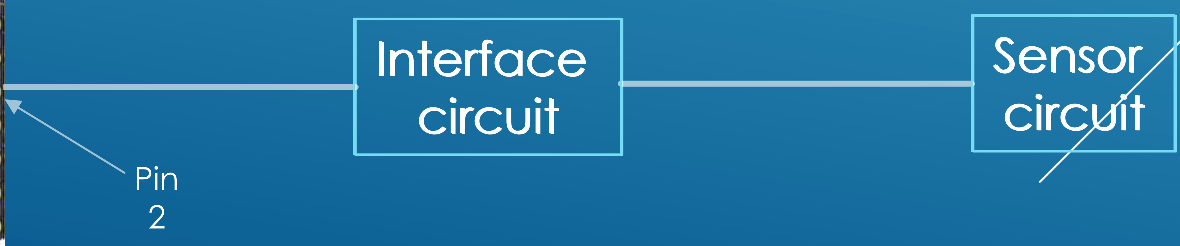
USING SENSORS AND OUTPUTS

SENSORS GENERATE DATA

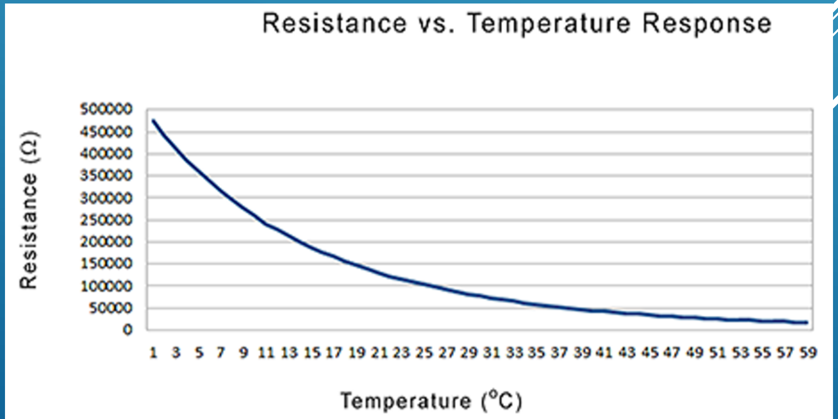
- A sensor converts some physical quantity into an electrical signal
  - Temperature, Light
  - Sound, Vibration
  - Pressure, Force
  - Humidity, Wind
  - Physical contact
- It is often necessary to add additional components (interface circuit) to convert the change into a useful voltage
- A microcontroller is usually equipped with multiple types of input pins, allowing a range of signal types to be measured



SENSOR OUTPUTS

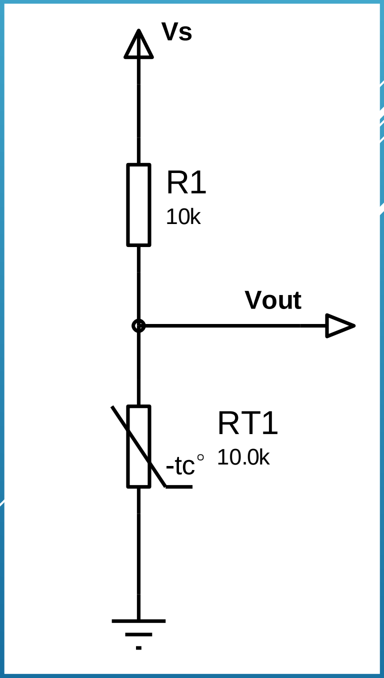
- The output from a sensor will be a signal of some kind, and it will vary with the physical quantity being 'sensed'
- There are different types of sensor output:
  1. **Binary** (two-state) – too high/too low, quantity detected, weight exceeds some reference value
  2. **Analogue** – voltage varies directly and proportionately with quantity, e.g. an LM35 outputs 10 mV/°C.

