

# Instructions:

## CLEVER CARD KIT FOR RASPBERRY PI



# KIT CONTENTS

Before you start making your Smartcard Raspberry Pi projects, please make sure that you have all the parts listed below:

RFID reader



6 x RFID cards



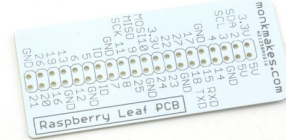
RFID key fob



10 x Female to female jumper wires



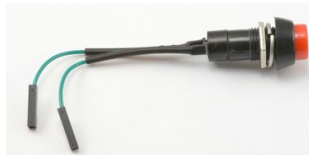
Raspberry Leaf GPIO template



Raspberry Squid RGB LED



Raspberry Squid Button



You will of course also need a Raspberry Pi with a recent version of Raspbian on it.

Technically speaking, the ‘Clever Cards’ of this kit are actually contactless RFID cards of the type known as MiFare® 1k (13.56MHz). These tags are available in credit card sized plastic cards, key fobs and even self-adhesive labels.

While you are setting up your Raspberry Pi to use this kit you will need an Internet connection to your Raspberry Pi.

## CONNECTING THE CARD READER

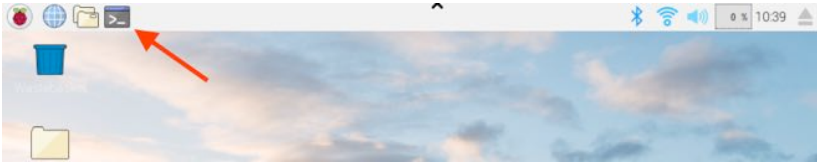
All the projects in this kit need the RFID Reader to be connected to the GPIO connector of your Raspberry Pi using the jumper wires provided. Place the Raspberry Leaf over the GPIO pins so that you can see which pin is which and then connect the wires as follows:

Lead Color	Smartcard Reader Pin	Raspberry Pi Pin
Orange	SDA	8
Yellow	SCK	SCK 11
White	MOSI	MOSI 10
Green	MISO	MISO 9
IRQ Not connected		
Blue	GND	GND
Gray	RST	25
Red	3.3V	3.3V

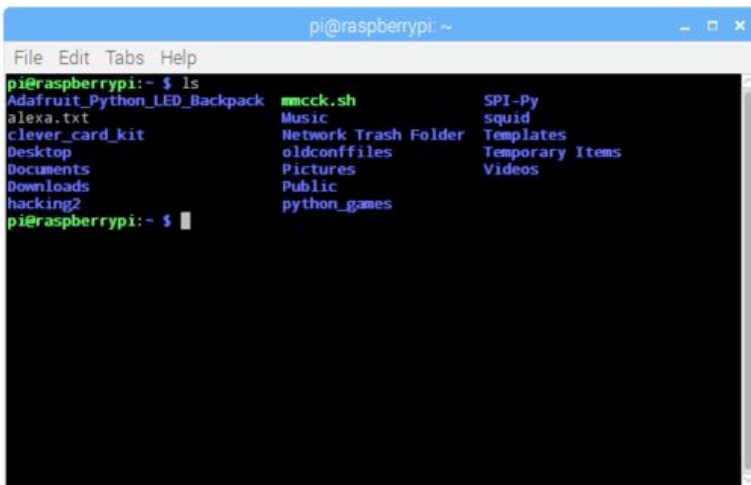
# USING THE TERMINAL

While you are setting up your Raspberry Pi and when running the programs that this kit uses you will need to run Linux commands in the LX Terminal.

To start an LX Terminal window, click on the terminal icon indicated by the arrow below.



This will open an LX Terminal like the one shown below.



When the LX Terminal is ready for you to enter a command, it will prompt you with the \$ character. For example, type the command 'ls'. This will show you a list of files in your 'current directory'.

In the instructions in this booklet, you will see the \$ symbol with the commands that you are asked to type. This is just as a reminder that you are using the LX Terminal, you do not actually need to type the \$ again.

# SETTING UP YOUR RASPBERRY PI

Before you can use the reader, you need to configure your Raspberry Pi and install some software. Your Raspberry Pi must be connected to the Internet for this.

