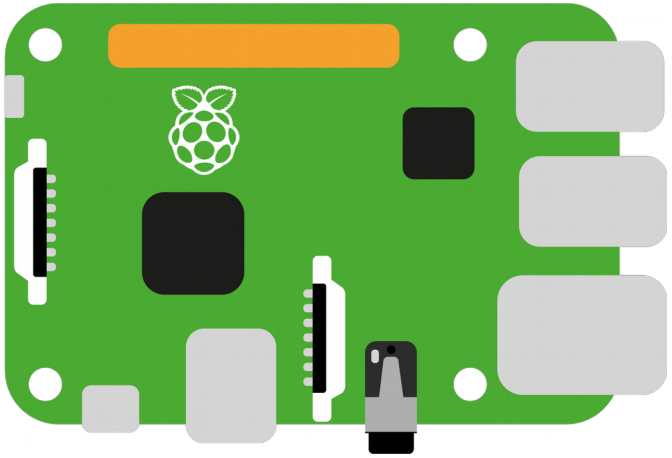


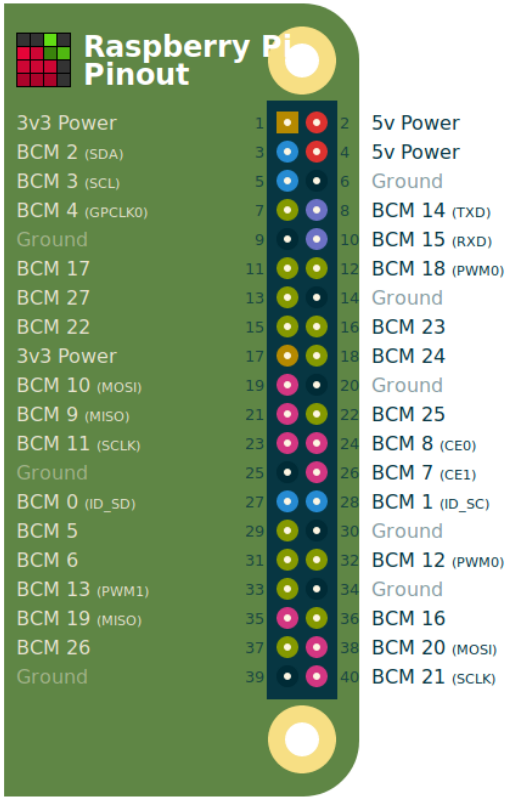
Programming paradigms for physical computing and IoT

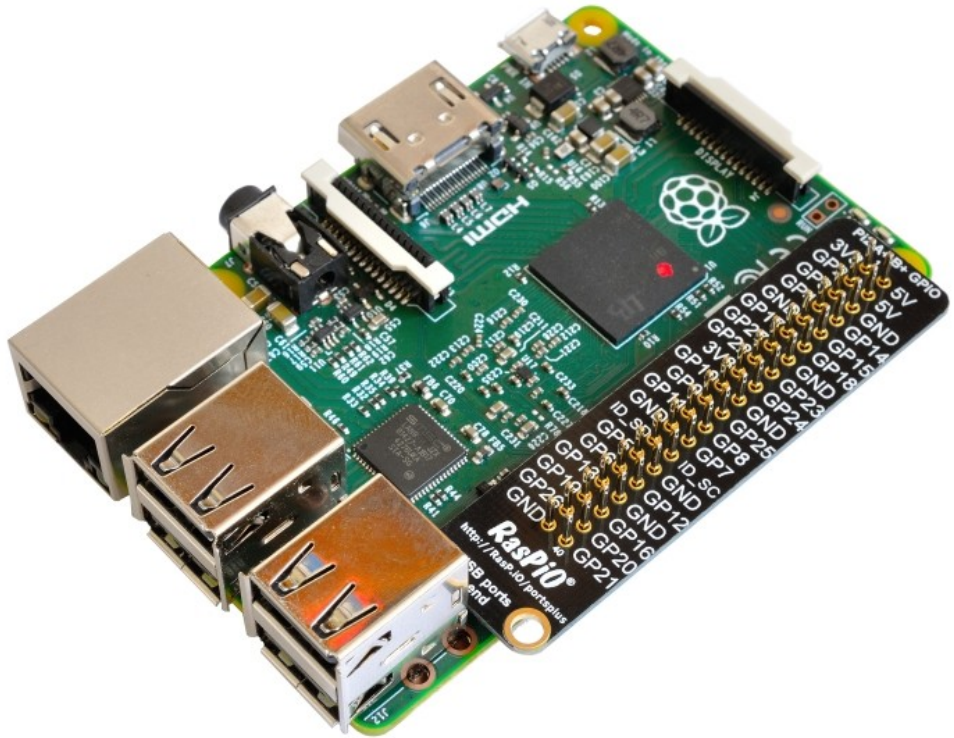
Ben Nuttall

Raspberry Pi Foundation

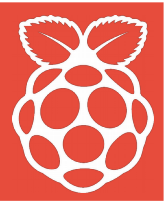
UK Charity 1129409

GPIO Pins – General Purpose Input/Output





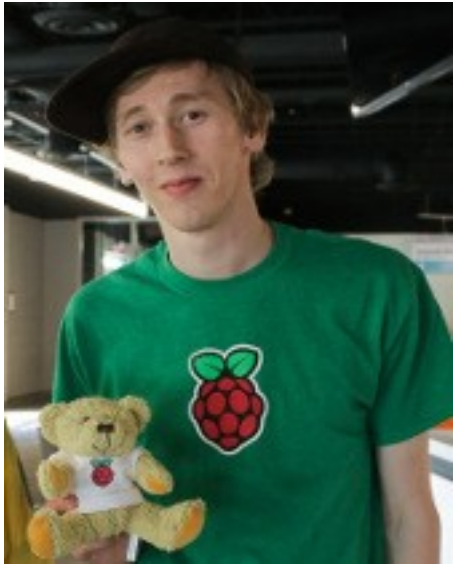
GPIO Zero – a friendly API for GPIO devices



```
from gpiozero import LED

led = LED(2)

led.blink()
```

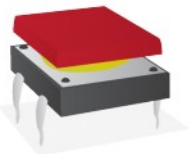
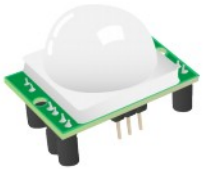
GPIO Zero – a friendly API for GPIO devices

- **Zero-boilerplate Pythonic library**
- **Intended for use in education**
- **Simple, guessable API with commonly used names and sensible default values**
- **Simple introduction, smooth learning curve**
- **Multi-paradigm**
- **Extendable**

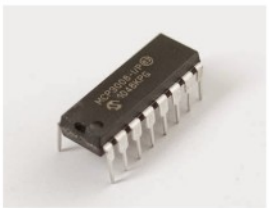


GPIO Zero supports...