     In some instances, the output may not be linear:

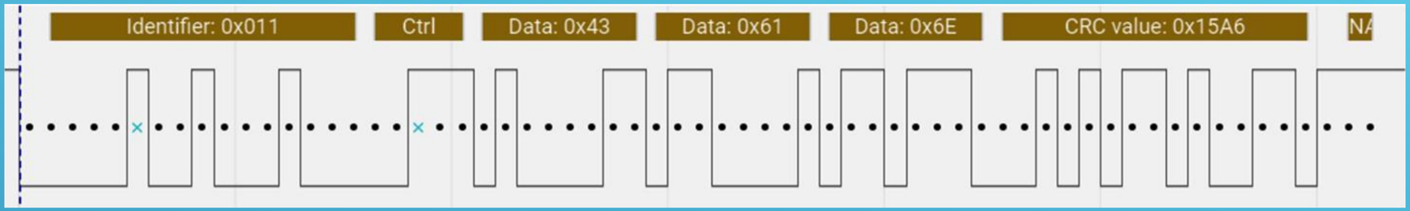


SENSOR OUTPUTS - continued

- When the response is non-linear, software in the microcontroller must be written to convert the output, e.g. a look-up table
- The ‘output’ may be a change in resistance, rather than a voltage, e.g. temperature-dependent resistor. A circuit is needed to convert the change to a voltage. The manufacturer will usually tell you how to do this on an associated ‘data sheet’, and may even suggest a suitable circuit



SENSOR OUTPUTS - DIGITAL



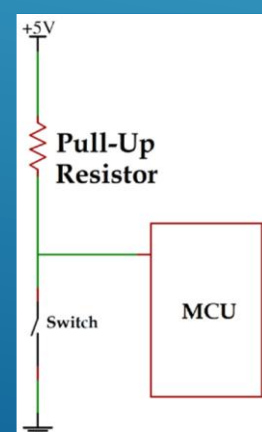
3. **Digitally-encoded**. This kind of sensor often contains a processor, which provides conversion, makes the measurement, and outputs a digitally-encoded signal.
   This type tends to be most accurate

   Drivers: the above type often have 'libraries' or driver-software to convert the digital pulses back into numbers. Main types are SPI, I^2C, DS1820, DHT11/22

EXAMPLES – 1. BINARY

- Any kind of switch has a ‘binary’ output, either ON or OFF. Some sensors behave like this, e.g. PIR (heat sensitive ‘person-detector’)
- A switch will only output a voltage if you provide an interface circuit – a resistor and voltage source is enough
- Choose a pin on the ESP32 to be the input pin

DC 12V Automatic PIR Infrared Motion Sensor Intelligent Switch Black 180 Degree Infrared Ray Human Body...



READING BINARY INPUTS

- Binary inputs are the easiest to read
- Configure a pin of your choice (23) to be input [line 3]
- Value of pin (1 or 0) can be printed out using: `print(pin.value())`
- 'pin' is the name we chose on line 2

