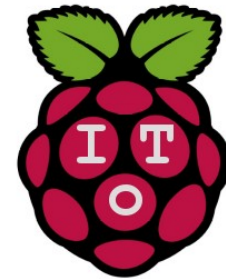




Internet of Things



Язык программирования **Ruby** в проектах IoT

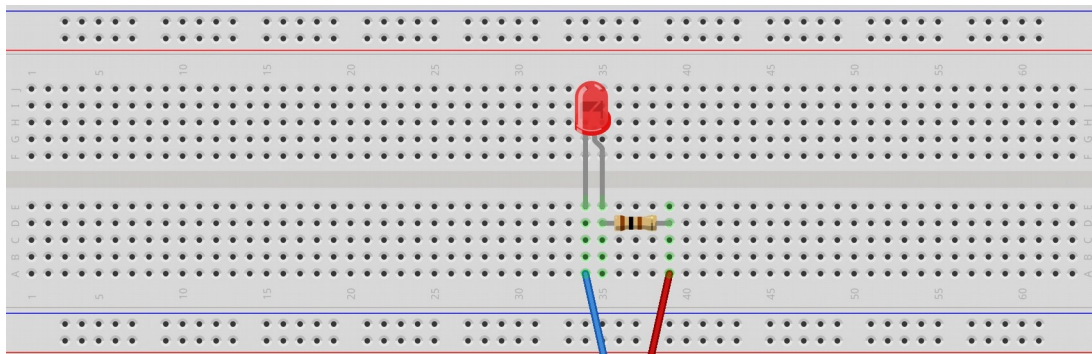
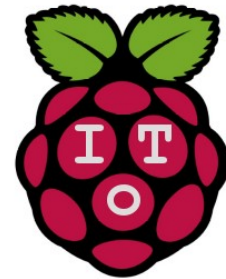
Шадринск
2018-2019

М. В. Шохирев

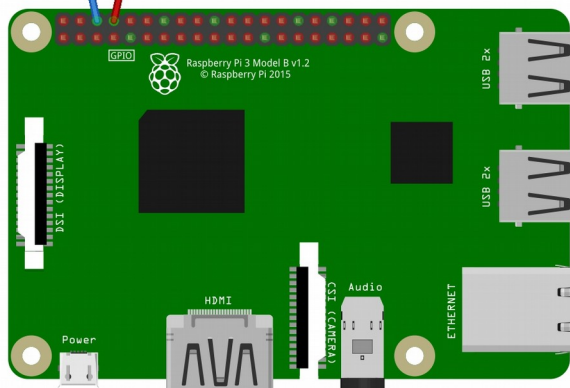


Светодиод (LED) :

ПОДКЛЮЧЕНИЕ



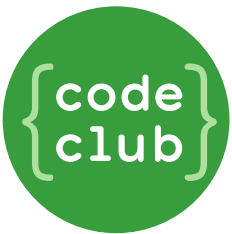
Подключать светодиод через резистор (сопротивление) на 330 Ом нужно, чтобы уменьшить силу тока, проходящего через LED, для соблюдения ограничений Raspberry Pi на максимальный ток с каждого контакта GPIO.



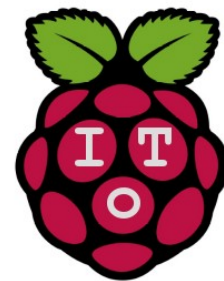
fritzing

Короткую ножку светодиода (LED) подключить (обычно синим или чёрным проводом) к контакту [6] «земля» (GND) на GPIO у Raspberry Pi.

Длинную ножку светодиода подключить (любым цветным проводом) к резистору на 330 Ом, который подключить к соседнему физическому контакту [8] на GPIO у Raspberry Pi, имеющему логический номер BCM (25).



LED : запуск программы на языке Ruby



Теперь мы знаем, что можно управлять физическими объектами, считывая данные (показания датчиков) из определённых системных файлов и записывая в них значения (посылая сигналы включения и выключения).

Но записывать эти действия в командных файлах на **bash** громоздко и неудобно: из-за его ограниченных возможностей и необходимости подробно описывать все действия.

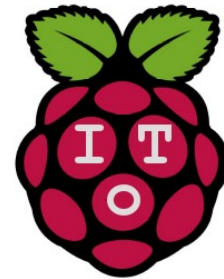
Поэтому программы управления обычно пишут на языках высокого уровня, где есть готовые библиотеки для взаимодействия с различными физическими устройствами. Мы научимся писать программы на языке **Ruby**.