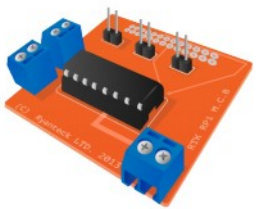


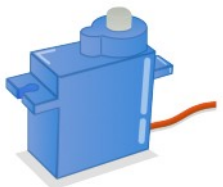





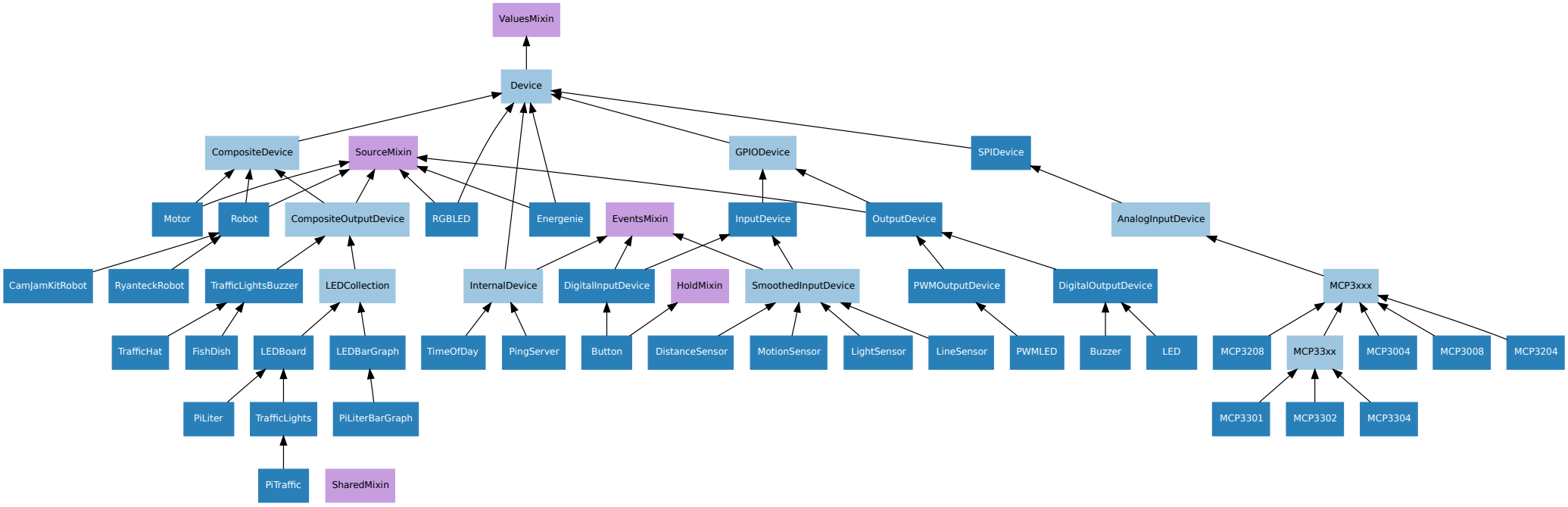





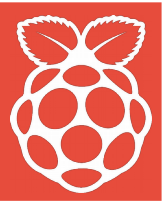





GPIO Zero device hierarchy

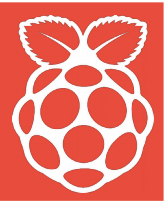


Multi-paradigm: procedural (polling)



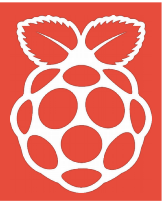
```
from gpiozero import LED, Button

led = LED(17)
button = Button(4)

while True:
    if button.is_pressed:
        led.on()
    else:
        led.off()
```

Multi-paradigm: procedural (blocking)



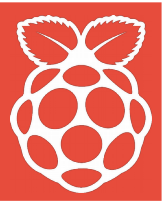
```
from gpiozero import LED, Button

led = LED(17)
button = Button(4)

while True:
    button.wait_for_press()
    led.on()
    button.wait_for_release()
    led.off()
```

Multi-paradigm: event-driven (callbacks)



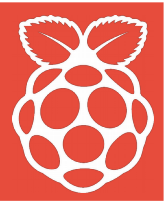
```
from gpiozero import LED, Button

led = LED(17)
button = Button(4)

button.when_pressed = led.on
button.when_released = led.off
```

Multi-paradigm: declarative



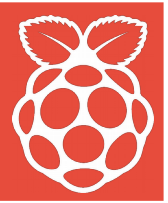
```
from gpiozero import LED, Button

led = LED(17)
button = Button(4)

led.source = button.values
```

.value



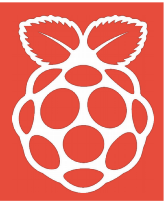
```
>>> led = PWMLED(17)
>>> led.value
0.0
>>> led.on()
>>> led.value
1.0
>>> led.value = 0
```

.value



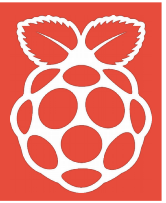
```
>>> led = PWMLED(17)
>>> pot = MCP3008()
>>> led.value
0.0
>>> pot.value
0.510145879738202
>>> led.value = pot.value
```

