Οι ανιχνευτες σωματιδιων του LHC ειναι απο τις εντυπωσιακοτερες μηχανες... (LHC is the most impressive machine ever built)

...και ισως προσφερονται σαν πιθανες ιδεες για μετρησεις με τους μαθητες σας and will be possibly a great way to attract students in STEM activities)

(https://www.mycloud.ch/s/S00CA3ADC5EB87832A4A2D2BF2BFB58126F8B438C91)

Α. Τσιρου (ΕΚΠΑ), Piero Giorgio Verdini (INFN Pisa)

https://www.mycloud.ch/s/S00CA3ADC5EB87832A4A2D2BF2BFB58126F8B438C91

The ones that have their own laptops must install the following(the CERN laptops have them, but then they stay at CERN…):

Arduino IDE 2.1.1

Arduino IDE libraries:

AHT20 1.0.1 by Dvarrel (https://github.com/dvarrel/AHT20)

BMP180MI 1.0.1 by Gregor Christandl https://bitbucket.org/christandlg/bmp180mi

LM75A Arduino Library 1.0.1 by M2M Solutions AB (https://github.com/m2m-solutions/M2M_LM75A)

Το CERN ειναι το μερος που «παιρνουν σαρκα και οστα» τρια πραγματα:

1. Γινεται βασικη ερευνα για να επιβεβαιωσει η απορριψει τα μοντελα που περιγραφουν το συμπαν και την εξελιξη του. Η ερευνα αυτη απαιτει την δημιουργια, συντηρηση, και λειτουργια οργανων που μπορουν να δωσουν δεδομενα σε στατιστικα ικανες ποσοτητες για τις παραπανω μελετες.
2. Η δημιουργια/εξελιξη και χρηση τεχνολογιας ειναι δεδομενη και η εκφραση «τεχνολογια αιχμης » ειναι σχεδον κενη διοτι το προσωπικο του CERN δημι-ουργει διαρκως τεχνολογια για να κανει δυνατη την πραγματοποιηση του πρωτου στοχου. Η τεχνολογια αυτη εχει ξεκιναει απο το πραγματικο βαρυ hardware (τροποι χτησιματος και στηριξης) μεχρι το πιο αφαιρετικο software (ποιος δεν ξερει το WEB!).
3. Οι ανθρωποι δουλευουν πραγματικα μαζι και αποδεικνυουν, με συγχωρειτε για την παραφραση το:

«Εξ ανθρὠπου τα χείρω...Και εξ ανθρὠπου τα κρείττω»

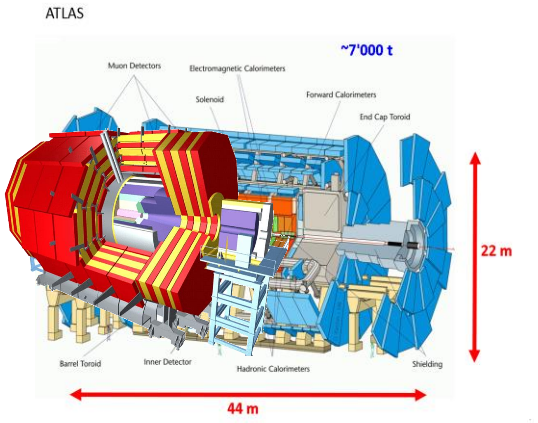
Ετσι, ειναι το μερος που οι νεοι ανθρωποι λατρευουν να δουλευουν ΜΑΖΙ, και ειναι το μερος που λατρευει τους νεους ανθρωπους.

Γι' αυτό εισαστε και ειμαστε εδω.....

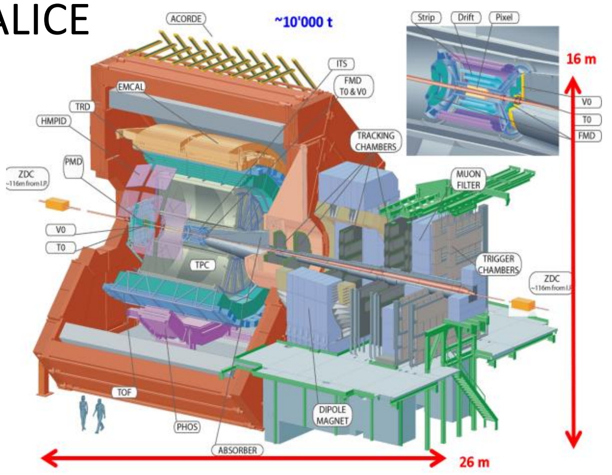
What do the detectors “do”:

1. They force the particles created during collisions to interact with matter losing energy via different, well parametrized, processes.
2. They record, practically taking pictures of the whole process.
3. Their construction and operation is such that they can perform adequately for providing the statistics and resolution required.

