



Samsung Innovation Campus

| Vạn vật kết nối - IoT

Together for Tomorrow!
Enabling People

Education for Future Generations

Chương 4.

Raspberry Pi với Python

Internet vạn vật kết nối – IoT

Mô tả chương học



Mục tiêu của chương học

- ✓ Thực hành lập trình cho mạch Raspberry Pi bằng Python thông qua dự án “Toy”.
- ✓ Tìm hiểu các nội dung cơ bản về thao tác trên thiết bị IoT qua ngôn ngữ Python.



Nội dung chương học

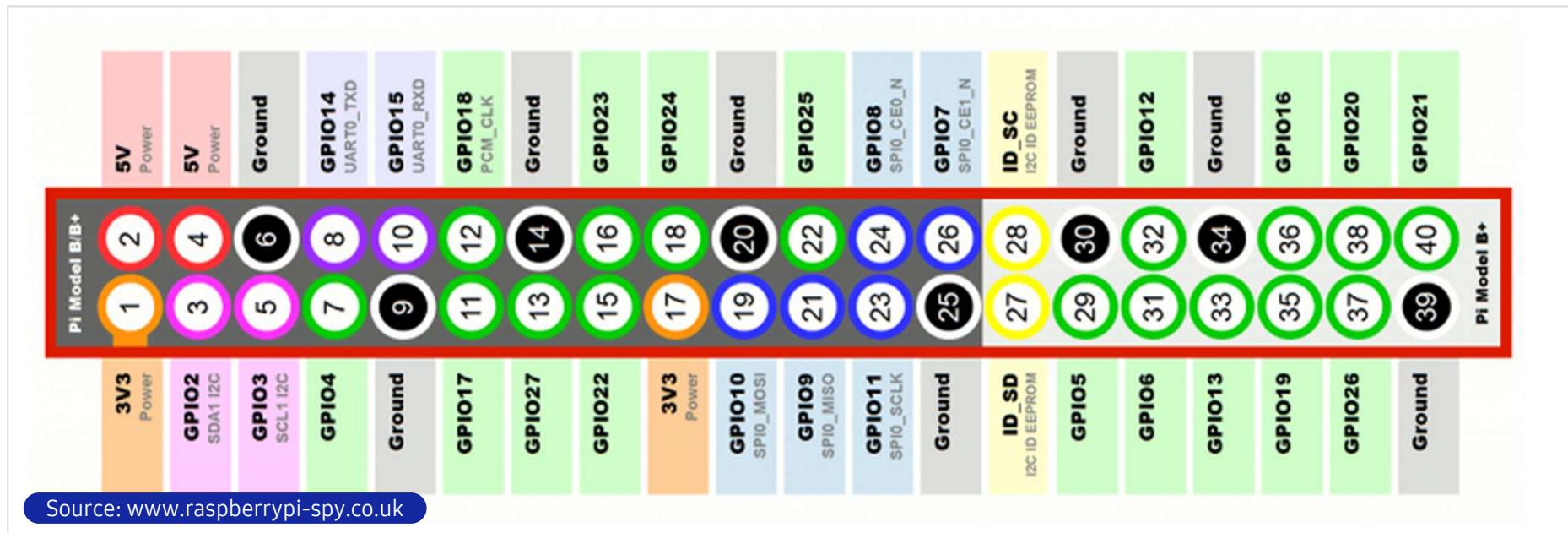
- ✓ Bài 1. Dự án Giám sát mức độ sử dụng CPU
- ✓ Bài 2. Dự án Kiểm soát dải giá trị đo của cảm biến
- ✓ Bài 3. Dự án Trực quan hóa dữ liệu nhiệt độ

Mở đầu

Thứ tự đánh số các chân xuất – nhập tín hiệu (GPIO)

| Nắm được thứ tự đánh số các chân GPIO trước khi bắt đầu với Dự án Toy

- ▶ Trong chương học này, chúng ta sẽ thông qua Dự án Toy để thực hành sử dụng Python nhằm lập trình cho mạch Raspberry Pi.
- ▶ Khởi tạo và quản lý thư mục có tên là Toy_project đặt tại đường dẫn /home/pi trên bộ nhớ Raspberry Pi để thực hiện và thao tác với dự án.



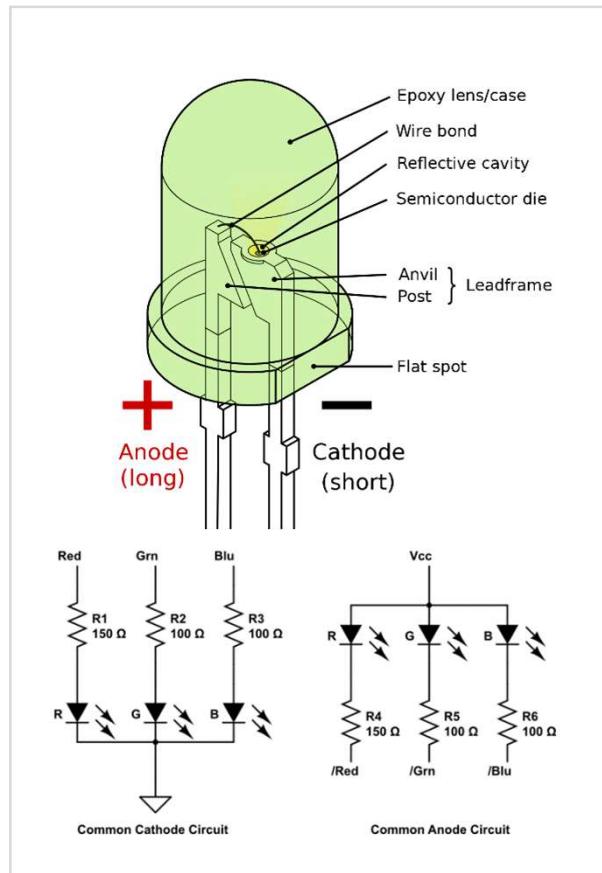
Bài 1.

Dự án Giám sát mức độ hoạt động của CPU

- | 1.1. Điều khiển LED
- | 1.2. Giám sát mạch Raspberry Pi
- | 1.3. File Text trong Python
- | 1.4. Dự án Toy

LED

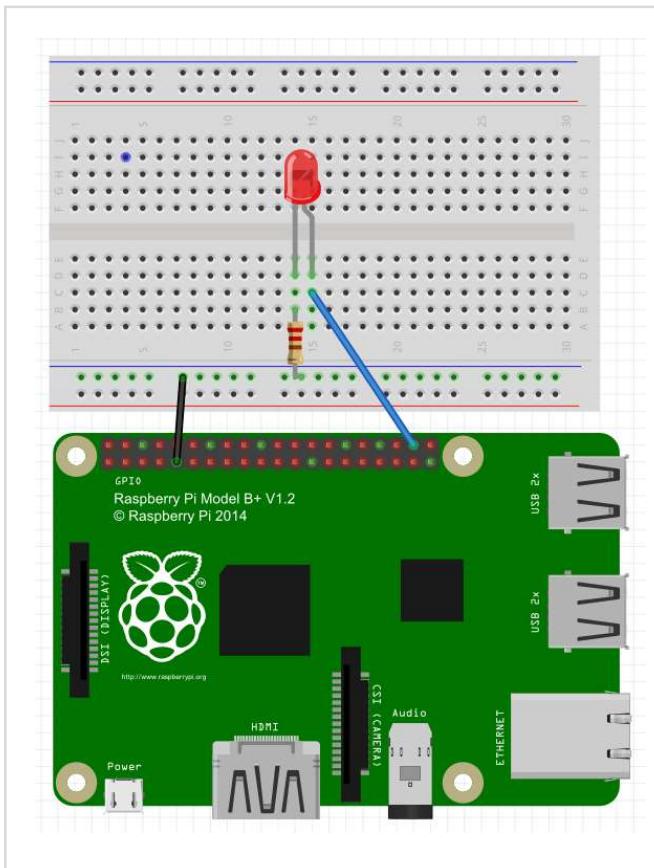
| LED là gì?



- ▶ Diode phát quang (light-emitting diode – LED) là linh kiện bán dẫn có thể phát quang khi cung cấp nguồn điện cho nó.
- ▶ Đèn LED có nhiều ưu điểm khi so sánh với loại bóng sợi đốt, chẳng hạn như tiêu thụ lượng điện năng thấp hơn, tuổi thọ cao hơn, cải thiện độ bền vật lý, kích thước nhỏ hơn và chuyển đổi trạng thái bật-tắt nhanh hơn.
- ▶ Tuy nhiên, đèn LED cũng có nhiều nhược điểm, chẳng hạn như các hạn chế điện tính đối với điện áp thấp và thường là nguồn điện 1 chiều - DC (không phải xoay chiều - AC), không có khả năng cung cấp ánh sáng ổn định từ nguồn điện xung DC hoặc nguồn AC, có nhiệt độ hoạt động tối đa và nhiệt độ lưu trữ thấp hơn.
- ▶ Dòng điện chạy theo chiều từ Anode (cực dương) → Cathode (cực âm) như diode thông thường.
- ▶ Các chân GPIO và bộ thư viện GPIO Zero trên Raspberry Pi được sử dụng để kết nối và điều khiển LED.
- ▶ Hãy xem cách Điều khiển LED thông qua các ví dụ sử dụng thư viện GPIO Zero để có thể tự thực hiện hiệu quả.

Bật LED

| Bật LED



- ▶ Sơ đồ mạch
 - Chân GPIO 20 của Raspberry Pi – Cực dương (anode) của LED
 - Cực âm (Cathode) của LED – Điện trở (khoảng 2K ôm)
 - Chân còn lại của điện trở - Hàng chân âm trên mạch test (BreakBoard)
 - Hàng chân âm trên mạch test – Chân GND trên cổng GPIO của Raspberry Pi
- ▶ Sơ đồ mạch cho thấy tín hiệu từ GPIO truyền qua cực dương đến cực âm của LED, qua điện trở và cuối cùng đến nguồn âm (Ground - GND) của Raspberry Pi.
- ▶ Sau khi chuẩn bị xong mạch điện, chuyển tiếp sang lập trình Python.

| Chương trình Python (led1.py)

```
1  from gpiozero import LED  
2  
3  led_red = LED(20)  
4  
5  while True:  
6      led_red.on()
```

- ▶ dòng 1: Khai báo thư viện LED trong thư viện gpiozero.
- ▶ dòng 3: Gán giá trị thứ tự chân GPIO được kết nối với đèn LED dưới dạng tham số, chân GPIO 20.
- ▶ dòng 5-6: vòng lặp vô tận while điều khiển bật LED
- ▶ Tham khảo thêm các ví dụ liên quan đến điều khiển LED bằng thư viện GPIO Zero tại website:
https://gpiozero.readthedocs.io/en/stable/api_output.html

| Thực thi led1.py

- ▶ Thực thi chương trình led1.py trên Raspberry Pi.

```
$ python3 led1.py
```

```
pi@raspberrypi:~/Toy_projects $ python led1.py
```

- ▶ Kiểm tra xem đèn LED có sáng không.

