1 Hardware Description

The following three programs are the python program which is used for controlling an eight times relays expansion board. The expansion board can be added on top of a raspberry pi 4 model B through the 40 Pin Header. Furthermore, there are 8 relays and ten phototransistor optocouplers mounted on the PCB. Each relay and optocoupler channel have an indication LED on the PCB that can be controlled or read out by the python program through the specific GPIO pins. Moreover, the output of the optocoupler is connected to pull-up resistors to ensure a well-defined logical level at the output pins under all conditions.

For the connection and wiring of the prototype are made by myself with the help of jumper wires and crimping tool. From this I learned how to strip and crimp the wire so that the cables can be connected to the female pins.

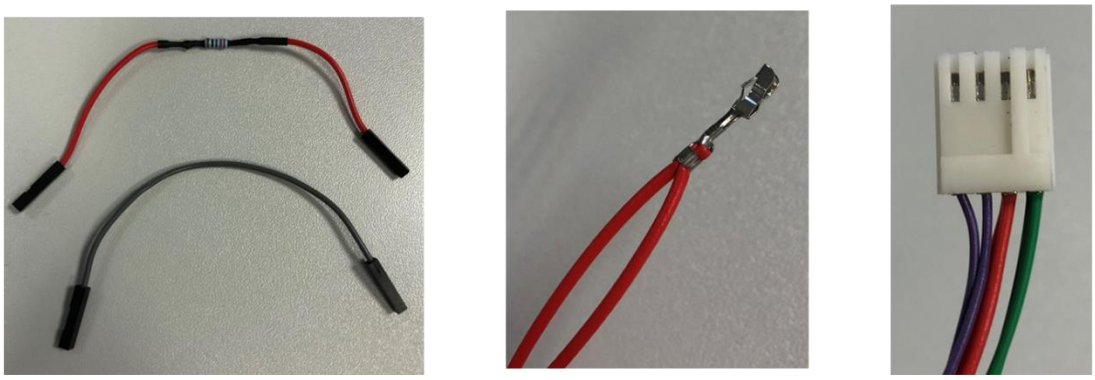


Figure 1 Hardware Connection and Wiring

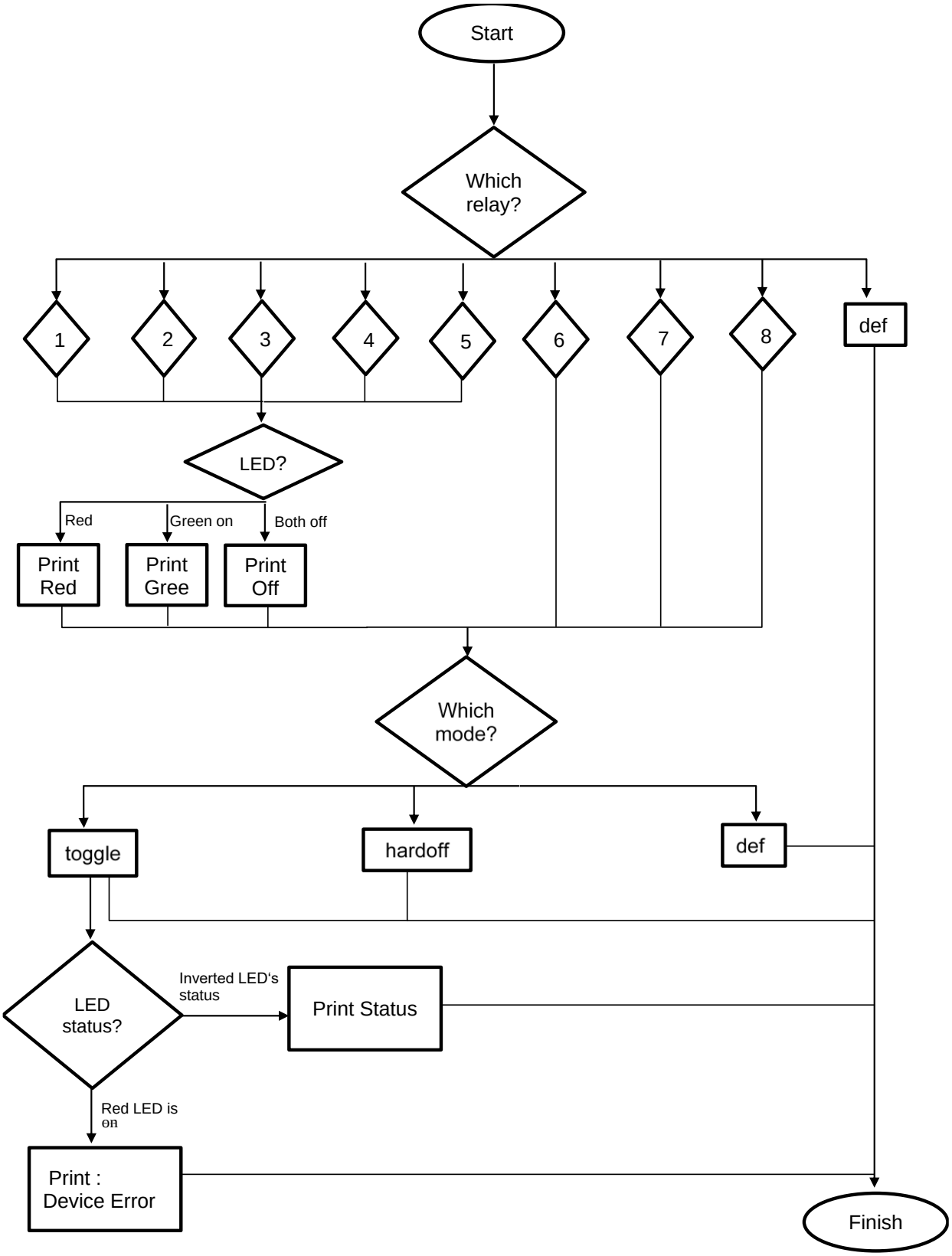
2 Software Description

The expansion board is controlled by a Python code, which runs on raspberry pi. The following programs are aimed to enable the user to access and read the status of devices remotely. These three programs can be run simultaneously through three different active terminals. Two programming codes are written using the module argparse, in order that the users can control the expansion board conveniently.

- **Program 1: power_relay.py**

  The first programming code written under the file name power_relay.py, facilitates the user to toggle and hard off the relay that is connected to the power button of an external device using argparse module. The argparse module is a module to write user-friendly command-line interfaces. The program defines the arguments it requires and ascertains them to solve the program. Furthermore, the argparse module automatically generates help, usage messages, and issues errors when invalid arguments are given.

- **Code Diagram**



- **Script Flow**

```python
class Power:
    '''A class to represent a relay and the action applied'''

    def __init__(self, num: int):
        '''initializing relays'''
        device_gpio_map = [
            [23, 17, 27],
            [24, 22, 10],
            [25, 9, 11],
            [8, 0, 5],
            [7, 6, 13],
            [1],
            [12],
            [16]
        ]

        if num < 1 or num > 9:
            raise ValueError("unknown relay number (1 to 8 allowed)")

        self.status = "none"

        index = num - 1

        self.device_has_led = len(device_gpio_map[index]) > 1
        self.power_button = OutputDevice(device_gpio_map[index][0])

        if self.device_has_led:
            self.green_led = DigitalInputDevice(device_gpio_map[index][1])
            self.red_led = DigitalInputDevice(device_gpio_map[index][2])