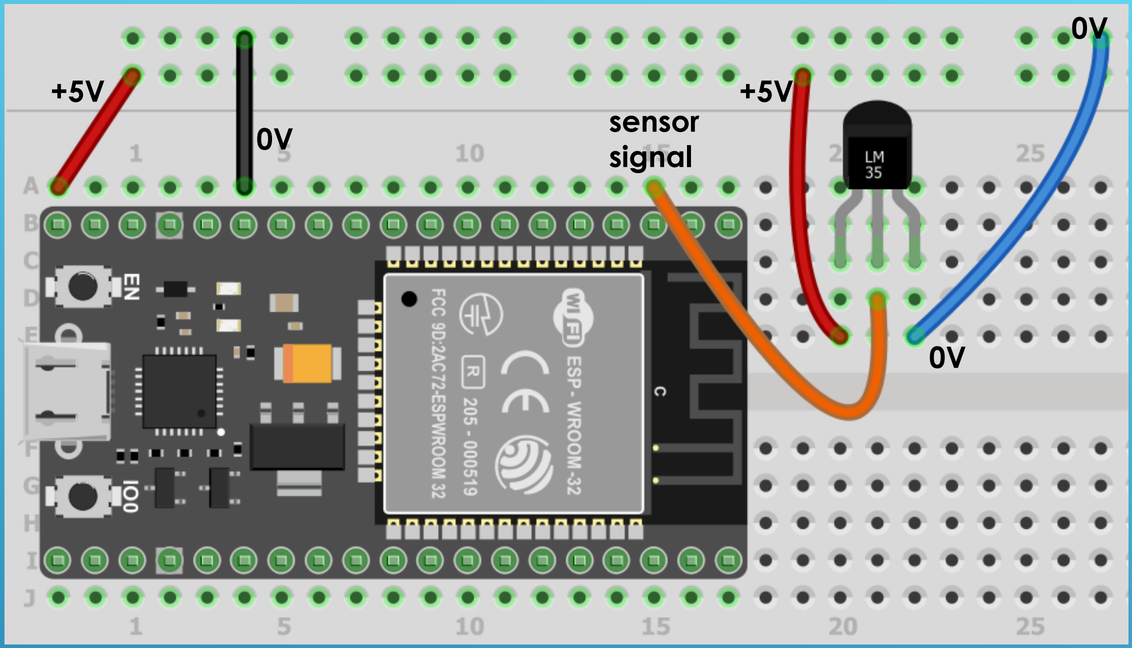
```
1 import time
2 from machine import Pin
3 pin = Pin(23, Pin.IN)
4
5 while(True):
6     print(pin.value())
7     time.sleep(3)
```

Shell

```
>>> %Run -c $EDITOR_CONTENT
 1
 1
 1
 0
 0
 0
 1
 1
```

Value of pin changes

EXAMPLES 2.ANALOGUE LM35 TEMPERATURE SENSOR



- LM35 temperature sensor is a 3-pin device, requiring +5V supply
- Pin1: +5V supply
- Pin2: temperature sensor signal, 10mV/degree C
- Pin3: 0V Ground
- Pin C1 converts analog voltage, V -> number, n

  where V = n x 1.0/4096

READING ANALOGUE INPUTS

- Analogue signals are read with a special input pin called an ‘ADC’ (analogue to digital converter)
- Pin 36 is an ADC-type pin (check ESP32 information)
- Line 6: pin36 is given the name ‘adc’ and configured to read analogue voltages
- Line 8: pin36 is read and the voltage is output – in this case it will need to be scaled to read in degrees centigrade

```
1 # ADC test
2
3 from machine import Pin, ADC
4 import time
5
6 adc = ADC(Pin(36))
7 while True:
8     print(adc.read())
9     time.sleep(1)
```

Shell

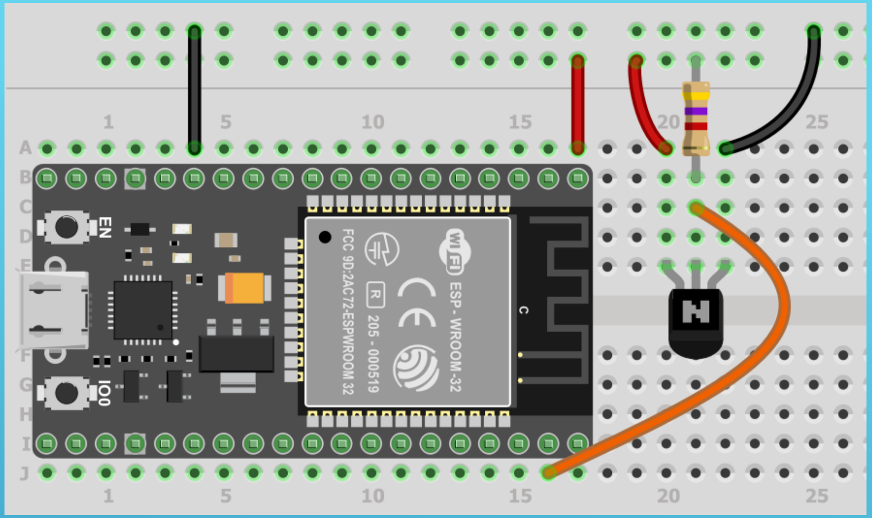
```
>>> %Run -c $EDITOR_CONTENT

 629
 624
 619
 625
 643
 701
 731
 768
 796
 811
 834
 849
 833
```