Nháy LED

I Chương trình Python (led2.py)

- Chương trình điều khiển đèn LED nhấp nháy theo một chu kỳ cụ thể.
- Phương thức (hàm) pause() của thư viện signal đưa tiến trình (chương trình) vào chế độ ngủ cho đến khi nhận được tín hiệu.
- Vì tiến trình sẽ khởi động lại khi nhận một tín hiệu mới thông qua hàm pause, nên chương trình có thể được thực hiện lại bất cứ khi nào một tín hiệu mới xuất hiện trong hàm blink() mà không cần dùng vòng lặp while như đoạn mã lệnh ở phần trước.

```
1  from gpiozero import LED  
2  from signal import pause  
3  
4  led_red = LED(20)  
5  
6  led_red.blink(1)  
7  pause()
```

- dòng 1: Khai báo thư viện lập trình với LED của thư viện gpiozero.
- dòng 2: Khai báo thư viện lập trình pause của thư viện signal.
- dòng 4: Gán giá trị thứ tự chân GPIO được kết nối với đèn LED dưới dạng tham số, GPIO 20
- dòng 6: Làm cho đèn LED nhấp nháy theo chu kỳ (Trong trường hợp này, bật trong 1 giây và tắt trong 1 giây).
- dòng 7: Khởi động lại tiến trình khi nhận được tín hiệu mới.

I Thực thi led2.py

- Thực thi mã lệnh led2.py trên Raspberry Pi.

```
$ python3 led2.py
```

```
pi@raspberrypi:~/Toy_projects $ python led2.py
```

- Thay đổi tham số trong phương thức blink và kiểm tra xem chu kỳ nhấp nháy có được thay đổi tương ứng không.
 - Khi sử dụng cú pháp `blink(on_time = 3, off_time = 1)`, LED được cài đặt để bật trong 3 giây và tắt trong 1 giây.

Điều khiển LED bằng chuỗi nhập từ bàn phím

I Chương trình Python (led3.py)

- ▶ Điều khiển đèn LED bằng cách nhập một chuỗi lệnh vào cửa sổ Terminal.

```
1  from gpiozero import LED  
2  
3  led_red = LED(20)  
4  
5  while True:  
6      s = input()  
7      if s == "on":  
8          led_red.on()  
9      elif s == "off":  
10         led_red.off()  
11      else:  
12          print("invalid command")
```

- dòng 1: Khai báo thư viện lập trình với LED từ thư viện gpiozero.
- dòng 3: Gán giá trị thứ tự chân GPIO được kết nối với đèn LED dưới dạng tham số.
- dòng 6: Chờ tới khi có chuỗi được người dùng nhập vào.
- dòng 7-10: Nếu chuỗi đầu vào là "on" hoặc "off", đèn LED được điều khiển theo lệnh tương ứng.
- dòng 11-12: Nếu chuỗi đầu được nhập không phải là "on" hoặc "off", hãy in ra thông báo " invalid command" cho biết rằng lệnh không phù hợp.

| Thực thi led3.py

- ▶ Thực thi led3.py trên Raspberry Pi.

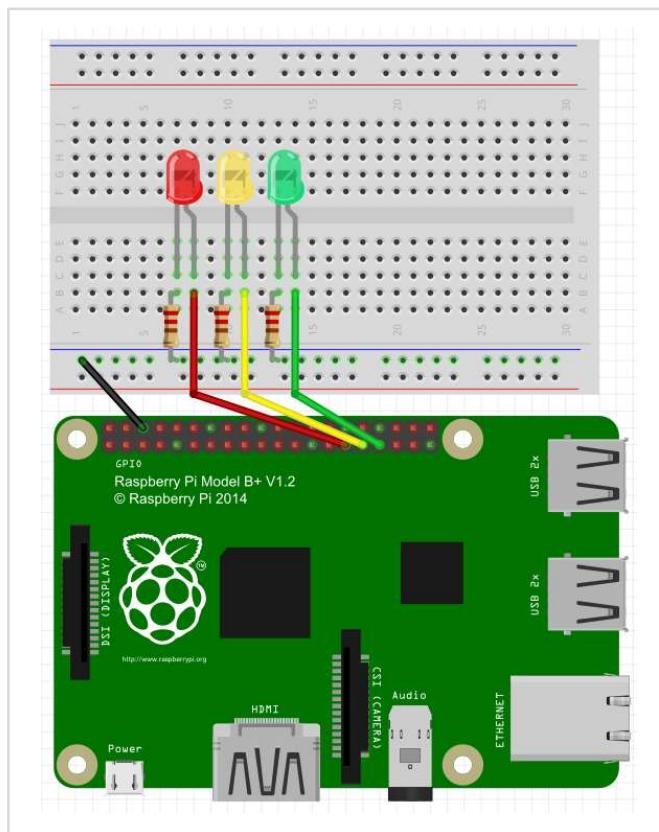
```
$ python3 led3.py
```

- ▶ Kiểm tra các thông báo và trạng thái của LED khi nhập vào chuỗi on, off hoặc chuỗi khác.

```
pi@raspberrypi:~/Toy_projects $ python led3.py  
LED on  
on  
LED off  
off  
LED on  
on  
LED off  
off  
nothing  
invalid command  
message  
on
```

Điều khiển LED

| Điều khiển LED



▶ Sơ đồ mạch

- Chân GPIO 5 của Raspberry Pi – Cực dương của LED đỏ
 - Chân GPIO 6 của Raspberry Pi – Cực dương của LED vàng
 - Chân GPIO 13 của Raspberry Pi – Cực dương của LED xanh
 - Cực âm của LED đỏ – Điện trở (2K Ôm)
 - Cực âm của LED vàng – Điện trở (2K Ôm)
 - Cực âm của LED xanh – Điện trở (2K Ôm)
 - Chân còn lại của các Điện trở - Hàng chân âm của breadboard
 - Hàng chân âm của breadboard – Chân GND trên Raspberry Pi
- ▶ Sau khi chuẩn bị xong mạch điện, chuyển tiếp sang lập trình Python.

| Chương trình Python (led4.py)

```
1  from gpiozero import LED
2  from time import sleep
3
4  led ={
5      "red" : LED(5),
6      "yellow" : LED(6),
7      "green" : LED(13)
8  }
9
10 while True:
11     for i in led.keys():
12         led[i].blink(on_time=1, off_time=1)
13         sleep(2)
```

- ▶ dòng 1: Khai báo thư viện lập trình với LED từ thư viện gpiozero.
- ▶ dòng 2: Khai báo thư viện sleep từ thư viện time để thiết lập thời gian chờ.
- ▶ dòng 4-7: Gán giá trị thứ tự chân GPIO kết nối với mỗi LED dưới dạng tham số. Liên kết các đối tượng LED tới các khóa của biến kiểu từ điển và đặt giá trị cho chúng.
- ▶ dòng 11-13: Phương thức blink sẽ bật đèn LED trong 1 giây và sau đó tắt trong 1 giây. Nó chạy tuần tự cho từng LED. Chu kỳ chờ được đặt là 2 giây trước khi chuyển sang chu kỳ điều khiển đèn LED tiếp.

| Thực thi led4.py

- ▶ Thực thi led4.py trên Raspberry Pi.

```
$ python3 led4.py
```

```
pi@raspberrypi:~/Toy_projects $ python led4.py
```

- ▶ Bạn có thể quan sát đèn LED nhấp nháy trong khi điều chỉnh các tham số on_time, off_time và sleep.

Bài 1.

Dự án Giám sát mức độ hoạt động của CPU

- | 1.1. Điều khiển LED
- | 1.2. Giám sát mạch Raspberry Pi
- | 1.3. File Text trong Python
- | 1.4. Dự án Toy

psutil

| Thư viện psutil

- ▶ psutil (python system and process utilities) là **một thư viện đa nền tảng để truy xuất thông tin về các tiến trình đang chạy và sử dụng hệ thống** (CPU, bộ nhớ, ổ đĩa, mạng, cảm biến) trong Python.
- ▶ Nó chủ yếu được dùng để giám sát hệ thống, lập hồ sơ, giới hạn tài nguyên của mỗi tiến trình và quản lý các tiến trình đang chạy.
- ▶ Tham khảo các hàm hệ thống sử dụng thư viện psutil tại:
 - <https://psutil.readthedocs.io/en/latest/#system-related-functions>

Các hàm hệ thống trong psutil

I Các hàm lấy thông tin về CPU (bộ xử lý trung tâm)

- ▶ Hàm `cpu_percent()`
 - Trả về một giá trị số nguyên thể hiện việc mức độ sử dụng (%) CPU của toàn hệ thống hiện tại.
 - Để ngắn gọn, chỉ cần đặt tham số “interval” là 1.
- ▶ Hàm `cpu_count()`
 - Trả về giá trị cho biết số lượng “logical CPUs” có trong hệ thống (giống như `os.cpu_count` trong Python 3.4) hoặc trả về giá trị “None” nếu không xác định được.
 - “logical CPUs” là giá trị được tính bằng số, tương ứng với số lõi CPU vật lý nhân với số luồng có thể chạy trên mỗi lõi (còn được gọi là Hyper Threading).

CPU

- ▶ Viết chương trình psutil_cpu.py để thử nghiệm các hàm CPU trong thư viện psutil.

```
1 import psutil  
2  
3 print(psutil.cpu_percent(interval = 1))  
4 print(psutil.cpu_count())
```

- dòng 1: Khai báo thư viện psutil.
- dòng 3: Kiểm tra % CPU được sử dụng bằng phương thức cpu_percent() trong thư viện psutil.
- dòng 4: Kiểm tra số lượng lõi CPU vật lý bằng phương thức cpu_count() trong thư viện psutil.

- ▶ Thực thi psutil_cpu.py.

```
$ python3 psutil_cpu.py
```

```
pi@raspberrypi:~/Toy_projects $ python3 psutil_cpu.py  
1.3  
4
```

I Các hàm trả về thông tin của Ổ đĩa (disk)

▶ Hàm disk_partitions()

- Trả về tất cả các phân vùng trên ổ đĩa, dưới dạng danh sách các tuple được định danh bao gồm kiểu device, mount point, và filesystem. Chức năng này tương tự như lệnh "df" trên UNIX.

▶ Hàm disk_usage()

- Trả về số liệu thống kê dung lượng ổ đĩa với mỗi phân vùng theo đường dẫn được gọi tới dưới dạng tuple được định danh. Bao gồm tổng dung lượng tính bằng byte, dung lượng đã sử dụng, dung lượng trống và tỷ lệ phần trăm đã sử dụng.
- OSError được báo nếu đường dẫn không tồn tại.

I Ổ đĩa

- ▶ Viết chương trình psutil_disk.py để kiểm tra các hàm liên quan đến ổ đĩa trong thư viện psutil.

```
1 import psutil  
2  
3 print(psutil.disk_partitions())  
4 print(psutil.disk_usage('/'))
```

- dòng 1: Khai báo thư viện psutil.
- dòng 2: Sử dụng phương thức disk_partitions() trong thư viện psutil để kiểm tra toàn bộ các phân vùng ổ đĩa được gắn vào hệ thống; bao gồm các kiểu: device, mount point, and filesystem.
- dòng 4: Sử dụng phương thức disk_usage() trong thư viện psutil để kiểm tra thống kê sử dụng ổ đĩa bao gồm tổng dung lượng tính bằng byte, dung lượng đã sử dụng, dung lượng trống và tỷ lệ phần trăm đã sử dụng.

```
$ python3 psutil_disk.py
```

```
pi@raspberrypi:~/Toy_projects $ python3 psutil_disk.py  
[sdiskpart(device='/dev/root', mountpoint '/', fstype='ext4', opts='rw,noatime'), sdiskpart(device='/dev/mmcblk0p1', mountpoint='/boot', fstype='vfat', opts='rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=ascii,shortname=mixed,errors=remount-ro')]  
sdiskusage(total=15348269056, used=8577150976, free=6092939264, percent=58.5)
```

Bài 1.

