пример программы

1.15.27 Датчик Холла (магнитное поле)

`Скачать файл с классов`

Датчиком Холла измеряет величину магнитного поля используя эффект Холла.

Если вдоль датчика пропустить электрический ток, а перпендикулярно плоскости пластинки создать магнитное
поле, то на боковых плоскостях пластинки D возникнет электрическое поле, которое называют полем Холла. Магнитное поле позволяет фиксировать разницу потенциалов



Класс

class Hall_Magnetic

Класс используется для определения объектов, имеющих поведение магнитного датчика

Пример объявления объекта:

```
magnetic=Hall_Magnetic(pin0)
```

magnetic.get_signal()

Команда позволяет получить аналоговый сигнал с датчика

magnetic.get_status(*base_status: int*, *span=20*)

Команда возвращает **True**, если сигнал не вышел из диапазона (-span **base_status** +span)

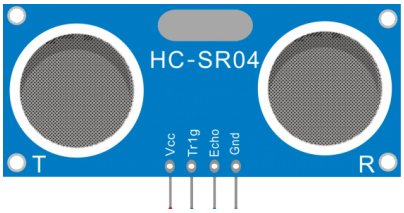
Пример программы

```
from microbit import Image, display, pin0, sleep

from Hall_Magnetic import Hall_Magnetic

magnetic = Hall_Magnetic(pin0)

while True:
    display.scroll(str(magnetic.get_signal()))
    sleep(1000)

    if magnetic.get_status(16, 10):
        display.show(Image.YES)
    else:
        display.show(Image.NO)
    sleep(1000)
```

1.15.28 Датчик Ультразвука

Скачать файл с классов

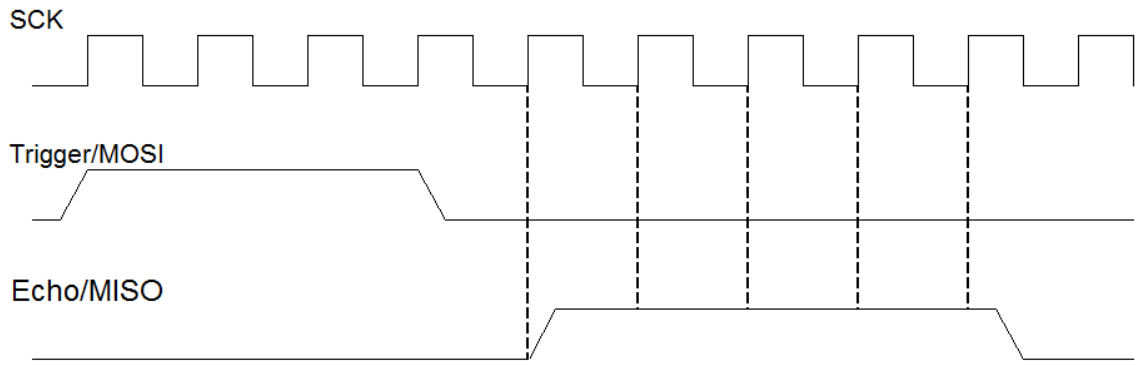
Ультразвуковой дальномер HC-SR04 – это помещенные на одну плату приемник и передатчик ультразвукового сигнала. Излучатель генерирует сигнал, который, отразившись от препятствия, на приемник. Измерив время, за которое сигнал проходит до объекта и обратно, можно оценить расстояние. Кроме самих приемника и передатчика, на плате находится еще и необходимая обвязка, чтобы сделать работу с этим датчиком простой и удобной.



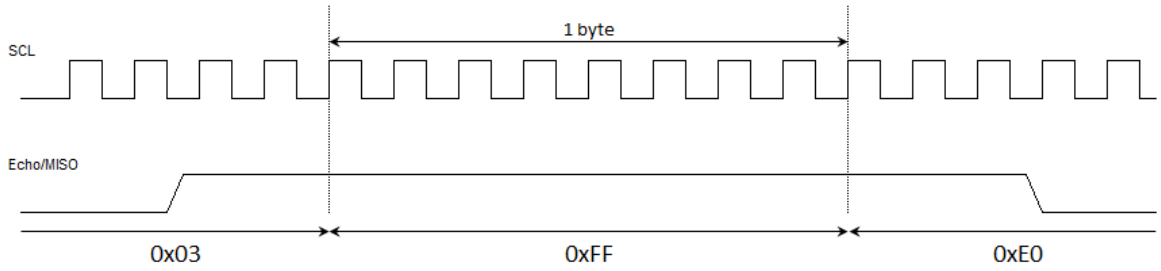
Класс позволяет считывать расстояние от ультразвукового датчика HCSR04 или аналогичного.