        else:
            print(f"Device {num} has no LED support, no status checks available")

    def main(self, act):
        '''main process'''
        self.status_led()

        self.mode_option(act)

        self.waiting_led(self.status)
        self.status_led()
```

```python
        self.waiting_led(self.status)
        self.status_led()

    def status_led(self):
        '''check the led status'''
        if not self.device_has_led:
            return

        if self.green_led.value and not self.red_led.value:
            print("status led : red")
            self.status = "red"
        elif self.red_led.value and not self.green_led.value:
            print("status led : green")
            self.status = "green"
        else:
            print("status led : off")
            self.status = "off"

    def mode_option(self, action):
        '''action applied to the relay'''
        if action == "toggle":
            self.toggle_power_btn()
        elif action == "hardoff":
            self.hardoff()
        elif action == "none":
            return 0
        else:
            raise ValueError("unknown action requested")

    def toggle_power_btn(self):
        '''toggle relay'''
        print("toggling power button")
        self.power_button.on()
        sleep(0.5)
        self.power_button.off()
        sleep(1)

    def hardoff(self):
        '''hardoff relay'''
        print("longpress power button (this block 6 secs)")
```

Figure 2 First Project's Python Code (Part 1)

To simplify the Python program, a code template Class Power is applied. The Class includes all the functions that are needed for the program, namely, to determine the activated relay, to toggle or hard-off the relay, and to check the LED status. The instance of the class is represented by using the "Self" parameter. The "Self" parameter is allowing access to the attributes and methods of the class in Python. It binds the attributes with the given arguments. Inside the Power Class, the program begins to initialize the GPIO Pins from relay number one to eight into arrays called 'device_gpio_map'. The first until fifth relay have an array length of 3, because each of these 5 relays connected to a green and a red LED. The rest of the relay don't have any LED connected, so the length of the array or the declared GPIO Pins is only one. Moreover, the program will raise an error if the given number is below 1 or above 9, otherwise it will initiate a new variable 'index', indicating the index of the array 'device_gpio_map' that label the selected relay. Afterwards, the program will check the length of the selected array, so the program knows whether the selected relay has any LED that its status needs to be checked and controlled.

This LED indication will be then save in a Boolean variable named ‘device_has_led’.