Dự án Giám sát mức độ hoạt động của CPU

- | 1.1. Điều khiển LED
- | 1.2. Giám sát mạch Raspberry Pi
- | **1.3. File Text trong Python**
- | 1.4. Dự án Toy

File Text trong Python

| Tạo file Text

- ▶ Bạn có thể mở một tệp bằng phương thức open().
- ▶ Có 2 tham số của phương thức open() là path và mode. Tham số path chỉ định đường dẫn của tệp cần mở, và tham số mode chỉ định chế độ làm việc với file, với ba chế độ khác nhau tuỳ theo mục đích sử dụng.
 - Path: có thể được sử dụng bất kể đường dẫn tuyệt đối hay tương đối. Nếu tệp không tồn tại, nó sẽ tự động tạo mới một tệp.
 - r (read mode): chỉ đọc.
 - w (write mode): chỉ ghi, xóa nội dung hiện có và ghi đè lên nội dung mới.
 - a (append mode): thêm nội dung mới vào cuối nội dung của tệp hiện có.
- ▶ Sau khi kết thúc chỉnh sửa, nên đóng tệp đang mở bằng phương thức close(). Có thể bỏ qua thao tác này, nhưng sẽ có lỗi nếu ta cố gắng sử dụng lại một tệp đã mở ở chế độ ghi mà không đóng nó trước đó.

Tạo file Text

- Nhập chuỗi ~/Toy_projects/make_txtfile.py và thực thi câu lệnh.

```
1 f = open("test.txt", 'w')
2 f.close()
```

- Dòng 1: với việc chỉ viết tên file, câu lệnh có mục đích tìm kiếm file test.txt ngay trong thư mục chứa file python để mở file và ghi (write) dữ liệu vào file.
 - Nếu file không tồn tại, một file test.txt mới sẽ được khởi tạo và mở.
- line 2: Đóng file đang được mở.

- Chạy file make_textfile.py trong Terminal và kiểm tra kết quả.
- Có thể thấy rằng file test.txt mới đã được tạo.

```
pi@raspberrypi:~/Toy_projects $ ls
cpu_usage_log.txt led1.py led4.py psutil_cpu.py temp_humid.py ultrasonic_span.py
cpu_usage.py led2.py make_txtfile.py psutil_cpu.pyc temp_visualization.py
distance_log.txt led3.py pot.py psutil_disk.py ultrasonic.py
pi@raspberrypi:~/Toy_projects $ python3 make_txtfile.py
pi@raspberrypi:~/Toy_projects $ ls
cpu_usage_log.txt led1.py led4.py psutil_cpu.py temp_humid.py ultrasonic.py
cpu_usage.py led2.py make_txtfile.py psutil_cpu.pyc temp_visualization.py ultrasonic_span.py
distance_log.txt led3.py pot.py psutil_disk.py test.txt
pi@raspberrypi:~/Toy_projects $
```

I Ghi lên file text

- Tạo file write_txtfile.py để mở và ghi nội dung lên file đó.

```
1 f = open("test.txt", 'w')
2 for i in range(1, 7):
3     data = f"Line {i}\n"
4     f.write(data)
5 f.close()
```

- Dòng 1: Mở file test.txt ở chế độ ghi (write).
- Dòng 2-4: vòng lặp để ghi nội dung mới (data) lên file đã mở thông qua câu lệnh f.write(data)
- Dòng 5: Đóng file đã mở sau khi hoàn thành các thao tác liên quan đến nó.

- Chạy file write_textfile và kiểm tra nội dung.

```
pi@raspberrypi:~/Toy_projects $ python3 write_txtfile.py
pi@raspberrypi:~/Toy_projects $ nano test.txt
```

- Có thể quan sát thấy nội dung của file test.txt đã được ghi lên đúng như mong muốn.

```
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
```

I Ghép nối vào file text

- Tạo file append_txtfile.py để ghép nối nội dung mới vào nội dung hiện có của file.

```
1 f = open("test.txt", 'a')
2 for i in range(9, 11):
3     data = f"Added Line {i}!\n"
4     f.write(data)
5 f.close()
```

- Dòng 1: Mở file test.txt ở chế độ ghép nối (append).
- Dòng 2-4: một vòng lặp để ghi (f.write(data)) nội dung được ghép nối (data) vào file text
- Dòng 5: Đóng file đã mở sau khi hoàn thành các thao tác liên quan đến nó.

- Chạy file append_textfile đã tạo và kiểm tra nội dung.

```
pi@raspberrypi:~/Toy_projects $ python3 append_txtfile.py
pi@raspberrypi:~/Toy_projects $ nano test.txt
```

- Có thể quan sát thấy nội dung của file test.txt đã được ghi đúng như mong muốn.

```
GNU nano 3.2                         test.txt
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8
Added Line 9!
Added Line 10!
```

Đọc File Text trong Python

I Đọc file text

- ▶ Tiếp theo, người học sẽ tìm hiểu và thực hành sử dụng các phương thức `readline()`, `readlines()` và `read()` để đọc một tập tin ngoại vi trong Python.
- ▶ `readline()`
 - Đọc một dòng từ file ngoại vi và trả về dòng đó dưới dạng chuỗi.
 - Nếu không còn hàng nào để đọc, nó sẽ trả về một chuỗi rỗng.
- ▶ `readlines()`
 - Trả về một danh sách chứa tất cả các dòng trong file ngoại vi.
 - Ví dụ: nó trả về một danh sách như `["dòng 1\n", "dòng 2\n", ... , "đã thêm Dòng 10!\n"]`
- ▶ `read()`
 - Trả về toàn bộ nội dung tệp (dưới dạng chuỗi).

| Tạo file read_txtfile.py và sử dụng các phương thức/hàm khác nhau liên quan đến đọc file text.

```
1  f = open("test.txt", 'r')
2  print("readline() method")
3  print(f.readline())
4  print(f.readline())
5  f.close()
6
7  f = open("test.txt", 'r')
8  print("readlines() method")
9  lines = f.readlines()
10 for line in lines:
11     print(line.strip())
12 f.close()
13
14 f = open("test.txt", 'r')
15 print("read() method")
16 data = f.read()
17 print(data)
18 f.close()
```

- ▶ Dòng 1-2: Mở file test.txt ở chế độ đọc và in ra thông báo "readline() method".
- ▶ Dòng 3-4: Hàm Readline() trả về một dòng văn bản trong file test.txt. Khi câu lệnh readline() đầu tiên được thực thi, "line 1\n" được trả về. Khi câu lệnh readline() thứ hai được thực thi, "line 2\n" được trả về. Hàm print trong Python in ra "line1\n\n" vì nó sẽ chèn một dòng mới (tương ứng với ký tự điều khiển '\n' sau khi đã in ra các tham số).
- ▶ Dòng 5: Đóng file sau khi đọc xong.
- ▶ Dòng 7-8: Mở file test.txt ở chế độ đọc và in ra thông báo "readlines() method".

| Tạo file read_txtfile.py và sử dụng các phương thức (method) khác nhau liên quan đến đọc file text.

```
1  f = open("test.txt", 'r')
2  print("readline() method")
3  print(f.readline())
4  print(f.readline())
5  f.close()
6
7  f = open("test.txt", 'r')
8  print("readlines() method")
9  lines = f.readlines()
10 for line in lines:
11     print(line.strip())
12 f.close()
13
14 f = open("test.txt", 'r')
15 print("read() method")
16 data = f.read()
17 print(data)
18 f.close()
```

- ▶ Dòng 9-11: Toàn bộ nội dung của file được lưu trong biến lines. Vòng lặp sẽ duyệt qua tuần tự các dòng (line) trong Lines và in nội dung của dòng văn bản ra màn hình. Với hàm print trong mã lệnh, điều khiển xuất '\n' bị xóa khỏi nội dung bằng cách sử dụng phương thức strip(), tránh việc dòng trống chèn vào hai lần như mã lệnh ở phần trước.
- ▶ Dòng 12: Đóng file sau khi đọc xong.
- ▶ Dòng 14-15: Mở file test.txt ở chế độ đọc và in ra thông báo "read() method".
- ▶ Dòng 16-17: Lưu trữ một chuỗi chứa tất cả nội dung của f trong data và in ra nội dung.
- ▶ Dòng 18: Đóng file sau khi đọc xong.

| Kiểm tra kết quả chương trình

```
pi@raspberrypi:~/Toy_projects $ python3 read_txtfile.py
readline() method
line 1
line 2
readlines() method
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
Added Line 9!
Added Line 10!
read() method
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
Added Line 9!
Added Line 10!
```

- ▶ Chạy file read_textfile và kiểm tra nội dung.
- ▶ Có thể thấy phần sử dụng readline() in ra các dòng văn bản có chứa hai kí tự \n, tương ứng với 2 dòng trống.
- ▶ Có thể thấy phần mã lệnh sử dụng readlines() in ra dòng văn bản chỉ có một kí tự \n nhờ vào hàm strip().
- ▶ Trong phần mã lệnh sử dụng hàm read(), có thể thấy toàn bộ nội dung của file đều được in bằng một lệnh print().

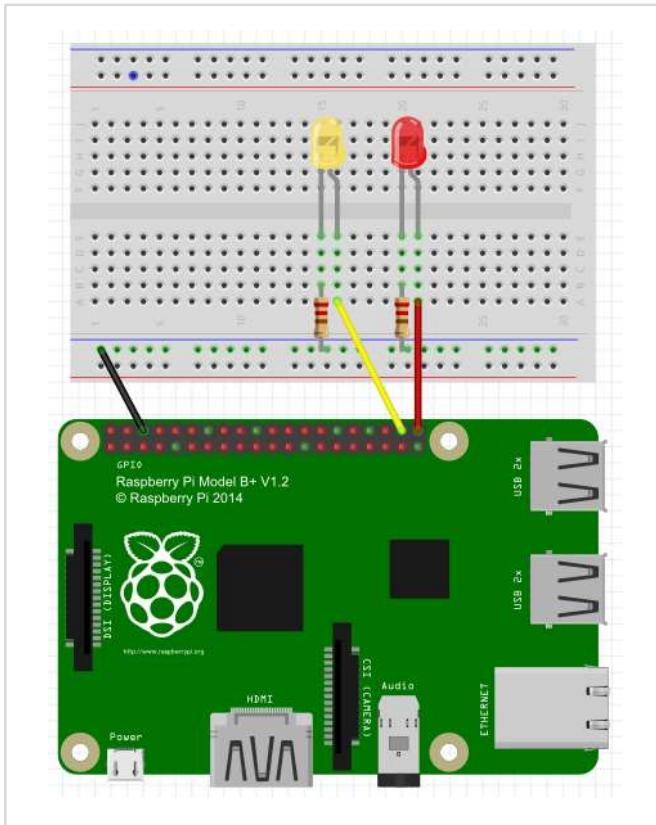
Bài 1.