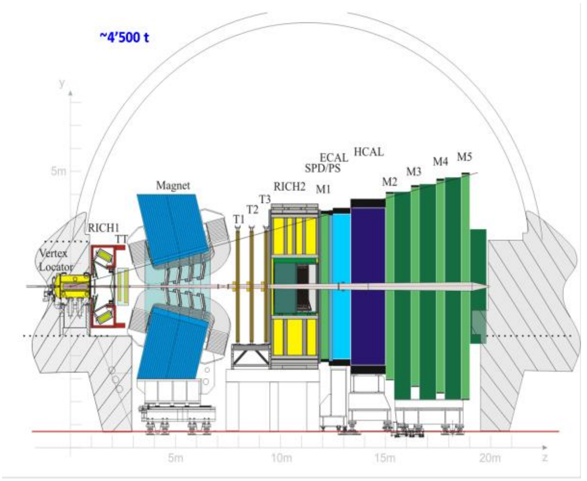
ATLAS



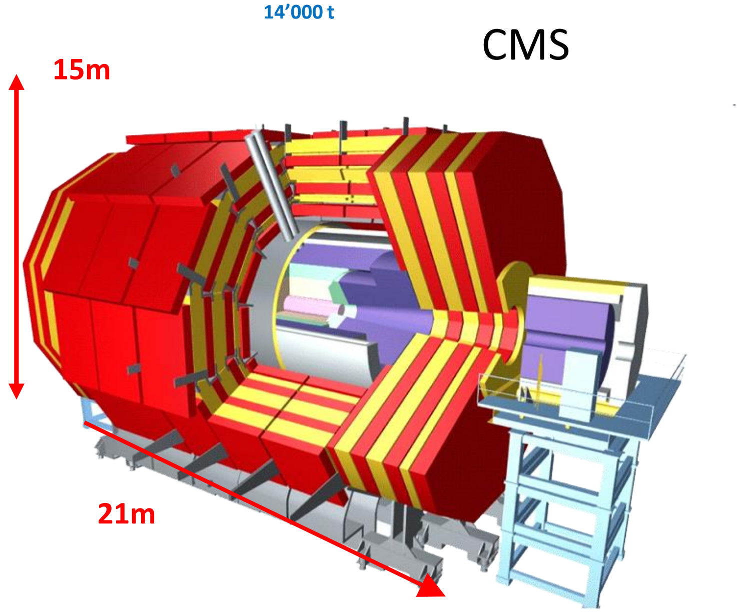
ALICE

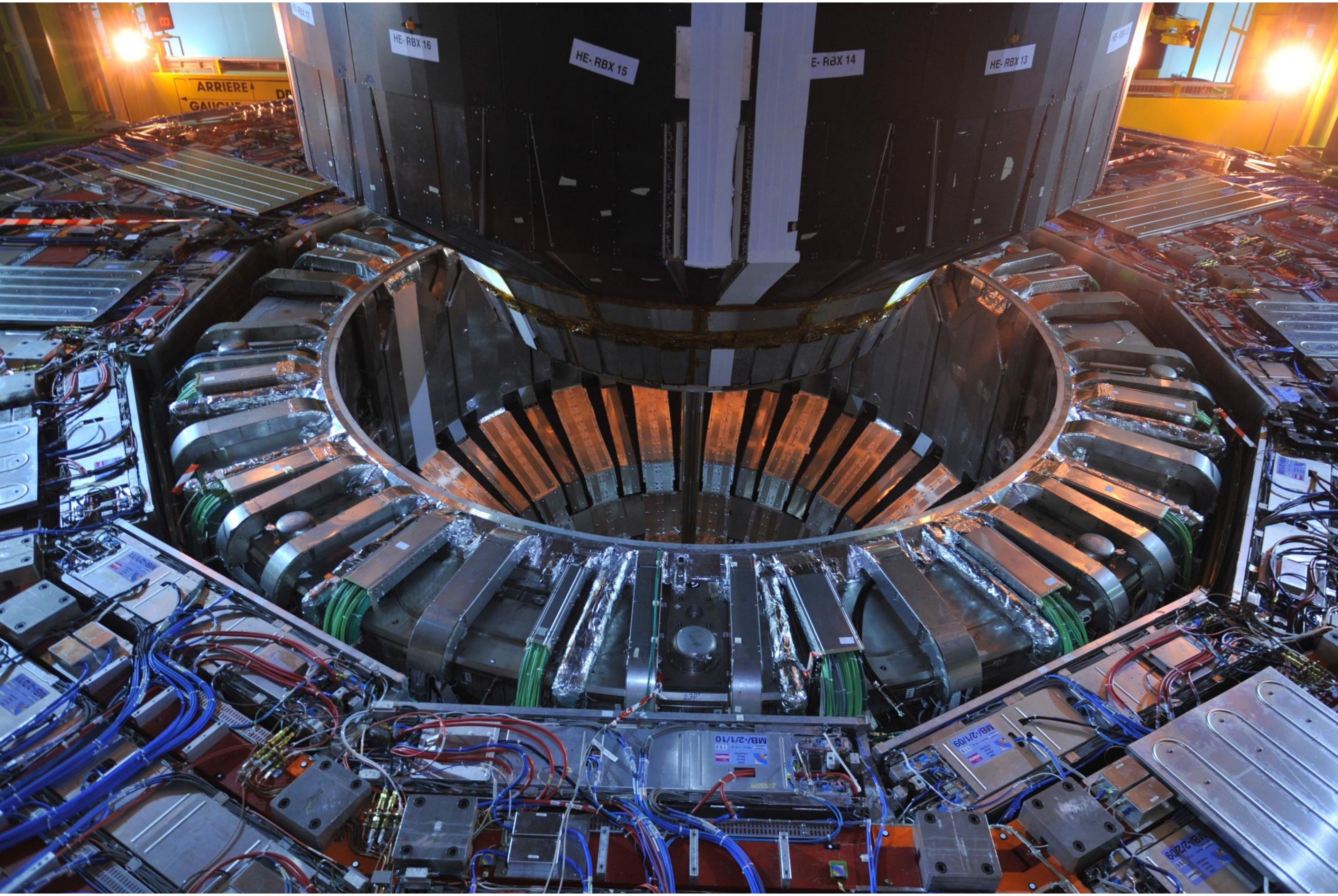


LHCb

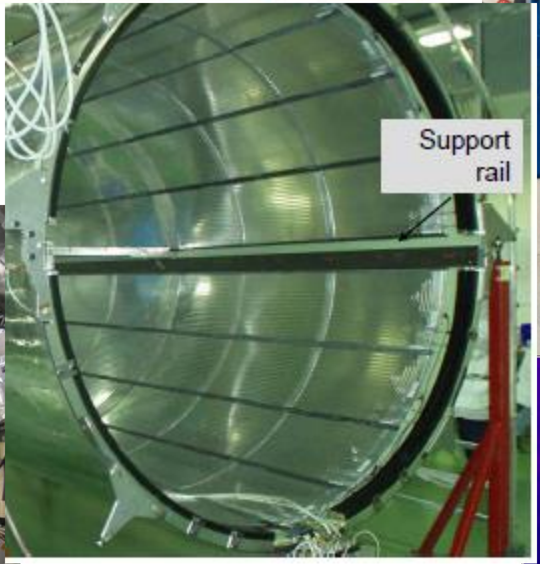


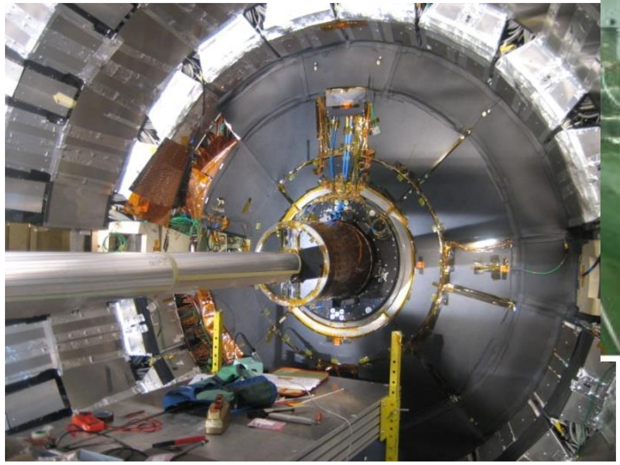
CMS

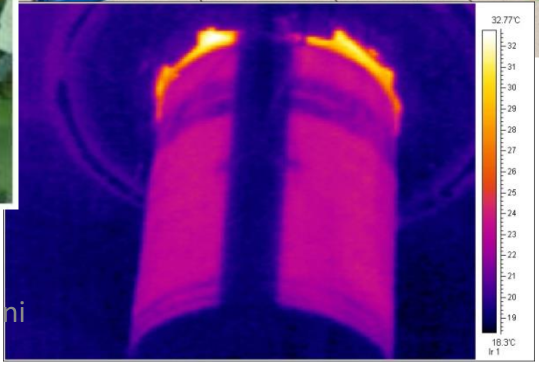



HE-RBX 16
HE-RBX 15
E-RBX 14
HE-RBX 13
ARRIERE
GAUCHE
MB/-2/1/10

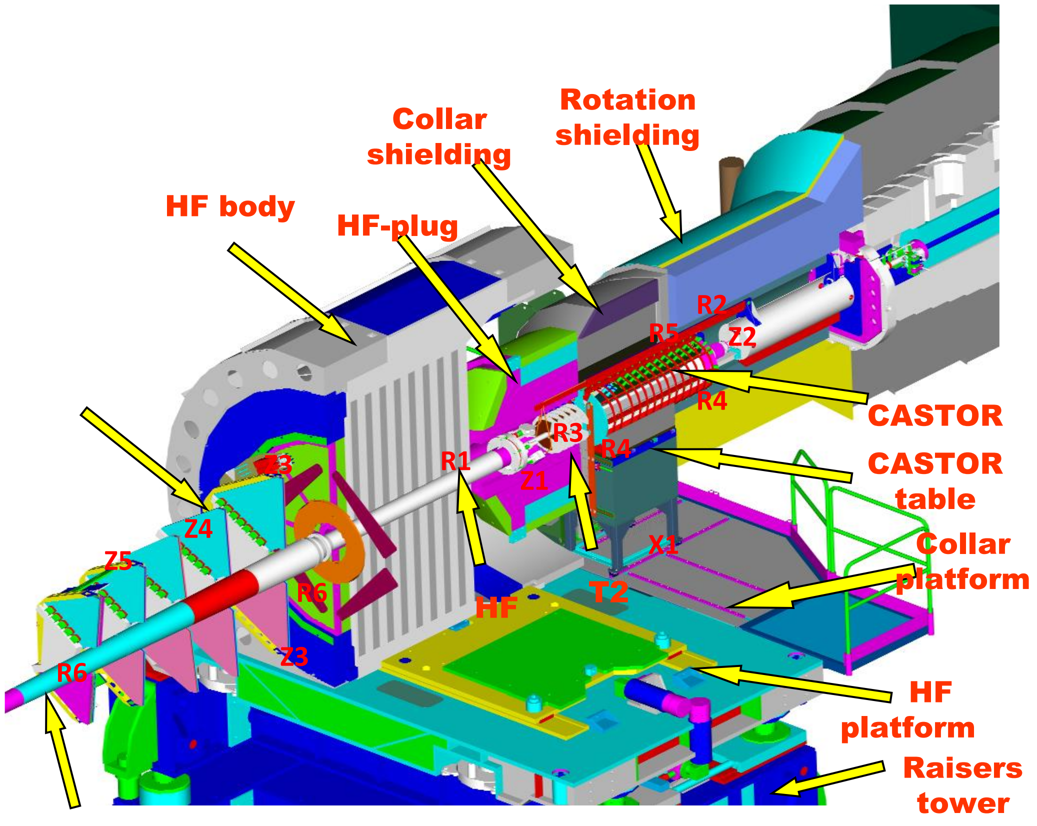
Smart insulator systems(~ (-20 -40)^{0}C to ~ +17(+9)^{0}C ECAL)





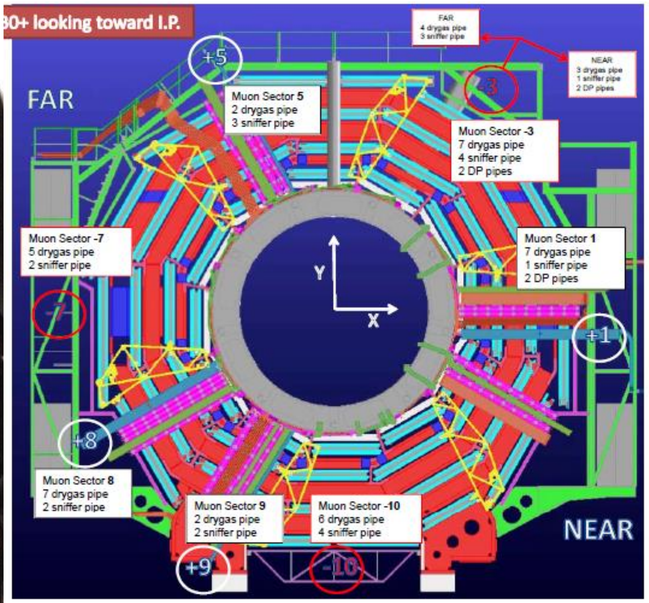


•Movement control systems (subdetectors weigh up to 2 tons, how do you move them without sensors when opening/closing the detector?)



…but finding a sensor and readout system for "us" because of magnetic field and radiation is very difficult (we manage) but not for you that you can pretend doing the same thing with your students using easier to have equipment….

•Environmental control and monitoring systems(dry air, Nitrogen, temperature)



Dewpoint problem for all systems that are cooled below ~13 deg C…

Η πλατφορμα-ecosystem(software+hardware+sensors, κοινοτητα) Arduino https://www.arduino.cc/εχει δωσει τεραστια ωθηση στην μετρηση, σε ολα τα εργαστηρια, τις σχολικες ταξεις και τα ανωτερα ιδρυματα. Το Arduino hardware ειναι προσιτο και απευθυνεται και σε μη ειδικους. Το περιβαλλον προγραμματισμου ειναι πολυ πιο φιλικο και τα εργαλεια που παρεχονται διευκολυνουν την χρηση της πλατφορμας απο ολους. Στην αρχη τα Arduino προγραμματιζονταν σε C++ μεσα στο περιβαλλον Arduino.ide. Αυτο Το περιβαλλον βοηθαει φοβερα στον προγραμματισμο αλλα το προβλημα της γλωσσας παραμενει διοτι η C++ δουλευεται απο ειδικους μονον. Οταν ομως τα Arduino αρχισαν να χρησιμοποιουνται στα σχολεια υπηρξαν διαφορες προσπαθειες για απλες γλωσσες προγραμματισμου (graphical –Scratch και Blockly). Οι γλωσσες αυτες δεν μπορουν να χρησιμοποιηθουν σε ολο το φασμα δυνατοτητων του Arduino και απαιτουν πολλες φορες ειδικες εκδοσεις του hardware. Επισης, η εκμαθηση τους για μαθητες που προσβλεπουν σε Τεχνικο/επιστημονικο μελλον προσφερουν μια κακη προπαιδευση.
Στο μεταξυ, η γλωσσα προγραμματισμου Python (https://micropython.org/) **εκανε την επανασταση (μια ακομα!) στον χωρο προγραμματισμου, με διαρκως περισσοτερους οπαδους-εφαρμογες. Στην Python (διερμηνευμενη γλωσσα) ο κυκλος προγραμματισμου ειναι πολυ πιο συντομος και αμεσος.**
Η Python μεταφυτευτηκε στο Arduino hardware σαν μPython και ειναι μια γλωσσα που παρεχει την δυνατοτητα πληρους αξιοποιησης της πλατφορμας και παρα πολλα ηλεκτρονικα κομματια παρεχουν το software ετοιμο. Επιπλεον ειναι η ιδανικη γλωσσα να επιδειξουμε σε μαθητες που εχουν τεχνολογικες/επιστημονικες ανησυχιες.

Το Arduino και το προγραμμα LED blink

Η πραγματικη επανασταση του συστηματος Arduino ειναι ολα τα επιπεδα αφαιρεσης αναμεσα στην δουλεια της CPU και στα ηλεκτρονικα κομματια , (I/O pins).

Where before you would need to manipulate the registers with cryptic instructions such as

```
DDRD = 0b11111111; // declaring port D as output, because we know the LED is connected to port D, pin 0
PORTD |= 0b10000000; // pin 0 of port D set HIGH, all other pins left unchanged, this is an OR operation
_delay_ms(1000); // delay of one second
PORTD &= 0b01111111; // pin 0 of port D set LOW, all others left unchanged, this is an AND operation
```

now you can achieve the same result with more meaningful and understandable instructions:

```
pinMode(LED_BUILTIN, OUTPUT); // we do not need to remember which port and pin the LED is connected to
digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level), no need to manipulate bits
delay(1000);                                     // wait for a second
digitalWrite(LED_BUILTIN, LOW);   // turn the LED off (LOW is the voltage level), again no bit manipulation needed
```

Αναλυση προγραμματος

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}


void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000
}
```

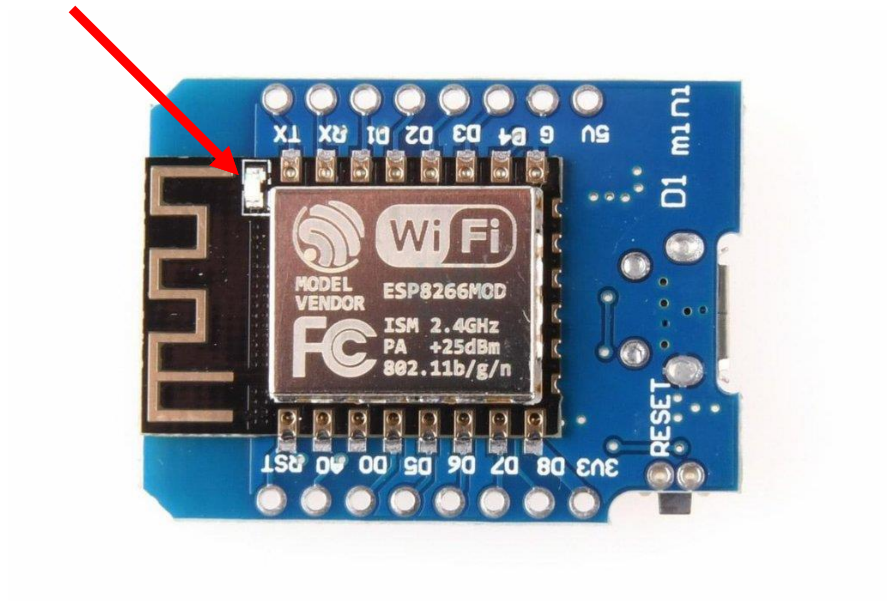
The setup() function is executed only once, at the beginning of the program. In this case, it handles the mechanisms needed to make the pin the on-board LED is connected to an output (so we can turn the LED on and off)

The loop() function, instead, is supposed to keep running "forever". In this case, it makes the LED pin take a HIGH value (+5 V) so the LED turns on, waits for 1 second, makes the LED pin take a LOW (0 V) value so the LED turns off, then waits 1 more second before ending and restarting again, and again, and again...

Το ιδιο "blinky" παραδειγμα σε MicroPython

- We can now try the LED blink example in MicroPython: in Thonny, create a new file (click on the blank page icon) and type in the following, being very careful to respect the indentation (use the Tab key "-->"):