.value



```
>>> while True:
...     led.value = pot.value
```

Source / Values



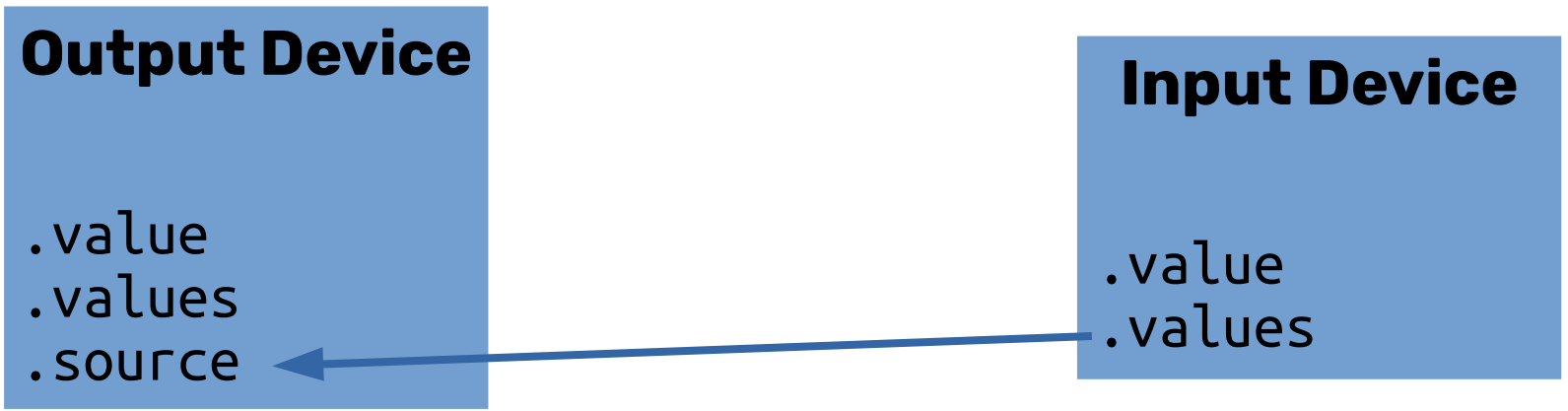
Output Device

```
.value
.values
.source
```

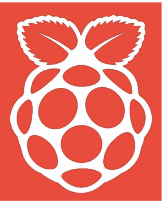
Input Device

```
.value
.values
```

Source / Values

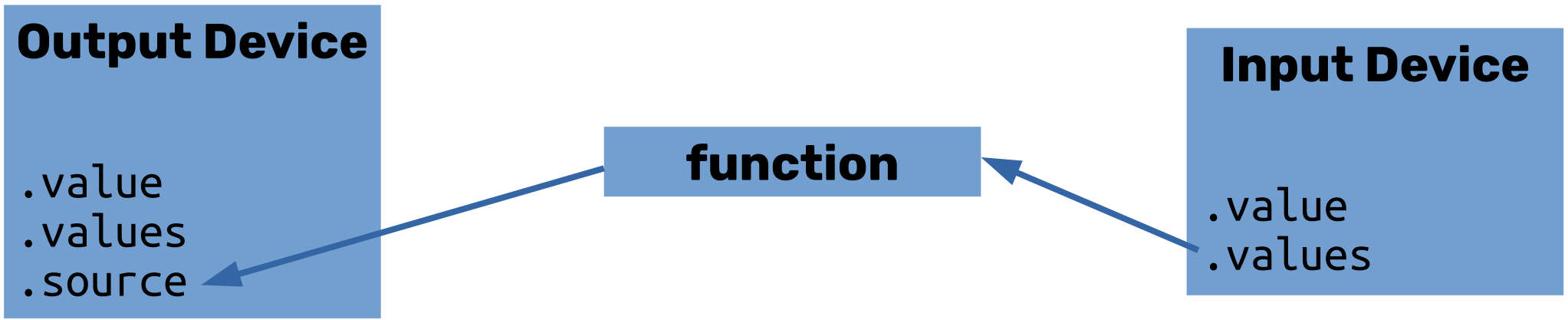


Source / Values

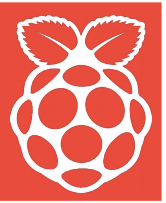


```
from gpiozero import LED, Button

led = LED(17)
button = Button(2)

led.source = button.values
```

Processing values

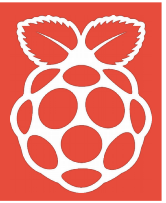


Source tools



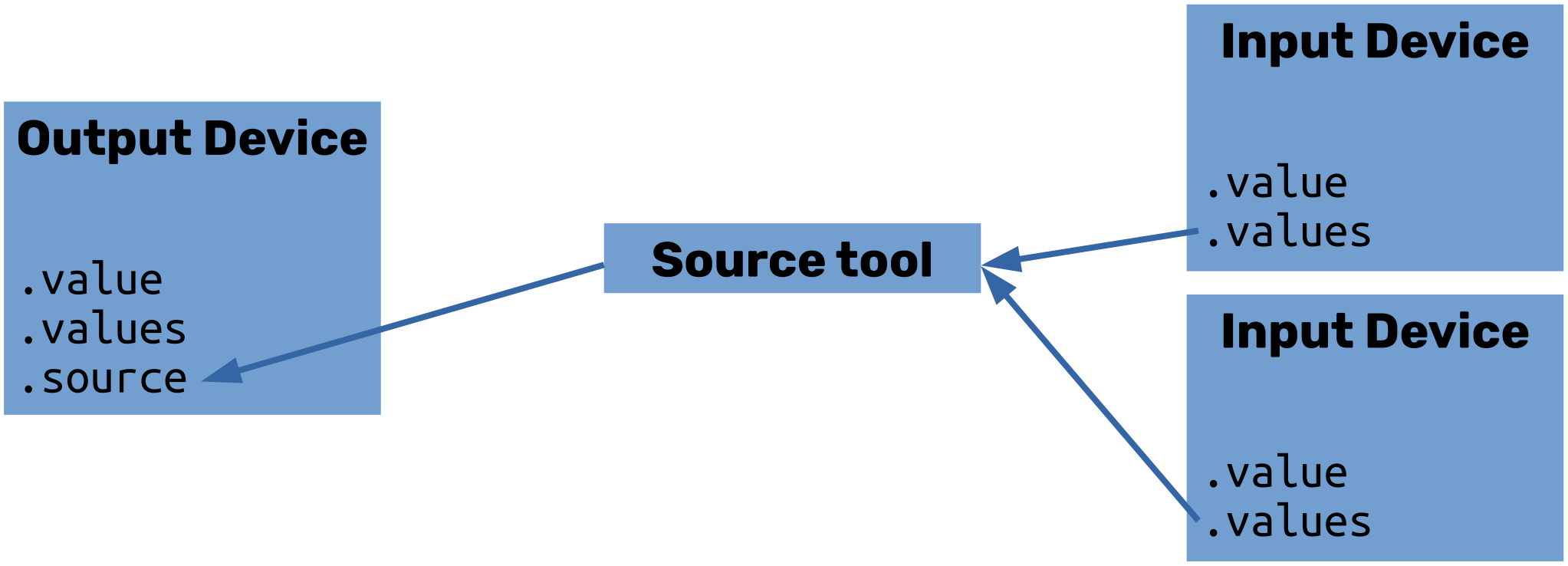
```
from gpiozero import Button, LED
from gpiozero.tools import negated

led = LED(4)
btn = Button(17)

led.source = negated(btn.values)
```