768 = 18.75°C

SCALING THE INPUT

- The ADC reads a voltage 0 –> 1 V as a number between 0 – 4096 (ESP data sheet, or see *Micropython* web site)
- So actual voltage = adc/4096
- LM35 outputs 10mV / degree,
  i.e. adc = temp/100
- Temperature reading = adc x 100/4096
- If adc = 768, then
  T = 768 x 100/4096 = 18.75°C



3. DIGITALLY ENCODED

- DS18B20 is a temperature sensor that outputs a digitally encoded temperature in degrees C (no scaling needed)
- Read data sheet to see how to connect (suitable connections shown above)
- Need Python 'library' modules to read the digitally-encoded data – in this case we must `import onewire and ds18x20` modules
- Note: we are using input pin GPIO 23 (orange wire above)

READING DS18B20

```
1  import machine, onewire, ds18x20, time
2
3  led = machine.Pin(2, machine.Pin.OUT)
4  led.value(0)     #just to be sure
5  pin = machine.Pin(23)
6  wire = onewire.OneWire(pin)
7  ds = ds18x20.DS18X20(wire)
8
9  addr = ds.scan().pop()
10
11 for i in range(5):
12 ds.convert_temp()
13 led.value(1)
14 time.sleep_ms(750)
15 led.value(0)
16 temp = ds.read_temp(addr)
17 print(temp)
18 time.sleep(3)
```

- Code looks more complex because modules are imported

3: use pin 2 as LED output

5: pin 23 is sensor input pin

6,7: creates digital input reader on pin 23

9: gets address of sensor (checks one is present)

12-17: reads sensor and flashes LED to confirm

18: 3 second wait

All this code is from Micropython website

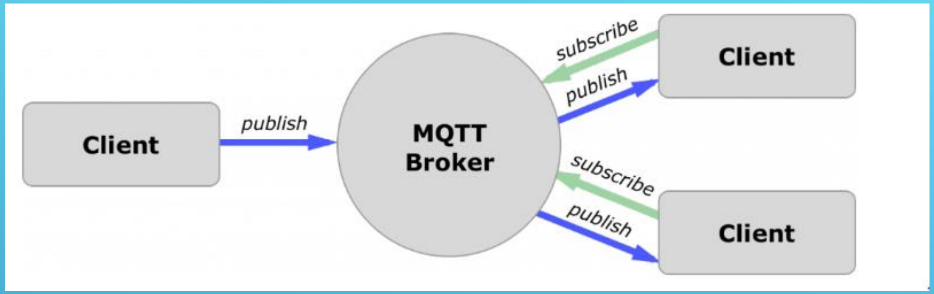
RUNNING THE CODE

```
>>> %Run -c $EDITOR_CONTENT
 21.875
 21.875
 21.875
 23.6875
 26.125
>>>
```

I touched sensor with my finger

- Although the code is more complex, it can be copied directly from Micropython web site
- Check wiring – you need a resistor between the middle pin and power supply (see data sheet)
- Check wiring – you must not get outer pins the wrong way around (see data sheet)
- Output is already converted to °C, and is formatted to several decimal places
- LED flashes each time a reading is taken