```
import time
from machine import Pin
ledpin = machine.Pin(2, Pin.OUT)
while(True):
    ledpin.on()
    time.sleep(0.5)
    ledpin.off()
    time.sleep(0.5)
```



Τι συμβαινει στο"blinky" προγραμμα;

import time
from machine import Pin
	These two lines are similar to #include <library.h> in C/C++

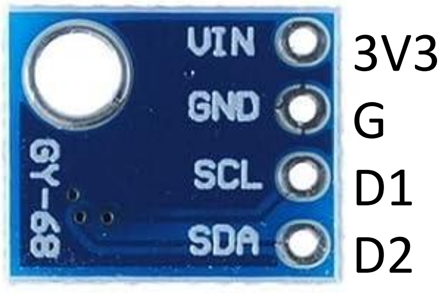
ledpin = machine.Pin(2, Pin.OUT)	here we define the LED pin
while(True):	now we loop forever
ledpin.on()	turn the pin to 1 (LED off)
time.sleep(0.5)	wait half a second
ledpin.off()	turn the pin to 0 (LED on)
time.sleep(0.5)	wait half a second

Οι αισθητηρες που θα χρησιμοποιησουμε μετρανε ο,τι χρειαζομαστε και με τους ανιχνευτες

- **Θερμοκρασια**
- **Αποσταση**
- **Υγρασια**
- **ροη**
- **πιεση**
- **Και ο,τι αλλο….**

Η διαφορα ειναι οτι εσεις μπορειτε να χρησιμοποιητε ψηφιακους αισθητηρες... εμεις ΟΧΙ.
Παρακατω βλεπετε αισθητηρες που χρησιμοποιουνται καθημερινα απο ολους μας....

Ο αισθητηρας BMP180 (θερμοκρασια, πιεση)



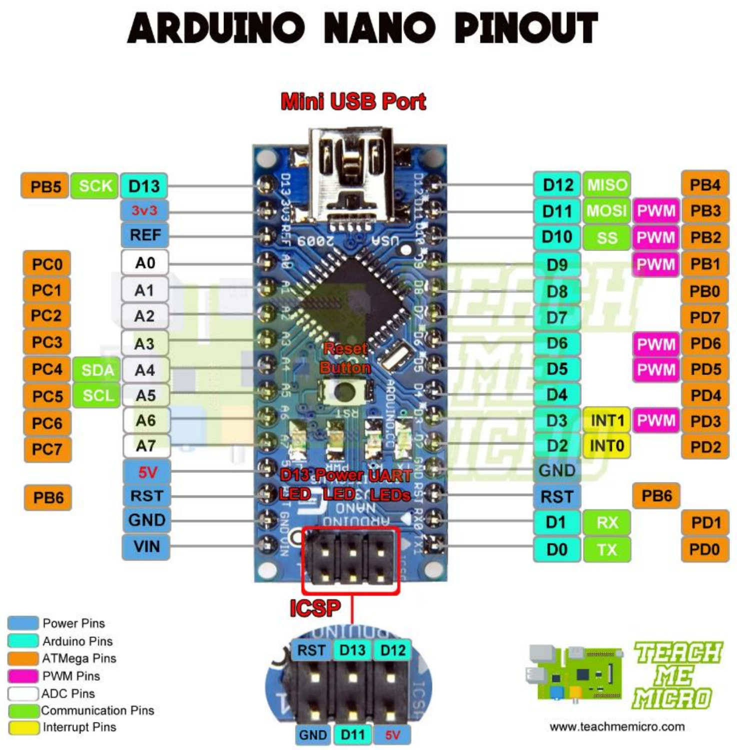
BMP180 (Bosch):

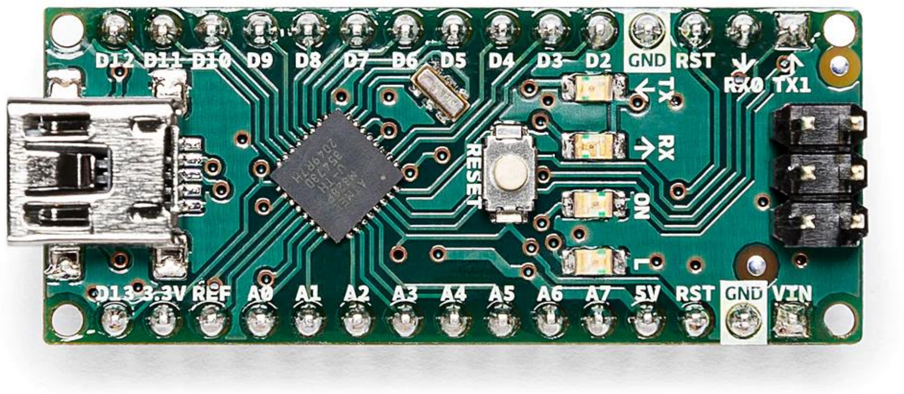
- VIN: power voltage, 3.3 V - pin "3V3" on CPU board;
- GND: power return, GND - pin "G" on CPU board;
- SCL: input I2C clock line (ρυθμος επικοινωνιας!)- pin "D1" on CPU board, GPIO 5;
- SDA: input/output I2C data line - pin "D2" on CPU board, GPIO 4;
- measures Temperature between 0 and 65 C, pressures between 300 and 1100 hPa (corresponding to +9000 and -500 m relative to sea level)

Arduino βιβλιοθηκες για αισθητηρες! Πολλες!

Another great strength of the Arduino "system" is the availability of "libraries" to handle many, many types of hardware. From the Arduino IDE, under "Tools", select "Library Manager" and enter, for example, BMP180. Scroll down until you find "BMP180MI" and click on "Install". After a short while, you will have this library available to call in your programs. Most libraries also come with example programs you can open, read, modify and play with. The BMP180 is not an exception. Similarly, you can search for AHT20 and install a library for the AHT20 (and AHT21) sensor, and an example program, or search for LM75A to get a library for the LM75A sensor, with examples.

Arduino Nano pinout: which pins can do what




D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
TX RX ON L RESET
D13 3.3V REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN

Beyond the Arduino IDE: MicroPython

https://micropython.org/

Why MicroPython?

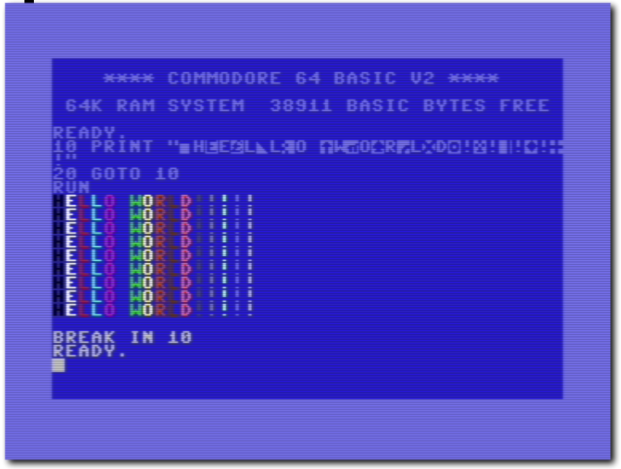
- MCU speeds and memory are increasing every year
- easier to write and read than C/C++
- easier to test than C/C++
- available on a large number of microcontroller families
- large selection of modules (libraries) available
- easier access to IoT protocols
- exception handling is superior
- complex data types (lists, dictionaries...) are immediately available

Why NOT MicroPython?

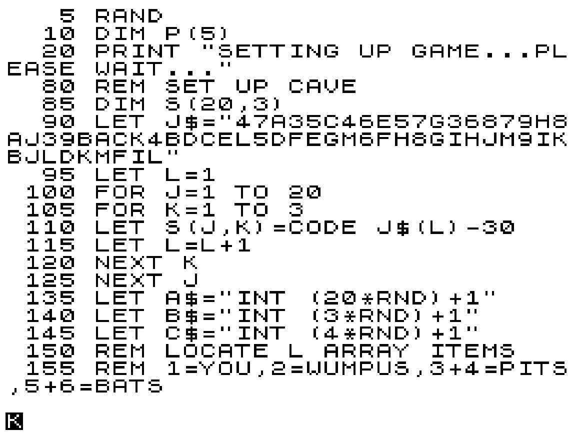
- less efficient (interpreted vs compiled) both in terms of CPU cycles and of memory. This is less often a problem than one would think, as many programs spend large amounts of time "waiting" for an event in an idle loop.
- errors can remain undiscovered until the buggy section of code is executed, while in the case of compiled code syntax errors are found at compilation. Logical errors, on the other hand...
- cannot run on the "smallest" (and usually least expensive) systems with limited resources.

Interpreted vs Compiled

- Already in 1952, interpreters were used to ease programming
- in 1958, John McCarthy began developing a LISP interpreter
- by the early 1970s, BASIC was used on mainframe computers
- in the mid 1970s, the advent of microcomputers caused an explosive growth of BASIC, followed by Eiffel, JavaScript, Lua, MATLAB, Perl, PHP, R, Ruby, VBScript...



```
    **** COMMODORE 64 BASIC V2 ****

 64K RAM SYSTEM  38911 BASIC BYTES FREE

READY.
10 PRINT "HELLO WORLD!!!!!
!"
20 GOTO 10
RUN
HELLO WORLD!!!!!
HELLO WORLD!!!!!
HELLO WORLD!!!!!
HELLO WORLD!!!!!
HELLO WORLD!!!!!
HELLO WORLD!!!!!
HELLO WORLD!!!!!
HELLO WORLD!!!!!
HELLO WORLD!!!!!

BREAK IN 10
READY.
```



```
   5 RAND
  10 DIM P(5)
  20 PRINT "SETTING UP GAME...PL
EASE WAIT..."
  80 REM SET UP CAVE
  85 DIM S(20,3)
  90 LET J$="47A35C46E57G36879H8
AJ39BACK4BDCEL5DFEGM6FH8GIHJM9IK
BJLDKMFIL"
  95 LET L=1
 100 FOR J=1 TO 20
 105 FOR K=1 TO 3
 110 LET S(J,K)=CODE J$(L)-30
 115 LET L=L+1
 120 NEXT K
 125 NEXT J
 135 LET A$="INT (20*RND)+1"
 140 LET B$="INT (3*RND)+1"
 145 LET C$="INT (4*RND)+1"
 150 REM LOCATE L ARRAY ITEMS
 155 REM 1=YOU,2=WUMPUS,3+4=PITS
,5+6=BATS

K
```

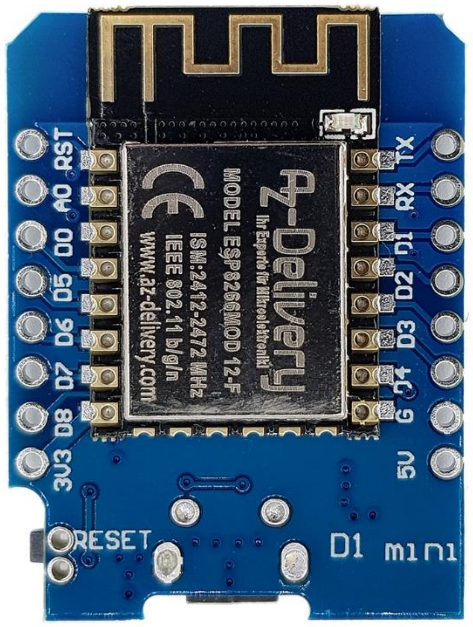
Why interpreted?

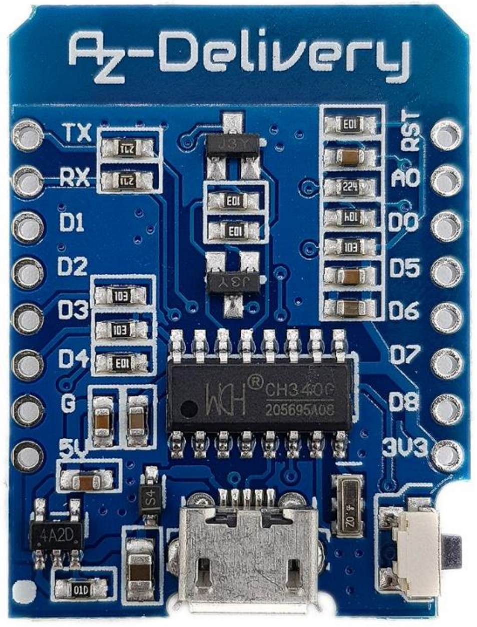
- The software development cycle is faster, since any modifications to the code do not require a compilation of the source code.
- An interpreted program can be distributed as source code without the need for recompilation, and apart from architecture-dependent features, does not need any adaptation to the new system.
- Once you are satisfied with your MicroPython script, you can just save it in the microcontroller's filesystem as "main.py" and it will be run avery time the microcontroller starts up.

Hardware and Software

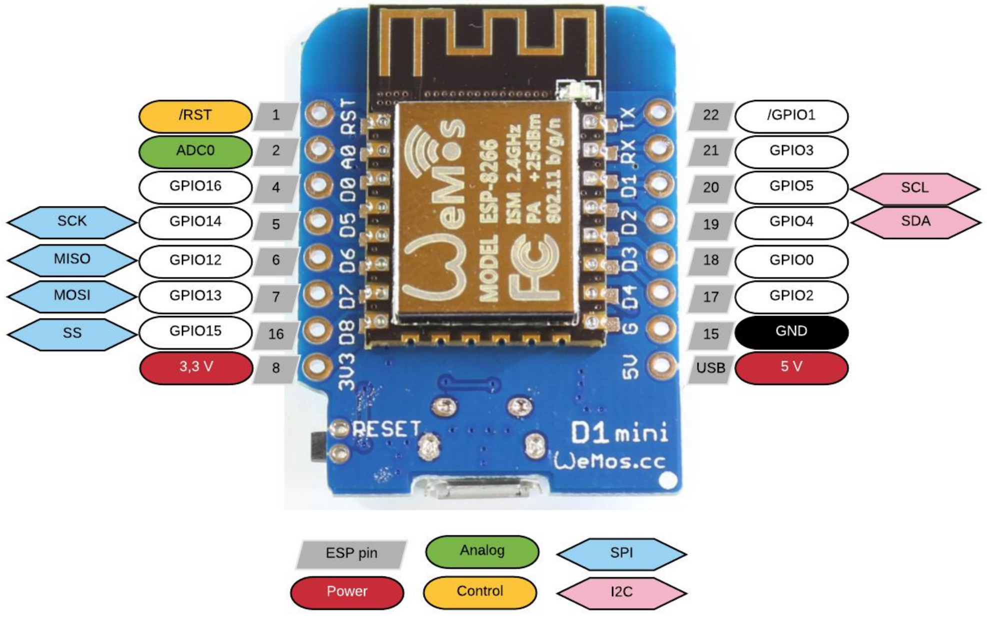
- Hardware:
  - D1 Mini (ESP8266) CPU board (https://www.wemos.cc/en/latest/d1/d1_mini.html);
  - BMP180 (Temperature and Pressure) sensor board;
- Software:
  - Thonny IDE (https://thonny.org/);
  - MicroPython firmware (https://micropython.org/download/esp8266/);
  - BMP180 MicroPython module (https://github.com/micropython-IMU/micropython-bmp180)
  - Optional: esptool (https://github.com/espressif/esptool/releases);

The CPU board: ESP8266 with 4 MBytes Flash





The CPU board pins and their functions



Note that pins have more than one denomination, depending whether you are using the Arduino IDE or MicroPython to program the CPU. In MicroPython, the GPIOxx number is used, so GPIO4 will be "Pin 4".
The board has an LED, connected to GPIO pin 2 (also D4) so that the LED will be ON if the pin outputs 0, and off if the pin outputs 1.

Which are the best pins to use?

Label	GPIO	Serial	SPI	I2C	Other	Comment	Input	Out
D0	16				Wakeup	HIGH at boot	No interrupt	OK
D1	5			SCL			OK	OK
D2	4			SDA			OK	OK
D3	0				FLASH	boot fails if LOW	Pulled up	OK
D4	2	TXD1			Board LED	boot fails if LOW	Pulled up	OK
D5	14		CLK				OK	OK
D6	12		MISO				OK	OK
D7	13	RXD2	MOSI				OK	OK
D8	15	TXD2	CS			boot fails if HIGH	Pulled down	Maybe
TX	1	TXD0				HIGH at boot	Maybe	NO
RX	3	RXD0				boot fails if LOW	NO	Maybe
A0	0				ADC		Analog	NO

Installing firmware with Thonny 4.1.2

The newest (as of today) version of Thonny can even fetch your firmware for you, and allows you to specify more parameters for the upload of firmware to your microcontroller...

The "blinky" example

- We can now try the LED blink example in MicroPython: in Thonny, create a new file (click on the blank page icon) and type in the following, being very careful to respect the indentation (use the Tab key "-->"):

```
import time
from machine import Pin
ledpin = machine.Pin(2, Pin.OUT)
while(True):
    ledpin.on()
    time.sleep(0.5)
    ledpin.off()
    time.sleep(0.5)
```

What is going on in our "blinky" program?