## Step 1. Update your Raspberry Pi

Make sure that your Raspberry Pi is up to date by running the following commands in the LX Terminal:

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

If you get error messages during the upgrade, **restart your Raspberry Pi and run both commands again.**

If the LX Terminal asks “Do you want to continue? [y/n]” press “y” to continue.

Note that these commands could take a considerable time to complete if your Raspberry Pi has a lot to update.

## Step 2. Configure your Raspberry Pi

The RFID (Radio Frequency ID) hardware uses an interface of the Raspberry Pi called SPI (Serial Programming Interface) and by default SPI is not enabled, so you will need to enable it using the Raspberry Pi Configuration tool that you will find on the start menu under the Preferences section.

When you have turned SPI on and updated, reboot using:

```
$ sudo reboot
```

## Step 3. Install the Software

Start by opening an LX Terminal on your Raspberry Pi and issue the following commands to fetch an installation script to install all the software that you will need for the projects in this book.

```
$ wget http://monkmakes.com/downloads/mmckk.sh  
$ chmod 511 mmckk.sh  
$ ./mmckk.sh
```

This will take a while and from time to time the script will prompt you to press Y to confirm installation of one of the software components. If you have any problems, or error messages please see the Troubleshooting section at the end of this booklet.

You are now ready to get on and build your first project.

## PROJECT 1. READING FROM A CARD

This first project will show that everything is working OK with the card reader. When you hold a tag next to the reader, it will display the unique ID of the card.

Run the program called 01\_read.py from the LX Terminal using the commands:

```
$ cd /home/pi/clever_card_kit  
$ python3 01_read.py
```

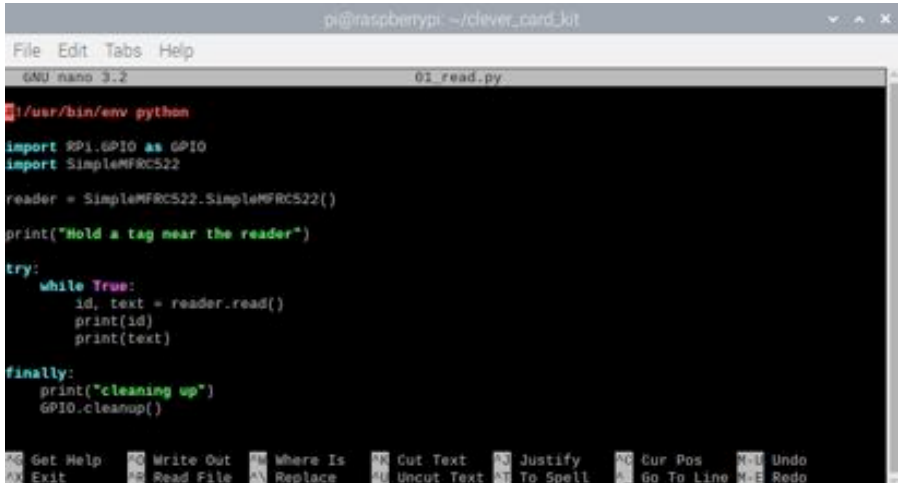
When you want to exit the program, press CTRL-c

The first 'cd' command sets the working directory. You will only need to do this once, the first time you run one of the programs, or after you restart your Raspberry Pi or change directory.

## The Code for Project 1

If you want to take a look at the code for Project 1 open it in the nano editor using the command:

```
$ nano 01_read.py
```



```
pi@raspberrypi: ~/clever_card_kit
File Edit Tabs Help
GNU nano 3.2 01_read.py
! /usr/bin/env python
import RPi.GPIO as GPIO
import SimpleMFRC522

reader = SimpleMFRC522.SimpleMFRC522()

print("Hold a tag near the reader")

try:
    while True:
        id, text = reader.read()
        print(id)
        print(text)
finally:
    print("cleaning up")
    GPIO.cleanup()
```

The *import* commands specify the library files that the code uses.

The *reader* variable contains a reference to SimpleMFRC522 a Python class that handles all communication with the reader hardware.