Dự án Giám sát mức độ hoạt động của CPU

- | 1.1. Điều khiển LED
- | 1.2. Giám sát mạch Raspberry Pi
- | 1.3. File Text trong Python
- | **1.4. Dự án Toy**

Ứng dụng giám sát hoạt động của CPU

I Tổng quan về ứng dụng



- ▶ Một ứng dụng giám sát việc sử dụng CPU, đặt cảnh báo khi mức sử dụng CPU vượt quá một tỷ lệ phần trăm nhất định và ghi lại mức sử dụng CPU dưới dạng tệp nhật ký (log).
- ▶ Sơ đồ ghép mạch
 - Chân GPIO 20 của Raspberry Pi – Cực dương của LED vàng
 - Chân GPIO 21 của Raspberry Pi – Cực dương của LED đỏ
 - Cực âm của LED vàng – điện trở (2K Ôm)
 - Cực âm của LED đỏ – điện trở (2K Ôm)
 - Chân còn lại của các Điện trở - Hàng chân âm của breadboard
 - Hàng chân âm của breadboard – Chân GND trên Raspberry Pi

Giám sát việc sử dụng CPU

| Thư viện psutil

- ▶ Bạn có thể sử dụng CPU bằng cách sử dụng thư viện psutil được cung cấp theo mặc định trong Raspberry Pi.
- ▶ Phương thức `cpu_percent()` của psutil trả về mức sử dụng CPU.
- ▶ Khi `percpu` được đặt thành `True` làm tham số của `cpu_percent()`, mức sử dụng CPU được trả về dưới dạng phần trăm.
- ▶ Thay vì mức sử dụng của từng CPU, chúng tôi tính toán trung bình mức sử dụng của tất cả các CPU để tìm ra tổng mức sử dụng CPU.
- ▶ Return type của `cpu_percent()` nhận ra một danh sách chứa việc sử dụng tất cả các CPU và tính toán trung bình.

```
11 import psutil  
12 cpu_usage = psutil.cpu_percent(interval = 1, percpu = True)
```

Lưu nhật ký với Datetime

I Lưu nhật ký ngày giờ với Datetime

- ▶ Chương trình thực hiện ghi nhận lại mức độ sử dụng (%) CPU tại từng thời điểm, dưới dạng nhật ký.
- ▶ Hàm `datetime.now()` là một phương thức trả về thời gian hiện tại của hệ thống. Trong trường hợp này, nó trả về một đối tượng `datetime` thay vì một chuỗi.
- ▶ Phương thức `strftime()` trả về một đối tượng `datetime` ở định dạng đã xác định.
 - `%Y`: năm, `%m`: tháng, `%d`: ngày
 - `%H`: giờ, `%M`: phút, `%S`: giây
- ▶ Với các phương thức này, thông tin thời gian ở định dạng mong muốn được lưu/ ghi vào file nhật ký.

```
11     from datetime import datetime
12     data = f"{datetime.now().strftime('%Y/%m/%d %HH %MM %SS')}" \
13         f"cpu usage(%) : {cpu_usage_mean}\n"
```

Đoạn mã Python để giám sát CPU

| ~/Toy_projects/cpu_usage.py(1)

- ▶ Tạo tập lệnh ~/Toy_projects/cpu_usage.py.

```
1  from gpiozero import LED
2  import psutil
3  from time import sleep
4  from datetime import datetime
5
6  led_yellow = LED(20)
7  led_red = LED(21)
8  file = open("/home/pi/Toy_projects/cpu_usage_log.txt", "w")
```

- dòng 1: Khai báo thư viện LED từ thư viện gpiozero.
- dòng 2: Khai báo thư viện lập trình psutil để giám sát mức sử dụng CPU.
- dòng 3: Khai báo thư viện sleep từ thư viện time cho xử lý định thời chờ.
- dòng 4: Khai báo thư viện datetime để xác định thông tin thời gian hệ thống.
- dòng 6-7: Gán số chân GPIO được kết nối với đèn LED làm tham số.
- dòng 8: Tạo tệp văn bản và mở ở chế độ ghi để lưu thông tin nhật ký.

Đoạn mã Python để giám sát CPU

| ~/Toy_projects/cpu_usage.py(2)

- ▶ Tạo tập lệnh ~/Toy_projects/cpu_usage.py.

```

10     while True:
11         cpu_usage = psutil.cpu_percent(interval = 1, percpu = True)
12         cpu_usage_mean = sum([i/len(cpu_usage) for i in cpu_usage])
13         cpu_usage_mean = round(cpu_usage_mean, 3)
14         print(f"cpu usage(%): {cpu_usage_mean}%")
15         if 60 > cpu_usage_mean > 30: # usage over 30%
16             led_yellow.on()
17             led_red.off()
18         elif cpu_usage_mean >= 60: # usage over 60%
19             led_yellow.on()
20             led_red.on()
21         else:
22             led_yellow.off()
23             led_red.off()
24         data = f"{datetime.now().strftime('%Y/%m/%d %H %M %S')} " \
25                 f"cpu usage(%): {cpu_usage_mean}% \n"
26         file.write(data)
27         sleep(1)
28     file.close()
```

- dòng 11-13: Sử dụng phương thức `cpu_percent()` để xác định mức sử dụng CPU. Tính giá trị trung bình của mỗi lần sử dụng CPU và lưu trữ giá trị làm tròn theo `cpu_usage_mean`.
- dòng 14: In/xuất thông tin mức sử dụng CPU vào cửa sổ Terminal.
- dòng 15-17: Đèn LED màu vàng được bật khi sử dụng CPU trong khoảng 30 ~ 60%.
- dòng 18-20: Cả đèn LED vàng và đèn LED đỏ đều bật khi mức sử dụng CPU trên 60%.
- dòng 21-23: Nếu không, cả hai đèn LED đều tắt.
- dòng 24-27: Đặt dấu thời gian ở định dạng Năm / Tháng / Giờ ngày / Phút / Giây. Lưu dấu thời gian và mức sử dụng CPU trong dữ liệu, ghi dữ liệu vào file nhật ký và thực hiện chờ 1 giây trước khi thực hiện lại xử lý trong vòng lặp.
- dòng 28: Đóng tệp nhật ký sau khi hoàn thành thao tác với tệp nhật ký.

Kết quả của dự án giám sát CPU

Mở Terminal trên Raspberry Pi

- Thực thi tệp tập lệnh python.

```
$ python3 cpu_usage.py
```

```
pi@raspberrypi:~ $ cd Toy_projects/
pi@raspberrypi:~/Toy_projects $ python3 cpu_usage.py
cpu usage(%) : 17.35%
cpu usage(%) : 6.1%
cpu usage(%) : 53.7%
cpu usage(%) : 88.25%
```

- Kiểm tra việc sử dụng CPU có được giám sát trong khi thực hiện các tác vụ khác nhau như truy cập Internet trên Raspberry Pi để quan sát xem việc sử dụng CPU có thay đổi hay không.
- Kiểm tra xem ánh sáng LED có thay đổi theo mức sử dụng CPU được chỉ định cũng được thực hiện đúng cách hay không.

Kết quả của dự án giám sát CPU

File Log

- ▶ Kiểm tra xem tệp log có được tạo thành công không.

```
$ ls  
pi@raspberrypi:~/Toy_projects $ ls  
cpu_usage_log.txt  cpu_usage.py
```

- ▶ Kiểm tra nội dung của tệp log.

```
pi@raspberrypi:~/Toy_projects $ nano cpu_usage_log.txt  
2021/09/30 04H 07M 55S cpu usage(%) : 17.35%  
2021/09/30 04H 07M 57S cpu usage(%) : 6.1%  
2021/09/30 04H 07M 59S cpu usage(%) : 53.7%  
2021/09/30 04H 08M 01S cpu usage(%) : 88.25%  
2021/09/30 04H 08M 03S cpu usage(%) : 89.05%  
2021/09/30 04H 08M 05S cpu usage(%) : 76.675%  
2021/09/30 04H 08M 07S cpu usage(%) : 81.125%  
2021/09/30 04H 08M 09S cpu usage(%) : 46.725%  
2021/09/30 04H 08M 11S cpu usage(%) : 4.5%
```

Bài 2.

Dự án Kiểm soát dài giá trị đo của cảm biến

- | 2.1. Sử dụng chiết áp
- | 2.2. Sử dụng cảm biến siêu âm
- | 2.3. Dự án Toy

Chiết áp

| Chiết áp (Điện trở ba cực)



- ▶ Chiết áp là một điện trở ba cực có tiếp điểm trượt hoặc quay tạo thành một bộ chia điện áp có thể điều chỉnh.
- ▶ Nếu chỉ có hai thiết bị đầu cuối được sử dụng, một đầu và cần gạt nước, nó hoạt động như một điện trở thay đổi hoặc bộ biến trở.
- ▶ Dụng cụ đo được gọi là chiết áp về cơ bản là một bộ chia điện áp được sử dụng để đo điện thế (điện áp). Các thành phần được đặt tên bởi vì chúng là các triển khai của cùng một nguyên tắc.
- ▶ Chiết áp thường được sử dụng để điều khiển các thiết bị điện, chẳng hạn như điều khiển âm lượng trên thiết bị âm thanh.
- ▶ Chiết áp được vận hành bởi một cơ chế có thể được sử dụng làm đầu dò vị trí.

MCP3008

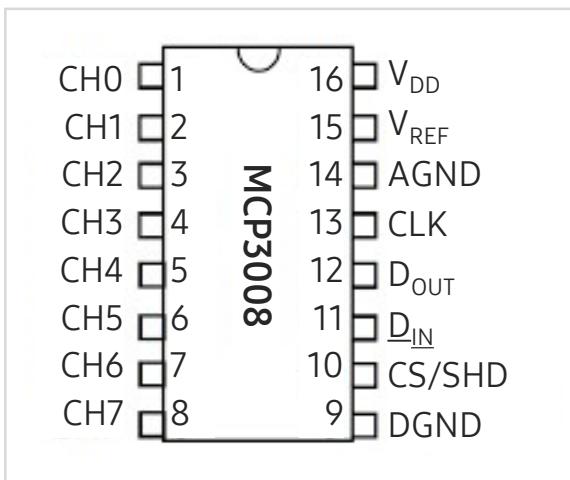
| CP3008 (ADC, Bộ chuyển đổi tương tự - số)



- ▶ Bộ chuyển đổi tương tự sang kỹ thuật số (ADC) 10-bit MCP3008 với hiệu suất cao và tiêu thụ điện năng thấp, kích thước nhỏ. Đó là lý tưởng cho các ứng dụng điều khiển nhúng.
- ▶ MCP3008 có kiến trúc thanh ghi xấp xỉ (SAR) liên tiếp, hỗ trợ giao tiếp với mạch ngoài qua SPI (theo tiêu chuẩn công nghiệp). Nó cho phép thực hiện biến đổi ADC với độ phân giải 10-bit, tương tự như bộ ADC tích hợp bên trong bất kỳ vi điều khiển PIC® nào.
- ▶ MCP3008 xử lý lấy mẫu 200k mẫu/giây, với 8 kênh đầu vào, tiêu thụ điện năng thấp (chế độ chờ điển hình 5nA, hoạt động điển hình 425µA) và được đóng gói theo các chuẩn chân cắm PDIP hoặc SOIC 16 chân.
- ▶ Các ứng dụng cho MCP3008 bao gồm thu thập dữ liệu, thiết bị đo đạc và đo lường, bộ ghi dữ liệu đa kênh, PC công nghiệp, điều khiển động cơ, robot, tự động hóa công nghiệp, cảm biến thông minh, thiết bị cầm tay và thiết bị y tế gia đình.

