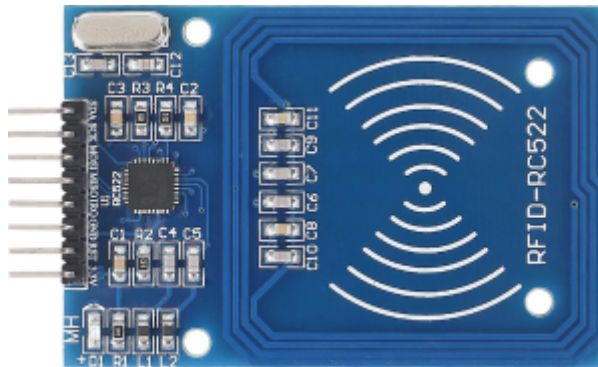


Pico with RFID-RCS522

Aug 2026



Introduction



RFID stands for **radio-frequency identification**, which is a wireless technology that uses radio waves to identify, track, and communicate information between a tag and a reader.

RC522 is a low-cost, 13.56 MHz radio frequency identification (RFID) reader and writer module powered by the NXP MFRC522 chip, commonly used with microcontrollers like Arduino and Raspberry Pi.

Key Specifications

- Operating Frequency: 13.56 MHz
- Operating Voltage: 2.5V to 3.3V (Note: Always power VCC with 3.3V; supplying 5V can damage the module, though data pins are mostly 5V tolerant)
- Operating Current: 13 mA to 26 mA
- Read Range: About 2 cm to 5 cm depending on the tag
- Supported Protocols: ISO/IEC 14443 Type A (such as MIFARE 1K, 4K, and Ultralight)
- Communication Interfaces: SPI (default), I2C, and UART

Pinout Configuration

- VCC: Power supply, strictly 3.3V
- RST: Reset pin
- GND: Ground reference
- IRQ: Interrupt pin to alert the microcontroller
- MISO / SCL / Tx: Master-In-Slave-Out for SPI
- MOSI: Master-Out-Slave-In for SPI
- SCK: Serial Clock for SPI
- SS / SDA / Rx: Slave Select / Signal input

RFID Frequencies

There are three main frequency bands (with a fourth microwave band sometimes included).

Low Frequency (LF) Range:

- 125 kHz to 134.2 kHz (or 120–150 kHz).
- Use cases: Animal tracking (pet microchips) and key fobs.

- Range distance: Short (up to 10 cm or a few inches).

High Frequency (HF)

- Range: 13.56 MHz.
- Use cases: Smart cards, payment badges, and NFC (Near Field Communication).
- Range distance: Short (up to 1 meter).

Ultra-High Frequency (UHF)

- Range: 860 MHz to 960 MHz (plus a 433 MHz band for active tags).
- Use cases: Warehouse inventory, toll passes, and supply chains.
- Range distance: Long (from a few meters up to 100 meters for active tags).

Microwave Frequency (Optional 4th Band)

