African School of Fundamental Physics and Applications

Small Physics Experiments

# An Introduction to Physics Experimentation

Uli Raich

This workshop provides hands-on exercises on sensor readout and experiment control with actuators.

It is part of the African School of Fundamental Physics 2022

## Physics Experimentation

In many of today’s physics experiments some electronics is used to capture signals from sensors and to control the experiments through actuators

Sensors can measure

- Temperature, relative air humidity, soil moisture, air pressure
- Air pollution through particle detection or gaz analysis
- Magnetics fields (e.g the earth magnetic field, which allows to build a compass)
- Movement through accelerometers or gyroscopes
- Distance through ultrasound or lasers

African School of Fundamental Physics and Applications

## Actuators

Actuators are used to control the experiment

These can be:

- Digital on/off signals, relays
- Analog signal levels controlling e.g. an amplifier
- Different types of motors

African School of Fundamental Physics and Applications

# How to build a simple experiment

A typical experiment consists of

- The sensor and actuator
- A computer to read the sensor values and/or drive the actuators
- A program reading the results and displaying and storing them
- An analysis program for data evaluation

## Types of computers

We all use PCs and hand phones, which are controlled by computers.

- The PCs are rather powerful, they have large resources of disk space and memory and they are equipped with network interfaces, but they lack other hardware interfaces. PCs are relatively expensive.
- Microcontrollers have much less resources and usually no hard disk. Modern µCs have WiFi interfaces and they have hardware interfaces to
  - GPIO (General Purpose IO lines)
  - I2C instrumentation bus
  - SPI serial peripheral interface
- Prices can be very low (ours costs 4 US$)

African School of Fundamental Physics and Applications

## Course layout

There will be short lecture parts explaining the system and giving a few online examples

Most of the time will be spent on hands-on exercises where you will set up a simple experiment yourself.

The sensors will be simulated by a simple pushbutton switch for digital signals, and a slider potentiometer for analogue signals.

We also have a single color user programmable LED as well as a rgb LED ring with seven LEDs used as a simple display.

Data can also be transferred to the PC and plotted there.