The first function that's written is function main(). This function directs the program to the functions that needs to be done. Next function is function status_led(), that’s written to check the status’ of the led. However, as soon as the program enters the function, it needs to be checked whether the selected relay has any LEDThis function identifies the given arguments so thatthe program will check the variable ‘device_has_led’ with if statement. If the variable is true then the program checks which LED is currently on or if both LEDs are off. Third function if the function ‘mode_option’, which lets the program determine which action will be applied to the chosen relay and call the specific function. The actions are included toggle (turned on for 0,5 seconds then turned off), hard off (turned on for 6 seconds then turned off), and default mode (nothing needs to be done).

```
    def hardoff(self):
        '''hardoff relay'''
        print("longpress power button (this block 6 secs)")
        self.power_button.on()
        sleep(6)
        self.power_button.off()

    def waiting_led(self, state):
        '''waiting led until specific order occured'''
        if not self.device_has_led:
            return

        if state == "off":
            print("device is turning on")
            print("waiting for the green LED...")
            self.green_led.wait_for_inactive(
                timeout=120) and self.red_led.wait_for_active(timeout=120)
            if self.red_led.value and not self.green_led.value:
                print("device has been turned on")
            else:
                print("Timeout : DEVICE ERROR. Waited for 2 minutes")
        else:
            print("device is turning off")
            print("waiting for the green and red LED...")
            self.green_led.wait_for_active(
                timeout=None) and self.red_led.wait_for_active(timeout=None)
            print("device has been turned off")


if __name__ == '__main__':
    PARSER = argparse.ArgumentParser(
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    PARSER.add_argument("number", type=int, help="number of relay")
    PARSER.add_argument("action", default="none",
                        nargs="?", help="toggle, hardoff")

    ARGS = PARSER.parse_args()
    POWER = Power(ARGS.number)
    POWER.main(ARGS.action)
```

Figure 3 First Program's Python Code (Part 2)

The last function is waiting_led(). This function aims to put the program to sleep function and wait until the desired outcome signal is received. Lastly (if the selected relay has LED) the program will wait until the specific LED to be turned

on or off, depending on the order, and check their status again before it ends their cycle.

- **Application of the Program**

  To run the program in the terminal without any error, a few parameters need to be set. The following parameter should be given after the command "./power_relay.py":

Table 1 Parameter of Program 1

Parameter	Syntax	Intention
Help	-h	a string containing a brief description of the argument
	--help	
Relay	1 to 8 (only one number per session)	the relay option, which the action will be applied
	"none" (default)	
Action	toggle	the action that will be applied to the chosen relay
	hardoff	
	"none" (default)	

The help strings can include various formats and description depending on what is written in the program. The help value is requested by the user and with this argument, a description as follow will be displayed.

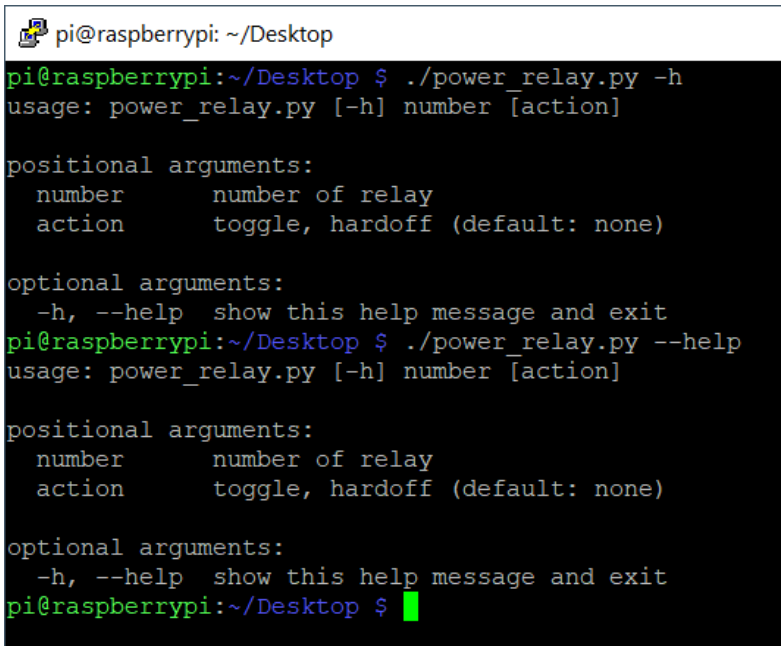


Figure 4 Help Parameter

Relay and action arguments must be given simultaneously and only one argument type is required for each argument. For example, the user would like to toggle the first relay. An argument "1 toggle" is required after the command "./power_relay.py". Additionally, the placement of the argument is important to note. The number of the desired relay must be the first argument and followed by the action argument.