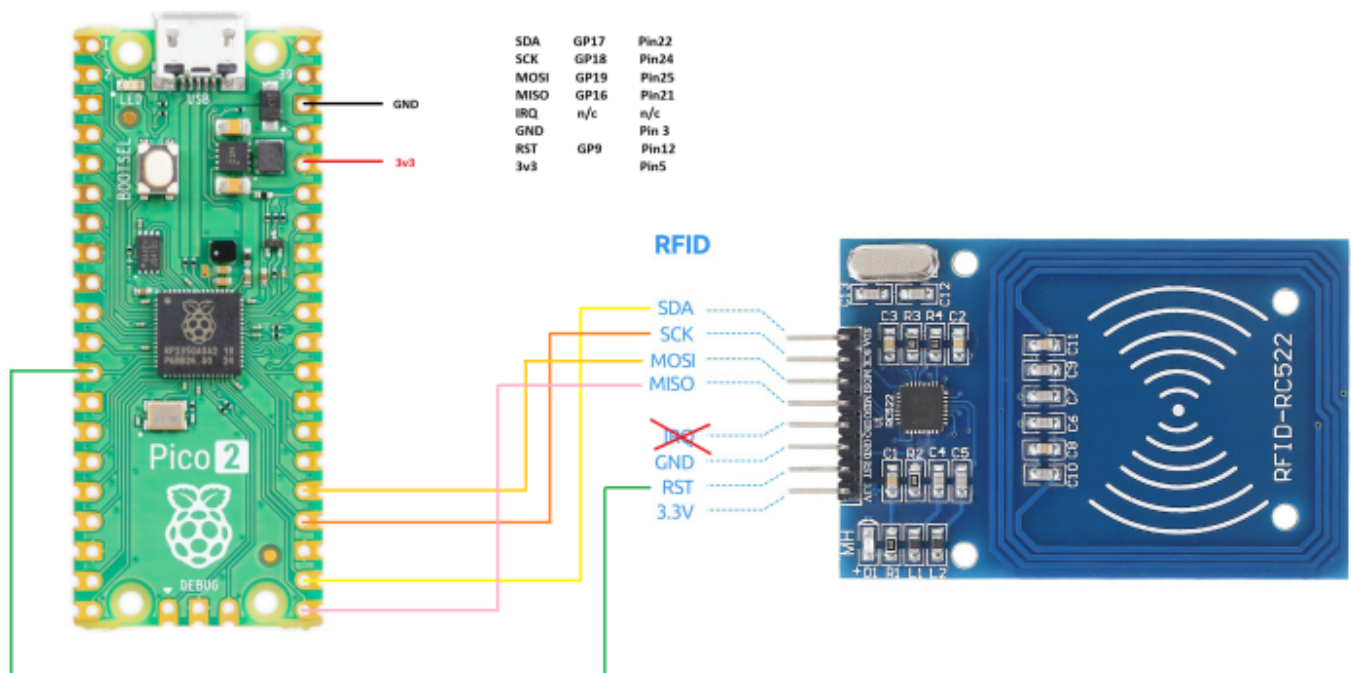
- Range: 2.45 GHz.
- Use cases: Long-range active tracking and toll collection.
- Range distance: Very long.

The device we will be using is the High Frequency (HF) 13.56 MHz Range.

Wiring for Raspberry Pi Pico

Here is the wiring for a Raspberry Pi Pico. Remember we are using:

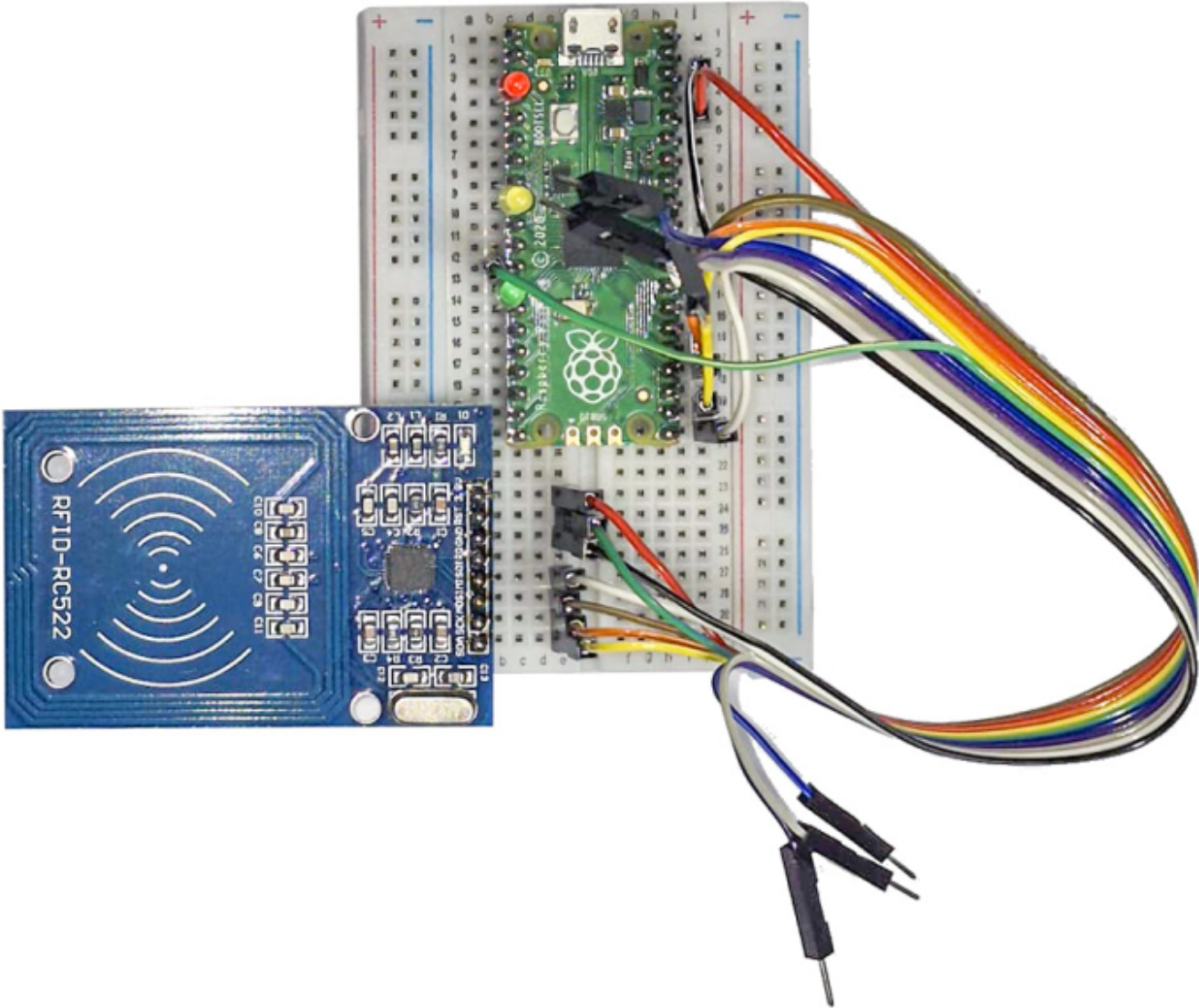
- SPI Protocol
- SPI Interface 0



Here are the pin connections, in case the diagram is not clear.

Signal	GPIO	Pico Pin Number
SDA	GP17	Pin 22
SCK	GP18	Pin 24
MOSI	GP19	Pin 25
MISO	GP16	Pin 21
IRQ	n/c	n/c
GND	—	Pin 3
RST	GP9	Pin 12
3V3	—	Pin 5

Once you have wired up your Pico and RFID-RCS522 we can move on to some code.



Some test Python Scripts

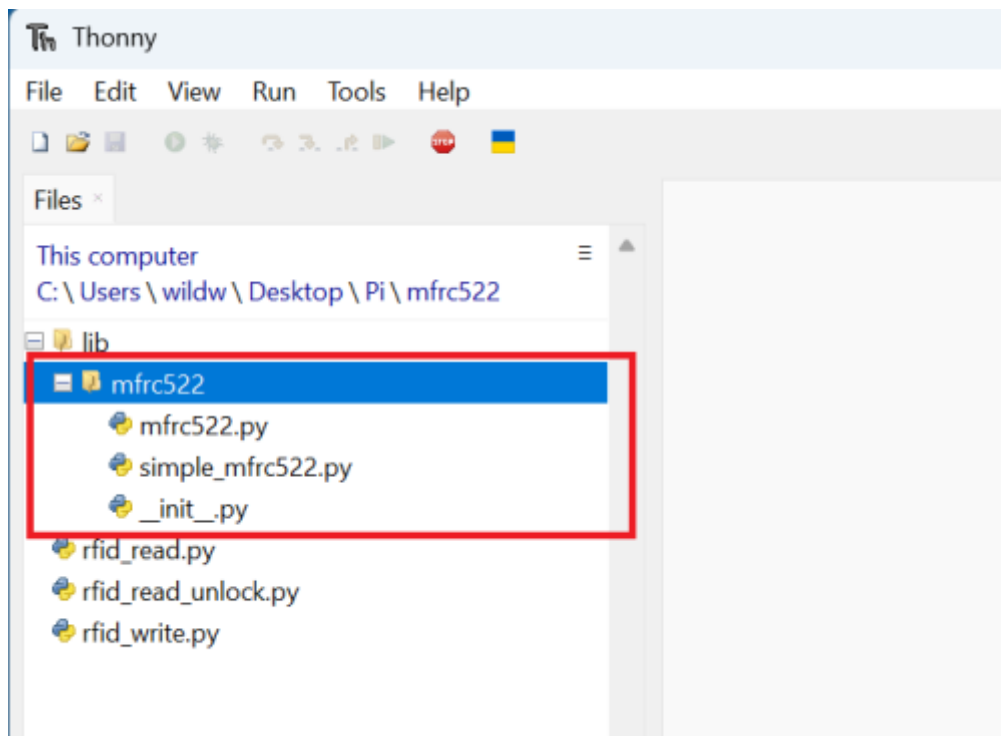
We will start with two scripts.