```
import time
from machine import Pin
ledpin = machine.Pin(2,
Pin.OUT)
while(True):
   ledpin.on()
   time.sleep(0.5)
   ledpin.off()
   time.sleep(0.5)
```

These two lines are similar to #include <library.h> in C/C++

here we define the LED pin

now we loop forever

turn the pin to 1 (LED off)

wait half a second

turn the pin to 0 (LED on)

wait half a second

Another program: I2C scanner

```
from machine import Pin
i2c = machine.I2C(scl=Pin(5), sda=Pin(4))
print('Scanning i2c bus...')
devices = i2c.scan()
if len(devices) == 0:
   print("No i2c devices found!")
else:
   print('found:', len(devices), 'i2c devices')
for device in devices:
   print("Decimal address: ", device, " | Hexadecimal 
address: ", hex(device))
```

Similar to #include<machine/Pin.h>

Declare an I2C interface on GPIO pins 5 and 4

Create a list of all devices present on the I2C bus

If the list has zero length, so no elements, it means no devices were found, so print a disappointed message

If the list is not empty, print the number of devices found, then for each element of the list print the I2C address that generated a positive response

The I2C bus

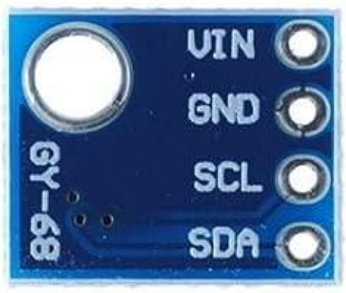
The I2C was designed by Philips in the 1980s for communications between integrated circuits on the same circuit board. However, it is not limited to this, and many digital sensors make use of it for their readout, since it requires few lines and can connect multiple sensors.

We will use I2C-based sensors in this lecture.

The main characteristics are:

- only two bus lines needed, SDA (Serial DAta) and SCL (Serial CLock)
- synchronous bus, so no strict baud rate requirements exist
- multi-master system, with provisions for arbitration and collision detection
- each device on the bus is addressable by its own unique address

The BMP180 Sensor board

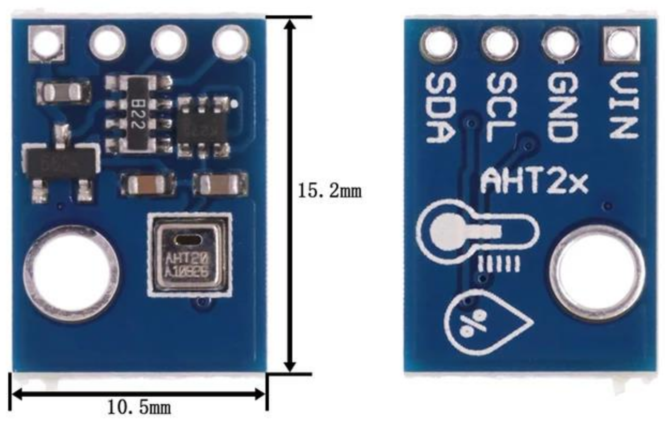


3V3
G
D1
D2

BMP180:

- VIN: power voltage, 3.3 V - pin "3V3" on CPU board;
- GND: power return, GND - pin "G" on CPU board;
- SCL: input I2C clock line - pin "D1" on CPU board, GPIO 5;
- SDA: input/output I2C data line - pin "D2" on CPU board, GPIO 4;
- measures Temperature between 0 and 65 C, pressures between 300 and 1100 hPa (corresponding to +9000 and -500 m relative to sea level)

The AHT21 Sensor board



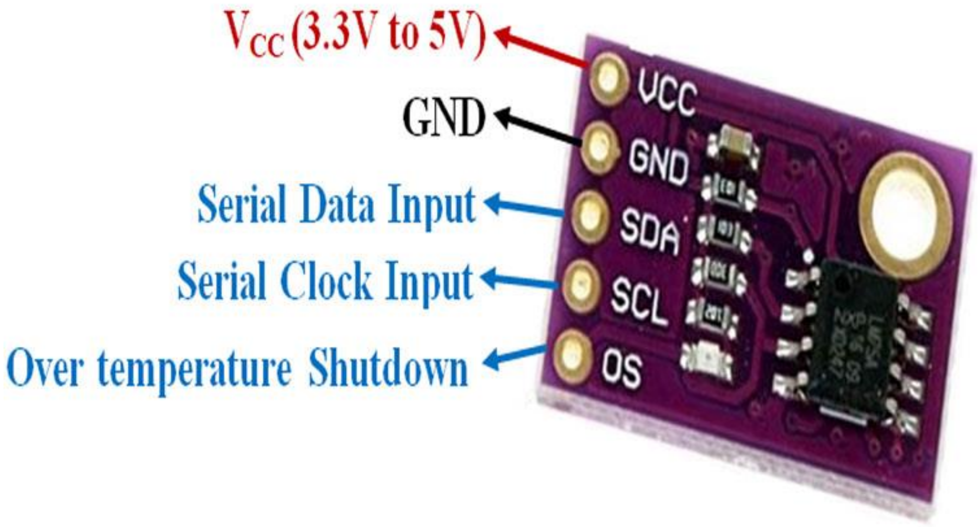
AHT21:

- VIN: power voltage, 3.3 V - pin "3V3" on CPU board;
- GND: power return, GND - pin "G" on CPU board;
- SCL: input I2C clock line - pin "D1" on CPU board, GPIO 5;
- SDA: input/output I2C data line - pin "D2" on CPU board, GPIO 4;
- measures Temperature between -40 and 120 C, relative humidity between 0 and 100 %

What if there IS no existing software module?

- Suppose you find a new sensor for which nobody has published a module yet. How difficult is it to develop your own software?
- The LT75 is a temperature sensor with a digital output that becomes active when the temperature exceeds a programmable threshold. It is an I2C slave and contains four registers:
  - Temperature, 9 bits encoded over 2 bytes, read-only (address 0)
  - Configuration, 8 bits (address 1)
  - Hysteresis, 9 bits encoded over 2 bytes (address 2)
  - Threshold, 9 bits encoded over 2 bytes (address 3)
- We can start by simply reading the temperature register.

The LM75A sensor board and its connections



On the back, the module allows you to change three bits of the I2C address. Leave them unconnected for now.

The quick and dirty code (but we have a module for the LM75a... feel free to try it)

```
# WARNING! Ugly code! But it does work...
from machine import I2C, Pin
# Create an instance of the I2C bus with the right pin numbers
i2c = I2C(scl=Pin(5), sda=Pin(4))
# Create a 2-byte buffer for storing the temperature measurement
tbuffer = bytearray(2)
# Read two bytes from the temperature register into the newly created buffer. The LM75 address is 0x48
i2c.readfrom_mem_into(0x48, 0, tbuffer)
# Convert 9 bits to a temperature value. The integer part is in tbuffer[0] and the 0.5 in the MSB of tbuffer[1]
temperature = (tbuffer[0]) + (tbuffer[1] >> 7) / 2
print(temperature)
```

Python and executing actions at a given time

MicroPython has extensions which allow the easy use of most features of the microcontrollers it runs on, and this includes the timers. It is possible to create a Timer (which can in some cases be virtual - with no corresponding hardware timer in the CPU) with a given period (or just a single execution after a specific delay), and associate with it a so-called "callback function" to be executed when the Timer runs out (the period has expired). In the next example, we will use two, to schedule the readout of temperature and relative humidity from the AHT21 sensors every 5 and 30 seconds respectively.

```
import time
import micropython
from machine import Pin, I2C, Timer
import ahtx0
i2c = I2C(scl=Pin(5), sda=Pin(4))
aht = ahtx0.AHT10(i2c)
t1 = Timer(1)
t1.init(mode = Timer.PERIODIC, period = 5000,
callback = lambda t :
micropython.schedule(print, f'T =
```

Explaining the Timer listing (1)

```
import time
import micropython
from machine import Pin,
I2C, Timer
import ahtx0
i2c = I2C(scl=Pin(5),
sda=Pin(4))
aht = ahtx0.AHT21(i2c)
```

These lines contain the code we already used to prepare the readout of the AHT21 sensor:

- include the necessary modules (libraries)
- setup the I2C bus
- create an AHT21 object

Explaining the Timer listing (2)

```
t1 = Timer(1)
t1.init(
    mode = Timer.PERIODIC,
    period = 5000,
    callback = lambda t :
micropython.schedule(print,
f'T = {aht.temperature:.1f} C')
)
```

Create a Timer object with id = 1

Set it up so that it runs in periodic mode (restarts immediately after expiring), with a period of 5000 milliseconds, and with an anonymous callback function that requests that the Python system print "as soon as possible" the (formatted) temperature value.

The same happens for humidity.

Explaining the Timer listing (3)

A few pythonic notes:

1. we should NOT specify directly print() as the callback function since it would be executed in Interrupt mode. Better to ask Python to print "as soon as possible" but outside of Interrupt mode, using the micropython.schedule() call.
2. Since the callback functions are not really useful for anything else, we do not need to give them a name, but we can create them as anonymous using the "lambda" notation. n Python, a lambda function is a small, anonymous function that can take any number of arguments, but can have only one expression. Perfect here.
3. f'T = {aht.temperature:.1f} C' is a string containing a format specification, namely the value of the temperature with exactly one decimal digit.

More callback functions

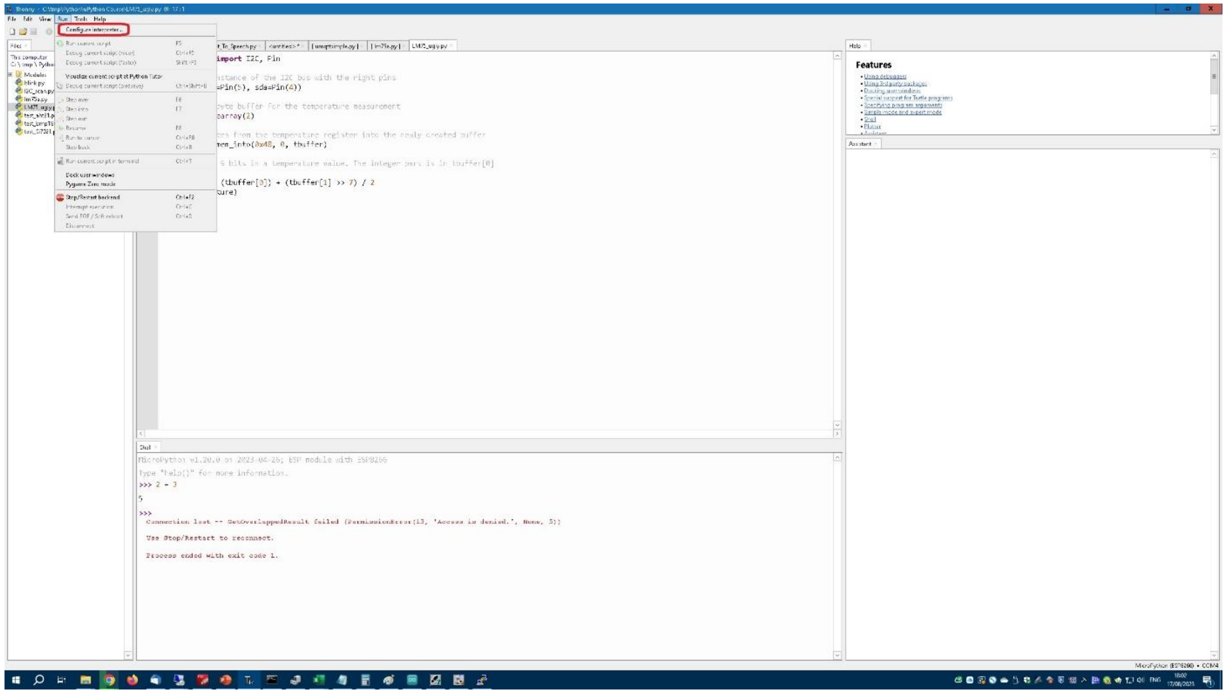
It is also possible to associate a callback function with an interrupt caused by an event on an input (or even an output) pin. Then the CPU could execute any kind of main loop without the need to check the pin while stil responding (and quickly) to the arrival of a signal.

In the following example, a callback function (that prints the number of the pin) is associated with an interrupt generated by a falling edge (1 to 0) on pin 4 (D2). In the main loop, pin 5 (D1), defined as an output, changes value regularly, printing "Pin(5)" every time a falling edge is generated. By connecting pin 5 to pin 4, the program starts printing additionally "Pin(4)" just after "Pin(5)", with no evident instruction to do so.

Pin callback example

```
from machine import Pin
import time
p0 = Pin(5, Pin.OUT)
p1 = Pin(4, Pin.IN)
p1.irq(handler = lambda p : print(p), trigger = Pin.IRQ_FALLING)
while True:
   p0.value(0)
   time.sleep(0.5)
   p0.value(1)
   time.sleep(4.5)
```

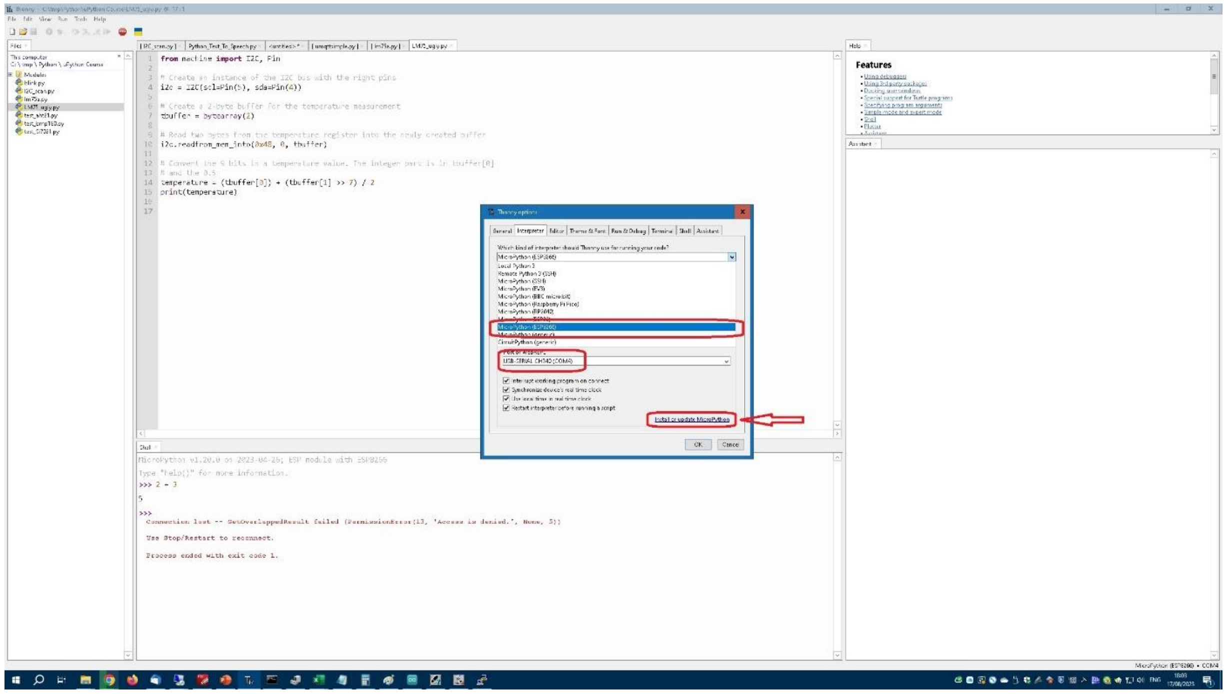
Using Thonny to install the ESP8266 firmware



```
import I2C, Pin

nstance of the I2C bus with the right pins
=Pin(5), sda=Pin(4))

byte buffer for the temperature measurement
earray(2)

tes from the temperature register into the newly created buffer
mem_into(0x48, 0, tbuffer)

9 bits in a temperature value. The integer part is in tbuffer[0]

 (tbuffer[0]) + (tbuffer[1] >> 7) / 2
ture)
```

```
MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>> 2 + 3
5
>>>
 Connection lost -- GetOverlappedResult failed (PermissionError(13, 'Access is denied.', None, 5))

 Use Stop/Restart to reconnect.