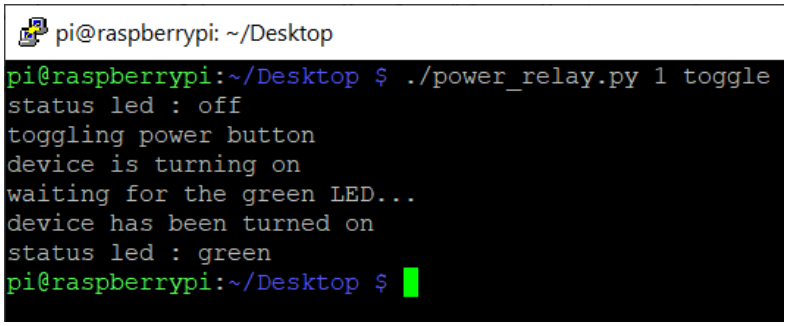
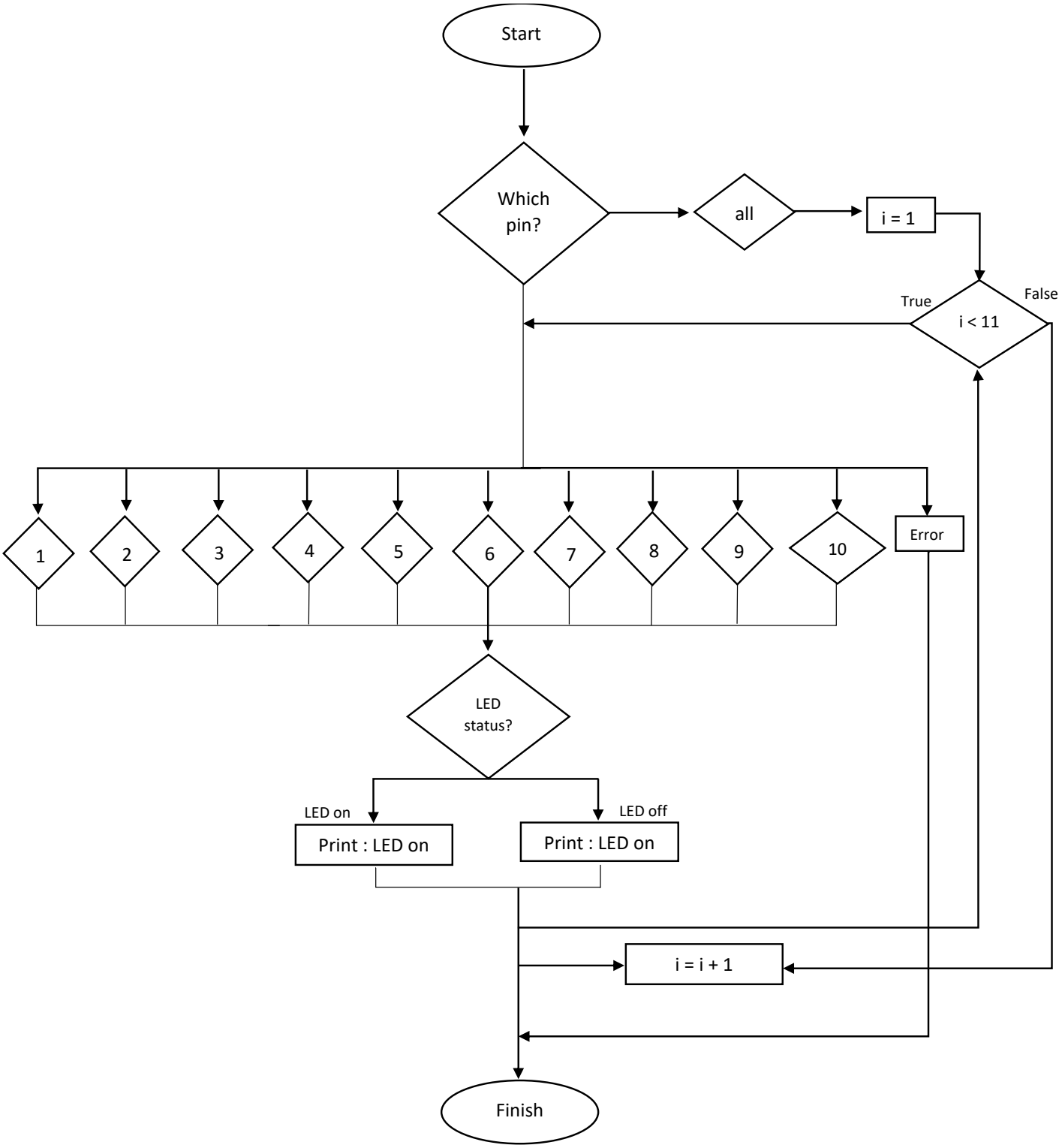


Figure 5 Relay and action parameter

- **Program 2: read_status.py**

  The second programming code written under the file name read_status.py, allows the user to read the status of ten LEDs using argparse module. These LEDs are connected to external devices.