First script to Read an RFID Tag. Second Script to Write to an RFID Tag.

Once we have written to a tag we will use the first script to ensure that we can read what was written.

Before we begin, we have to copy some files to our Raspberry Pi Pico, driver files basically so that we can read and write to our RFID tags.

First of all, we need to create a folder on our Pico under the lib folder (if you don't have the lib folder, create this first). The folder name is **mfr522**.



In this /lib/mfr522 folder, we need to put the following three files:

- `_init__.py`
- `mfr522.py`
- `simple_mfr522.py`

These are the drivers for the RFID card we are using. You can download them from

here

.

In the top level folder, we need to add these two files:

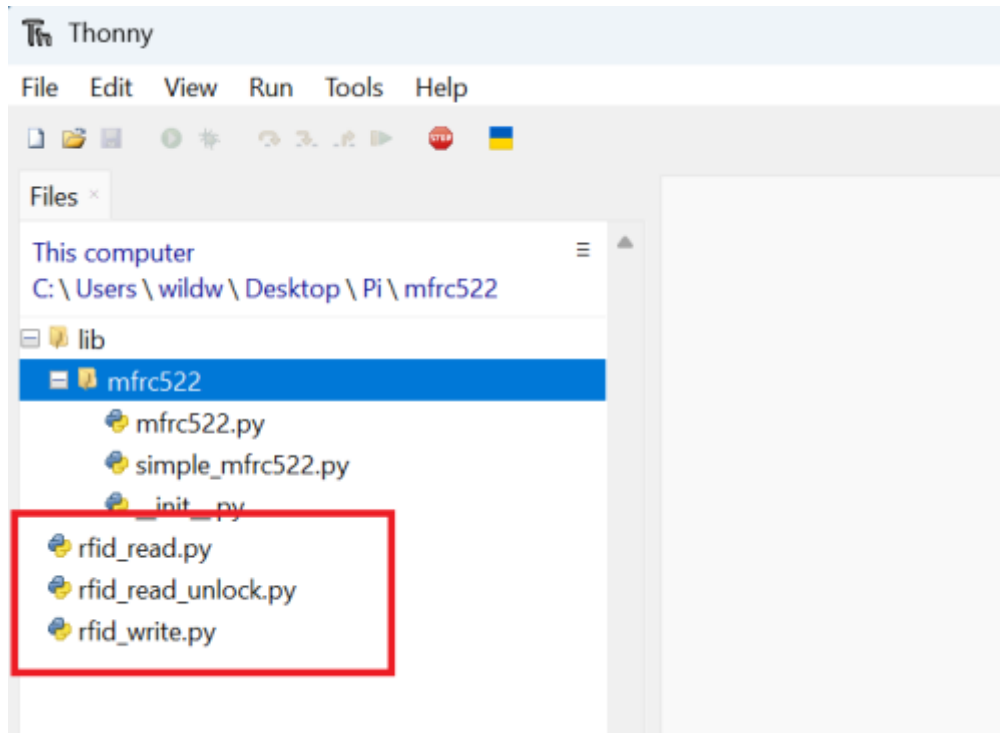
- `rfid_read.py`
- `rfid_write.py`

These are the two Python scripts we will use to read and write to and from our RFID Tags, you can download them

here

.

Here you can see the two files (plus one extra we will not use at this time).



At this point you should have a fully wired Pico/RFID board, plus all the software installed (in the correct folder structure).

A Quick Note About RFID Tags

If you purchased an RFID kit, then it likely came with a few items.

RFID Writer/Reader RFID Tag (Keyring) RFID Card.



If you don't have the tag or card at this time, then you need to acquire these to proceed with the testing.

These tags have various storage capabilities as laid out below:

RFID tag storage capacity ranges from 64 bits for basic identification tags up to 8 kilobytes or more for advanced data-storing tags.

Typical Capacities by Memory Type

- **Basic / Low-Memory Tags:** 64 to 128 bits. These store a basic serial number or electronic product code (EPC) that links to an external database.

- **Medium-Memory Tags:** 512 bits to 2 kilobytes (KB). These store extra item details, user data, or small web links (such as NFC NTAG chips with 144 to 888 bytes of user memory).
- **High-Memory Tags:** Up to 8 KB or more. Specialized industries like aerospace use these to store complete repair and parts histories directly on the item.