 Process ended with exit code 1.
```

1. Select the CPU and the port



```
from machine import I2C, Pin

# Create an instance of the I2C bus with the right pins
i2c = I2C(scl=Pin(5), sda=Pin(4))

# Create a 2-byte buffer for the temperature measurement
tbuffer = bytearray(2)

# Read two bytes from the temperature register into the newly created buffer
i2c.readfrom_mem_into(0x48, 0, tbuffer)

# Convert the 9 bits in a temperature value. The integer part is in tbuffer[0]
# and the 0.5
temperature = (tbuffer[0]) + (tbuffer[1] >> 7) / 2
print(temperature)
```

Thonny options — Interpreter

Which kind of interpreter should Thonny use for running your code?

MicroPython (ESP8266)

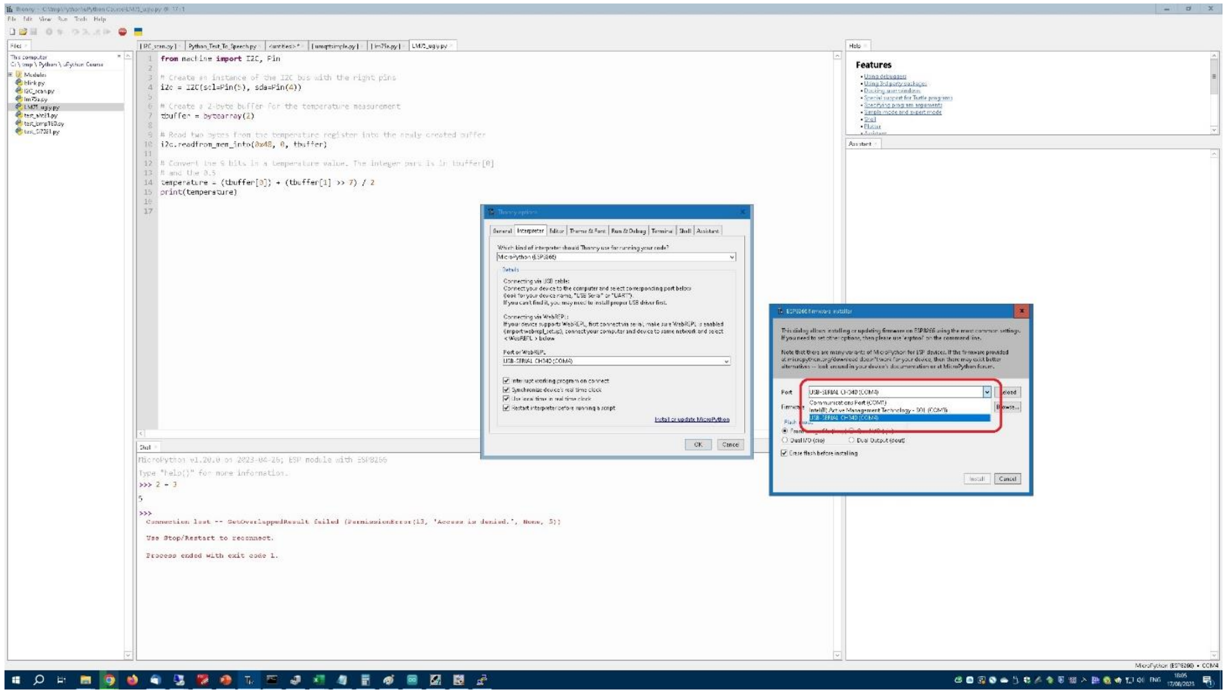
Port or WebREPL: USB-SERIAL CH340 (COM4)

- [✓] Interrupt working program on connect
- [✓] Synchronize device's real time clock
- [✓] Use local time in real time clock
- [✓] Restart interpreter before running a script

Install or update MicroPython

```
MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>> 2 + 3
5
>>>
  Connection lost -- GetOverlappedResult failed (PermissionError(13, 'Access is denied.', None, 5))

  Use Stop/Restart to reconnect.

  Process ended with exit code 1.
```

2. Find the firmware on your computer



```
from machine import I2C, Pin

# Create an instance of the I2C bus with the right pins
i2c = I2C(scl=Pin(5), sda=Pin(4))

# Create a 2-byte buffer for the temperature measurement
tbuffer = bytearray(2)

# Read two bytes from the temperature register into the newly created buffer
i2c.readfrom_mem_into(0x48, 0, tbuffer)

# Convert the 9 bits in a temperature value. The integer part is in tbuffer[0]
# and the 0.5
temperature = (tbuffer[0]) + (tbuffer[1] >> 7) / 2
print(temperature)
```

```
MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>> 2 + 3
5
>>>
  Connection lost -- GetOverlappedResult failed (PermissionError(13, 'Access is denied.', None, 5))

  Use Stop/Restart to reconnect.

  Process ended with exit code 1.
```

Thonny options

Which kind of interpreter should Thonny use for running your code?
MicroPython (ESP8266)

Details

Connecting via USB cable:
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

Connecting via WebREPL:
If your device supports WebREPL, first connect via serial, make sure WebREPL is enabled
(import webrepl_setup), connect your computer and device to same network and select
< WebREPL > below

Port or WebREPL
USB-SERIAL CH340 (COM4)

- [✓] Interrupt working program on connect
- [✓] Synchronize device's real time clock
- [✓] Use local time in real time clock
- [✓] Restart interpreter before running a script

ESP8266 firmware installer

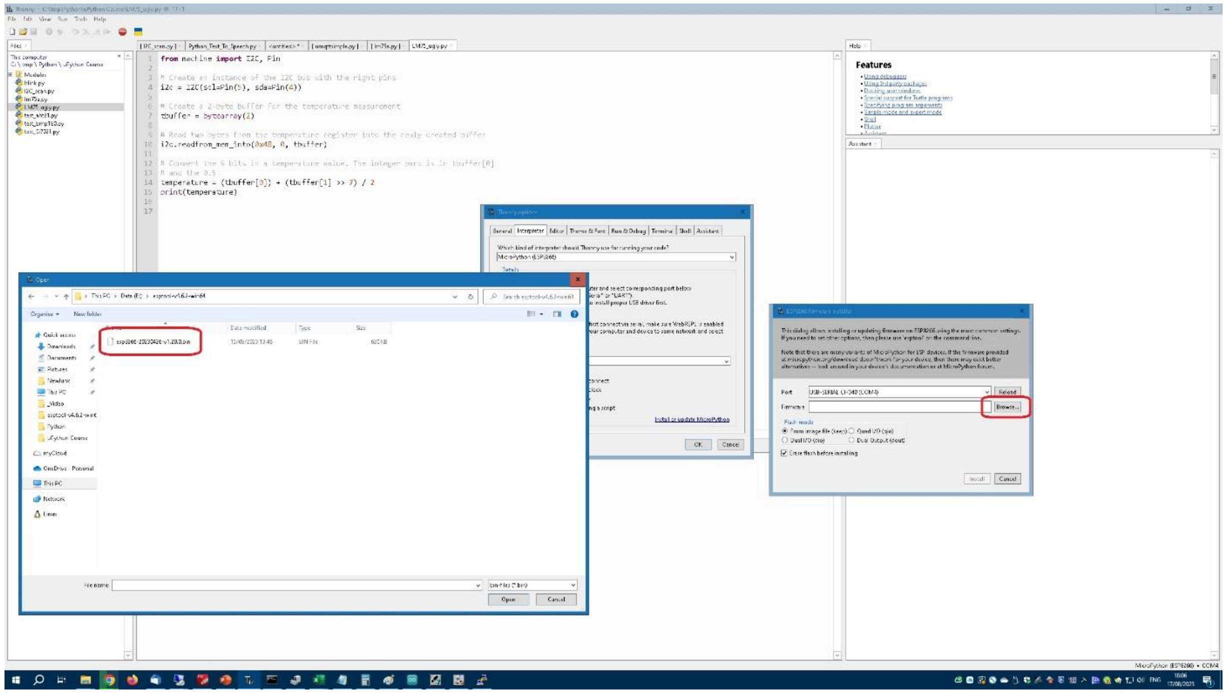
This dialog allows installing or updating firmware on ESP8266 using the most common settings.
If you need to set other options, then please use 'esptool' on the command line.

Note that there are many variants of MicroPython for ESP devices. If the firmware provided at micropython.org/download doesn't work for your device, then there may exist better alternatives -- look around in your device's documentation or at MicroPython forum.

Port: USB-SERIAL CH340 (COM4)
- Communications Port (COM1)
- Intel(R) Active Management Technology - SOL (COM3)
- USB-SERIAL CH340 (COM4)

- [✓] Erase flash before installing

3. Select the firmware binary



```python
from machine import I2C, Pin

# Create an instance of the I2C bus with the right pins
i2c = I2C(scl=Pin(5), sda=Pin(4))

# Create a 2-byte buffer for the temperature measurement
tbuffer = bytearray(2)

# Read two bytes from the temperature register into the newly created buffer
i2c.readfrom_mem_into(0x48, 0, tbuffer)

# Convert the 9 bits in a temperature value. The integer part is in tbuffer[0]
# and the 0.5
temperature = (tbuffer[0]) + (tbuffer[1] >> 7) / 2
print(temperature)
```

This PC > Data (E:) > esptool-v4.6.2-win64

Name	Date modified	Type	Size
esp8266-20230426-v1.20.0.bin	13/08/2023 13:46	BIN File	620 KB

ESP8266 firmware installer

This dialog allows installing or updating firmware on ESP8266 using the most common settings. If you need to set other options, then please use 'esptool' on the command line.

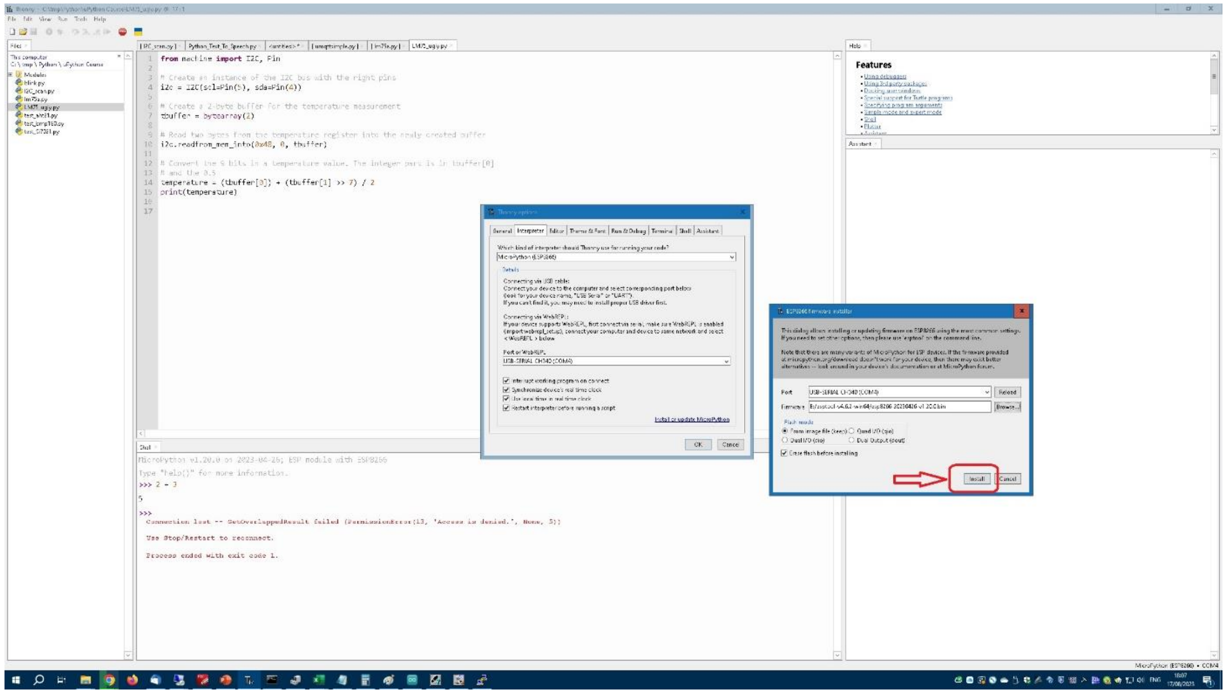
Note that there are many variants of MicroPython for ESP devices. If the firmware provided at micropython.org/download doesn't work for your device, then there may exist better alternatives -- look around in your device's documentation or at MicroPython forum.

Port: USB-SERIAL CH340 (COM4)
Firmware:

Flash mode
[✓ From image file (keep)] Quad I/O (qio)
Dual I/O (dio) Dual Output (dout)

[✓ Erase flash before installing]

4. Start the installation (it will take some time)



```python
from machine import I2C, Pin

# Create an instance of the I2C bus with the right pins
i2c = I2C(scl=Pin(5), sda=Pin(4))

# Create a 2-byte buffer for the temperature measurement
tbuffer = bytearray(2)

# Read two bytes from the temperature register into the newly created buffer
i2c.readfrom_mem_into(0x48, 0, tbuffer)

# Convert the 9 bits in a temperature value. The integer part is in tbuffer[0]
# and the 0.5
temperature = (tbuffer[0]) + (tbuffer[1] >> 7) / 2
print(temperature)
```

```
MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>> 2 + 3
5
>>>
  Connection lost -- GetOverlappedResult failed (PermissionError(13, 'Access is denied.', None, 5))

  Use Stop/Restart to reconnect.