Для примера можно в терминальном окне запустить программу управления светодиодом на языке **Ruby**:

```
cd ~/projects  
./test_led.rb
```



LED: управление из программы на Ruby



```
#!/usr/bin/ruby # указать путь к интерпретатору ruby

require "led" # подключить библиотеку led.rb

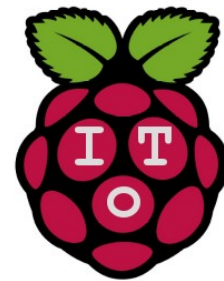
led = LED.new(14) # создать объект led класса Led (Светодиод)
5.times do # 5 раз повторить команды от do до end
  led.on() # скомандовать объекту led включиться (on)
  sleep 1 # приостановить выполнение на 1 секунду
  led.off() # скомандовать объекту led выключиться (off)
  sleep 1 # приостановить выполнение на 1 секунду
end

led.blink(3) # скомандовать объекту led помигать (blink) 3 раза

# Запуск в терминальном окне: ./test_led.rb
```



Среда разработки Geany



The screenshot shows a Raspberry Pi desktop environment. On the left, the application menu is open, displaying various categories and applications. The 'Geany' application is highlighted in the 'Программирование' (Programming) category. A red arrow points from the 'Geany' icon in the menu to the 'Geany' application window. The application window is titled 'без имени - Geany' and shows the 'Файл' (File) menu open. The menu options include 'Создать' (Create), 'Создать из шаблона' (Create from template), 'Открыть...' (Open...), 'Открыть выбранный файл' (Open selected file), 'Недавние файлы' (Recent files), 'Сохранить' (Save), 'Сохранить как...' (Save as...), 'Сохранить все' (Save all), 'Загрузить заново' (Reload), 'Загрузить заново как' (Reload as), 'Свойства' (Properties), 'Параметры страницы' (Page properties), 'Печать...' (Print...), 'Закрыть' (Close), 'Закрыть остальные' (Close others), 'Закрыть все' (Close all), and 'Выход' (Exit). A text box with the description 'Быстрая и легковесная среда разработки, использующая GTK2' (Fast and lightweight development environment using GTK2) is positioned near the application menu. The desktop background is a landscape image of a road stretching into the distance. The system tray at the top right shows the time as 12:40 and the battery level at 55%.

Программирование

- Arduino IDE
- BlueJ Java IDE
- Geany
- Greenfoot Java IDE
- Mathematica
- Node-RED
- Python 3 (IDLE)
- Scratch
- Scratch 2
- Sense HAT Emulator
- Sonic Pi
- Thonny Python IDE
- Wolfram

Образовательные

Офис

Интернет

Аудио и видео

Графика

Игры

Системные

Стандартные

Электроника

Help

Параметры

Run...

Shutdown...

Быстрая и легковесная среда разработки, использующая GTK2

Файл Правка Поиск Вид Документ Проект Сборка Инструменты Справка

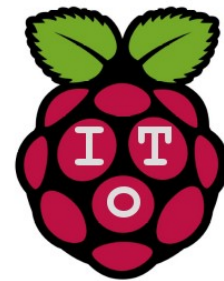
- Создать Ctrl+N
- Создать из шаблона
- Открыть... Ctrl+O
- Открыть выбранный файл Shift+Ctrl+O
- Недавние файлы
- Сохранить Ctrl+S
- Сохранить как...
- Сохранить все Shift+Ctrl+S
- Загрузить заново Ctrl+R
- Загрузить заново как
- Свойства
- Параметры страницы
- Печать... Ctrl+P
- Закрыть Ctrl+W
- Закрыть остальные
- Закрыть все Shift+Ctrl+W
- Выход Ctrl+Q

без имени - Geany

Это Geany 1.29.