- **Code Diagram**



- **Script Flow**

```python
class ReadStatus:
    '''A class represent the LED pins and check their status'''
    def __init__(self):
        '''initializing led pins'''
        self.pins = [
            Button(17),
            Button(27),
            Button(22),
            Button(10),
            Button(9),
            Button(11),
            Button(0),
            Button(5),
            Button(6),
            Button(13)
        ]

    def state(self, num):
        '''checking the status of a specific'''
        int_num = int(0 if num is None else num)
        if int_num < 1 or int_num > 10:
            raise ValueError("unknown pin number (1 to 10 allowed)")

        index = int_num - 1
        pin = self.pins[index]

        if pin.is_pressed:
            print(f"{int_num} is high")
        else:
            print(f"{int_num} is low")
```

```python
if __name__ == '__main__':
    PARSER = argparse.ArgumentParser(
        formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    PARSER.add_argument("number", type=int, nargs="?",
                        help="pin number from 1 to 10")
    PARSER.add_argument('--all', action='store_true',
                        help="print out all LEDs state")

    ARGS = PARSER.parse_args()
    READ_STATUS = ReadStatus()

    if ARGS.all:
        for i in range(1, 11):
            READ_STATUS.state(i)
    else:
        READ_STATUS.state(ARGS.number)
```

Figure 6 Second Program's Python Code

Generally, the second programming code is designed in a similar way to the first code with argparse module and class template. However, there is a particular difference in comparison to the first program. Only one argument is needed for this program and that is namely the number of the pin that is going to be read, from one to ten. Then the status of the selected pin will be read. The program also provides the option to read the status of all pins. If the given argument is lower than one or greater than 10 and not 'all', than the program will raise an error.

- **Application of the Program**

To run the program, an argument to select the pin is required. The following arguments are allowed to be written after the command "./read_status.py":

Table 2 Parameter of Program 2

Parameter	Syntax	Intention
Help	-h	a string containing a brief description of the argument
	--help	
Pin	1 to 10 (only one number per session)	the pin option, whose status is going to be read
	all (status of all pins)	

In Figure 33 is shown how the help parameter for the second program is described.

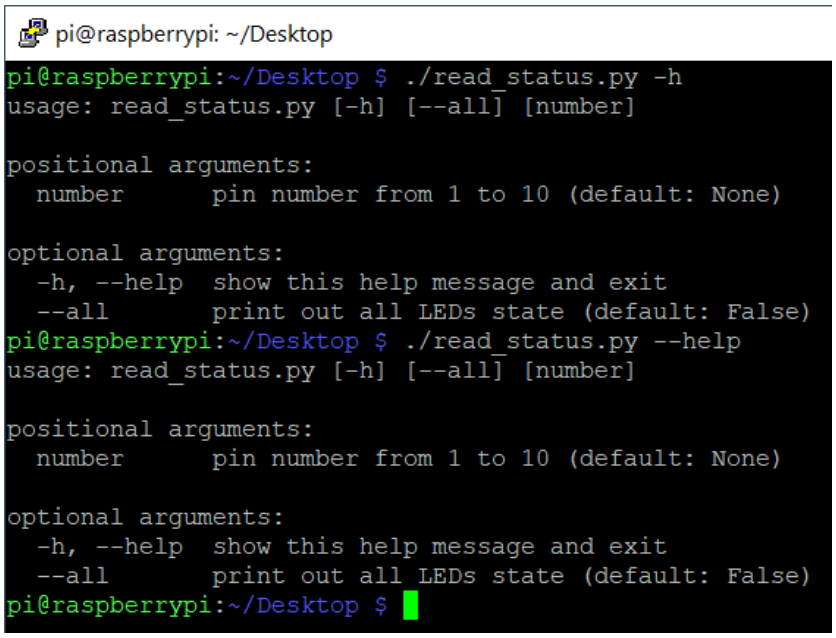


Figure 7 Help Parameter

In the following Figure is shown that the program is able to read the state of a pin or all pin