  Process ended with exit code 1.
```

Thonny options

Which kind of interpreter should Thonny use for running your code?
MicroPython (ESP8266)

Details

Connecting via USB cable:
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

Connecting via WebREPL:
If your device supports WebREPL, first connect via serial, make sure WebREPL is enabled
(import webrepl_setup), connect your computer and device to same network and select
< WebREPL > below

Port or WebREPL: USB-SERIAL CH340 (COM4)

- [x] Interrupt working program on connect
- [x] Synchronize device's real time clock
- [x] Use local time in real time clock
- [x] Restart interpreter before running a script

ESP8266 firmware installer

This dialog allows installing or updating firmware on ESP8266 using the most common settings.
If you need to set other options, then please use 'esptool' on the command line.

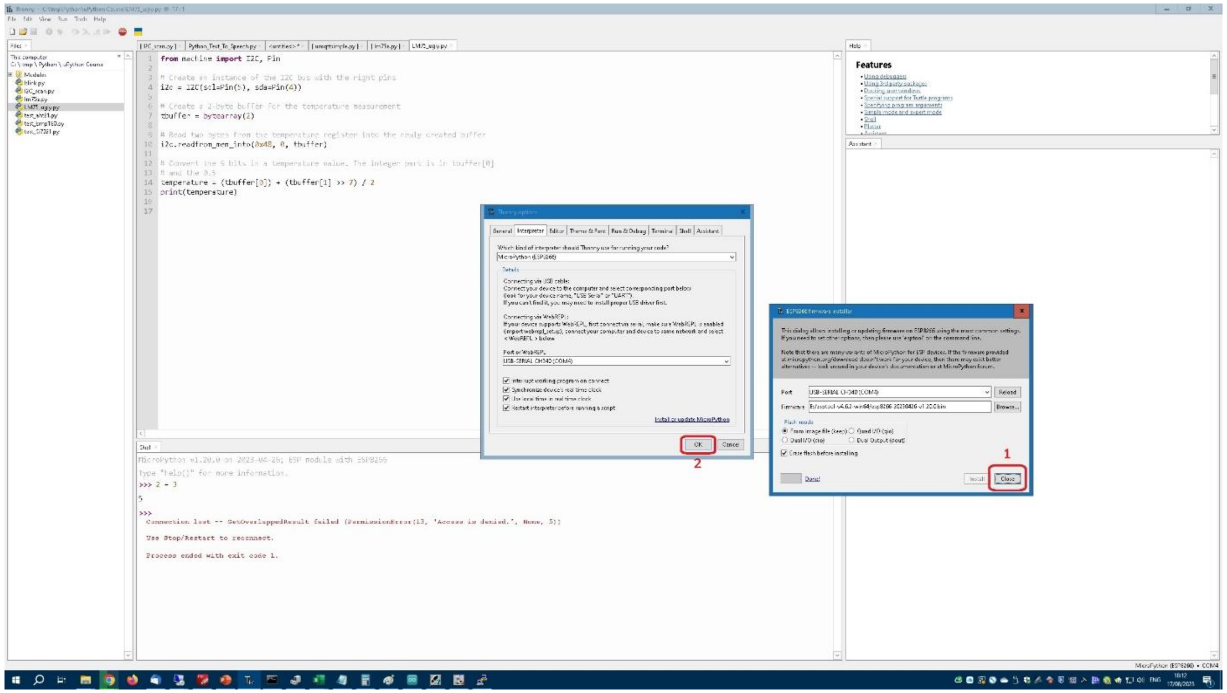
Note that there are many variants of MicroPython for ESP devices. If the firmware provided at micropython.org/download doesn't work for your device, then there may exist better alternatives -- look around in your device's documentation or at MicroPython forum.

Port: USB-SERIAL CH340 (COM4)
Firmware: E:/esptool-v4.6.2-win64/esp8266-20230426-v1.20.0.bin

Flash mode: [✓ From image file (keep)] Quad I/O (qio) Dual I/O (dio) Dual Output (dout)

- [x] Erase flash before installing

5. Once done, close the pop-ups



```
from machine import I2C, Pin

# Create an instance of the I2C bus with the right pins
i2c = I2C(scl=Pin(5), sda=Pin(4))

# Create a 2-byte buffer for the temperature measurement
tbuffer = bytearray(2)

# Read two bytes from the temperature register into the newly created buffer
i2c.readfrom_mem_into(0x48, 0, tbuffer)

# Convert the 9 bits in a temperature value. The integer part is in tbuffer[0]
# and the 0.5
temperature = (tbuffer[0]) + (tbuffer[1] >> 7) / 2
print(temperature)
```

```
MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>> 2 + 3
5
>>>
  Connection lost -- GetOverlappedResult failed (PermissionError(13, 'Access is denied.', None, 5))

  Use Stop/Restart to reconnect.

  Process ended with exit code 1.
```

Which kind of interpreter should Thonny use for running your code?
MicroPython (ESP8266)

Details

Connecting via USB cable:
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

Connecting via WebREPL:
If your device supports WebREPL, first connect via serial, make sure WebREPL is enabled
(import webrepl_setup), connect your computer and device to same network and select
< WebREPL > below

Port or WebREPL
USB-SERIAL CH340 (COM4)

- [✓] Interrupt working program on connect
- [✓] Synchronize device's real time clock
- [✓] Use local time in real time clock
- [✓] Restart interpreter before running a script

This dialog allows installing or updating firmware on ESP8266 using the most common settings.
If you need to set other options, then please use 'esptool' on the command line.

Note that there are many variants of MicroPython for ESP devices. If the firmware provided at micropython.org/download doesn't work for your device, then there may exist better alternatives -- look around in your device's documentation or at MicroPython forum.

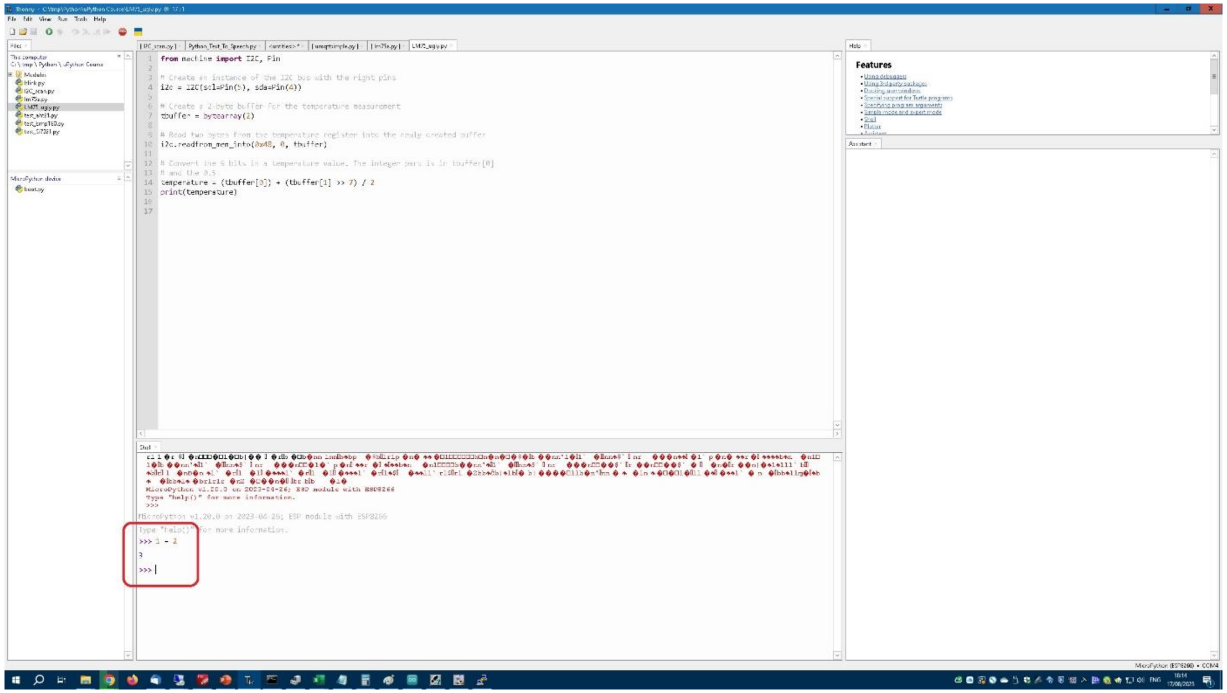
Port: USB-SERIAL CH340 (COM4)
Firmware: E:/esptool-v4.6.2-win64/esp8266-20230426-v1.20.0.bin

Flash mode: [✓ From image file (keep)] Quad I/O (qio) Dual I/O (dio) Dual Output (dout)

- [✓] Erase flash before installing

Done!

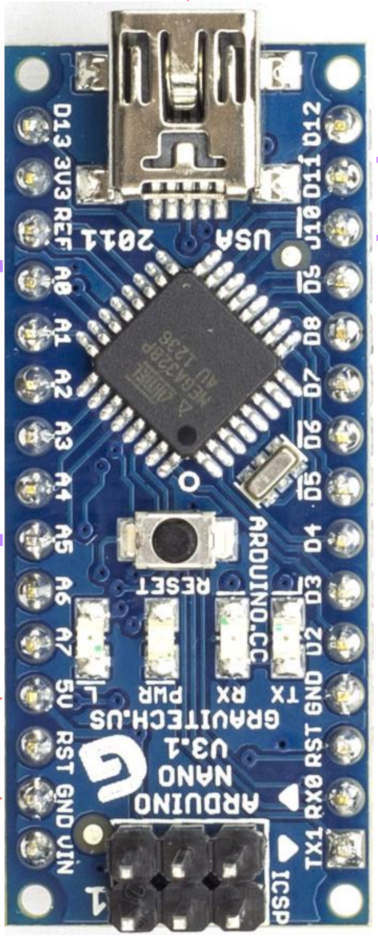
6. Finally, test that you have a working setup



```python
from machine import I2C, Pin

# Create an instance of the I2C bus with the right pins
i2c = I2C(scl=Pin(5), sda=Pin(4))

# Create a 2-byte buffer for the temperature measurement
tbuffer = bytearray(2)

# Read two bytes from the temperature register into the newly created buffer
i2c.readfrom_mem_into(0x48, 0, tbuffer)

# Convert the 9 bits in a temperature value. The integer part is in tbuffer[0]
# and the 0.5
temperature = (tbuffer[0]) + (tbuffer[1] >> 7) / 2
print(temperature)
```