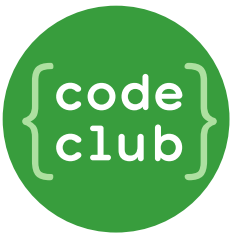
Текст программы на **Ruby** в **Geany**



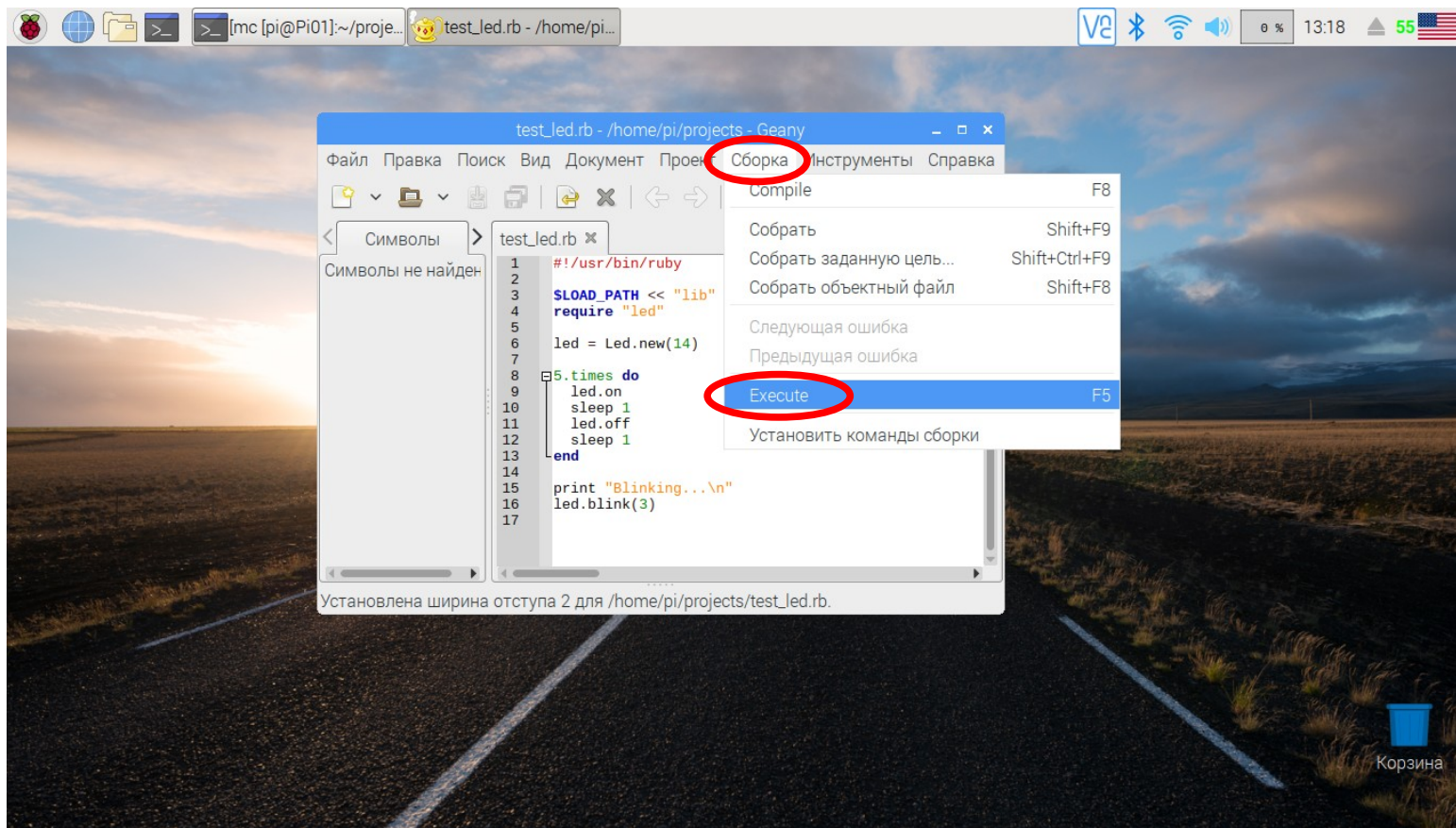
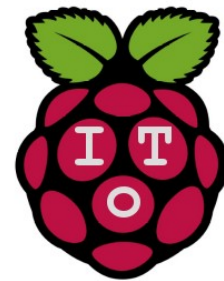
The screenshot shows a Raspberry Pi desktop environment. On the left is the application menu with categories like 'Программирование' (Programming), 'Образовательные' (Educational), 'Офис' (Office), 'Интернет' (Internet), 'Аудио и видео' (Audio and video), 'Графика' (Graphics), 'Игры' (Games), 'Системные' (System), 'Стандартные' (Standard), 'Электроника' (Electronics), 'Help', 'Параметры' (Parameters), 'Run...', and 'Shutdown...'. The 'Программирование' category is expanded, showing various IDEs: Arduino IDE, BlueJ Java IDE, Geany, Greenfoot Java IDE, Mathematica, Node-RED, Python 3 (IDLE), Scratch, Scratch 2, Sense HAT Emulator, Sonic Pi, Thonny Python IDE, and Wolfram. A red arrow points from the 'Geany' icon to the Geany IDE window. The Geany window is titled 'test_led.rb - /home/pi/projects - Geany' and contains a Ruby script for blinking an LED. The script is as follows:

```
1  #!/usr/bin/ruby
2
3  $LOAD_PATH << "lib"
4  require "led"
5
6  led = Led.new(14)
7
8  5.times do
9    led.on
10   sleep 1
11   led.off
12   sleep 1
13 end
14
15 print "Blinking...\n"
16 led.blink(3)
17
```