Mạch đọc giá trị của Chiết áp

I Chân trên MCP3008



- ▶ **SCLK** : Serial CLock
- ▶ **CE** : Chip Enable (CS - Chip Selection)
- ▶ **MOSI** : Master Out Slave In
- ▶ **MISO** : Master In Slave Out
- ▶ **MOMI** : Master Out Master In
- ▶ **CH0 – CH7** : Kênh nhận giá trị tương tự từ các nguồn khác nhau

MCP3008 Datasheet

Chiết áp

| Nhận giá trị của chiết áp

```
import time
import spidev

spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 1000000

def ReadVol(vol):
    adc = spi.xfer2([1, (8 + vol) << 4, 0])
    data = ((adc[1] & 3) << 8) + adc[2]
    return data

mcp3008 = 0

while True:
    a_1 = ReadVol(mcp3008)
    print('readvol : ', a_1, 'Voltage:', 3.3 * a_1 / 1024)
    time.sleep(0.5)
```

- ▶ Sử dụng cổng SPI để giao tiếp với MCP3008, có thể nhận được giá trị tương tự của chiết áp như trong đoạn chương trình bên trái.
- ▶ Với thư viện gpiozero, chương trình có thể hiệu quả hơn và dễ đọc hơn nhiều với cùng chức năng.

I Đọc giá trị của chiết áp với GPIO Zero

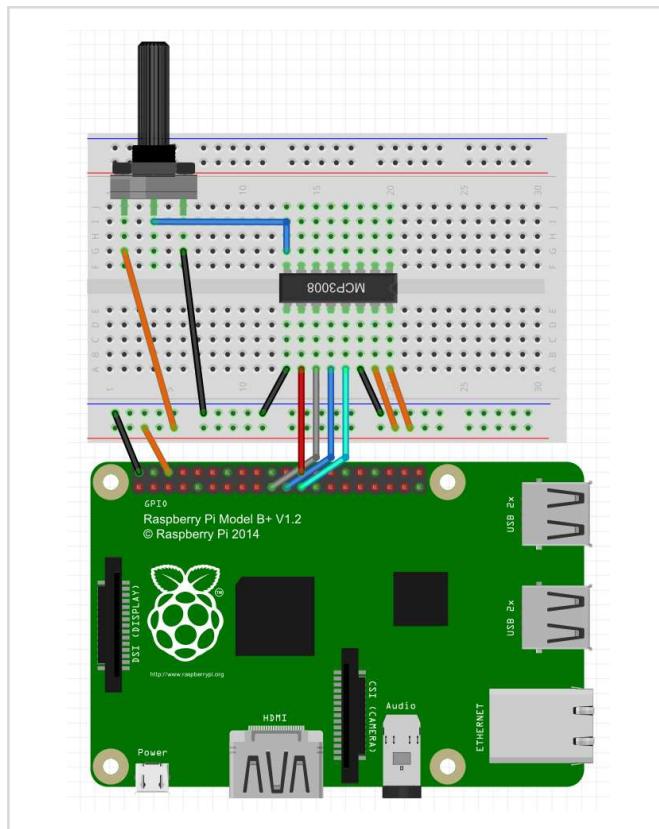
- Đây là một Tập lệnh Python (pot.py) đọc giá trị tương tự của chiết áp bằng cách sử dụng gpiozero và MCP3008.
- Đoạn mã dưới đây có hình thức đơn giản và dễ đọc hơn nhiều so với đoạn mã được nhập bằng giao tiếp qua cổng SPI.

```
1  from gpiozero import MCP3008
2  from time import sleep
3
4  pot = MCP3008(7)
5
6  while True:
7      print(pot.value * 100)
8      sleep(1)
```

- dòng 1: Khai báo thư viện lập trình với MCP3008 từ thư viện gpiozero.
- dòng 2: Khai báo thư viện sleep từ thư viện time để thiết lập thời gian chờ.
- dòng 4: Gán chỉ số kênh tín hiệu đầu vào (dưới dạng tham số) của MCP3008 được kết nối với chiết áp.
- dòng 7-8: hàm pot.value thực hiện đo và chuyển đổi giá trị điện áp trên triết áp từ dạng tương tự sang dạng số,\
- Vòng lặp sẽ thực hiện chờ 1 giây trước khi thực hiện phép đo tiếp theo.

Mạch đọc giá trị của chiết áp

Đọc giá trị của chiết áp với GPIO Zero



- ▶ Một ví dụ về việc đọc giá trị chiết áp và xuất giá trị trên terminal bằng MCP 3008 và GPIO Zero.
- ▶ Sơ đồ mạch
 - VCC của chiết áp – VCCs
 - GND của chiết áp – GNDs
 - OUT của chiết áp – MCP3008 ch7
 - MCP3008 VDD & VREF – VCCs
 - MCP3008 AGND – GNDs
 - MCP3008 CLK – Raspberry Pi GPIO 11 (SCLK)
 - MCP3008 DOUT – Raspberry Pi GPIO 9 (MISO)
 - MCP3008 DIN – Raspberry Pi GPIO 10 (MOSI)
 - MCP3008 CS/SHDN – Raspberry Pi GPIO 8 (CE0)
 - MCP3008 DGND – GNDs
 - Raspberry Pi GND – GNDs

Giá trị đầu ra khi đo chiết áp

| Terminal

- ▶ Thực thi tệp tập lệnh python.

```
$ python3 pot.py
```

```
pi@raspberrypi:~/Toy_projects $ python3 pot.py
0.843765412
2.334129979
```

- ▶ Trong khi điều chỉnh chiết áp, hãy kiểm tra xem giá trị có thay đổi bình thường và phép đo có được thực hiện hay không.

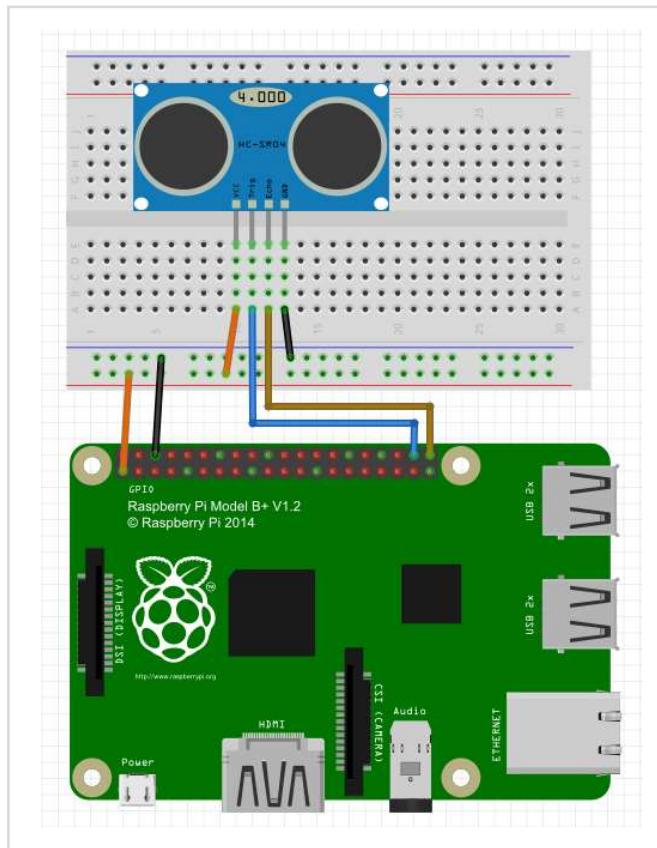
Bài 2.

Dự án Kiểm soát dài giá trị đo của cảm biến

- | 2.1. Sử dụng chiết áp
- | 2.2. Sử dụng cảm biến siêu âm
- | 2.3. Dự án Toy

Cảm biến siêu âm

| Phương pháp đo khoảng cách bằng cảm biến siêu âm



- ▶ Khoảng cách được tính bằng chênh lệch giữa thời gian sóng siêu âm được truyền đi và thời gian để nó chạm vào một vật thể và quay trở lại đầu thu của cảm biến.
 - Echo - Trig
- ▶ Không giống như chiết áp, giá trị này là giá trị số (digital), vì vậy không cần phải chuyển đổi thông qua MCP3008.
- ▶ Sơ đồ mạch
 - HC-SR04 VCC – VCCs
 - HC-SR04 Echo - Raspberry Pi GPIO 21
 - HC-SR04 Trig - Raspberry Pi GPIO 20
 - HC-SR04 GND – GNDs
 - Raspberry Pi GND – GNDs

Đọc giá trị của cảm biến siêu âm

| Tập lệnh Python sử dụng hàm GPIO của Thư viện RPi.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
GPIO.setup(18, GPIO.IN)

try:
    stop = 0
    while True:
        GPIO.output(17, False)
        time.sleep(0.5)

        GPIO.output(17, True)
        time.sleep(0.00001)
        GPIO.output(17, False)

        while GPIO.input(18) == 0:
            start = time.time()

        while GPIO.input(18) == 1:
            stop = time.time()

        time_interval = stop - start
        distance = time_interval * 17000
        distance = round(distance, 2)

        print(f"distance : {distance}")

except KeyboardInterrupt:
    GPIO.cleanup()
```

- ▶ Có thể thực hiện bằng cách sử dụng hàm GPIO của thư viện RPi như trình bày ở hình bên trái.
- ▶ Để giảm bớt sự dài dòng khi lập trình, có thể sử dụng thư viện GPIO Zero để cho hiệu quả và tính linh hoạt cao hơn.

I Chương trình ultrasonic.py

- ▶ Có thể thực hiện chức năng trong mã lệnh phần trước thông qua thư viện gpiozero như sau:

```
1  from gpiozero import DistanceSensor  
2  
3  ultrasonic = DistanceSensor(echo = 21, trigger = 20)  
4  
5  while True:  
6      print(round(ultrasonic.distance * 100, 3))
```

- dòng 1: Khai báo thư viện lập trình DistanceSensor từ thư viện gpiozero để giao tiếp, đọc giá trị từ cảm biến siêu âm (HC-SR04)
- dòng 3: Chỉ định chân GPIO kết nối với chân echo và trigger của Cảm biến siêu âm dưới dạng tham số cho hàm khởi tạo DistanceSensor
- dòng 6: Tính khoảng cách đo được từ cảm biến siêu âm, thực hiện nhân với 100 và làm tròn thích hợp để giá trị khoảng cách xác định trong chương trình phù hợp, rẽ quan sát hơn.

Giá trị đầu ra của cảm biến siêu âm

| Terminal

- ▶ Thực thi tệp tập lệnh python.

```
$ python3 ultrasonic.py
```

```
pi@raspberrypi:~/Toy_projects $ python3 ultrasonic.py
```

- ▶ Thay đổi khoảng cách giữa đối tượng đo và cảm biến siêu âm. Quan sát trên cửa sổ Terminal, ta có thể thấy giá trị khoảng cách đo được của cảm biến được in ra khác nhau tùy thuộc vào khoảng cách đo.