African School of Fundamental Physics and Applications

# Simulating an experiment

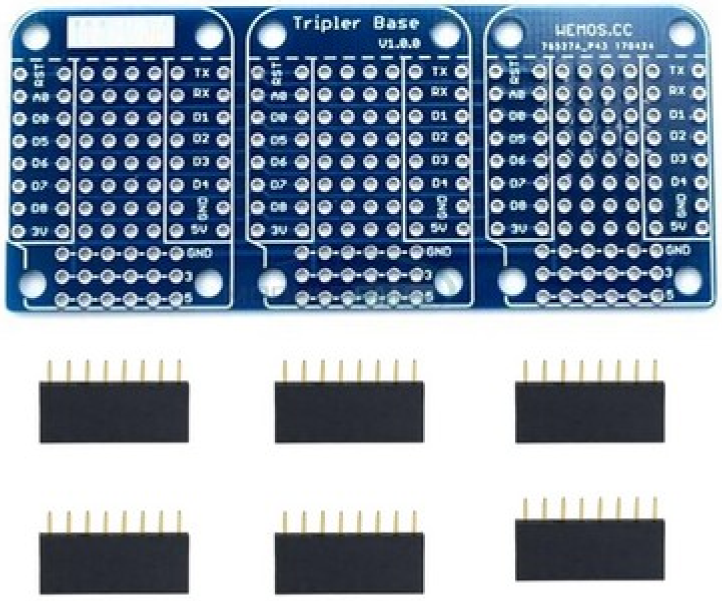
To simulate an experiment we have

an ESP32 micro-controller

The CPU card has a user programmable LED and an USB to serial interface

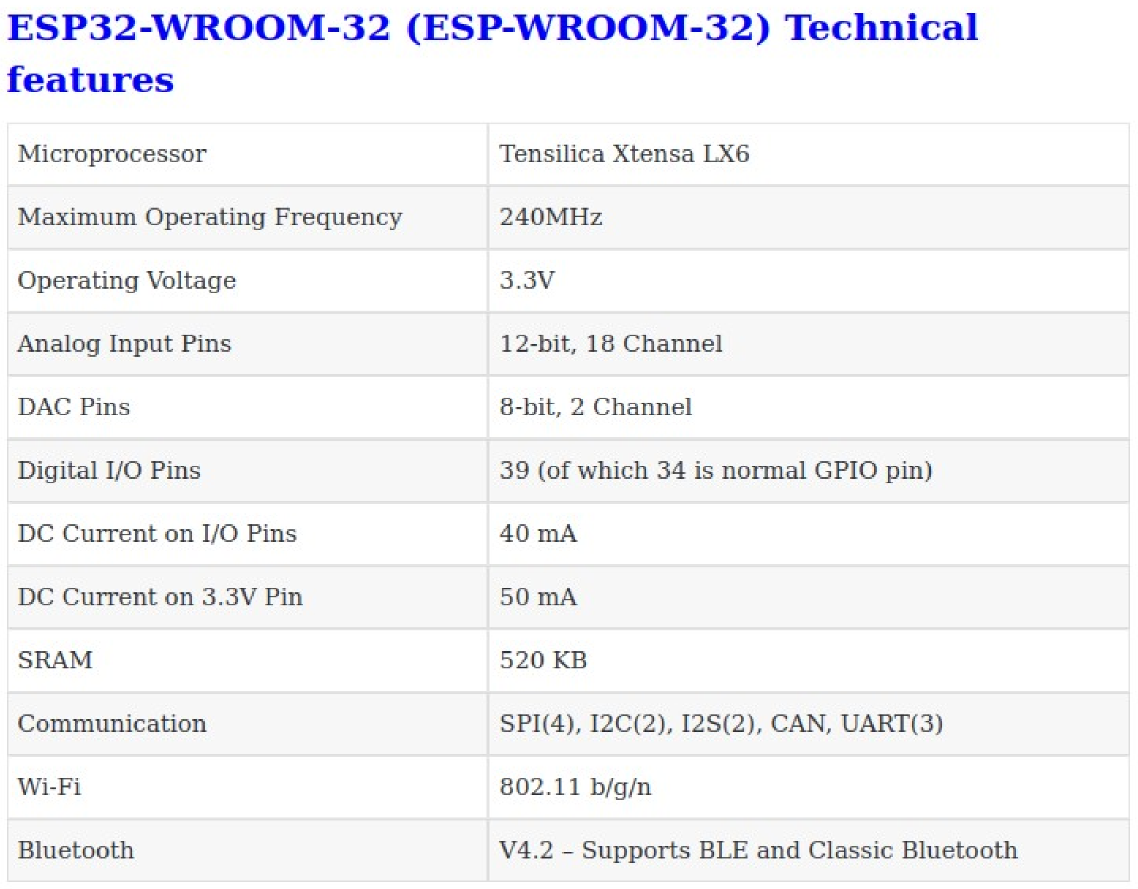
a bus system

to connect the sensor and the uC



African School of Fundamental Physics and Applications

## ESP32 specifications



**ESP32-WROOM-32 (ESP-WROOM-32) Technical features**

| | |
|---|---|
| Microprocessor | Tensilica Xtensa LX6 |
| Maximum Operating Frequency | 240MHz |
| Operating Voltage | 3.3V |
| Analog Input Pins | 12-bit, 18 Channel |
| DAC Pins | 8-bit, 2 Channel |
| Digital I/O Pins | 39 (of which 34 is normal GPIO pin) |
| DC Current on I/O Pins | 40 mA |
| DC Current on 3.3V Pin | 50 mA |
| SRAM | 520 KB |
| Communication | SPI(4), I2C(2), I2S(2), CAN, UART(3) |
| Wi-Fi | 802.11 b/g/n |
| Bluetooth | V4.2 – Supports BLE and Classic Bluetooth |