Below the code editor, a status bar message reads: 'Установлена ширина отступа 2 для /home/pi/projects/test_led.rb.'

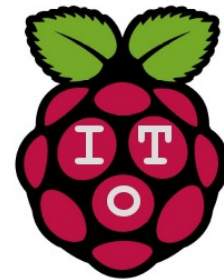


Запуск программы на **Ruby** из **Geany**





Выполнение программы на **Ruby** из **Geany**



The screenshot shows a Raspberry Pi desktop environment. The Geany IDE is open, editing a file named `test_led.rb` located at `/home/pi/projects`. The code in the editor is as follows:

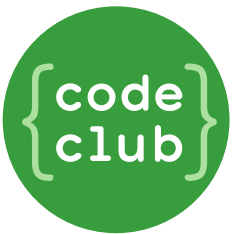
```
1 #!/usr/bin/ruby
2
3 $LOAD_PATH << "lib"
4 require "led"
5
6 led = Led.new(14)
7
8 5.times do
9   led.on
10  sleep 1
11  led.off
12  sleep 1
13 end
14
15 print "Blinking...\n"
16 led.blink(3)
17
```

A red circle highlights the run button (a green play icon) in the Geany toolbar. A callout points to it with the text: "Кнопка запуска программы на выполнение".

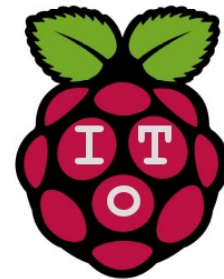
Below the editor, a terminal window shows the output of the program:

```
Бlinking...
(program exited with code: 0)
Press return to continue
```

A callout points to the terminal output with the text: "Вывод команды **print** показывается в терминальном окне".



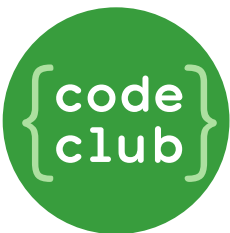
Классы и объекты: модель реальности



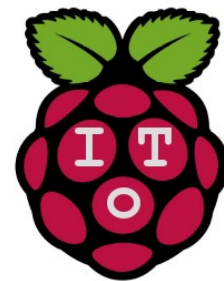
Ruby – объектно-ориентированный язык, который позволяет в программах моделировать объекты реального мира с их характерными свойствами и уникальным состоянием, а также взаимодействие между ними.

При разработке программы создаётся программная модель:

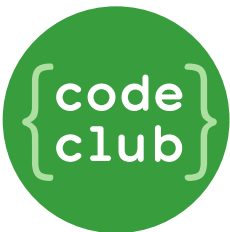
- предметы и понятия реальности описываются как **классы**, в которых
- описываются **объекты** внешнего мира (светодиоды, кнопки, датчики, реле, моторы),
- их **свойства** (название, цвет, контакты, показания, состояние) и
- возможные **действия** с ними (включить, выключить, считать показания, изменить состояние, увеличить скорость).