The try/finally block catches CTRL-c when you want to exit and leaves the GPIO pins in a safe state.

The real work happens on the line in bold. This waits for a tag to be read and then returns the contents of the tag as a pair of values:

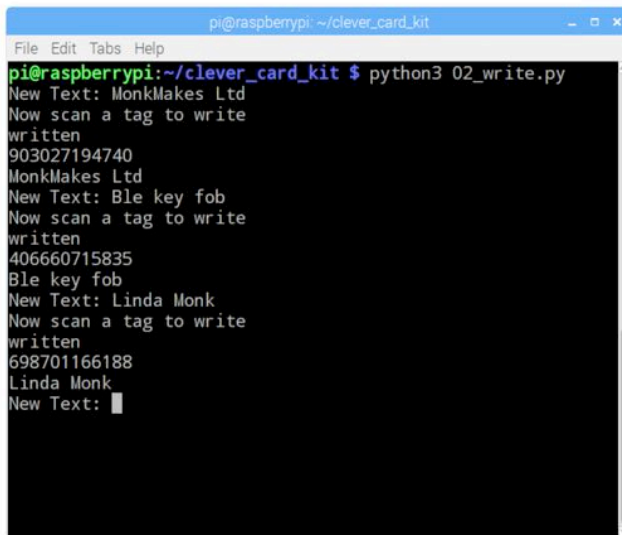
- **id** – the unique ID allocated to the tag when its made
- **text** – a custom field of 48 characters (technically sector 8 of the tags memory)

Use CTRL-x to leave the nano text editor. If you have made changes to the code then nano will prompt you to save the changes by pressing “y” or discard changes by pressing “n”. After pressing y, the text editor will let you rename the file. Simply press enter to keep the same name as before.

## PROJECT 2. WRITING TO A CARD

All RFID cards like these Clever Cards have an identifying number that is written onto them and cannot be changed. However, the cards used in this kit have 1 kilobyte of memory that you can read from or write to. Access to the memory is by sector and each sector has 48 usable bytes so can store 48 characters of text in a sector. By default, the SimpleMFRC522 class that we use to access the tag just gives us access to sector 8 allowing us to read and write short messages there. Run the program called 02\_write.py from the LX Terminal using the commands:

```
$ python3 02_write.py
```



```
pi@raspberrypi: ~/clever_card_kit
File Edit Tabs Help
pi@raspberrypi:~/clever_card_kit $ python3 02_write.py
New Text: MonkMakes Ltd
Now scan a tag to write
written
903027194740
MonkMakes Ltd
New Text: Ble key fob
Now scan a tag to write
written
406660715835
Ble key fob
New Text: Linda Monk
Now scan a tag to write
written
698701166188
Linda Monk
New Text: █
```

Enter the text that you want to write to the tag (up to 48 characters), press Enter and then hold a tag to the reader. You will get a confirmation message saying that the tag was written, the tag ID and new text. Repeat for as many tags as you like and use CTRL-c to quit the program. You can then check the newly written tags by running 01\_read.py again.

### The Code for Project 2

The code is very similar to the read.py example, except that it also prompts you to enter the new text that you want to write onto the tag.



```

import RPi.GPIO as GPIO
import SimpleMFRC522

reader = SimpleMFRC522.SimpleMFRC522()

try:
    while True:
        text = raw_input('New Text: ')
        print("Now scan a tag to write")
        id, text = reader.write(text)

        print("written")
        print(id)
        print(text)
finally:
    print("cleaning up")
    GPIO.cleanup()

```

The function *write* is supplied with the text (maximum 16 characters) to write onto the card. The function then reads the data back from the card to check that the write worked and these values are assigned to *id* and *text* in the same way as when using the *read* function.

The read and write commands cover 90% of what you want to do with an RFID tag, however, the SimpleMFRC522 library has a number of other options. You will find documentation for the library here on Github: [https://github.com/simonmonk/clever\\_card\\_kit](https://github.com/simonmonk/clever_card_kit)

## PROJECT 3. DOOR LOCK

This project won't actually lock doors for you, for that you would need an electro-mechanical door latch and driver electronics. But it will allow you to setup tags to turn the Squid LED green or red depending on whether the tag was Granted or Denied.