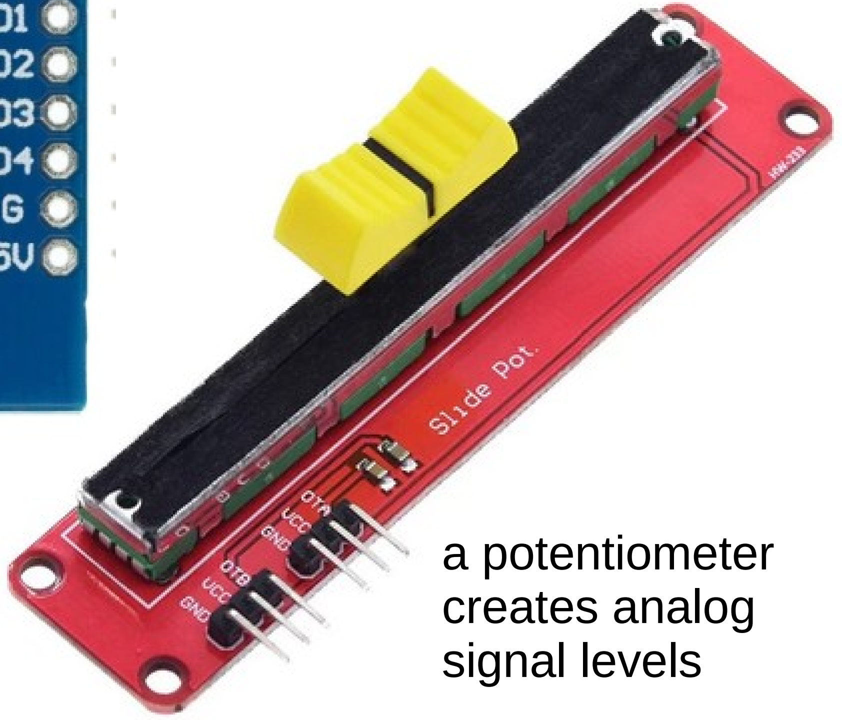
All this for 4 US$

African School of Fundamental Physics and Applications

## Sensor and actuator

a simple display with 7 rgb LEDs

a pushbutton allows to generate digital signals



a potentiometer creates analog signal levels

African School of Fundamental Physics and Applications

## How to use these devices

To interact with the devices we must write readout and control programs.

These days, a very popular programming language is **Python.**

Standard Python (CPython) uses huge libraries and can be very resource hungry but resources are scarce on the micro-controller.

A stripped down Python interpreter: MicroPython is already installed in flash memory on the CPU board.

We can connect the CPU card to the PC using a micro USB cable. The CPU converts USB to serial and we can use an serial terminal emulator for communication.

## Documentation

The documentation for the course is available at:

https://afnog.iotworkshop.africa/do/view/AFNOG/ASP2022

The MicroPython documentation is here:

https://docs.micropython.org/en/latest/

African School of Fundamental Physics and Applications

## Programming Tools

We have a series of tools allowing us to program the ESP32 micro-controller:

- thonny is an **I**ntegrated **D**evelopment **E**nvironment to
  - Edit programs and save them on the PC or the ESP32 file system
  - Transfer the programs to the ESP32
  - Run them
  - Print the results
  - Interact with the MicroPython Interpreter
- minicom is a serial terminal emulator that can talk to the MicroPython on the ESP32
- ampy allows to transfer file to and from the ESP32 via the serial line and to run programs

African School of Fundamental
Physics and Applications

## Connecting to the ESP32

Connection of the ESP32 micro-controller to the PC is simple:

Just connect the micro USB cable to the PC. This will also power the ESP32.