Remember:

- A 128-bit value equals 32 characters in hexadecimal or 16 characters in standard 8-bit text.
- A 1k bits (kilobits) equals 125 standard ASCII or UTF-8 characters using decimal math, or 128 characters using binary math.

Using the Read / Write Python Code

There are two main Python scripts here, one to read tags, one to write tags. Lets start with the tag reading Python code.

From Thonny, run the Python script `rfid_read.py`



The screenshot shows the Thonny IDE interface. The top menu bar includes File, Edit, View, Run, Tools, and Help. The left sidebar shows the file explorer with 'This computer' and 'RP2040 device' sections. The 'RP2040 device' section is expanded, showing a list of files including 'lib', 'traffic lights', '3 Sequence LEDs.py', 'Controlling onboard and 3 leds.py', 'GPIO14 LED.py', 'lcdtest.py', 'led_blink.py', 'led_flash_chatgpt.py', 'led_on.py', 'mian-traffic-light-Pico1 & 2.py', 'mian-traffic-light-Pico1W & 2W.py', 'rfid_read.py', 'rfid_read_unlock.py', 'rfid_write.py', 'Traffic-Light-With-Breakout-Button-better-dete', 'Traffic-Light-With-Breakout-Button.py', and 'traffic_light_with_main.py'. The 'rfid_read.py' file is selected. The main editor area shows the code for 'rfid_read.py' with line numbers 1 through 10. The code is as follows:

```
1 from mfrc522 import SimpleMFRC522
2
3 reader = SimpleMFRC522(spi_id=0,sck=18,miso=16,mosi=19,cs=17,rst=9)
4
5 def read():
6     print("Reading...Please place the card...")
7     id, text = reader.read()
8     print("ID: %s\nText: %s" % (id,text))
9
10 read()
```

Below the code editor is a 'Shell' window showing the MicroPython v1.28.0 on 2026-04-06; Raspberry Pi Pico with RP2040. It prompts the user to type "help()" for more information and shows the prompt '>>>'.

MicroPython (RP2040) • Board CDC @ COM4

Click the play button in Thonny.

The screenshot shows the Thonny IDE interface. The left sidebar displays the file explorer with the 'rfid_read.py' file selected. The main editor window shows the following Python code:

```

1 from mfrc522 import SimpleMFRC522
2
3 reader = SimpleMFRC522(spi_id=0,sck=18,miso=16,mosi=19,cs=17,rst=9)
4
5 def read():
6     print("Reading...Please place the card...")
7     id, text = reader.read()
8     print("ID: %s\nText: %s" % (id,text))
9
10 read()

```

The Shell window at the bottom shows the execution output:

```

>>> %Run -c $EDITOR_CONTENT
MPY: soft reboot
Reading...Please place the card...

```

Place your RFID Card/Keyring on the reader for a scan to occur.

The screenshot shows the Thonny IDE interface after an RFID scan. The main editor window shows the same Python code as before. The Shell window now displays the following output:

```

>>> %Run -c $EDITOR_CONTENT
MPY: soft reboot
Reading...Please place the card...
ID: 07 69 0D 34
Text: mary had a little lamb
>>>

```

Above we can see that the following was read from the RFID Tag.

- ID: 07 69 0D 64
- Text: mary had a little lamb

The ID is the Tag ID, the Text is something I have written to the RFID Tag previously.

Now we can write some data to the RFID Tag. Stop the running `rfid_read.py` and run the `rfid_write.py`.



The screenshot shows the Thonny IDE interface. The left sidebar displays the file explorer with the 'RP2040 device' selected, showing a list of files including 'rfid_write.py'. The main editor window displays the code for 'rfid_write.py':

```
1 from mfrc522 import SimpleMFRC522
2
3 reader = SimpleMFRC522(spi_id=0,sck=18,miso=16,mosi=19,cs=17,rst=9)
4
5 def write():
6     to_write = input("Please enter the message: ")
7     print("Writing...Please place the card...")
8     id, text = reader.write(to_write)
9     print("ID: %s\nText: %s" % (id,text))
10
11 write()
```

The Shell window at the bottom shows the execution output:

```
>>> %Run -c $EDITOR_CONTENT
MPY: soft reboot
Please enter the message:
```

Above we are being prompted to Enter a message, enter a short message, bearing in mind your Tag might not have a lot of memory. Press enter.