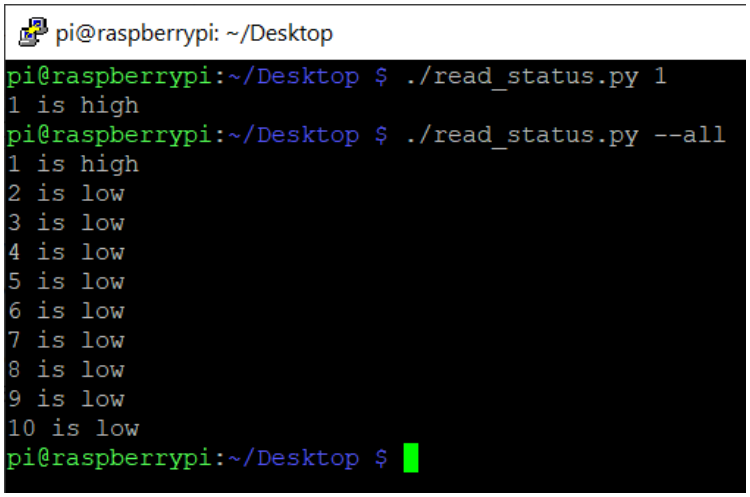
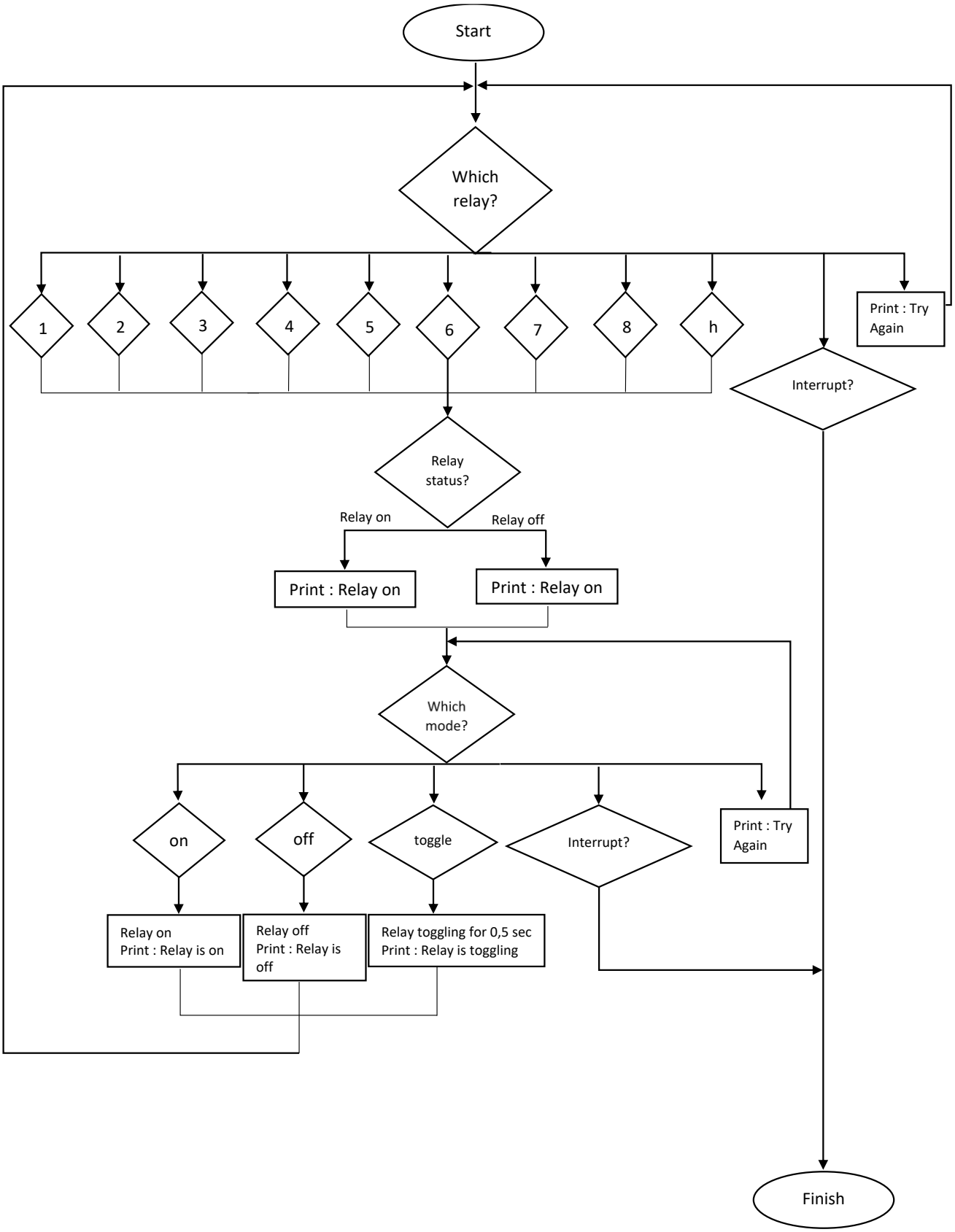