```
8.84  
3.219  
9.79  
14.637  
14.637
```

Bài 2.

Dự án Kiểm soát dài giá trị đo của cảm biến

- | 2.1. Sử dụng chiết áp
- | 2.2. Sử dụng cảm biến siêu âm
- | **2.3. Dự án Toy**

Dải giá trị đo của cảm biến siêu âm

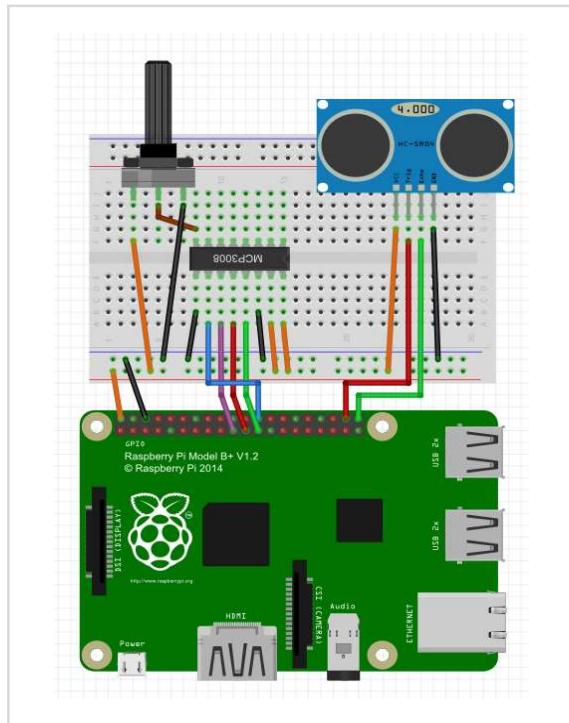
I Đo khoảng cách bằng cảm biến

- ▶ Cảm biến siêu âm là cảm biến xác định khoảng cách từ nó đến đối tượng gần nhất bằng cách sử dụng các xung.
- ▶ Trong trường hợp bạn chỉ muốn đo gần hơn một khoảng cách nhất định, sử dụng chiết áp để thay đổi dải đo của Cảm biến siêu âm để có thể sửa khoảng cách tham chiếu ngay lập tức.
- ▶ Trong dự án này, khoảng cách tham chiếu, hay dải đo được thay đổi thông qua chiết áp, trong khi cảm biến siêu âm đo khoảng cách thực tế từ cảm biến đến vật thể đặt trước nó.



Điều chỉnh dải đo của cảm biến siêu âm

| Tổng quan dự án



- ▶ Dự án đo khoảng cách bằng cảm biến siêu âm nhưng chỉ ghi lại nhật ký khi khoảng cách nhỏ hơn dải đo, được chỉ định bởi chiết áp.
- ▶ Sơ đồ mạch kết nối của dự án
 - HC-SR04 VCC – VCCs
 - HC-SR04 Echo - Raspberry Pi GPIO 21
 - HC-SR04 Trig - Raspberry Pi GPIO 20
 - HC-SR04 GND – GNDs
 - Chân VCC của chiết áp – VCCs
 - Chân GND của chiết áp – GNDs
 - Chân giữa của chiết áp – MCP3008 ch7
 - MCP3008 VDD & VREF – VCCs
 - MCP3008 AGND – GNDs
 - MCP3008 DIN - Raspberry Pi GPIO 10 (MOSI)
 - MCP3008 CS/SHDN - Raspberry Pi GPIO 8 (CEO)
 - MCP3008 DGND - GNDs
 - Raspberry Pi GND – GNDs

| ~/Toy_projects/ultrasonic_span.py (1)

- ▶ Tạo tập lệnh ~/Toy_projects/ultrasonic_span.py.

```
1  from gpiozero import MCP3008, DistanceSensor  
2  from time import sleep  
3  from datetime import datetime  
4  
5  pot = MCP3008(7)  
6  ultrasonic = DistanceSensor(echo = 21, trigger = 20)  
7  file = open("/home/pi/Toy_projects/distance_log.txt", "w")
```

- dòng 1: Khai báo các thư viện lập trình gpiozero, thư viện MCP3008 và DistanceSensor.
- dòng 2: Khai báo thư viện sleep từ thư viện time để thiết lập thời gian chờ.
- dòng 3: Khai báo thư viện datetime để xác định các mốc/dấu thời gian hệ thống.
- dòng 5-6: Chỉ định kênh tín hiệu vào cho bộ biến đổi ADC , kênh 7, và thiết lập biến pot để lưu giá trị từ MCP3008, xác định dải đo cho cảm biến siêu âm.
- dòng 7: Tạo tệp văn bản và mở ở chế độ ghi để lưu nhật ký.

| ~/Toy_projects/ultrasonic_span.py (2)

```
9  while True:
10     dist = ultrasonic.distance * 100
11     span = pot.value * 100 # for scaling
12     dist, span = round(dist, 3), round(span,3)
13     if dist <= span:
14         print(f"scaled span : {span}, dist : {dist}")
15         timestamp = datetime.now().strftime('%Y/%m/%d %HH %MM %SS')
16         data = f"{timestamp} --> "
17         data += f"distance : {dist} , span : {span}\n"
18         file.write(data)
19     else:
20         print(f"Distance > span!! scaled span : {span}, dist : {dist}")
21     sleep(1)
22 file.close()
```

| ~/Toy_projects/ultrasonic_span.py (2)

```

9  while True:
10     dist = ultrasonic.distance * 100
11     span = pot.value * 100 # for scaling
12     dist, span = round(dist, 3), round(span,3)
13     if dist <= span:
14         print(f"scaled span : {span}, dist : {dist}")
15         timestamp = datetime.now().strftime('%Y/%m/%d %H %M %S')
16         data = f"{timestamp} --> "
17         f"distance : {dist} , span : {span}\n"
18         file.write(data)
19     else:
20         print(f"Distance > span!! scaled span : {span}, dist : {dist}")
21         sleep(1)
22     file.close()

```

- dòng 10-11: Chuyển đổi giá trị khoảng cách đo được từ cảm biến siêu âm và dải đo từ chiết áp để có giá trị phù hợp.
- Với cảm biến siêu âm giá trị khoảng cách đo được được nhân với 100, do đó dải đo cũng được nhân với 100 để tương ứng.
- dòng 12: Làm tròn các giá trị một cách thích hợp để dễ xử lý.
- dòng 13-14: xuất thông báo ra màn hình nếu khoảng cách đo được nhỏ hơn dải đo (không hợp lệ).
- dòng 15-17: Ghi khoảng cách đo và dải đo kèm theo dấu thời gian vào tệp nhật ký.
- dòng 19-20: Chỉ in thông báo ra màn hình mà không ghi dữ liệu vào file nhật ký trong trường hợp khoảng cách đo được lớn hơn dải đo (không hợp lệ).
- dòng 21: Chờ 1 giây trước khi thực hiện lại xử lý trong vòng lặp while.
- dòng 22: Đóng file nhật ký sau khi thực hiện xong.

Điều chỉnh dải đo của cảm biến siêu âm

| Terminal

- ▶ Thực thi tệp tập lệnh python.

```
$ python3 ultrasonic_span.py
```

```
pi@raspberrypi:~/Toy_projects $ python3 ultrasonic_span.py
```

- ▶ Điều khiển Chiết áp và Cảm biến siêu âm để thay đổi khoảng cách đo và dải đo. Bạn có thể kiểm tra các thay đổi dựa trên các giá trị đo được trên thiết bị đầu cuối.
- ▶ Nếu Khoảng cách lớn hơn dải đo, bạn có thể thấy rằng giá trị đo được hiển thị cùng với thông báo "Span!".

```
scaled span : 48.705, dist : 5.581
scaled span : 36.199, dist : 4.774
scaled span : 36.199, dist : 14.621
Distance > span!! scaled span : 33.952, dist : 48.744
scaled span : 23.4, dist : 2.97
scaled span : 23.4, dist : 6.23
scaled span : 21.739, dist : 8.915
Distance > span!! scaled span : 18.906, dist : 22.383
Distance > span!! scaled span : 15.486, dist : 23.608
```

I Tệp nhật ký

- ▶ Kiểm tra xem tệp nhật ký có được tạo thành công không.

```
$ ls
```

```
pi@raspberrypi:~/Toy_projects $ ls
cpu_usage_log.txt  cpu_usage.py  distance_log.txt  ultrasonic_span.py
```

- ▶ Kiểm tra nội dung của tệp nhật ký.

```
pi@raspberrypi:~/Toy_projects $ nano distance_log.txt
```

```
2021/09/30 10H 14M 54S --> distance : 45.259 , span : 60.528
2021/09/30 10H 14M 55S --> distance : 44.49 , span : 58.378
2021/09/30 10H 14M 56S --> distance : 5.581 , span : 48.705
2021/09/30 10H 14M 57S --> distance : 4.774 , span : 36.199
2021/09/30 10H 14M 58S --> distance : 14.621 , span : 36.199
2021/09/30 10H 15M 00S --> distance : 2.97 , span : 23.4
2021/09/30 10H 15M 01S --> distance : 6.23 , span : 23.4
2021/09/30 10H 15M 02S --> distance : 8.915 , span : 21.739
```

Giá trị mẫu dữ liệu được ghi nhận
“dist : 2.97, span : 12.4”

Bài 3.

Dự án Trực quan hóa dữ liệu nhiệt độ

- | 3.1. Trực quan hóa dữ liệu
- | 3.2. Dự án Toy

Trực quan hóa

| Nhận thông tin chuyên sâu từ dữ liệu đã được trực quan hóa

- ▶ Trực quan hóa dữ liệu là **Quá trình thể hiện trực quan và cung cấp kết quả phân tích dữ liệu một cách dễ hiểu.**
- ▶ Nhu cầu về Trực quan hóa dữ liệu tăng lên để mô tả trực quan nhiều thông tin và cung cấp thông tin cần thiết một cách hiệu quả và rõ ràng.
- ▶ Kỹ thuật trực quan hóa truyền thống chủ yếu là hiển thị thông tin thống kê về nhật ký hệ thống hoặc kết quả phân tích thực nghiệm trong đồ thị. Về mặt trực quan hóa dữ liệu lớn, có những hạn chế trong việc xem xét tất cả dữ liệu. Cùng với các yếu tố kỹ thuật của trực quan hóa, tầm quan trọng của các yếu tố phương pháp trực quan giúp tóm tắt dữ liệu và xem xét nó ngay lập tức tăng lên.
- ▶ Chúng ta sẽ sử dụng thư viện cơ bản và phổ biến nhất **matplotlib**.



Ví dụ của Trực quan hóa dữ liệu

Sử dụng matplotlib.pyplot

| Vẽ biểu đồ đường đơn giản

- ▶ Cấu trúc dữ liệu chủ yếu được sử dụng để trực quan hóa là DataFrame của thư viện pandas. Đối với những người chưa quen với cấu trúc của DataFrame, chúng ta sẽ sử dụng [các loại cấu trúc dữ liệu cơ bản khác của Python](#) để trực quan hóa.
- ▶ Tạo một file mã lệnh plt_1.py để tạo một biểu đồ đường đơn giản sử dụng thư viện matplotlib.

```
1 import matplotlib.pyplot as plt
2
3 data = {
4     "x": [i for i in range(1, 10)],
5     "y": [i * i for i in range(1, 10)]
6 }
7
8 plt.plot(
9     data["x"],
10    data["y"],
11 )
12 plt.show()
```

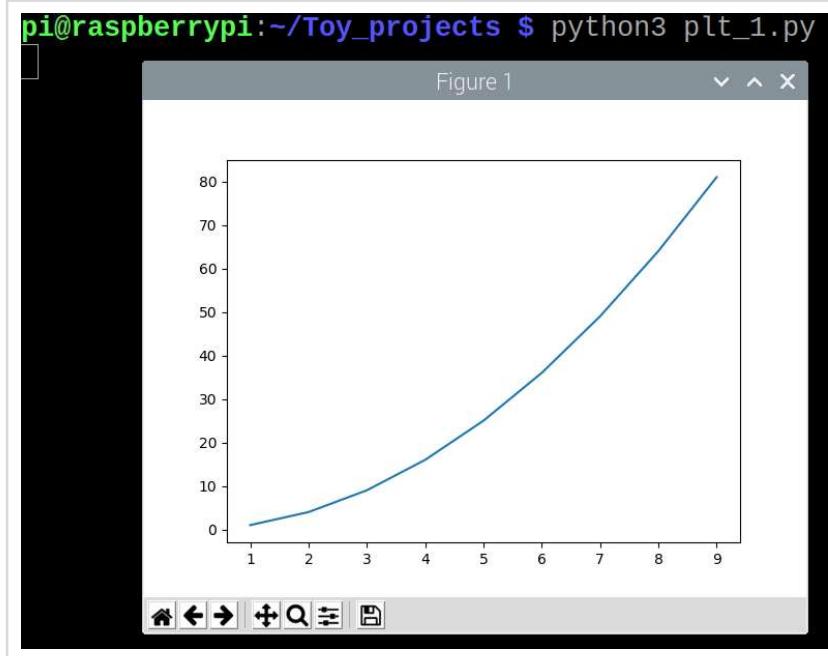
- dòng 1: Khai báo thư viện matplotlib.pyplot và định danh là plt.
- dòng 3-4: Gán x và y của data một cách tương ứng.
 - x : [1, 2, 3, ..., 8, 9]
 - y : [1, 4, 9, ..., 64, 81]
- dòng 8-11: Đặt dữ liệu trục x thành dữ liệu ["x"] và dữ liệu trục y thành dữ liệu ["y"] và vẽ một biểu đồ đường trên đối tượng plt.
- Dòng 12: In biểu đồ đường được vẽ trên đối tượng plt.

| Vẽ biểu đồ đường đơn giản

- ▶ Thực thi plt_1.py và kiểm tra kết quả.