Source tools – single source conversions

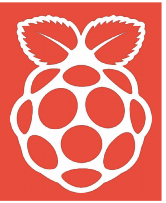


- absoluted
- booleanized
- clamped
- inverted
- negated
- post_delayed
- post_periodic_filtered
- pre_delayed
- pre_periodic_filtered
- quantized
- queued
- smoothed
- scaled

Combining values

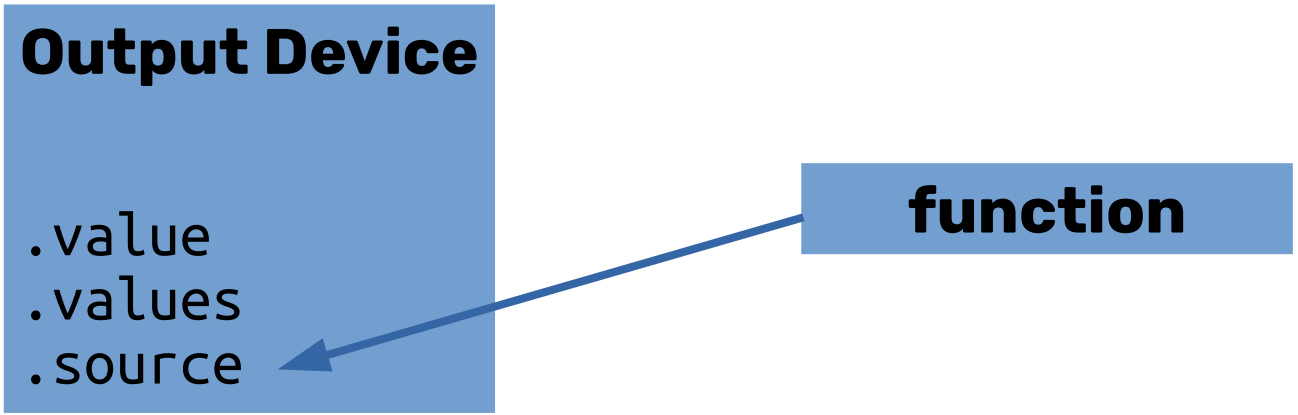


Source tools – combining sources

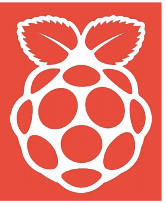


- all_values
- any_values
- averaged
- multiplied
- summed

Artificial values

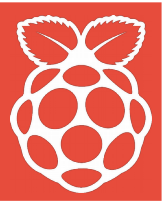


Source tools – artificial sources



- alternating_values
- cos_values
- ramping_values
- random_values
- sin_values

Internal Devices



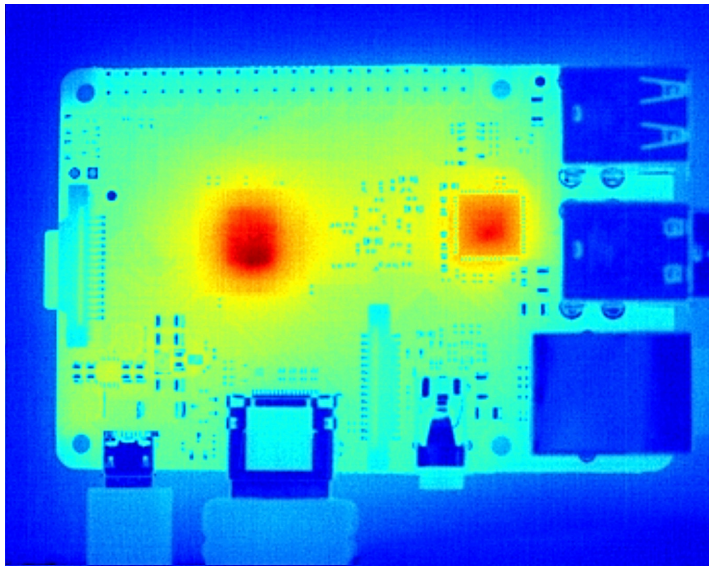
- TimeOfDay
- CPUTemperature
- PingServer
- More coming soon
- Make your own!

Energenie tortoise lamp

```
from gpiozero import Energenie, TimeOfDay
from datetime import time

lamp = Energenie(1)
daytime = TimeOfDay(time(9), time(18))

lamp.source = daytime.values
```

CPU Temperature bar graph

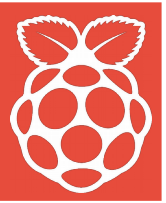
```
from gpiozero import LEDBarGraph, CPUTemperature

cpu = CPUTemperature(min_temp=50, max_temp=90)
leds = LEDBarGraph(2, 3, 4, 5, 6, 7, 8, pwm=True)

leds.source = cpu.values
```



Is the internet working?