USING MQTT



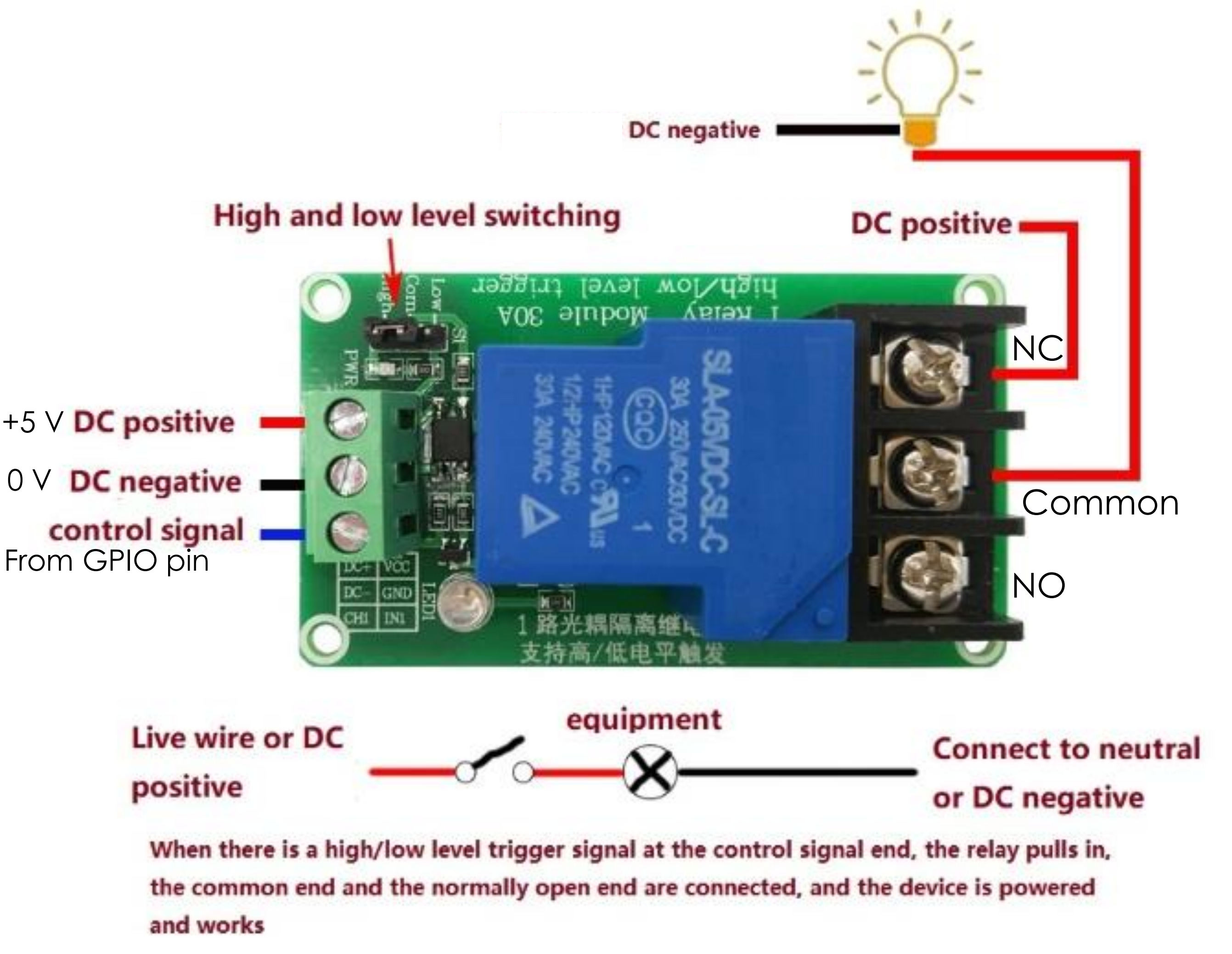
- Sensor readings should be stored as variables
- MQTT publish commands can then send these readings to a broker using a specific (unique) topic name
- Other clients can then ‘subscribe’ to that topic in order to get the reading
- Websites like ‘ThingSpeak’ can then analyse your data and plot graphs over time

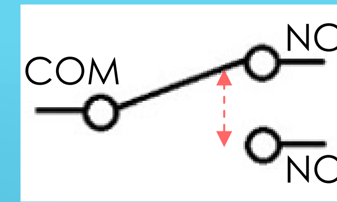
WHICH SENSOR?