Он использует внутреннее аппаратное устройство SPI для измерения длины возвращаемого эха, поэтому по умолчанию вы должны подключить контакт `echo` сонара к `pin14` micro:bit, а контакт `Trig` сонара к `pin15` micro:bit.

Датчик имеет один излучатель и приемник. Частота импульсов датчика около 10 микросекунд. Когда он получает сигнал от micro:bit, он посылает ультразвуковой сигнал через излучатель. Приемник улавливает отражение звука. Ширина импульса «эха» эквивалентна времени, которое требуется звуку, чтобы достичь объекта и вернуться. Это в два раза больше, чем расстояние до объекта.



Библиотека использует внутреннее аппаратное устройство spi для измерения «эха». SPI работает, используя один контакт для установки тактовой частоты с импульсами на требуемой частоте. Другой контакт используется для передачи битов в каждом тактовом цикле, а последний контакт используется для приема битов от другого устройства. Библиотека отправляет импульс через MOSI и ожидает получения чего-либо через MISO. Затем он измеряет длину возвращаемого импульса, считая эквивалентные «биты»



Класс

`class Ultrasonic`

Класс используется для определения объектов, имеющих поведение ультразвукового датчика

```
sonar = Ultrasonic( trigger, echo, speed_sound)
```

Параметр **speed_sound** - скорость распространения звука при комнотной температуре. По умолчанию показатель равен 343 000, Вы можете его корректировать:

`distance_mm()`

Возвращает расстояние до объекта в миллиметрах

`exception distance_cm`

Возвращает расстояние до объекта в сантиметрах

Пример программы

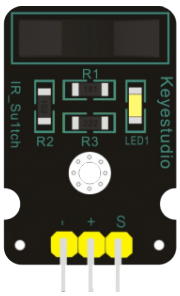
```
from microbit import display, pin1, pin2, sleep

from Ultrasonic import Ultrasonic

sonar = Ultrasonic(pin2, pin1)

while True:
    display.scroll(int(sonar.distance_mm()))
    sleep(1000)
```

1.15.29 Датчик Фотопрерыватель

Скачать файл с классов

Фотопрерыватели широко используются во многих областях, таких как измерение скорости, позиционирование и подсчет.

Небольшие бытовые приборы, оптические концевые выключатели, обнаружение объектов и так далее.



Класс

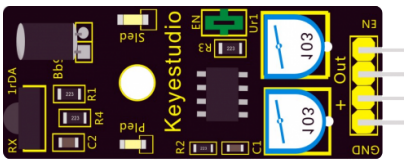
class Photo_Interrupter

??()

Пример программы

1.15.30 Датчик Инфракрасный датчик препятствий

Скачать файл с классов



Класс

```
class Infrared_Obstacle_Detector
```

```
??()
```

Пример программы

1.15.31 ИК - пульт

```
Скачать файл с классов
```

Класс

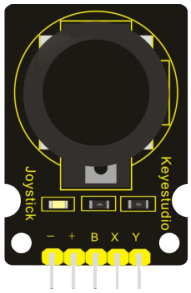
```
class IR_Remote
```

```
??()
```

Пример программы

1.15.32 Джойстик

```
Скачать файл с классов
```



Аналоговый джойстик чувствителен к тому, насколько далеко Вы перемещаете стик в любом конкретном направлении. Направление влево/вправо называют горизонтальной осью (осью X). Направление вверх/вниз называют вертикальной осью (осью Y).

При нажатии на стик происходит события нажатия на кнопку, сигнал можно снять с цифрового контакта **SW**.

Класс

```
class Joystick
```

Класс используется для определения объектов, имеющих поведение джойстика.

Пример объявления объекта:

```
joys=Joystick(pin0, pin1, pin2)
```

`joys.get_x()`
: Команда возвращает показание по оси X.

`joys.get_y()`
: Команда возвращает показание по оси Y.

`joys.click()`
: Команда возвращает сигнал нажатия на кнопку.

`joys.play_cross()`
: Команда возвращает сигналы **UP**, **Down**, **Left**, **Right**, **0**. Принцип работы цифрового джойстика.

`joys.play_diagonal()`
: Команда возвращает сигналы **Up**, **Down**, **Left**, **Right**, **0**. Диаганальные сигналы **Up-Left**, **Up-Right**, **Down-Left**, **Down-Right**. Принцип работы цифрового джойстика.

Пример программы

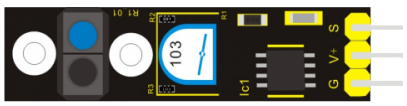
```
from microbit import pin0, sleep
from neopixel import NeoPixel

np = NeoPixel(pin0, 5)
while True:
    for i in range(5):
        np[i] = (255, 0, 0)
        np.show()
        sleep(1000)
        np.clear()
```

1.15.33 Модуль отслеживания линии

`Скачать файл с классов`

Датчик отслеживания линии позволяет определять цвет поверхности. Единица — чёрный или пустота, ноль — не чёрный.