To use the project, press the button to cycle round three modes indicated by the LED color:

- Green – listening – waiting for you to scan a tag. If it's a valid tag the LED will flash for 10 seconds during which the door (if it existed) would be unlocked.
- Cyan (light blue/green) – Granting – waiting for you to scan a tag so that it can be added to the list of valid tags for door access.
- Purple – Revoking – waiting for a tag to be scanned and then removed from the list of allowed tags.

The valid tags are stored in the file `allowed_tags.pickle` and you can also add and remove tag numbers here by editing the file.

### Step 1. Attach the Raspberry Squid LED

This project uses a Raspberry Squid RGB LED to indicate the status of the project. Green for door unlocked red for access denied etc. The first step is to attach the Squid LED that's included in your kit to your Raspberry Pi.

Attach the four color coded leads of the Squid LED to your Raspberry Pi as follows. The connectors are all in a row.

Lead Color	Raspberry Pi Pin
Red	18
Black	GND (below 18)
Green	23
Blue	24

You can check that the Squid LED is working correctly by running the program `test_led.py` using the command:

```
$ python3 test_led.py
```

This will make the LED shine red, then green, then blue, then white half brightness and then white full brightness.

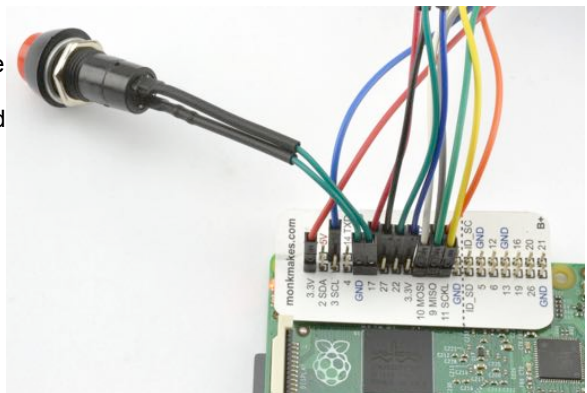
The program `test_led.py` uses the Squid library to set the color of the LED. You can find out more about this library on Github here:

<https://github.com/simonmonk/squid>



## Step 2. Attach the Raspberry Squid Button

The project also uses a Squid button to switch between modes. Attach the button with one lead going to pin 17 and the other lead going to the GND connection next to pin 17. It does not matter which button lead goes to which pin.



## Step 3. Run the Program

The program for this project is called `03_lock.py`. Run it using the command:

```
$ python3 03_lock.py
```

Try pressing the button to switch between the modes and add and remove some tags.

## The Code for Project 3

The code for project 3 is a lot more complicated than code we have met so far. Rather than list it in full here, open it in the editor using:

```
$ nano 03_lock.py
```

.. and then follow the discussion here:

The key to this type of program, where you switch between modes is to use a variable (in this case called *mode*) and set a value in the mode variable so that the program can know which mode it is currently in. In this program, the mode will always be one of the following values: LISTEN, GRANT or REVOKE.

The main loop of the program checks to see what mode the program is in and then calls the appropriate handler function.

```
try:
    while True:
        if mode == "LISTEN":
            handle_listen_mode()
        elif mode == "GRANT":
            handle_grant_mode()
        elif mode == "REVOKE":
            handle_revoke_mode()
finally:
    print("cleaning up")
    GPIO.cleanup()
```

The handler for LISTEN mode is as follows:

```
def handle_listen_mode():
    global mode, allowed_tags
    led.set_color(GREEN)
    id = reader.read_id_no_block()
    if id:
        if id in allowed_tags:
            unlock_door()
        else:
            print("Unknown Tag")
```

```

        flash(RED, 5, 0.1)
    if button.is_pressed():
        print("pressed")
        mode = 'GRANT'

```

The function requires access to two global variables, *mode* and *allowed\_tags*. The variable *allowed\_tags* is a list of the ids of those tags that have permission to open the door.