```
$ python3 plt_1.py
```

- ▶ Có thể kiểm tra biểu đồ đường được vẽ dựa trên dữ liệu được tạo trong mã lệnh.
- ▶ Đóng cửa sổ để thoát.



I Vẽ biểu đồ phân tán đơn giản

- ▶ Hãy tạo một biểu đồ phân tán thay vì một biểu đồ đường thẳng.
- ▶ Tạo tệp plt_2.py.

```
1 import matplotlib.pyplot as plt
2
3 data = {
4     "x" : [i for i in range(1, 10)],
5     "y" : [i * i for i in range(1, 10)]
6 }
7
8 plt.scatter(
9     data["x"],
10    data["y"],
11    marker = "o"
12 )
13 plt.show()
```

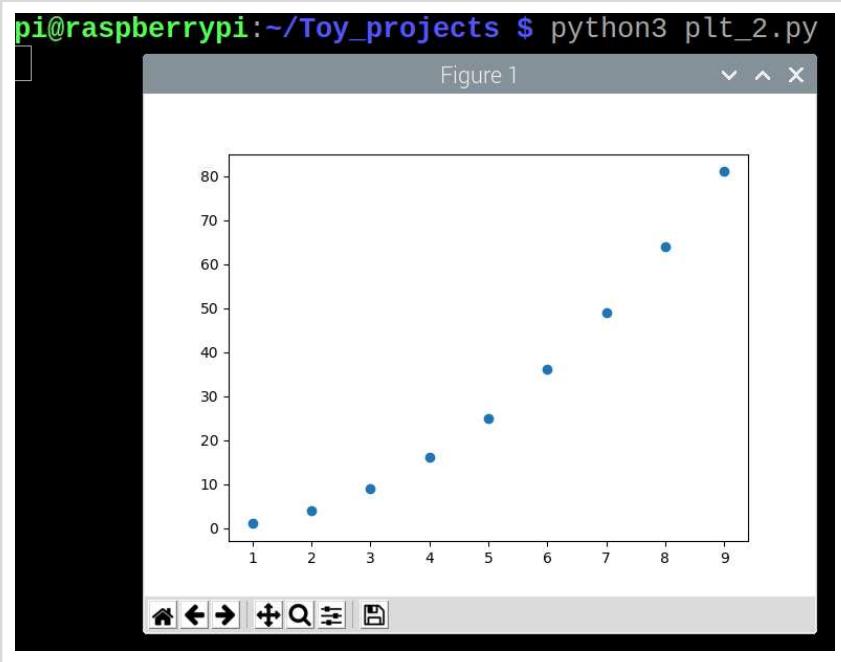
- dòng 1: Khai báo thư viện matplotlib.pyplot và định danh là plt.
- dòng 3-6: Gán x và y của biến data lần lượt là:
 - x: [1, 2, 3 ,..., 8, 9]
 - y: [1, 4, 9, ..., 64, 81]
- dòng 8-12: Đặt dữ liệu của đồ thị theo trục x là data1 ["x"] và trục y là data1 ["y"]. Đặt điểm đánh dấu trên đồ thị là kiểu vòng tròn, "o", và vẽ đồ thị trên đối tượng plt.
- Dòng 12: In/vẽ biểu đồ phân tán trên đối tượng plt.

| Vẽ biểu đồ phân tán đơn giản

- ▶ Thực hiện plt_2.py và kiểm tra kết quả.

```
$ python3 plt_2.py
```

- ▶ Bạn có thể kiểm tra biểu đồ phân tán được vẽ dựa trên dữ liệu được tạo.
- ▶ Đóng cửa sổ để thoát.



I Vẽ nhiều biểu đồ đồng thời

- ▶ Hãy tạo biểu đồ đường thẳng và biểu đồ phân tán trong cùng một không gian đồ thị.
- ▶ Tạo tệp plt_3.py.

```
1 import matplotlib.pyplot as plt
2
3 data1 = {
4     "x": [i for i in range(1, 10)],
5     "y": [i * i for i in range(1, 10)]
6 }
7
8 data2 = {
9     "x": [i for i in range(1, 10)],
10    "y": [i * 10 for i in range(1, 10)]
11 }
12
13 plt.plot(
14     data1["x"],
15     data1["y"],
16     color = "r"
17 )
18 plt.scatter(
19     data2["x"],
20     data2["y"],
21     marker = "^"
22 )
23 plt.show()
```

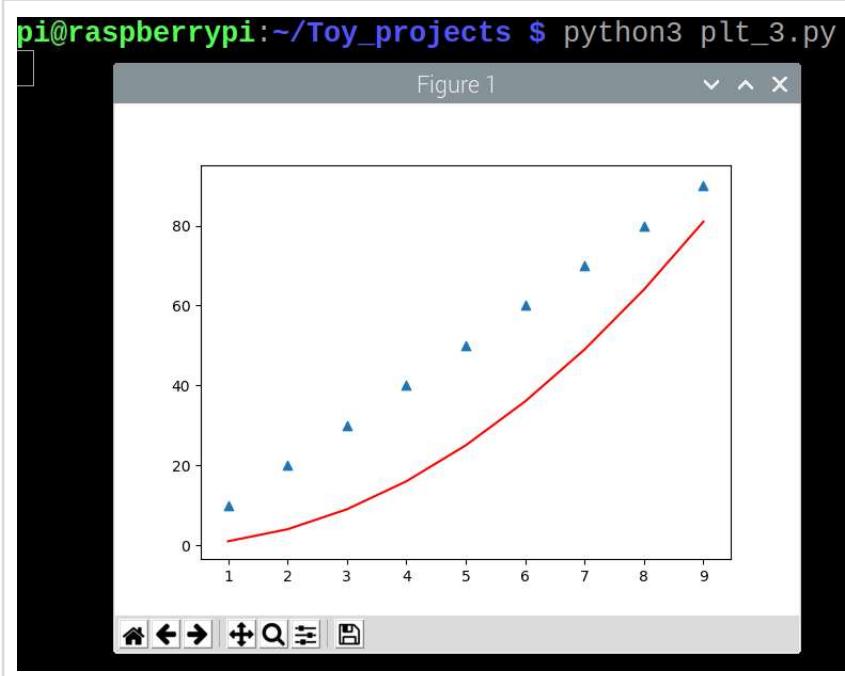
- dòng 1: Khai báo thư viện matplotlib.pyplot và định danh là plt.
- dòng 3-6: Thiết lập biến data1, là tập mẫu giá trị theo trục x và y cho đồ thị biểu diễn
- dòng 8-11: Thiết lập biến data2, tương tự như data1
- dòng 13-17: Đặt dữ liệu theo trục x là data1 ["x"] và trục y là data1 ["y"] cho đồ thị thứ nhất. Đặt thuộc tính màu là "r" (màu đỏ) và vẽ biểu đồ đường trên đối tượng plt.
- dòng 18-22: Đặt dữ liệu theo trục x là data1 ["x"] và trục y là data1 ["y"] cho đồ thị thứ hai. Đặt điểm đánh dấu trên đồ thị là loại hình tam giác, "^", và vẽ biểu đồ phân tán trên đối tượng plt.
- Dòng 23: In (hiển thị) các đồ thị được vẽ của đối tượng plt.

| Vẽ nhiều biểu đồ đồng thời

- ▶ Thực thi plt_3.py và kiểm tra kết quả.

```
$ python3 plt_3.py
```

- ▶ Bạn có thể kiểm tra biểu đồ phân tán và biểu đồ đường được vẽ dựa trên dữ liệu được tạo.
- ▶ Đóng cửa sổ để thoát.



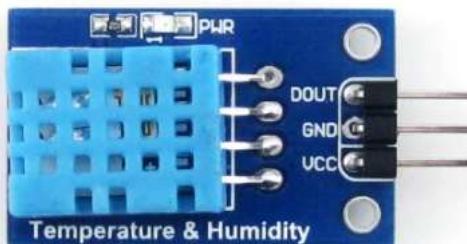
Bài 3.

Dự án Trực quan hoá dữ liệu nhiệt độ

- | 3.1. Trực quan hoá dữ liệu
- | 3.2. Dự án Toy

Cảm biến nhiệt độ & độ ẩm

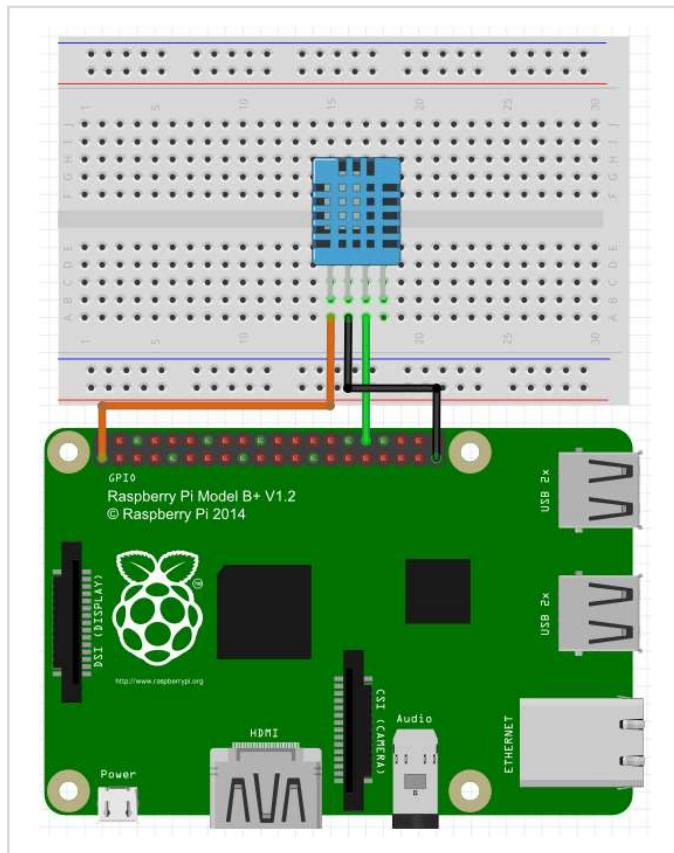
| DHT-11



- ▶ DHT-11 là một cảm biến kỹ thuật số đo nhiệt độ và độ ẩm và trả lại giá trị đo được ở đầu ra dưới dạng số.
- ▶ Vì giá trị đo được trả về qua chân DOUT, nó có thể được đọc trực tiếp mà không cần qua mạch chuyển đổi sử dụng IC MCP3008.
- ▶ Bây giờ hãy tiến hành một dự án Trực quan hoá dữ liệu sử dụng DHT-11 và một Nút bấm.

Trực quan hóa nhiệt độ

I Tổng quan dự án



- ▶ Dự án thực hiện biểu diễn trực quan hóa dữ liệu cảm biến nhiệt độ - độ ẩm theo thời gian thực.
- ▶ Sơ đồ mạch
 - Chân VCC của DHT-11 – Chân 5v của Raspberry Pi
 - Chân GND của DHT-11 – Chân GND của Raspberry Pi
 - Chân DOUT của DHT-11 – Chân GPIO 20 của Raspberry Pi

| Nhập các lệnh sau để giảm nguy cơ xảy ra lỗi.

- ▶ Cài đặt thư viện hỗ trợ lập trình, tính toán xử lý với ma trận, trước khi thực hiện dự án

```
$ sudo apt-get install libatlas-base-dev
```

- ▶ Loại bỏ xung đột tiềm ẩn với thư viện numpy hiện có.