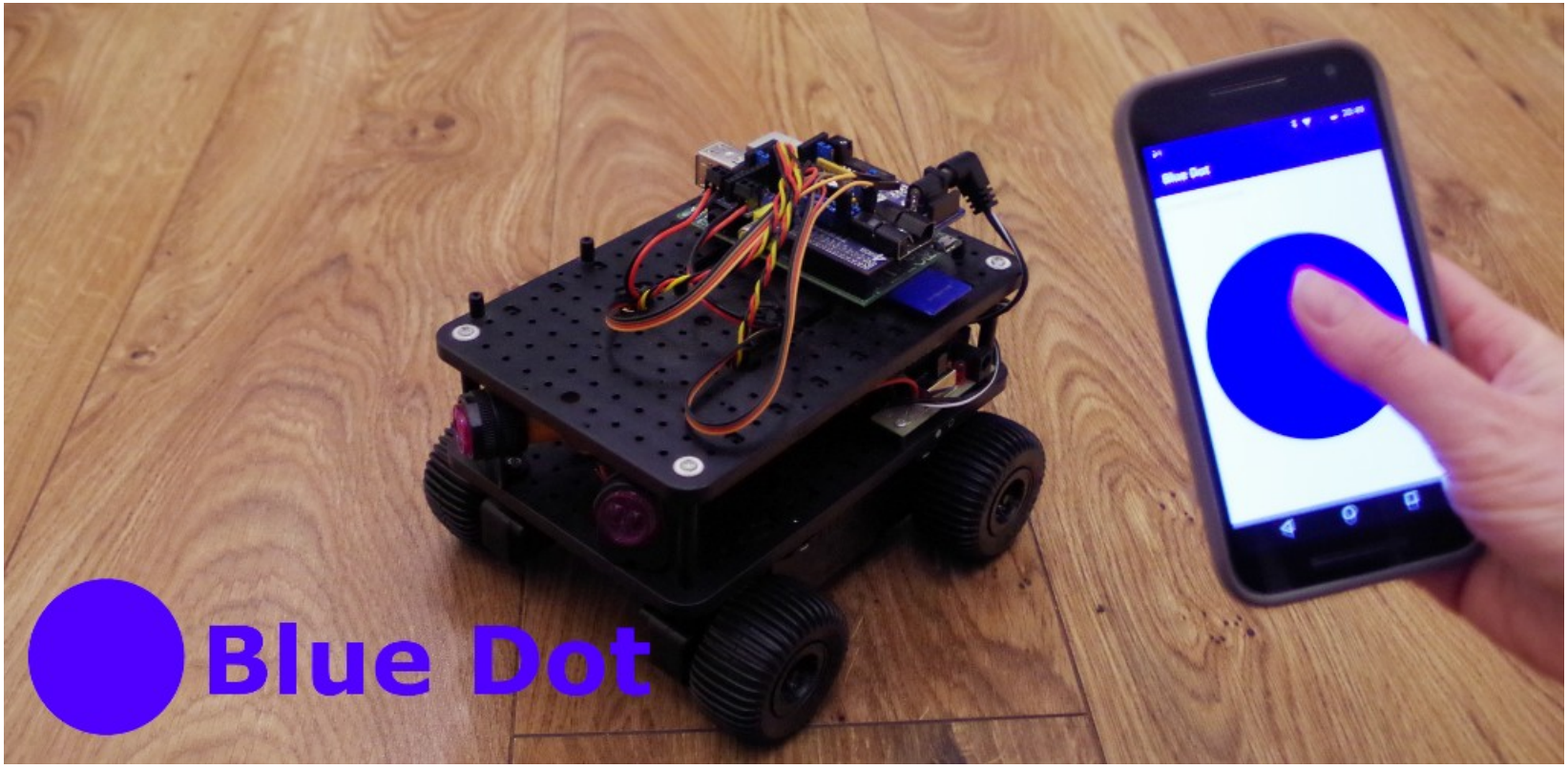
```
from gpiozero import LED, PingServer
from gpiozero.tools import negated

green = LED(17)
red = LED(18)
google = PingServer('google.com')

green.source = google.values
green.source_delay = 60
red.source = negated(green.values)
```

Custom internal devices

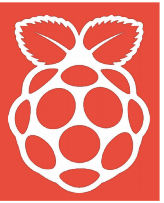


```
from gpiozero import InternalDevice

class FileReader(InternalDevice):
    @property
    def value(self):
        with open('value.txt') as f:
            return int(f.read().strip())
```

Blue Dot

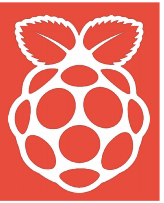


Multi-paradigm: procedural (polling)



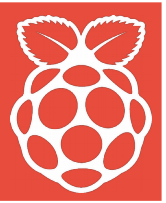
```
from gpiozero import LED
from bluedot import BlueDot

led = LED(17)
bd = BlueDot()

while True:
    if bd.is_pressed:
        led.on()
    else:
        led.off()
```

Multi-paradigm: procedural (blocking)



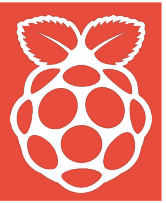
```
from gpiozero import LED
from bluedot import BlueDot

led = LED(17)
bd = BlueDot()

while True:
    bd.wait_for_press()
    led.on()
    bd.wait_for_release()
    led.off()
```

Multi-paradigm: event-driven (callbacks)



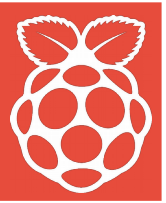
```
from gpiozero import LED
from bluedot import BlueDot

led = LED(17)
bd = BlueDot()

bd.when_pressed = led.on
bd.when_released = led.off
```

Multi-paradigm: declarative



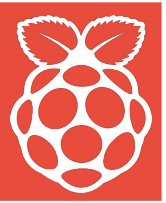
```
from gpiozero import LED
from bluedot import BlueDot

led = LED(17)
bd = BlueDot()

led.source = bd.values
```

GPIO Zero: cross-platform – distributed via apt/pip



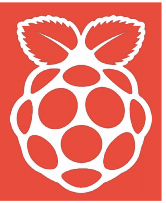
- **Raspberry Pi**
  - Raspbian, Debian, Ubuntu, etc
- **PC & Mac**
  - Raspberry Pi Desktop x86
  - Linux
  - Mac OS
  - Windows

Supporting multiple back-ends



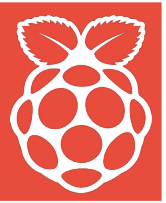
- **RPi.GPIO**
  - Low-level GPIO library, implemented in C (current default)
- **RPIO**
  - Low-level GPIO library, implemented in C (only supports Pi 1)
- **pigpio**
  - Low-level GPIO library, implemented in C
  - Runs as a daemon on the Pi, can accept remote commands
- **Native**
  - Pure Python, limited functionality, experimental (included in gpiozero)
- **Mock**
  - Pure Python, used in test suite, useful for testing (included in gpiozero)

MockPin



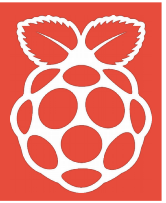
```
$ GPIOZERO_PIN_FACTORY=mock python3
>>> from gpiozero import LED
>>> led = LED(22)
>>> led.blink()
>>> led.value
True
>>> led.value
False
```

MockPin



```
>>> from gpiozero import LED, Button
>>> led = LED(22)
>>> button = Button(23)
>>> led.source = button.values
>>> led.value
False
>>> button.pin.drive_low()
>>> led.value
True
```