- Research information about the quantity that you wish to measure, and the available sensors
- Find examples of where people have used the sensor in their system, and see how they did it
- Learn, but do not copy
- **WARNING**: ESP32 pins may be damaged if you apply more than 3.3 V – always check data sheets to ensure voltage is not above this

DRIVING OUTPUT LOADS

- 'Load' is anything that you might want to operate using the ESP32
  - Led, light bulb
  - Motor, pump, solenoid/actuator
- The ESP32 can supply a maximum current of only 40mA (0.04 A) which will operate an LED or another logic circuit, but not much else
- To drive something like a motor (check the operating current), you can use a relay switch
- Many relays come on small circuit boards, and have some additional circuitry to allow the relay to be operated by a logic 1 from an ESP32 or other microcontroller. They usually have one 'changeover switch' for controlling a load



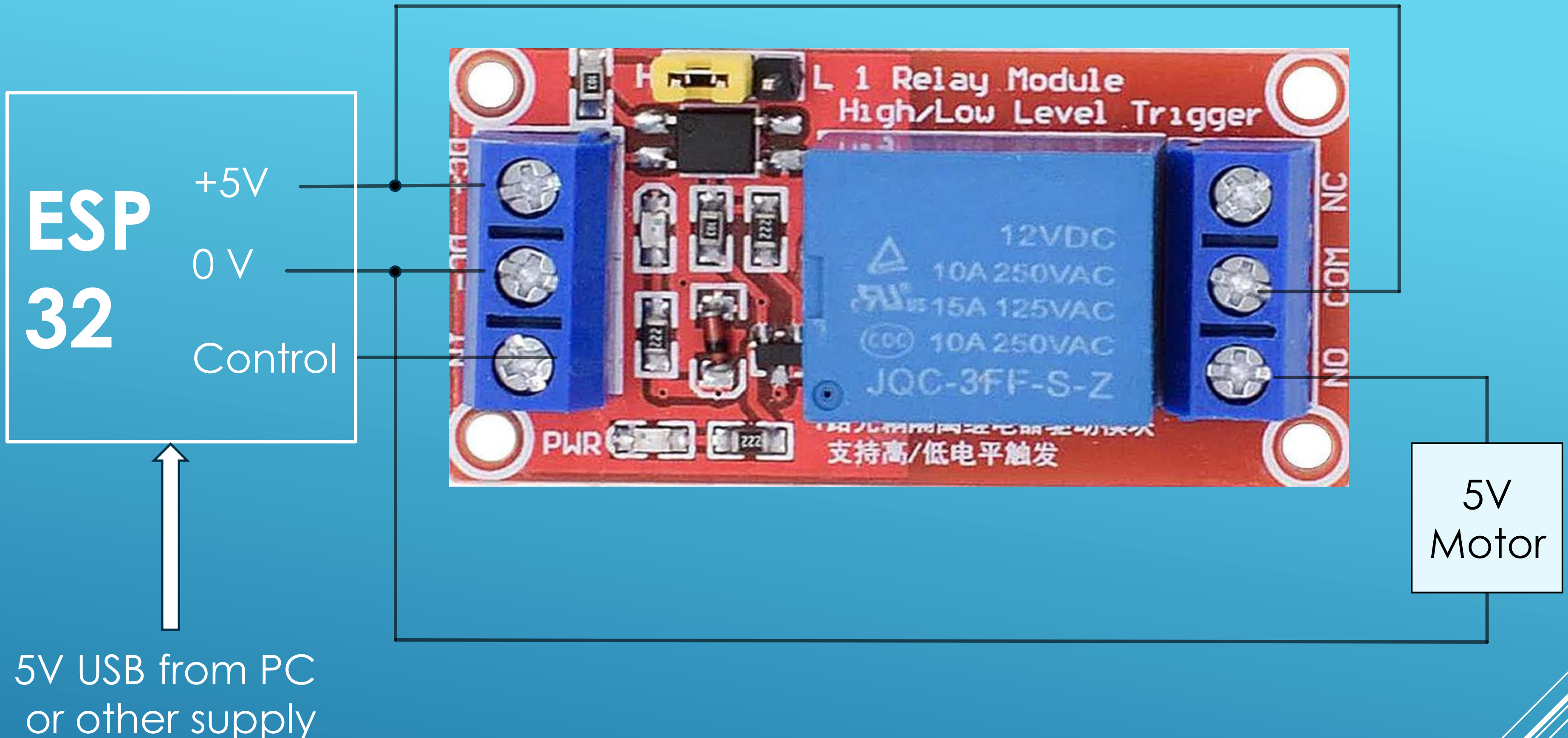


COMMON can be connected to NC or NO

NC=normally closed, Control=logic 0

NO=normally open Control=logic 1 will connect it to COM

The relay module can be powered from the +5V and 0V pins of the ESP32 (this will draw current directly from the USB cable plugged in to the ESP32) When the relay switches, it takes a large pulse of current; this may cause the ESP32 to reset.