```
$ pip3 uninstall numpy
```

- ▶ Cài đặt thư viện trực quan hóa.

```
$ sudo apt install python3-matplotlib python3-numpy python3-pandas
```

- | ~/Toy_projects/temp_visualization.py (1)
 - ▶ Tạo tập lệnh ~/Toy_projects/temp_visualization.py.

```
1  from matplotlib import pyplot as plt
2  from matplotlib import animation
3  import numpy as np
4  import Adafruit_DHT
5
6  sensor = Adafruit_DHT.DHT11
7  pin = 20
8
9  fig = plt.figure()
10 ax = plt.axes(xlim=(0, 30), ylim=(15, 45))
11 max_points = 30
12 line, = ax.plot(np.arange(max_points),
13                  np.ones(max_points, dtype=np.float) * np.nan, lw=1, c='blue', marker='d', ms=2)
```

| ~/Toy_projects/temp_visualization.py (1)

- ▶ Tạo tập lệnh ~/Toy_projects/temp_visualization.py.

```
1  from matplotlib import pyplot as plt
2  from matplotlib import animation
3  import numpy as np
4  import Adafruit_DHT
5
6  sensor = Adafruit_DHT.DHT11
7  pin = 20
8
9  fig = plt.figure()
10 ax = plt.axes(xlim=(0, 30), ylim=(15, 45))
11 max_points = 30
12 line, = ax.plot(np.arange(max_points),
13                  np.ones(max_points, dtype=np.float) * np.nan, lw=1, c='blue', marker='d', ms=2)
```

- dòng 1-2: Nhập thư viện matplotlib và nhập pyplot và hoạt ảnh. Trong trường hợp pyplot, đặt tên được gọi là plt.
- dòng 3: Gọi vào numpy với tên mã là np để tạo và tính toán ma trận nhanh chóng.
- dòng 4: Gọi vào Adafruit_DHT để sử dụng DHT-11 thuận tiện hơn.
- dòng 6-7: Gán Cảm biến DHT-11 cho biến cảm biến và assgin số chân GPIO cho biến pin.
- dòng 9: Tạo đối tượng matplotlib.pyplot.
- dòng 10: Đặt trực pyplot.
- dòng 11: Đặt lượng dữ liệu cần thể hiện bằng số điểm.
- dòng 12-13: Vẽ với max_points thành phần x có dạng (0, 1, 2... max_point-1) và các thành phần y có max_points trong ma trận chứa 1.

| ~/Toy_projects/temp_visualization.py (2)

```
15     def init():
16         return line
17
18     h, t = Adafruit_DHT.read_retry(sensor, pin)
19
20     def [animate](i):
21         h, t = Adafruit_DHT.read_retry(sensor, pin)
22         y = t
23         old_y = line.get_ydata()
24         new_y = np.r_[old_y[1:], y]
25         line.set_ydata(new_y)
26         return line,
27
28     anim = animation.FuncAnimation(fig, animate, init_func=init, frames=200, interval=20, blit=False)
29     plt.show()
```

| ~/Toy_projects/temp_visualization.py (2)

```
15     def init():
16         return line
17
18     h, t = Adafruit_DHT.read_retry(sensor, pin)
19
20     def animate(i):
21         h, t = Adafruit_DHT.read_retry(sensor, pin)
22         y = t
23
24         old_y = line.get_ydata()
25         new_y = np.r_[old_y[1:], y]
26         line.set_ydata(new_y)
27
28     anim = animation.FuncAnimation(fig, animate, init_func=init, frames=200, interval=20, blit=False)
29     plt.show()
```

- dòng 15-16: Định nghĩa hàm để khởi tạo.
- dòng 18: Lưu giá trị độ ẩm và nhiệt độ từ cảm biến DHT-11 vào 2 biến h và t tương ứng.
- dòng 21-22: Tương ứng giá trị nhiệt độ cảm biến với y để hiển thị động.
- dòng 23-25: Lấy dữ liệu theo trục y của biến line, di chuyển dữ liệu của np.r_ sang phải và thêm dữ liệu y vào để tạo thành một tập mới new_y, trước khi thiết lập mới dữ liệu theo trục y cho biến line.
- dòng 28: xử lý biểu diễn động (animation) cho trực quan hóa dữ liệu.
 - fig là một đối tượng plt
 - animate là một chức năng của animation
 - Hàm init để khởi tạo, cho tham số init_func param
- dòng 29: In dữ liệu trực quan hóa đã tạo lên một cửa sổ giao diện.

Kết quả của quá trình trực quan hóa dữ liệu nhiệt độ

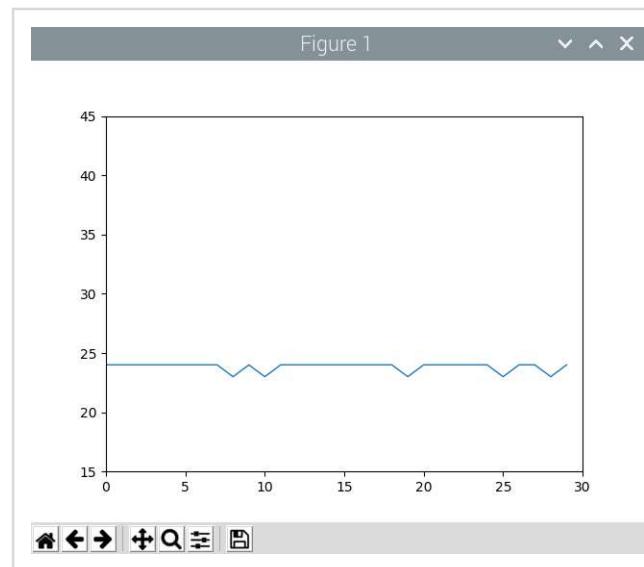
| Terminal

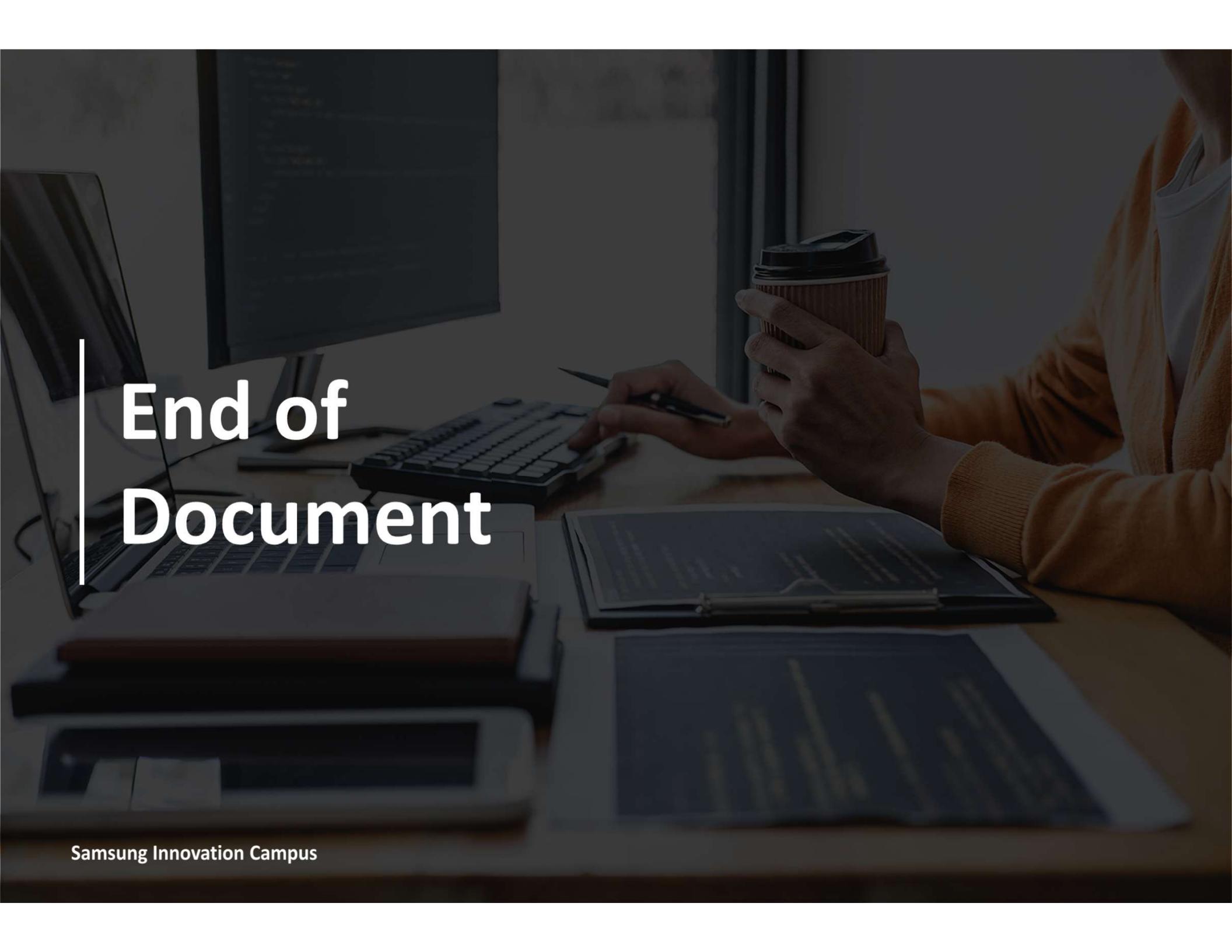
- ▶ Thực thi tập lệnh Python.

```
$ python3 temp_visualization.py
```

```
pi@raspberrypi:~/Toy_projects $ python3 temp_visualization.py
```

- ▶ Quan sát dữ liệu nhiệt độ được trực quan hóa trên màn hình. Có thể thấy thành quả dự án Trực quan hóa dữ liệu nhiệt độ, trong đó các giá trị đo được dịch chuyển theo thời gian thực tế (theo thời gian thực).



A photograph of a person working at a desk. They are wearing a light-colored long-sleeved shirt and are holding a brown paper coffee cup with a black lid in their right hand. Their left hand is on a keyboard. In the background, there are two computer monitors on stands. The monitor on the left shows a dark interface with some text and icons. The monitor on the right is mostly black. On the desk in front of the monitors, there is a stack of papers or books. The overall scene suggests a professional or academic environment.

End of Document



Together for Tomorrow!
Enabling People

Education for Future Generations

©2022 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of book.

This book is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.

To use this book other than the curriculum of Samsung Innovation Campus or to use the entire or part of this book, you must receive written consent from copyright holder.