```
[illegible]
MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>>
MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>> 1 + 2
3
>>>
```

USB connection with your PC


D13 3V3 REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN
D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
2011 USA
RESET
ARDUINO.CC
L PWR RX TX
GRAVITECH.US
ARDUINO NANO V3.1
ICSP

D10-D11-D13 "Drive" digital outputs

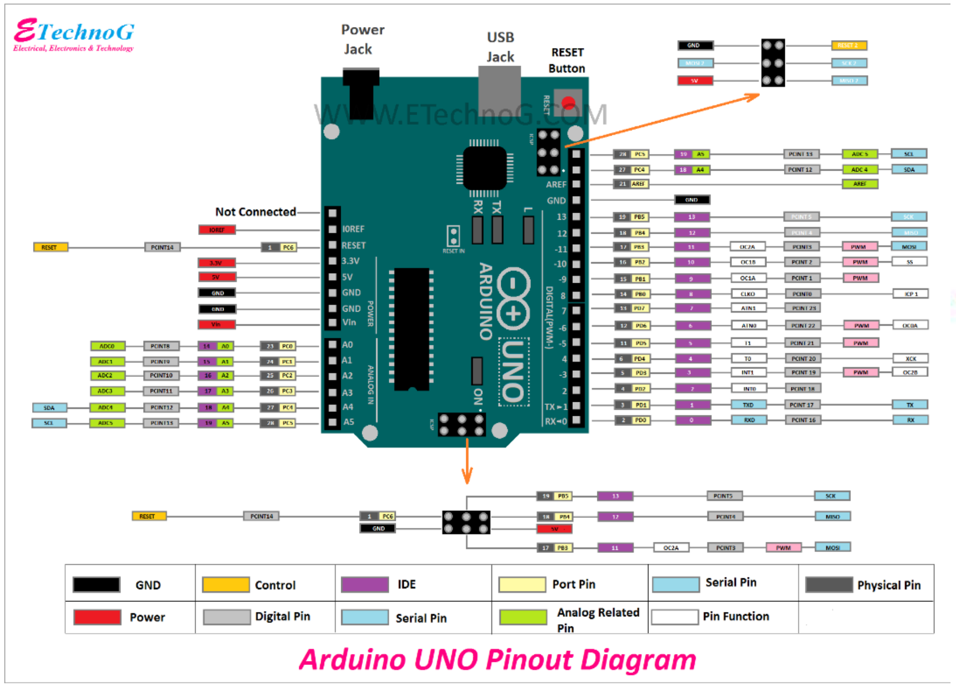
A0-A5 Read analog values

D2-D3 Read digital inputs

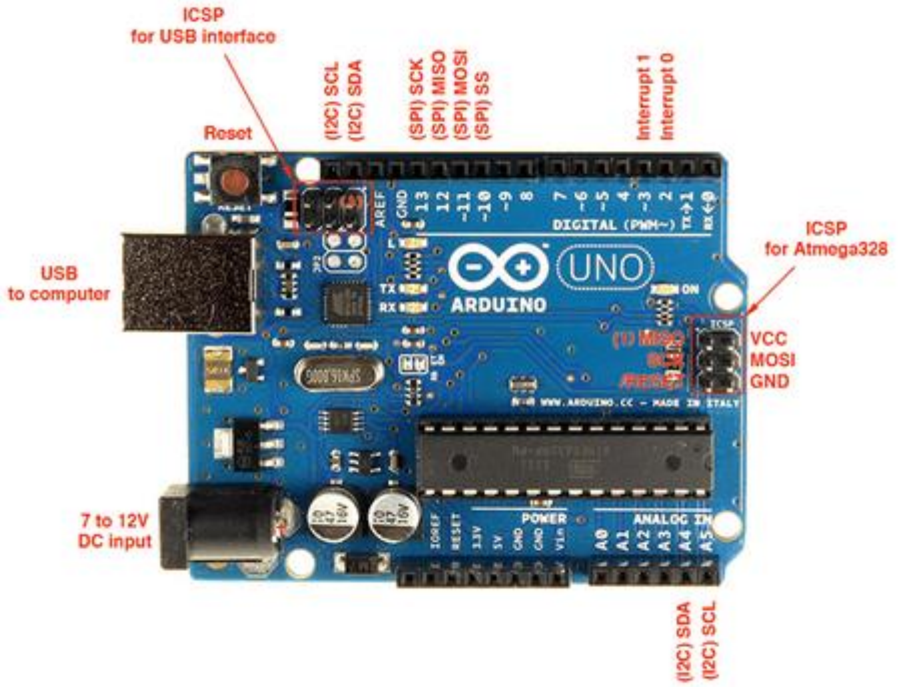
5V power. For (almost) every sensor

GROUND! Your reference!

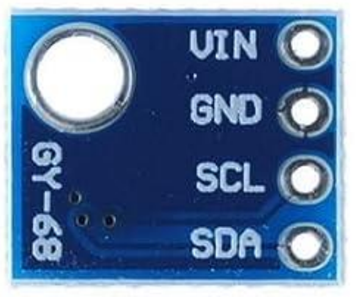
Arduino UNO pinout



Arduino UNO Pinout Diagram



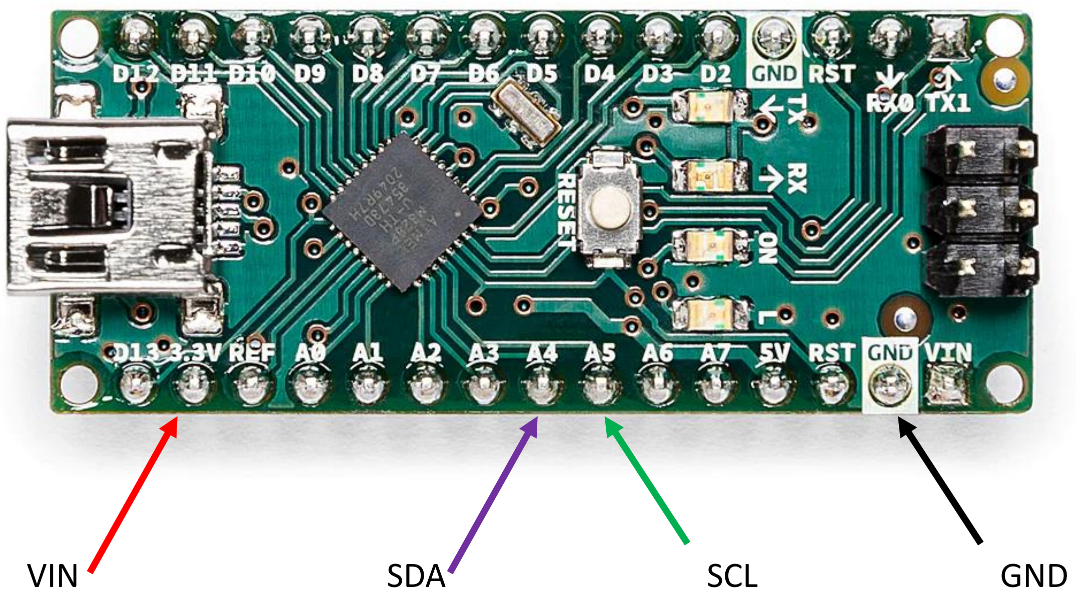
How do we connect our sensors? BMP180



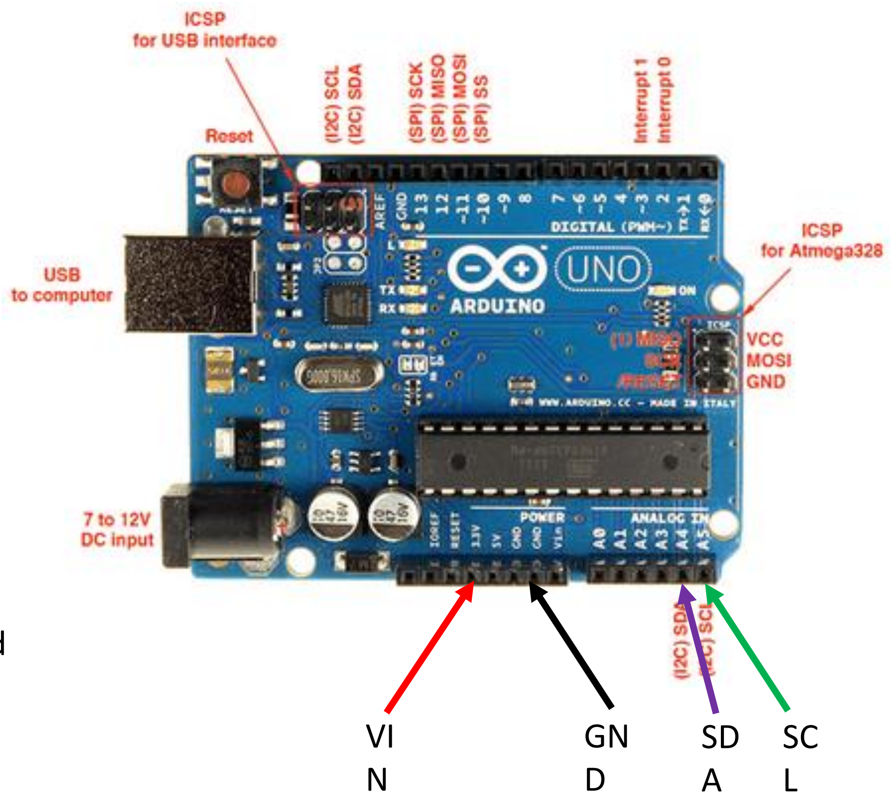
The BMP180 sensor board has four pins. We can connect them to our Arduino Nano as follows:

- VIN is the power supply for the sensor, which MUST be 3.3V and NOT 5V (otherwise you will destroy it). So, we need to connect it to the "3v3" pin of the Nano.
- GND is the power return, and it goes to one "GND" pin of the Nano (there are two available).
- SCL is the Serial CLock signal needed to read the sensor, and it needs to be connected to the "SCL" pin of the Nano, which is also called and labeled "A5".
- SDA is the Serial DAta line used to read the sensor, and it needs to be connected to the SDA line of the Nano, which is also called and labeled "A4".

Connections, in detail



Arduino UNO version of the connections:



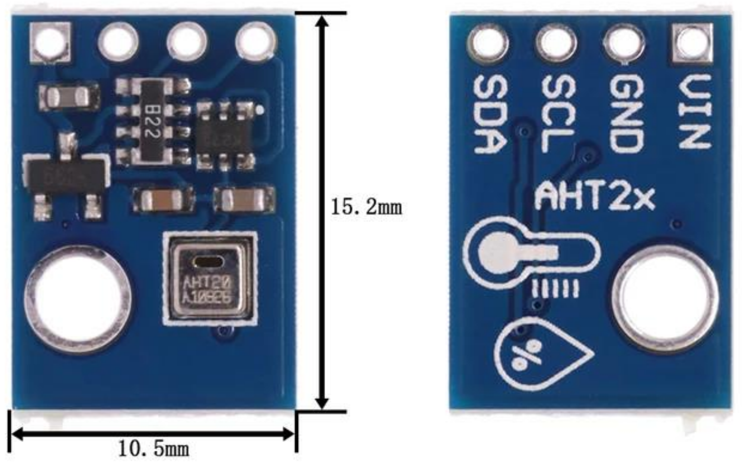
You can totally ignore the explanations for the pins you are not interested in, or check them out for the future...

Note that the pins are located in different positions.

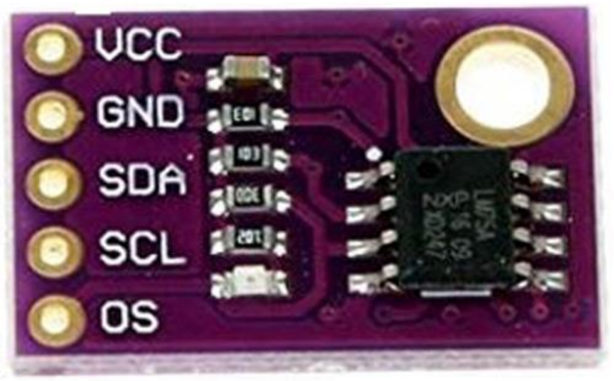
Connecting the AHT21 sensor board (αισθητηρας υγρασιας+θερμοκρασιας)

The AHT21 board also has four pins, in fact they are the same as for the BMP180 board but are not placed in the same positions, rather the other way around, so please be careful. You can disconnect the Arduino UNO from USB, disconnect the BMP180 board one wire at a time and reconnect each wire to the AHT21 immediately.

Write down which color wires you are using for which signal, and try to keep the association between signal and wire color always the same, and you will prevent trouble.



Connecting the LM75 board



The LM75 board seems to have one more connection, the one labeled "OS", but we do not really need it. It is an output that the LM75 makes active whenever the temperature exceeds a set limit (so it can be used as a thermostat).

In fact, ignoring "OS", the connections are similar to those for the AHT21, except for the exchange of SCL with SDA (this is an annoying fact, but there is no "standard" way of connecting these boards, as every producer feels free to locate the pins wherever it is more convenient. For him, not for us).

Arduino and LED blink program

The real innovation that the Arduino "system" brought is the layer of abstraction between the inner workings of the CPU and the access to the resources, such as I/O pins. Where before you would need to manipulate the registers with cryptic instructions such as

```
DDRD = 0b11111111; // declaring port D as output, because we know the LED is connected to port D, pin 0
PORTD |= 0b10000000; // pin 0 of port D set HIGH, all other pins left unchanged, this is an OR operation
_delay_ms(1000); // delay of one second
PORTD &= 0b01111111; // pin 0 of port D set LOW, all others left unchanged, this is an AND operation
```

now you can achieve the same result with more meaningful and understandable instructions:

```
pinMode(LED_BUILTIN, OUTPUT); // we do not need to remember which port and pin the LED is connected to
digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level), no need to manipulate bits
delay(1000);                      // wait for a second
digitalWrite(LED_BUILTIN, LOW);   // turn the LED off (LOW is the voltage level), again no bit manipulation needed
```

Let us analyze the LED blink program

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
```

The setup() function is executed only once, at the beginning of the program. In this case, it handles the mechanisms needed to make the pin the on-board LED is connected to an output (so we can turn the LED on and off)

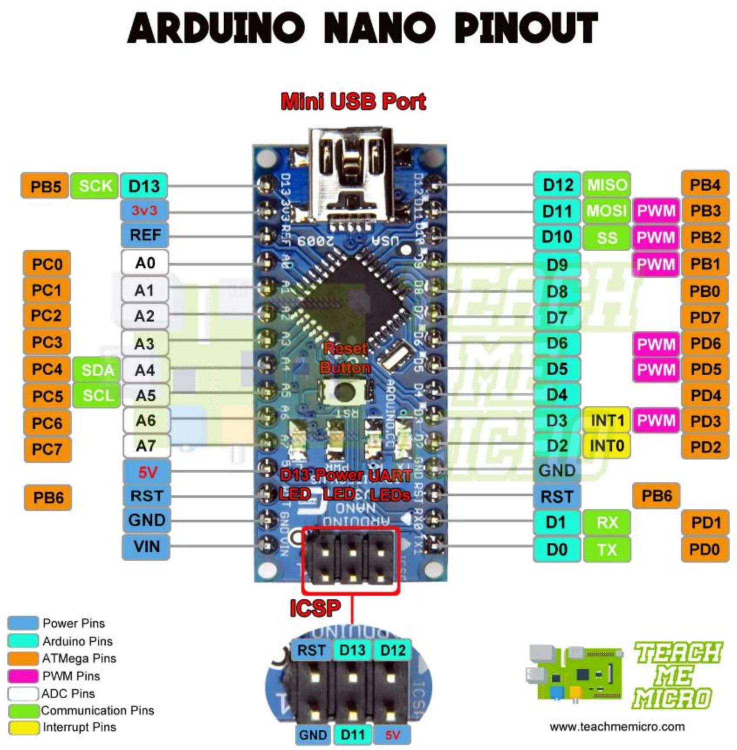
```
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000
}
```

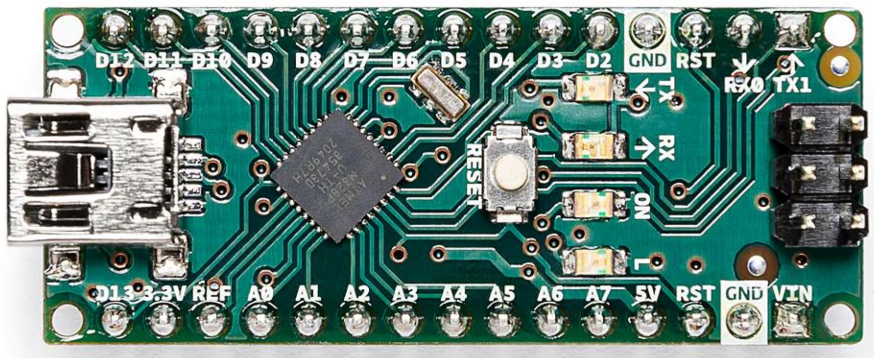
The loop() function, instead, is supposed to keep running "forever". In this case, it makes the LED pin take a HIGH value (+5 V) so the LED turns on, waits for 1 second, makes the LED pin take a LOW (0 V) value so the LED turns off, then waits 1 more second before ending and restarting again, and again, and again...