First the LED color is set to green and then *read\_id\_no\_block* is called. The function *read\_id\_no\_block* is like *read\_id* except that it does not wait for a tag to be presented. But if there is not tag within reading range the function just returns *None*. This is necessary because if you used *read\_id*, the program would be 'blocked' waiting for a tag and so pressing the button would not allow you to change mode.

Once a tag is presented to the reader (*if id:*) then the tag's id is compared with the list of *allowed\_tags* and if it is present the function *unlock\_door* is called.

If on the other hand the id is not in the list, then the LED is flashed red to indicate that the door will not be opened.

The final part of *handle\_listen\_mode* checks for a press of the button, and if there is one, it sets the mode to GRANT.

The other two handlers work in a similar manner, first checking for the presence of a tag, carrying out any necessary actions and then checking for a button press.

The allowed tags are stored in a file that is read when the program starts:

```

def load_tags():
    global allowed_tags
    try:
        with open('allowed_tags.pickle', 'rb') as handle:
            allowed_tags = pickle.load(handle)
            print("Loaded Tags")
            print(allowed_tags)
    except:
        pass

```

.. and then written back to the file every time a new tag is added to the *allowed\_tags* list.

```

def save_tags():
    global allowed_tags
    print("Saving Tags")
    print(allowed_tags)
    with open('allowed_tags.pickle', 'wb') as handle:
        pickle.dump(allowed_tags, handle)

```

## PROJECT 4. THING IDENTIFIER

This project allows your Raspberry Pi to recognize things and speak what they are. This is one project that could actually be of use to the visually impaired. It does of course require you to stick tags on all the things you want to be identified. For example, you could stick a card inside the cover of a book.



You can find a short video of this project in action here:

[https://youtu.be/\\_exIPFthUJw](https://youtu.be/_exIPFthUJw)

### Step 1. Prepare for Sound

To make this project, you will need to be able to hear the sounds that your Raspberry Pi makes. If your Pi is attached to a TV through an HDMI cable, then the sound will automatically be redirected to the TV, just turn up the volume.

Alternatively, you can use an external speaker or headphones attached to the audio socket on the Raspberry Pi.

You do not need the Squid LED or button to be attached for this project, so if you have just made Project 3, you can now take these off if you want to.

### Step 2. Program Some Tags

Next you need to program your tags with short messages (48 characters maximum) by running the tag writing program from project 2 (02\_write.py).

Run the program and set the text on a few tags as shown below:

### Step 3. Run the Identifier Program

The program for this project is called `04_identifier.py`. Run it using the command:

```
$ python3 04_identifier.py
```

You should here a message asking you to hold something up to the reader for identification. When you put one of the tags you just wrote next to the reader, it should read back the message using text to voice software.

## The Code for Project 4

The code for this project is pretty straightforward as the text just has to be read from the card and then passed to the text to speech software (festival).

```
import RPi.GPIO as GPIO
import SimpleMFRC522
import os

reader = SimpleMFRC522.SimpleMFRC522()

print("Hold a thing next to the reader")

def speak(message):
    try:
        os.system('echo ' + message + ' | festival --tts')
    except ValueError:
        print("No message on card! Please add a message to
the card before reading.")

try:
    while True:
        speak('Hold a thing next to the reader')
        id, text = reader.read()
        print(text)
        speak(text)

finally:
    print("cleaning up")
    GPIO.cleanup()
```

The function *speak* will just send any text passed to it to the text to speech software by calling the system command.



# PROJECT 5. LAUNCHER

As you saw in the code for project 4, it is possible to run system commands from within your Python program. In this project, rather than the commands being spoken by text to speech software, they are actually executed.

This means that anything that you can do from your Raspberry Pi's command line, you can do by holding a tag in front of the reader. This means that you could run system commands to do things like:

- Open a browser on a particular page
- Shut down your Raspberry Pi
- Change to a particular directory
- Play a sound file

You can find a short video of this project in action here:

<https://youtu.be/SdH3qKZH2G8>

You don't need anything connected to your Raspberry Pi except the card reader.

## Step 1. Program Some Tags

This project does not store the command on the tag itself, allowing you to have commands as long as you like. Instead the program just uses the tag's ID and then looks up the command from a Python dictionary, that is also saved to file so that the commands are remembered. This means that a separate program is required to setup the tags.