Переменный резистор, установленный на сенсоре, позволит регулировать чувствительность сенсора в широких пределах. Это позволяет откалибровать датчик под материал покрытия и внешнее освещение.



Класс

`class Line_Tracking`

Класс используется для определения объектов, имеющих поведение цифрового дачтика линии.

Пример объявления объекта:

```
line=Line_Tracking(pin0)
```

`line.calibrate()`

Команда позволяет провести каллибровку датчика. Вы вращаете подстроечный резистор, пока не погаснет светодиод. Пробуете поднести датчик к черному и белому полю попеременно. Светодиод должен гаснуть и загораться Подтвердите калибровку, удерживая **кнопку «A»**. Выводится показание датчика.

`line.get_signal()`

Команда позволяет получить аналоговый сигнал с датчика

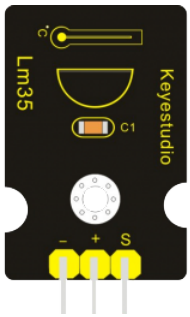
Пример программы

```
from microbit import display, pin0

from Line_Tracking import Line_Tracking

line = Line_Tracking(pin0)

line.calibrate()
while True:
    display.scroll(line.get_signal())
```

1.15.34 Датчик температуры LM35

`Скачать файл с классов`

Температурный датчик LM35 - представляет собой интегрированный сенсор, который может использоваться для измерения температуры с помощью электрического выходного сигнала, пропорционального температуре. Диапазон измерения от 0 до +100°C



Класс

class LM35_Linear

??()

Пример программы

1.15.35 Фоторезистор

Скачать файл с классов

Фоторезистор — компонент, меняющий сопротивление в зависимости от количества света падающего на него. В полной темноте он имеет максимальное сопротивление, а по мере роста освещённости сопротивление уменьшается.

Класс

class Photocell

Класс используется для определения объектов, имеющих поведение датчика фоторезистор.

Пример объявления объекта:

```
photo= Photocell(pin0)
```

photo.get_signal()

Команда позволяет получить аналоговый сигнал с датчика

Пример программы

```
from microbit import display, pin0

from Photocell import Photocell

photo = Photocell(pin0)

while True:
    display.scroll(photo.get_signal())
```

1.15.36 Датчик движения

`Скачать файл с классов`

ИК-датчик фиксирует движение людей и животных. Подобные датчики используются на раздвижных дверях, открывающихся автоматически при приближении человека.

Выводом с сенсора является бинарный цифровой сигнал: - движения нет логический ноль. - движение, сигнальный контакт устанавливается в логическую единицу

Класс

```
class PIR_Motion
```

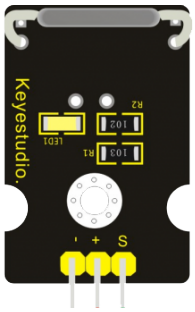
```
??()
```

Пример программы

1.15.37 Датчик геркона

`Скачать файл с классов`

Геркон — датчик срабатывает (замыкает) при воздействии на него магнитного поля, с достаточной силой. Любые механические контакты подвержены износу, чтобы уменьшить это влияние используют не механические датчики.



Класс

```
class Reed_Switch
```

Класс используется для определения объектов, имеющих поведение геркона

Пример объявления объекта:

```
switch=Push_Button(pin0)
```

`switch.is_pressed()`

Команда возвращает **True**, если есть магнитное поле рядом с герконом

`switch.count_pressed()`

Команда возвращает количество прошедших ее опросов методом **is_pressed()**, при которых геркон был замкнут

`switch.reset_pressed()`

Команда сбрасывает счетчик замыканий геркона или количество положительных результатов при вызоме метода **is_pressed()**

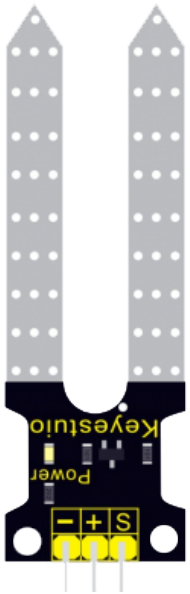
Пример программы

```
from microbit import Image, display, pin0, sleep

from Reed_Switch import Reed_Switch

switch = Reed_Switch(pin0)

while True:
    if switch.is_pressed():
        display.show(Image.YES)
        sleep(1000)
```

1.15.38 Датчик влажности

`Скачать файл с классов`

Принцип работы датчика - если в почве не хватает воды, аналоговое значение, выводимое датчиком, уменьшится, в противном случае оно увеличится.