- Example wiring for 5V operation from USB supply on ESP32
- Motor is energised when control = 1

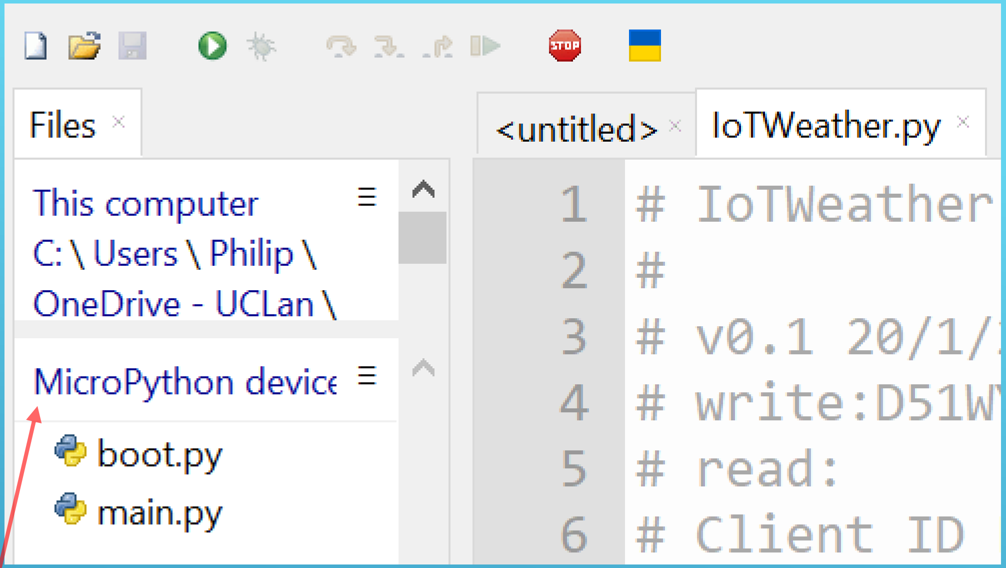
POWERING THE LOAD

5V USB supply

Wall-plug USB

- The load may require more than 5V to operate, in which case an additional battery/supply will be needed – it should be wired to the relay pins (COM, NC or NO)
- If the load only needs 5V and does not draw more than about 50mA of current, then it can be powered from the +5 V pin of the ESP32 (like the relay module)
  BUT, if it draws a lot of current there may not be enough available for the ESP32, which will protect itself by doing a reset
  - In this situation, use a plug-in 5V USB or wall-plug supply – these normally provide 1 or 2 amps
  - In this case, you cannot communicate from the PC – the USB is now plugged into the power supply

main.py



- The ESP32 normally gets its 'code file' from Thonny, and therefore cannot run without Thonny
- The ESP has its own file store, and can store files there in 'flash' memory without requiring power
- When the ESP is powered up from USB 5V supply, it first looks in its own flash memory for code, and if it finds a file called '*main.py*' it will run it
- If you save your finished program internally as main.py, then the ESP does not need to be 'hosted' by a PC running Thonny