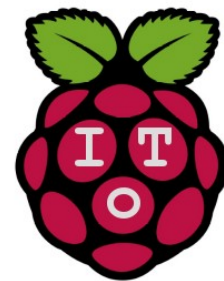
Классы и объекты: термины



Реалии	Объяснение	Термины ООП	Описание на Ruby
Несколько однотипных светодиодов (LED) разных цветов 	Похожие объекты с одинаковыми характеристиками	Класс «Светодиод»	<pre>class LED # описания end</pre>
Цвет, № контакта, рабочее напряжение, потребляемый ток, ...	Характеристики (свойства) этих объектов	Атрибуты в описании класса	<pre>attr_reader :pin attr_reader :color # другие атрибуты</pre>
Светодиоды можно включать и выключать	Их поведение заключается в переходах от состояния «включено» к «выключено» и наоборот	Объекты класса имеют <i>методы</i> on() и off()	<pre>def on() # включить end def off() # выключить end</pre>
Красный светодиод на пине 23 сначала включить, а потом выключить	Перевести светодиод в соответствующее состояние	Изменить <i>состояние</i> с помощью определённого метода	<pre>r = LED.new(23, :red) r.on() # включить r.off() # выключить</pre>



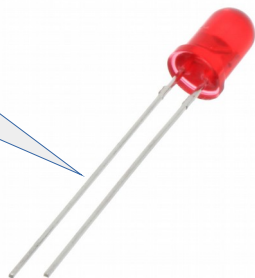
Классы и объекты: моделирование



Во время выполнения программы

- создаются объекты классов методом `new`: `led = LED.new(18, :red)`
- над объектами выполняются действия с помощью *МЕТОДОВ*: `led.on()`
- и у них изменяется *СОСТОЯНИЕ* (то есть значения свойств): `@state = :on`
- которое можно выяснить: `print "включен" if led.is_on?`

Физический
объект
в реальном мире

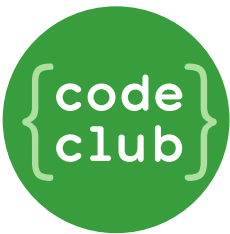


LED

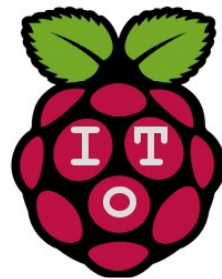
```
@pin = 18  
@color = :red  
@state = :on
```

```
on()  
off()  
is_on?()  
is_off?()
```

Информационный
объект
в программной
модели



Классы и объекты: пример



```
class LED
  attr_reader :pin
  attr_reader :color
  attr_reader :state

  def on
    @state = :on
  end
  def is_on?
    return @state==:on
  end
end

red = LED.new(18)
red.on
print(red.is_on?)

blue = LED.new(22)
print(blue.is_on?)
```

класс объектов: светодиоды
атрибут - свойство объекта @pin: № контакта
атрибут - свойство объекта @color: цвет
атрибут - свойство объекта @state: состояние