Arduino (sensor) Libraries

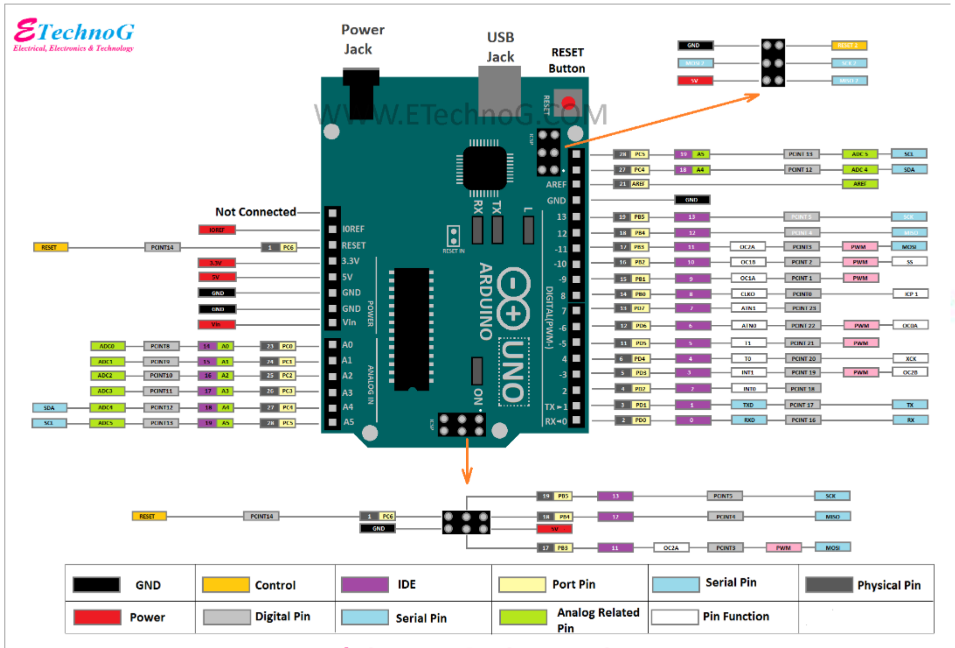
Another great strength of the Arduino "system" is the availability of "libraries" to handle many, many types of hardware. From the Arduino IDE, under "Tools", select "Library Manager" and enter, for example, BMP180. Scroll down until you find "BMP180MI" and click on "Install". After a short while, you will have this library available to call in your programs. Most libraries also come with example programs you can open, read, modify and play with. The BMP180 is not an exception. Similarly, you can search for AHT20 and install a library for the AHT20 (and AHT21) sensor, and an example program, or search for LM75A to get a library for the LM75A sensor, with examples.

Arduino Nano pinout: which pins can do what

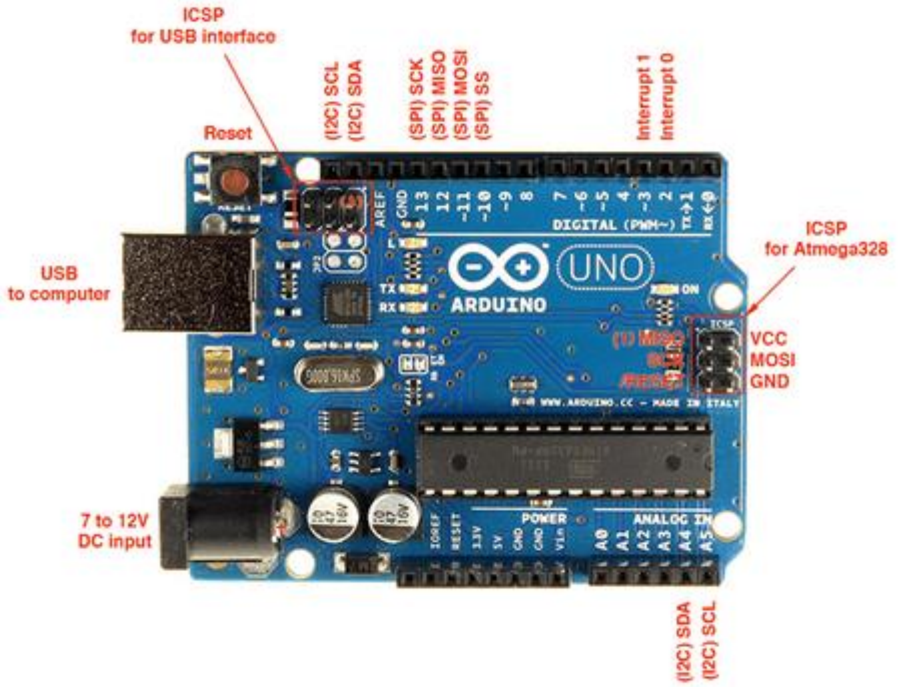



D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
TX RX ON L RESET
D13 3.3V REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN

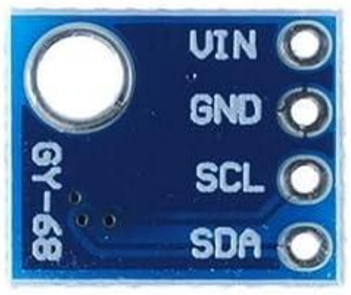
Arduino UNO pinout



Arduino UNO Pinout Diagram



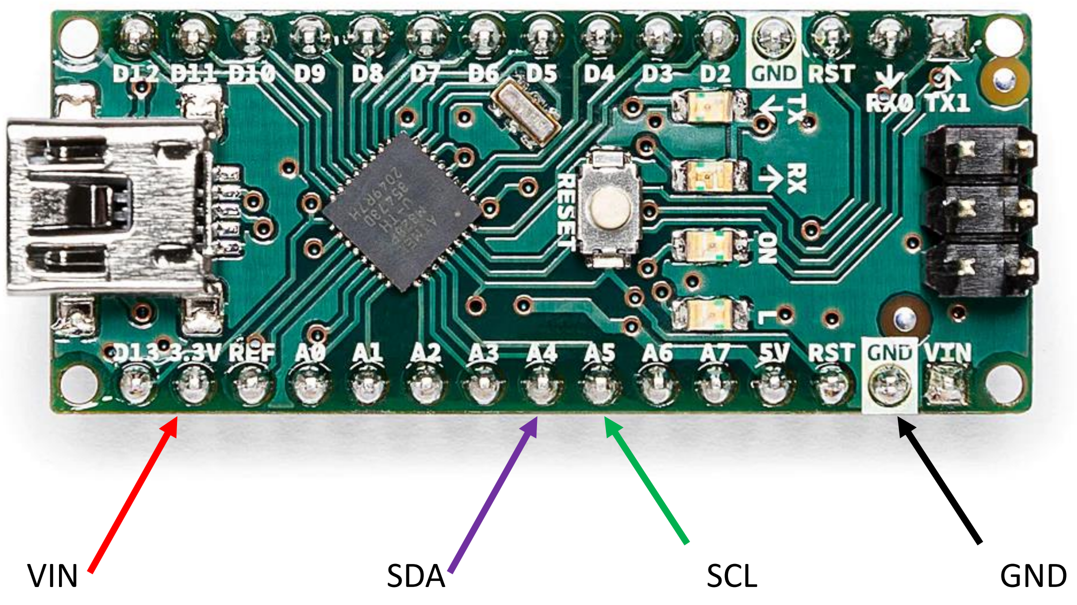
How do we connect our sensors? BMP180



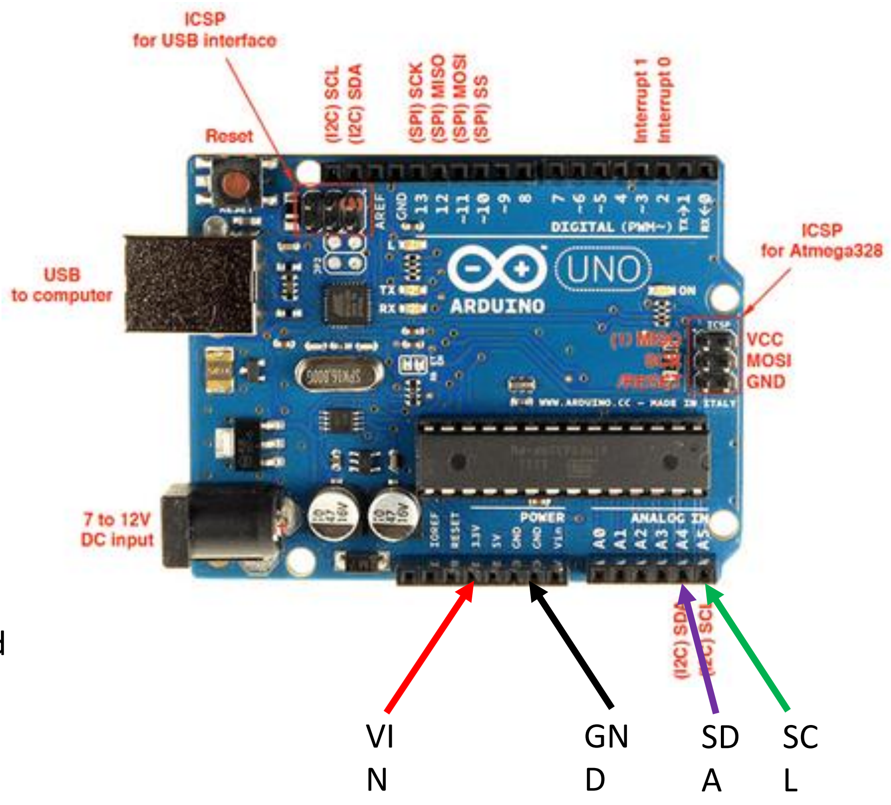
The BMP180 sensor board has four pins. We can connect them to our Arduino Nano as follows:

- VIN is the power supply for the sensor, which MUST be 3.3V and NOT 5V (otherwise you will destroy it). So, we need to connect it to the "3v3" pin of the Nano.
- GND is the power return, and it goes to one "GND" pin of the Nano (there are two available).
- SCL is the Serial CLock signal needed to read the sensor, and it needs to be connected to the "SCL" pin of the Nano, which is also called and labeled "A5".
- SDA is the Serial DAta line used to read the sensor, and it needs to be connected to the SDA line of the Nano, which is also called and labeled "A4".

Connections, in detail



Arduino UNO version of the connections:



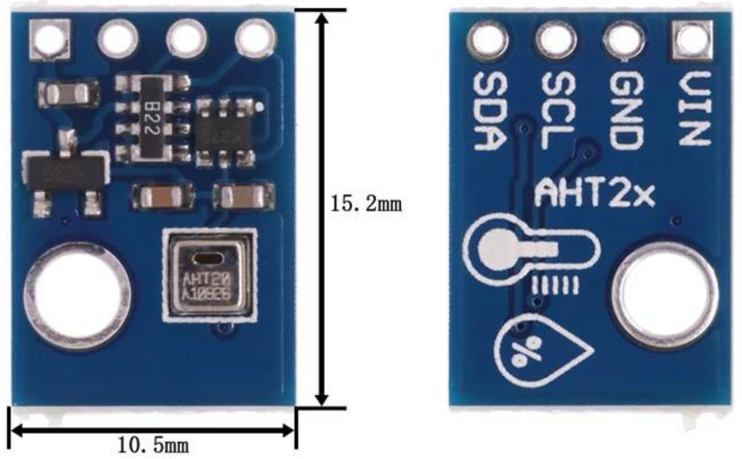
You can totally ignore the explanations for the pins you are not interested in, or check them out for the future...

Note that the pins are located in different positions.

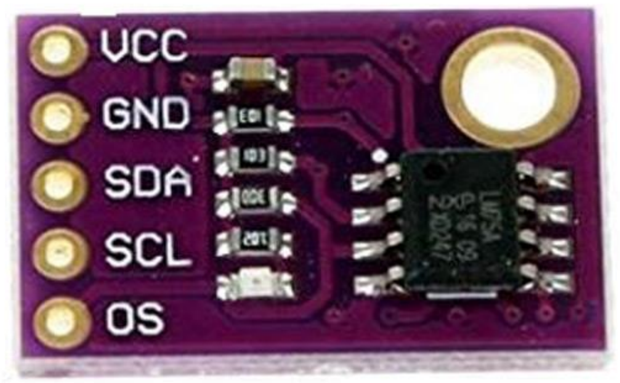
Connecting the AHT21 sensor board

The AHT21 board also has four pins, in fact they are the same as for the BMP180 board but are not placed in the same positions, rather the other way around, so please be careful. You can disconnect the Arduino UNO from USB, disconnect the BMP180 board one wire at a time and reconnect each wire to the AHT21 immediately.

Write down which color wires you are using for which signal, and try to keep the association between signal and wire color always the same, and you will prevent trouble.



Connecting the LM75 board



The LM75 board seems to have one more connection, the one labeled "OS", but we do not really need it. It is an output that the LM75 makes active whenever the temperature exceeds a set limit (so it can be used as a thermostat).

In fact, ignoring "OS", the connections are similar to those for the AHT21, except for the exchange of SCL with SDA (this is an annoying fact, but there is no "standard" way of connecting these boards, as every producer feels free to locate the pins wherever it is more convenient. For him, not for us).