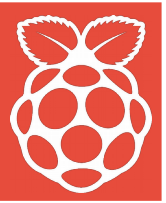
pigpio - remote GPIO from Pi or PC



```
$ GPIOZERO_PIN_FACTORY=pigpio PIGPIO_ADDR=192.168.0.2 python3 led.py
```

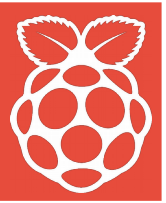
```
from gpiozero import LED

led = LED(22)

led.blink()
```

pigpio - remote GPIO from Pi or PC



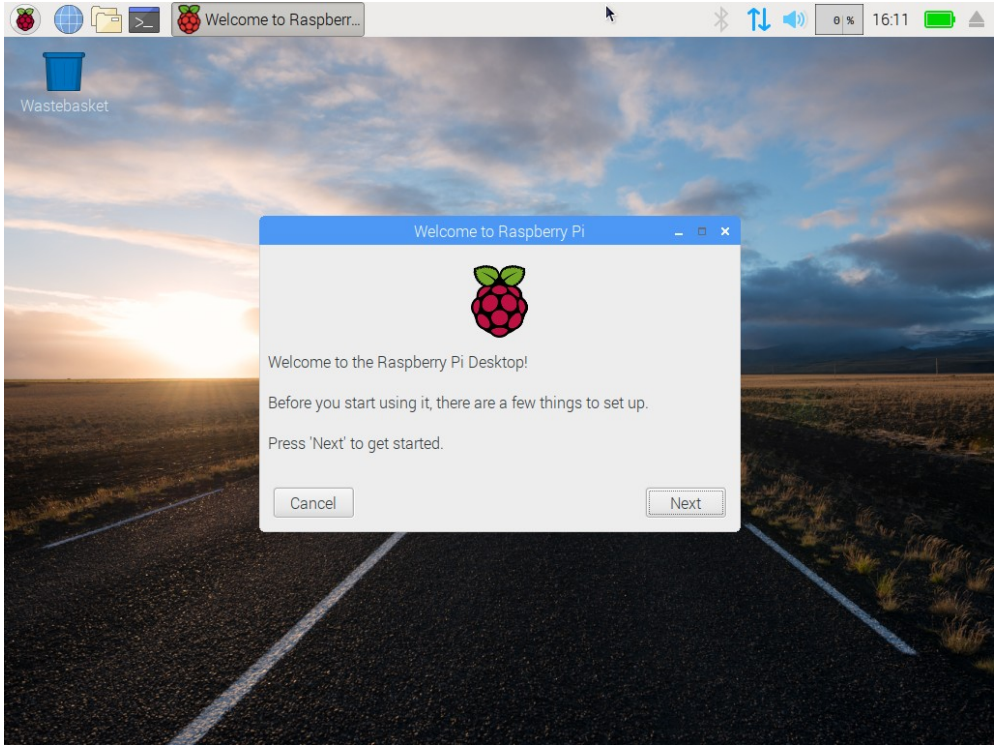
```
from gpiozero import LED
from gpiozero.pins.pigpio import PiGPIOFactory

factory = PiGPIOFactory('192.168.0.2')

led = LED(22, pin_factory=factory)

led.blink()
```

pigpio - remote GPIO from Pi or PC

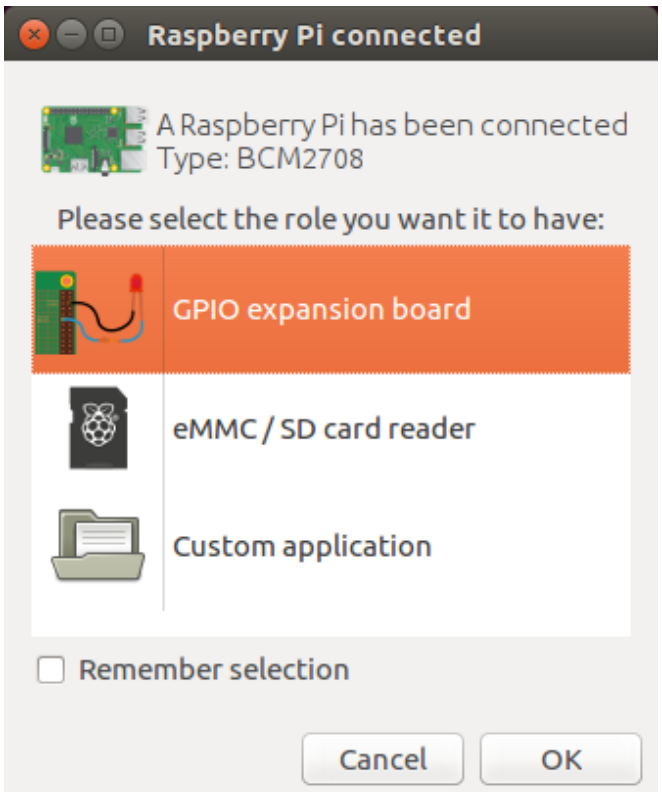


```
from gpiozero import LED, Button
from gpiozero.pins.pigpio import PiGPIOFactory

remote = PiGPIOFactory('192.168.0.2')

led = LED(22)
btn = Button(22, pin_factory=remote)

led.source = btn.values
```

Raspberry Pi Desktop x86 OS



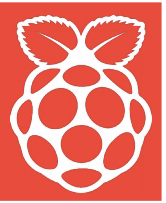
Pi Zero GPIO Expander



Pi Zero GPIO Expander

```
from gpiozero import LED
from gpiozero.pins.pigpio import PiGPIOFactory

pizero = PiGPIOFactory('fe80::1%usb0')

led = LED(22, pin_factory=pizero)

led.blink()
```