Figure 8 Reading State of given Pin Number

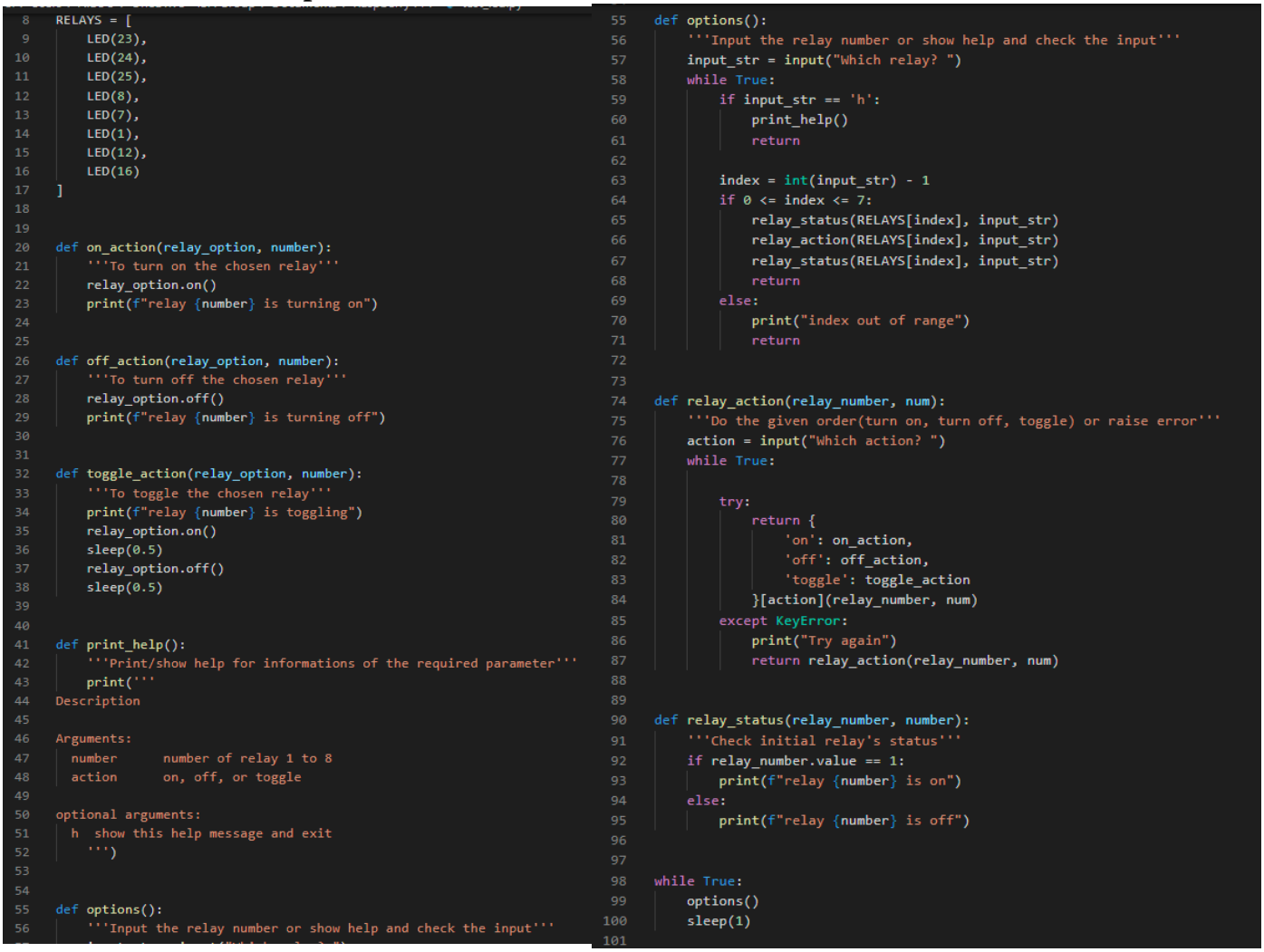
- **Program 3: test_led.py**

  One of the disadvantages of argparse is that the program is conducted once and can't be run in the background. This means that the relay can't be left on and another command is given simultaneously. This program code, that written under the file name test_led.py, has been written to overcome this problem. Without using argparse and deliver the argument manually allow the program to be left running in the background until the user decides to end the program. So, in this program, the user is enabled to turn on and turn off the relay as long as desired.

- **Code Diagram**



- **Script Flow**



```python
RELAYS = [
    LED(23),
    LED(24),
    LED(25),
    LED(8),
    LED(7),
    LED(1),
    LED(12),
    LED(16)
]


def on_action(relay_option, number):
    '''To turn on the chosen relay'''
    relay_option.on()
    print(f"relay {number} is turning on")


def off_action(relay_option, number):
    '''To turn off the chosen relay'''
    relay_option.off()
    print(f"relay {number} is turning off")


def toggle_action(relay_option, number):
    '''To toggle the chosen relay'''
    print(f"relay {number} is toggling")
    relay_option.on()
    sleep(0.5)
    relay_option.off()
    sleep(0.5)


def print_help():
    '''Print/show help for informations of the required parameter'''
    print('''
Description

Arguments:
  number      number of relay 1 to 8
  action      on, off, or toggle

optional arguments:
  h  show this help message and exit
    ''')


def options():
    '''Input the relay number or show help and check the input'''
    input_str = input("Which relay? ")
    while True:
        if input_str == 'h':
            print_help()
            return

        index = int(input_str) - 1
        if 0 <= index <= 7:
            relay_status(RELAYS[index], input_str)
            relay_action(RELAYS[index], input_str)
            relay_status(RELAYS[index], input_str)
            return
        else:
            print("index out of range")
            return


def relay_action(relay_number, num):
    '''Do the given order(turn on, turn off, toggle) or raise error'''
    action = input("Which action? ")
    while True:

        try:
            return {
                'on': on_action,
                'off': off_action,
                'toggle': toggle_action
            }[action](relay_number, num)
        except KeyError:
            print("Try again")
            return relay_action(relay_number, num)


def relay_status(relay_number, number):
    '''Check initial relay's status'''
    if relay_number.value == 1:
        print(f"relay {number} is on")
    else:
        print(f"relay {number} is off")


while True:
    options()
    sleep(1)
```