Run the program `05_launcher_setup.py` using the command:

```
$ python 05_launcher_setup.py
```

Enter the command: `chromium-browser` <http://google.com>

to open the Chromium browser on Google's home page, or use any other URL you like.

Present a tag to the reader for writing and then enter the next command for your next card, and so on for as many cards as you want to program.

Some useful command examples are listed below:

Command	Description
chromium-browser http://google.com	Open a browser window on a URL
sudo shutdown	Shut-down your Raspberry Pi
cd /home/pi/clever_card_kit	Change working directory
omxplayer example.mp3	Play a sound file

## Step 2. Run the Main Program

With your tags programmed, you can now just run the main program and then use your cards to trigger actions.

```
$ python3 05_launcher.py
```

## The Code for Project 5

The code for this project is similar to Project 3 in its use of a file to store tag data. However, in this case it uses a Python dictionary to associate a tag ID with a command.

```
import RPi.GPIO as GPIO
import SimpleMFRC522
import os, pickle, time
import pickle

reader = SimpleMFRC522.SimpleMFRC522()

tags = {}

def load_tags():
    global tags
    try:
        with open('command_tags.pickle', 'rb') as handle:
            tags = pickle.load(handle)
        print("Loaded Tags")
        print(tags)
    except:
        pass

def save_tags():
    global tags
    print("Saving Tags")
    print(tags)
    with open('command_tags.pickle', 'wb') as handle:
        pickle.dump(tags, handle)

load_tags()
```

```
try:
    while True:
        print("Hold a tag next to the reader")
        id = reader.read_id()
        command = tags.get(id, None)
        if command:
            print(command)
            os.system(command)
            time.sleep(2) # prevent too many repeats
finally:
    print("cleaning up")
    GPIO.cleanup()
```

## PROJECT 6. MONOPOLY ATM

You can use your Clever Card Kit with the game of Monopoly, so that when you need some cash, you can present your card to the Banker (equipped with a Raspberry Pi) and withdraw some funds. To be fair there is already a version of Monopoly with electronic money, but there is more fun in making your own.



This project is the first to use a graphical user interface. Run the program from the command line using the command:

```
$ python3 06_atm.py
```

.. and the window shown above will open.

Each player should have their own card and you can write your names on them so that they don't get mixed up.

To setup a card, enter a name and initial balance for the player and then click on the SAVE button. Present a card to the reader and the name and balance will be

written onto the card. Repeat this process for the other players.

When a player wants to change some of their card credit for hard cash, first click SCAN then hold the card next to the reader (leave it resting on it if you like). Enter a negative amount in the Amount field and click SAVE. To deposit cash into the Monopoly bank, do the same thing, but enter a positive number in the Amount field.

You could use adapt this project for say credits for treats.

## The Code for Project 6

This project uses the GUIZero (<https://lawsie.github.io/quizero/>) project to make a simple graphical user interface for the project.

```
from guizero import *
import RPi.GPIO as GPIO
import SimpleMFC522

reader = SimpleMFC522.SimpleMFC522()

app = App(title="Money Box", layout="grid", width=350, height=170)

def read_tag():
    id, text = reader.read()
    if id:
        values = text.split("/")
        if len(values) == 2:
            name_field.value = values[0]
            balance_field.value = values[1]

def write_tag():
    name = name_field.value
    balance = float(balance_field.value)
    adjustment = float(adjustment_field.value)
    balance += adjustment

    text = name + "/" + str(balance)
    reader.write(text)
    balance_field.value = str(balance)
    adjustment_field.value = "0"

PushButton(app, text="SCAN", command=read_tag, align="left", grid=[0,0])

Text(app, text="Name", align="left", grid=[0,1])
name_field = TextBox(app, text="-", align="left", width=20, grid=[1,1])
Text(app, text="Balance", align="left", grid=[0,2])
balance_field = TextBox(app, text="", align="left", width=10, grid=[1,2])
Text(app, text="Amount to + or -", align="left", grid=[0,3])
adjustment_field = TextBox(app, text="0", align="left", width=10, grid=[1,3])

PushButton(app, text="SAVE", command=write_tag, align="left", grid=[0,4])

try:
    app.display()
finally:
    print("cleaning up")
    GPIO.cleanup()
```

The function `read_tag` reads the text from a tag when it is presented and then splits

the string it reads around the '/' character that is used to separate the players name from their balance. Each of these values is then used to set the text of their two fields on the user interface.