метод - действие с объектом: включить
установить значение свойства объекта

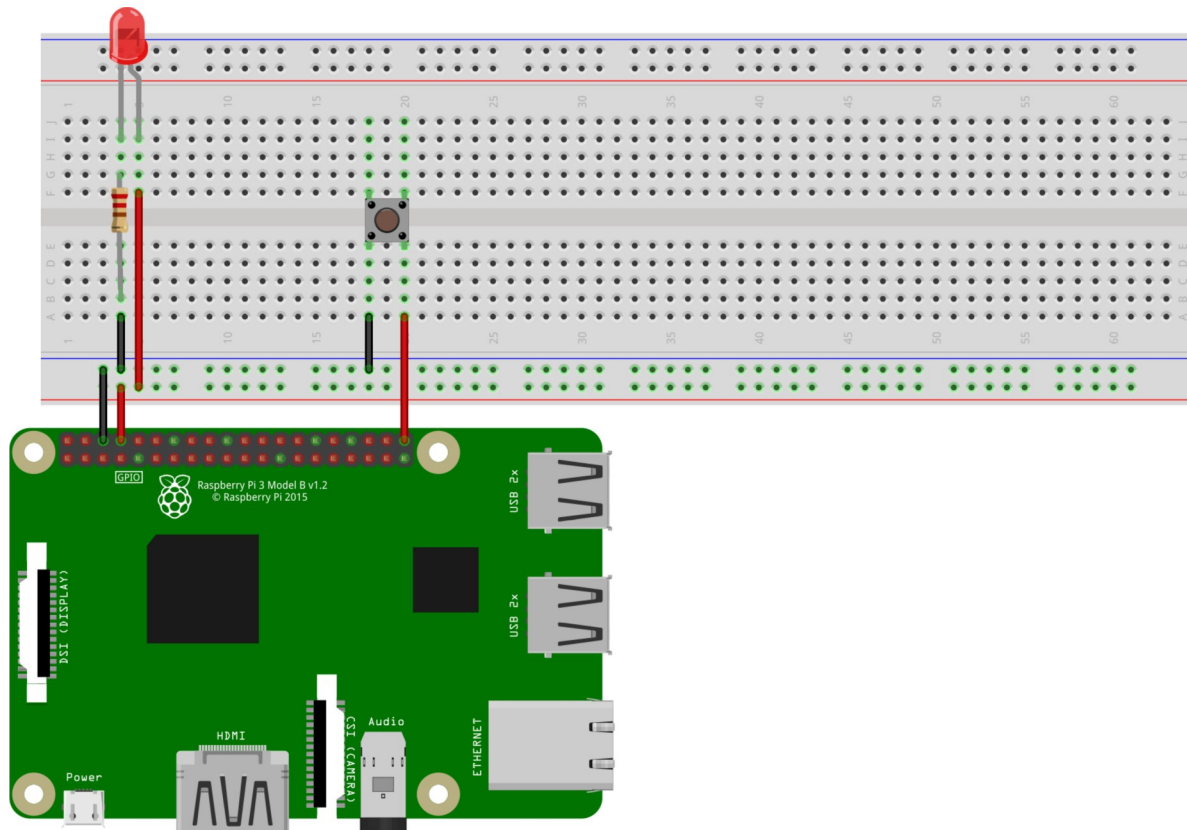
действие с объектом: проверить состояние
вернуть значение свойства @state объекта

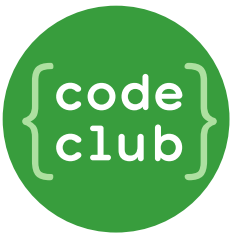
создать **объект** red класса Led с параметром 17
скомандовать объекту red: «Включись!»
спросить у объекта его состояние методом is_on?

создать ещё объект класса Led - с именем blue
спросить у объекта его состояние

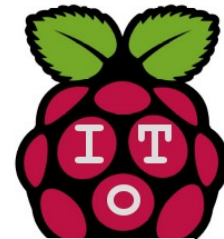


Кнопка : пример рецептора





Кнопка: управление на языке Ruby



```
#!/usr/bin/ruby  
require "button"
```

```
button = Button.new(21)
```

```
button.wait_for_press  
if button.long_press?  
  print "Длинное!\n"  
else  
  print "Короткое.\n"  
end
```

```
button.wait_for_presses(2)  
if button.double_press?  
  print "Двойное...\n"  
else  
  print "2 одинарных\n"  
end
```

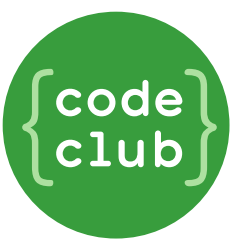
```
# подключить библиотеку button.rb
```

```
# создать объект класса Button
```