Класс

`class Soil_Humidity`

Класс используется для определения объектов, имеющих поведение датчика влажности почвы.

Пример объявления объекта:

```
soil= Soil_Humidity(pin0)
```

`soil.get_signal()`

Команда позволяет получить аналоговый сигнал с датчика

Пример программы

```
from microbit import display, pin0

from Soil_Humidity import Soil_Humidity

soil = Soil_Humidity(pin0)

while True:
    display.scroll(soil.get_signal())
```

1.15.39 Датчик пара

Скачать файл с классов

Датчик пара представляет собой аналоговый датчик, который может использоваться как простой датчик дождевой воды или датчик уровня жидкости. Когда влажность на чувствительной области этого датчика повышается, выходное напряжение его сигнального конца увеличивается.

Класс

class Steam

Класс используется для определения объектов, имеющих поведение датчика пара.

Пример объявления объекта:

```
steam_1= Steam(pin0)
```

steam_1.get_signal()

Команда позволяет получить аналоговый сигнал с датчика

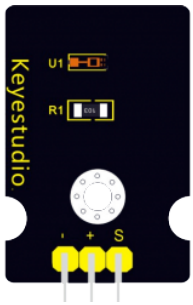
Пример программы

```
from microbit import display, pin0

from Steam import Steam

steam_1 = Steam(pin0)

while True:
    display.scroll(steam_1.get_signal())
```

1.15.40 Датчик окружающего света

Скачать файл с классов

Датчик чувствителен ко всему световому диапазону видимого света. Если уровень света очень низок, то уровень напряжения на выходе SIG будет также очень мал. При увеличении освещенности на выходе также будет увеличиваться напряжение.



Класс

```
class ATEMT6000_AmbientLight
```

Класс используется для определения объектов, имеющих поведение светочуствительного датчика.

Пример объявления объекта:

```
temp=TEMT(pin0)
```

```
photo.get_signal()
```

Команда позволяет получить аналоговый сигнал с датчика

Пример программы

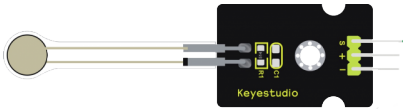
```
from microbit import display, pin0, sleep

from TEMT import TEMT

temp = TEMT(pin0)

while True:
    display.scroll(temp.get_signal())
    sleep(1000)
```

1.15.41 Датчик давления

Скачать файл с классов

Датчик использует гибкий чувствительный к давлению материал с ультратонкой пленкой. Он водонепроницаем и чувствителен к давлению.

Когда датчик определяет внешнее давление, сопротивление датчика изменится. Затем через схему можно преобразовать изменения сопротивления в изменения напряжения, вывести его на конце сигнала. Таким образом, мы можем получить условия изменения давления, обнаружив изменения сигнала.



Класс

```
class Thin_film_Pressure
```

Класс используется для определения объектов, имеющих поведение датчика давления.

Пример объявления объекта:

```
pressure= Thin_film_Pressure(pin0)
```

```
pressure.get_signal()
```

Команда позволяет получить аналоговый сигнал с датчика

Пример программы

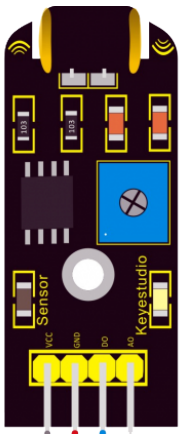
```
from microbit import display, pin0

from Thin_film_Pressure import Thin_film_Pressure

pressure = Thin_film_Pressure(pin0)

while True:
    display.scroll(pressure.get_signal())
```

1.15.42 Датчик вибрации

```
Скачать файл с классов
```

Датчик основан на чувствительном элементе 801S.

Внутренняя конструкция представляет собой металлический шар, закрепленный на специальной пружине в качестве шеста, а другой, окруженный им, как другой полюс. Как только вибрация достигает определенной амплитуды, два полюса соединяются.



Класс

```
class Vibration
```

```
??()
```

Пример программы

1.15.43 Датчик протечки

```
Скачать файл с классов
```

Датчик может измерять объем капли воды или количество воды по множеству следов открытых параллельных линий и уменьшения сопротивляемости.

Класс

```
class Water
```

Класс используется для определения объектов, имеющих поведение датчика протечки воды.

Пример объявления объекта:

```
wat= Water(pin0)
```

```
wat.get_signal()
```

Команда позволяет получить аналоговый сигнал с датчика

Пример программы

```
from microbit import display, pin0

from Water import Water

wat = Water(pin0)

while True:
    display.scroll(wat.get_signal())
```

Документация на Английском языке https://microbit-micropython.readthedocs.io/en/v2-docs/index.html

Содержание модулей Python

Алфавитный указатель

Символы

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W