Figure 9 Third Program's Python Code

Two required parameters, namely the number of relays and the action, are each going to be requested one after the other. The first function option() redirects the program to the function ‘relay_status’ and ‘relay_action’ with the specific relay as the argument. The function ‘relay_status’ review the state of the relay at the moment and print out whether the relay on or off. Afterwards, the function ‘relay_action’ runs the order on the second input parameter. The actions that could be applied to the relay is turning relay on, turning relay off and toggle the relay for 0,5s.

Since the program is going to be run in the background as long as the user requires to, the program can only be closed when the program is interrupted. The interruption can be conducted through shortcut Ctrl + C or by closing the terminal window

- **Application of the Program**

In comparison to the other two programming codes, this program has a different way to give the parameter on the terminal. Instead of giving the arguments after the command "python3 test_led.py", the argument should be given after the program has been run and the question in the following Table has been shown.

Table 3 Parameter of Program 3

Question	Parameter	Syntax	Intention
Which relay?	Relay	1 to 8 (only one number per session)	the relay option, which the action will be applied
	Help	h	a string containing a brief description of the argument
Which status?	Action	on	the action that will be applied to the chosen relay
		off	
		toggle	

To run the program, the user needs to write the command "python3 test_led.py" without any arguments. Next, the program is going to respond with the questions shown in Table 3 and the corresponding argument must be given after these questions.

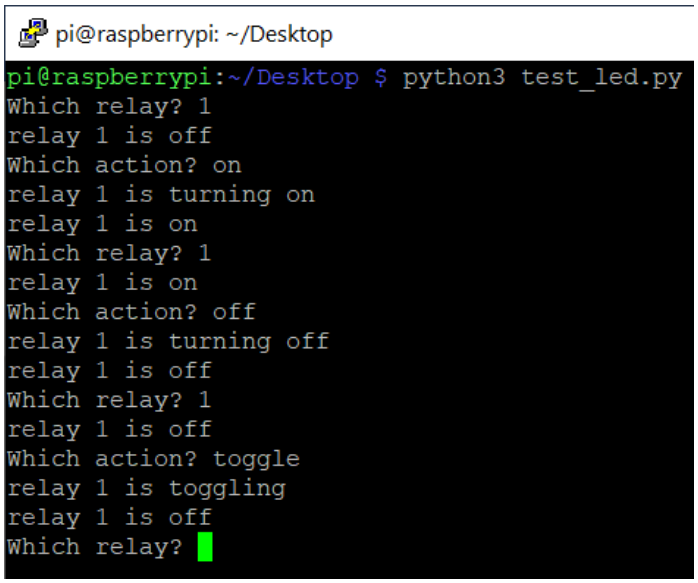


Figure 10 Parameter of the Program 3

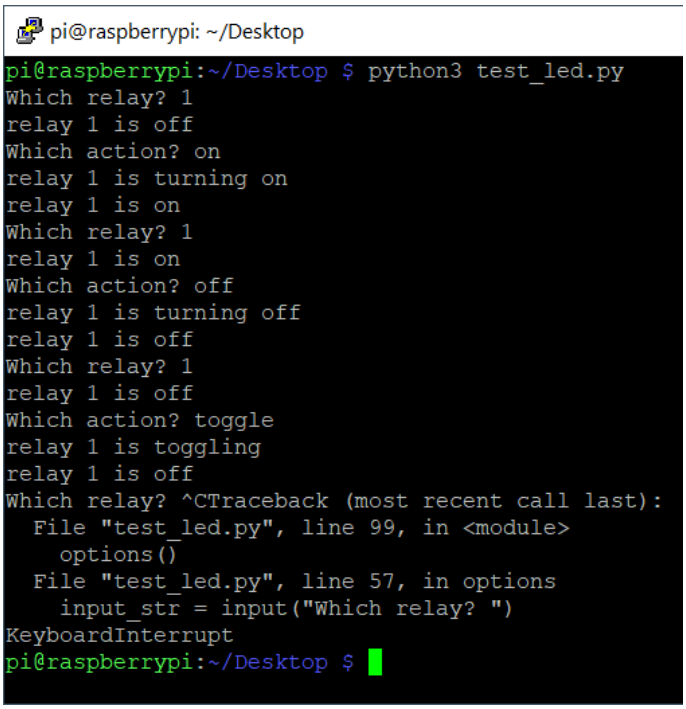


Figure 11 Interrupting The Program with Ctrl + C