The function *write\_tag* does the reverse, fetching the name, balance and adjustment values from the fields before adding the adjustment to the balance, writing the data back to the tag and then updating the fields.

The rest of the code defines the fields and buttons to be displayed. For example, the first PushButton with a text of SCAN will call the command *read\_tag* when it is pressed.

The buttons and fields are laid out in a grid, with 0,0 being the top left of the window.

# MAKE YOUR OWN PROJECTS

If, these projects will have made you want to create some projects of your own, then here is the documentation for the SimpleMFRC522 Python class that the projects in this kit all use.

## Import and Create an Instance

```
import SimpleMFRC522
reader = SimpleMFRC522.SimpleMFRC522()
```

### read()

This function takes no parameters and waits for a card to be presented to the reader. It then returns two values: id (the card id) and text (the 48 characters stored in sector 8 of the card)

```
id, text = reader.read()
print(id)
print(text)
```

### read\_id()

This function is like 'read' but only returns the id, ignoring any text stored on the card.

```
id = reader.read_id()
print(id)
```

### write(text)

This function takes a text string and just waits for a card to be presented to the reader. It then writes the first 48 characters of text (padding if needed) to sector 8 of the card. It then returns the id and text that were just written.

```
text = raw_input('New Text: ')
print("Now scan a tag to write")
id, text = reader.write(text)
print("written")
print(id)
print(text)
```

### read\_no\_block()

This function takes no parameters and if no card is being presented to the reader immediately returns two values None and None, indicating that no id or text were read. If a card is on the reader, then it returns two values: id (the card id) and text (the 48 characters stored in sector 8 of the card)

The code example below would normally be in a loop.

```
id, text = reader.read_no_block()
if id:
    print(id)
```

### **read\_id\_no\_block()**

This function is like 'read\_no\_block' but only returns the id, ignoring any text stored on the card. If there is no card present, None is returned.

```
id = reader.read_id_no_block()
print(id)
```

### **write\_no\_block(text)**

This function takes a text string and if no card is being presented to the reader immediately returns None, None. If a card is on the reader, then it writes the text string to sector 8 of the card and returns two values: id (the card id) and text (the 48 characters stored in sector 8 of the card)

The code example below would normally be in a loop.

```
id, text = reader.write_no_block('hello')
if id:
    print(id)
    print(text)
```



# TROUBLESHOOTING

**Problem:** Error messages during installation of the software.

**Solution:** Make sure that your system is up-to-date as described at the top of page 5. If that doesn't work, try a fresh install of Raspbian using NOOBS (<https://www.raspberrypi.org/documentation/installation/noobs.md>)

**Problem:** Error messages when running the programs.

**Solution:** Make sure that SPI is enabled (see Configure your Raspberry Pi on page 5). Check for error messages during software installation. Check the connections between the reader and your Raspberry Pi (see page 3)

**Problem:** No error messages, but my RFID card doesn't scan.

**Solution:** Try other cards – there may be something wrong with the card, or it might be the wrong type. Also, if this starts happening after installing other software, then this may be a result of that software using the SPI interface. Try a fresh install of Raspbian.

## Support

For the most up-to-date help on this kit, see <http://monkmakes.com/cck> or contact MonkMakes support at [support@monkmakes.com](mailto:support@monkmakes.com)

# NOTES



## ALSO AVAILABLE FROM MONKMAKES

As well as this kit, MonkMakes makes all sorts of kits and gadgets to help with your projects.

Find out more, as well as where to buy here: <https://monkmakes.com> you can also follow MonkMakes on Twitter @monkmakes.



From left to right: Electronics Starter Kit for micro:bit, Electronics Starter Kit for Raspberry Pi and Servo Kit for Raspberry Pi.