The screenshot shows the Thonny IDE interface. The left sidebar displays the file explorer with the 'RP2040 device' selected, showing a list of files including 'rfid_write.py'. The main editor window displays the code for 'rfid_write.py':

```
1 from mfrc522 import SimpleMFRC522
2
3 reader = SimpleMFRC522(spi_id=0,sck=18,miso=16,mosi=19,cs=17,rst=9)
4
5 def write():
6     to_write = input("Please enter the message: ")
7     print("Writing...Please place the card...")
8     id, text = reader.write(to_write)
9     print("ID: %s\nText: %s" % (id,text))
10
11 write()
```

The Shell window at the bottom shows the execution output:

```
>>> %Run -c $EDITOR_CONTENT
MPY: soft reboot
Please enter the message: Writing to Keyfob Yo! Yo! Yo!
Writing...Please place the card...
```

Now the Python rfid_write.py tells us it is writing to the card.

We now have to run the rfid_read.py again, to confirm that the write was successful.

```

1 from mfr522 import SimpleMFRC522
2
3 reader = SimpleMFRC522(spi_id=0,sck=18,miso=16,mosi=19,cs=17,rst=9)
4
5 def write():
6     to_write = input("Please enter the message: ")
7     print("Writing...Please place the card...")
8     id, text = reader.write(to_write)
9     print("ID: %s\nText: %s" % (id,text))
10
11 write()

```

```

>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Please enter the message: Writing to Keyfob Yo! Yo! Yo!
Writing...Please place the card...
Card detected 07 69 0D 34
ID: 07 69 0D 34
Text: Writing to Keyfob Yo! Yo! Yo!
>>>

```

Once running, tap your RFID card and you should see the message that you typed in during the `rfid_write.py` process.

At some point in the future we will come back to this and add some code to allow us to make real world changes when RFID tags are scanned, maybe turn on/off an LED or display an RFID tag ID: on an LCD screen.

Practical Example

This is a good example to show what you can do reading RFID tags. Now, I want to make something very, very clear. RFID tags have Unique IDs (UID) and you should be using this UID to ensure that only tags you have authorised can be used on your system. For this example, we are not using UIDs, because it's just a simple example of using an RFID tag to do something.

I am assuming you have read this entire article, if you have not, you will probably have a lot of issues, so I suggest you have a good read to understand the RFID Python scripts.

We are going to create 3 RFID tags. If you look in the previous section above, you can see we have used two Python scripts.

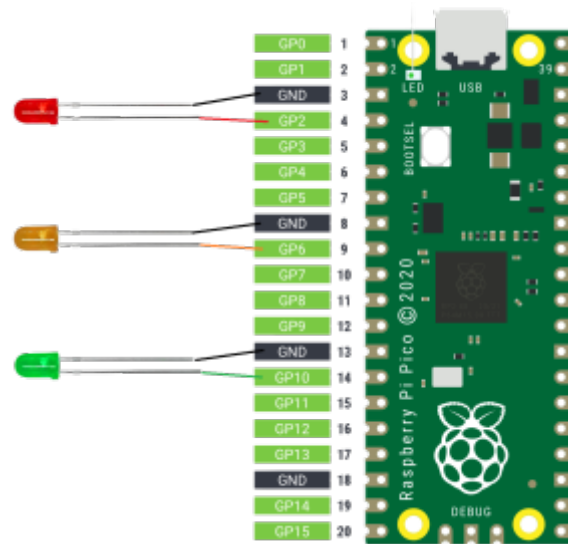
- `rfid_read.py`
- `rfid_write.py`

Initially, we are going to use the `rfid_write.py`. Using this Python script, create three RFID tags, RED, AMBER and GREEN.

What we are going to do is to turn on some LEDs depending on which RFID Tag is scanned.

The Hardware

Connect three LEDs, as shown in the diagram below.



If you used the same GPIO Pins that we used in this example to connect the RFID Reader/Writer, then the pins for the LEDs above will be free.

From:
<https://wiki.alanwalker.uk/> - WalkerWiki - wiki.alanwalker.uk

Permanent link:
https://wiki.alanwalker.uk/doku.php?id=pico_with_rfid-rcs522&rev=1788299028

Last update: 2026/09/01 21:43