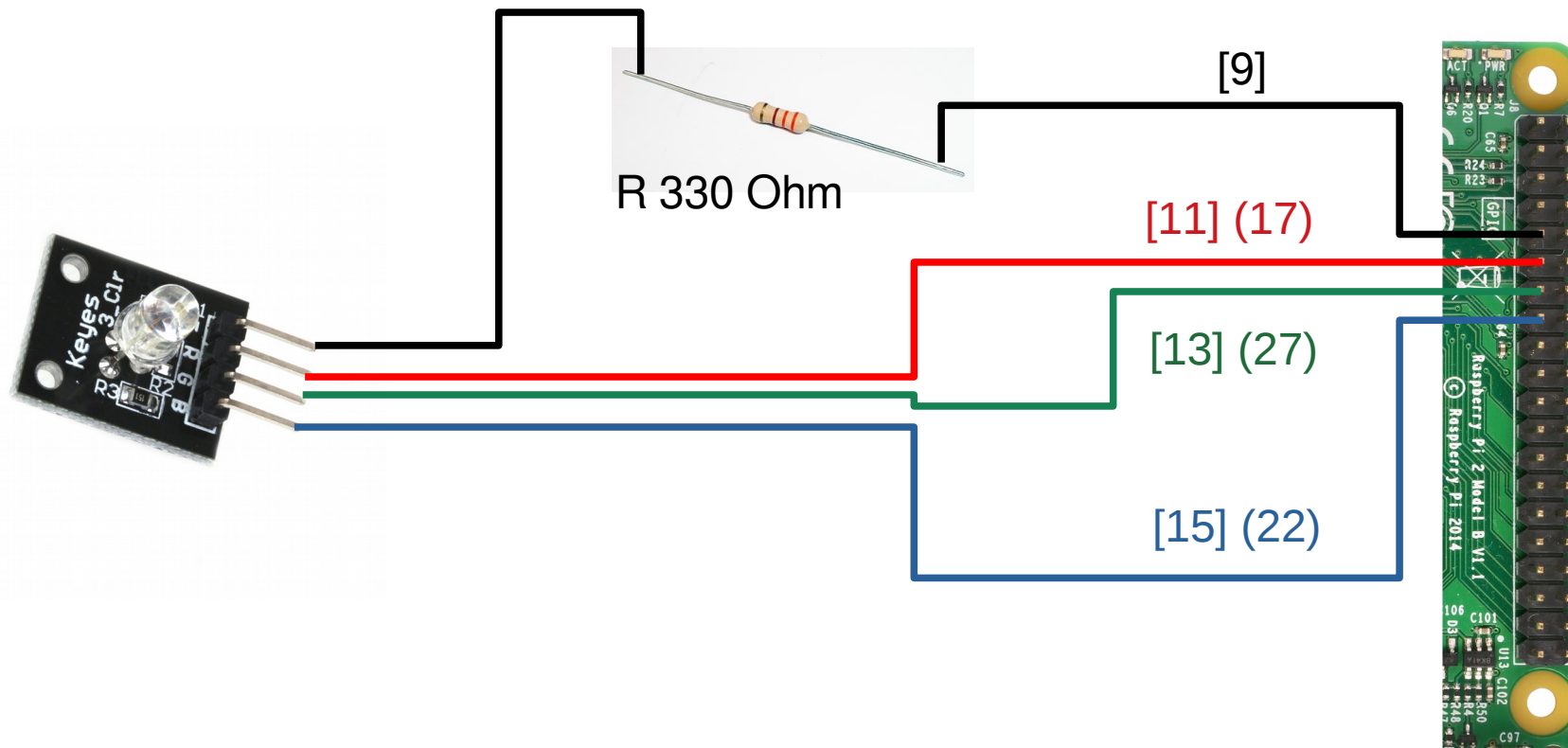
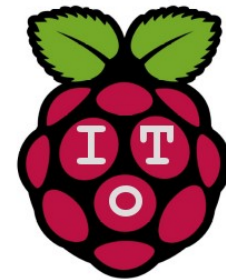
```
# ожидать нажатия на кнопку  
# было длинное нажатие?
```

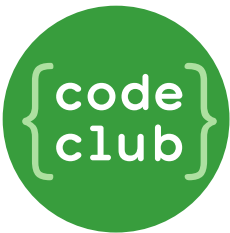
```
# ожидать 2 нажатия на кнопку  
# было двойное нажатие?
```

```
# было одиночное нажатие
```

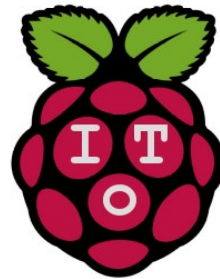


Светодиод **RGB** : подключение





RGB: управление из программы на Ruby



```
#!/usr/bin/ruby
```

```
require "led_rgb"      # подключить библиотеку lib/led.rb
```

```
rgb = LED::RGB.new(17, 27, 22) # объект rgb класса LedRgb
3.times do                    # 3 раза повторить команды от do до end
  [:red, :green, :blue, :yellow, :aqua].each do |color|
    rgb.on(color)             # скомандовать rgb включиться очередным цветом
    sleep 1                   # приостановить выполнение на 1 секунду
    rgb.off                   # скомандовать rgb выключиться
    sleep 1                   # приостановить выполнение на 1 секунду
  end
end
```

```
rgb.blink(:red, 5)        # скомандовать rgb помигать красным 5 раз
```




На каком языке
легче программировать?



*Что вам не понятно
из изученного материала?*