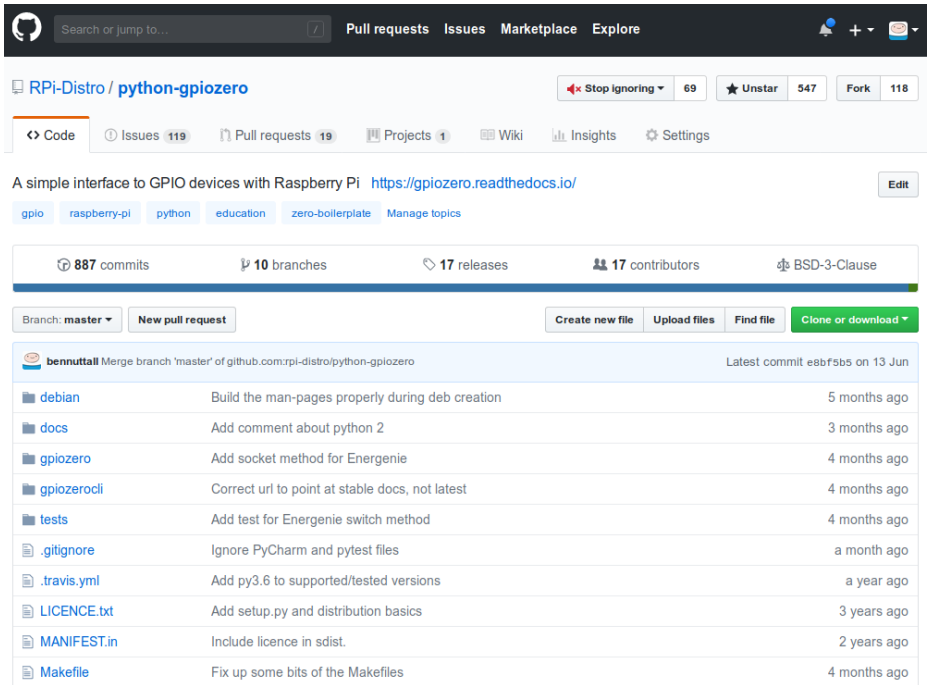
IoT devices?

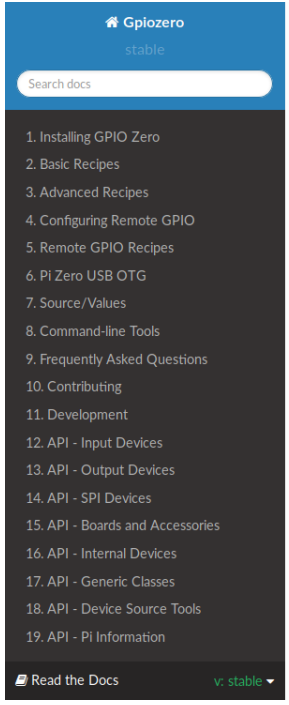


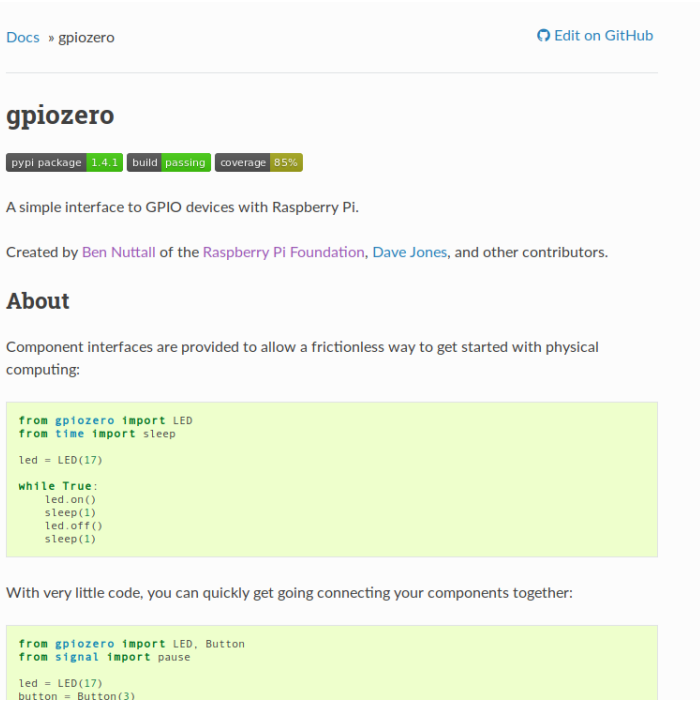
```
from somelib import GardenLight, LightSensor, MotionSensor
from gpiozero.tools import all_values, negated

garden = GardenLight()
light = LightSensor()
motion = MotionSensor()

garden.source = all_values(negated(light.values), motion.values)
```

Z-Wave devices & asyncio

GPIO Zero on GitHub & ReadTheDocs







piwheels

- Python package repository providing **Arm platform wheels** for Raspberry Pi
- **Builds automated** from PyPI releases, plus manual builds e.g. opencv & tensorflow
- Raspbian is pre-configured to use **piwheels.org** as an additional index to PyPI
- Massively reduces **pip install** time for Raspberry Pi users
- **Natively compiled** on Raspberry Pi 3 hardware (Mythic Beasts Pi cloud)
- Repo hosted on single Raspberry Pi serving **300-400k packages per month**

Raspberry Jam

- Independently organised community events around the world
- Family-friendly
- Mix of meetup / conference / workshop styles
- Makers, hackers, programmers & beginners come together
- Find one near you – or start your own!
- raspberrypi.org/jam

CoderDojo

- **Free coding clubs for young people**
- **Find one near you and volunteer as a mentor – or start a new Dojo in your area**
- **coderdojo.com**

Raspberry Pi & Python poster session today!

Python and Raspberry Pi

Explore physical computing and more using Python on Raspberry Pi

The Raspberry Pi is a small affordable computer which runs a Debian-based operating system called Raspbian. It has been designed for the purpose of education, and it is also used by hobbyists and in industry across the globe.

The Raspberry Pi Foundation

The Raspberry Pi Foundation is a charity that works to put the power of digital making into the hands of people all over the world by making computing accessible to all. More than 20 million Raspberry Pi computers have been sold since the first product launch in 2012, and all sales profits go towards the Foundation's educational programmes, courses, and resources.

Current Raspberry Pi models

Raspberry Pi 3 Model B+

- 64-bit quad-core Armv8 CPU @ 1.4GHz
- VideoCore IV GPU
- 1GB RAM
- $35

Raspberry Pi Zero / Zero W

- 32-bit single core Armv6 CPU @ 1GHz
- VideoCore IV GPU
- 512MB RAM
- $5/$10

Raspbian operating system

- Based on Debian Stretch
- Optimised for Raspberry Pi hardware
- Custom lightweight desktop theme
- Includes Python 3.5 and 2.7

GPIO pins

These pins (General-Purpose Input/Output pins) allow you to connect electronic components and to program physical devices, e.g. sensors or lights.They are useful for home automation and a diversity of maker projects. You can connect components directly to the pins using jumper wires, or use a breadboard and allow components to share use of some pins. The pins include:

- 3V3 (a constant supply of 3.3 volts)
- 5V (a constant supply of 5 volts)
- GND (ground pins — 0 volts)
- GPIO (general-purpose pins)
- SPI (Serial Peripheral Interface)
- I2C (Inter-Integrated Circuit)
- UART (Universal Asynchronous Receiver/Transmitter)

pinout.xyz

Add-on boards/HATs

Instead of connecting components to GPIO pins, you can use add-on boards which consist of embedded components on a PCB (printed circuit board) and sit on top of the Pi's GPIO pins. HAT (Hardware Attached on Top) add-on boards are very useful for extending the capabilities of your Raspberry Pi without needing to wire up or solder components. The Foundation has specified a HAT standard to determine which add-on boards can be considered HATs. An ever-growing range of HATs is available from the community of Raspberry Pi accessory retailers.

rpf.io/hats

Python and GPIO Zero

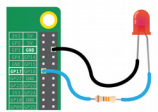
You can control the GPIO pins using a wide range of programming languages, but the easiest and most popular one is Python. The GPIO Zero library provides a simple interface to GPIO devices, and includes support for a range of components and add-on boards. With just a few lines of code you can flash an LED:

```
from gpiozero import LED
from time import sleep

led = LED(17)

while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

Or even:

```
from gpiozero import LED

led = LED(17)

led.blink()
```

GPIO Zero works with a selection of low-level GPIO libraries including pigpio, which supports remote GPIO. This means you can run Python code on one Raspberry Pi to control devices on another Pi, or even multiple Pis, or you can run the code on a PC or Mac. The library also provides a 'mock pin' tool which allows you to your test code without Raspberry Pi hardware.



Camera Module and picamera

The Camera Module is an official Raspberry Pi accessory and comes in two versions: a visible-light camera, and an infrared camera. The current versions have an 8-megapixel resolution. The Camera Module can be controlled with the command-line tools `raspistill` and `raspivid`, or with the Python library picamera:

```
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
sleep(10)
camera.capture('/home/pi/image.jpg')
camera.stop_preview()
```

With the help of picamera and GPIO Zero, you will be able to create physical projects like photo booths. You can also expand your Camera Module project to include web streaming via an HTTP server, image processing using the Python Imaging Library, computer vision using OpenCV, and more.

rpf.io/gswcam

Sense HAT

The Sense HAT is a Raspberry Pi add-on board made especially for ESA astronaut Tim Peake to take to the International Space Station for our Astro Pi programme. This programme gives young people across Europe the opportunity to run their Python code in space!
The Sense HAT comprises:

- Temperature sensor
- Humidity sensor
- Pressure sensor
- Accelerometer
- Gyroscope
- Magnetometer
- 8×8 RGB LED matrix
- Mini joystick

The Sense HAT's Python library provides easy access to the sensors and the display:

```
from sense_hat import SenseHat

sense = SenseHat()

while True:
    r = 255 * sense.humidity / 100
    sense.clear(r, 0, 0)
```

In order to make the Astro Pi competition more accessible, we also provide Sense HAT emulators: an online version on trinket.io and a desktop version (just run `pip install sense_emu`). rpf.io/sh

piwheels

piwheels is a Python package repository which provides Arm wheels (pre-compiled binaries) for all packages on PyPI, compiled natively on a cluster of Raspberry Pi 3s. It provides wheels compatible with all Raspberry Pi models and for all major Python versions. This vastly reduces install time for packages like numpy.

In Raspbian, pip is configured to use the piwheels server as an additional index, giving your Pi automatic access to the wheels. The main PyPI server can be used as a fallback for any missing packages.

rpf.io/piwheels

Multi-paradigm programming

Python is a multi-paradigm programming language, so you can write code in a number of different styles, like procedural, event-driven, object-oriented and functional. For simple tasks, the same outcome can be achieved using one of several styles.

GPIO Zero makes it easy to get started, and enables users to progress along the learning curve towards more advanced programming techniques. For example, if you want to make a push button control an LED, the easiest way to do this is via procedural programming using a while loop:

```
from gpiozero import LED, Button

led = LED(17)
button = Button(2)

while True:
    if button.is_pressed:
        led.on()
    else:
        led.off()
```

Another way to achieve the same thing is to use events (callbacks):

```
from gpiozero import LED, Button

led = LED(17)
button = Button(2)

button.when_pressed = led.on
button.when_released = led.off
```

You could even use a declarative approach, and set the LED's behaviour in a single line:

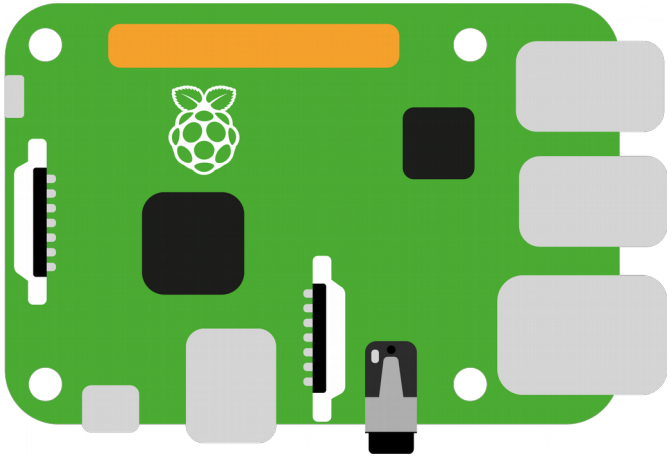
```
from gpiozero import LED, Button

led = LED(17)
button = Button(2)

led.source = button.values
```

rpf.io/gpiozero

Get involved

Use your programming skills to become a digital maker, contribute to open-source projects, and volunteer to engage more young people in making things with Python!

- Raspberry Jams are family-friendly community events for people of all ages — find your nearest one, or find out how to start your own, at rpf.io/jam
- Code Club is a worldwide network of volunteer-led coding clubs for children aged 9-13. rpf.io/cc
- CoderDojo is a global network for free computer programming clubs for young people. rpf.io/cd
- See the Raspberry Pi website for more of our initiatives and programmes! rpf.io



Programming paradigms for physical computing and IoT

Ben Nuttall

Raspberry Pi Foundation

UK Charity 1129409