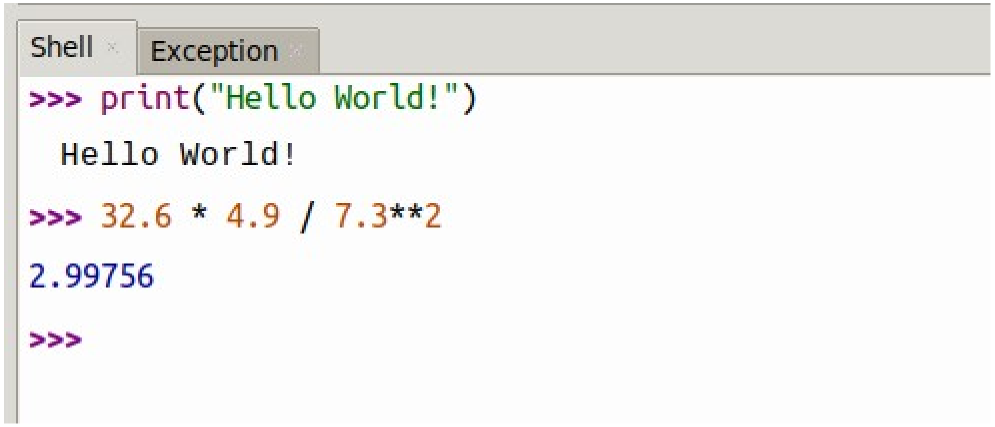
You should see the power LED on the ESP32 coming on.

African School of Fundamental Physics and Applications

## Talking to MicroPython

Since MicroPython is an interpreted language we can talk to the interpreter directly. When started we get a prompt saying that the REPL (**R**ead, **E**xecute and **P**rint **L**oop) is ready to receive commands.

Python can be used as a calculator.



```
>>> print("Hello World!")
 Hello World!
>>> 32.6 * 4.9 / 7.3**2
2.99756
>>>
```

African School of Fundamental Physics and Applications

# Programming

Talking to the interpreter is fine but we want to be able to write a program and have it executed.

For those who don’t know Python, here is a link to the Python tutorial.

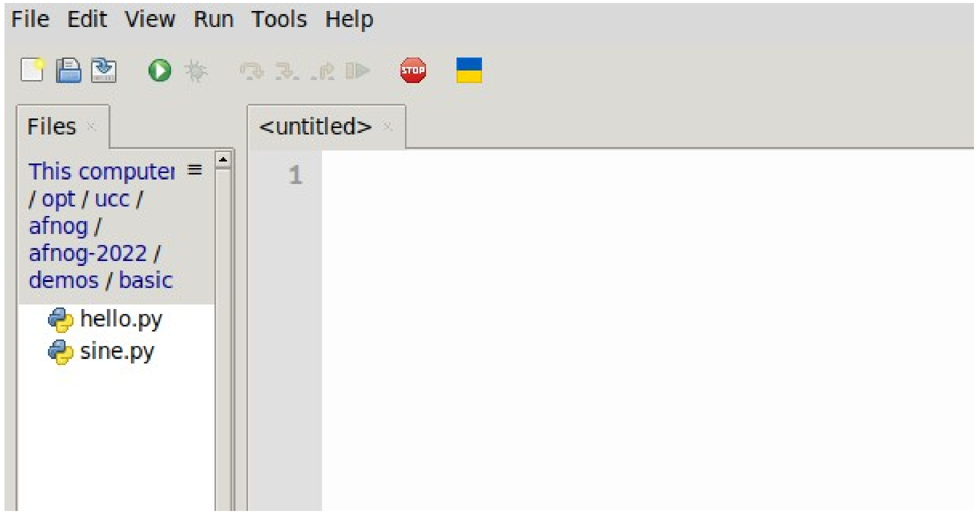
and here the link to the MicroPython documentation

Create a new Python file with File →New

This will create an editor window labeled <untitled>

Save this (empty file) and the window title will change to the filename

Now let’s write our hello world program and execute it



African School of Fundamental Physics and Applications

## Tools

The main tool we will be using is *thonny*

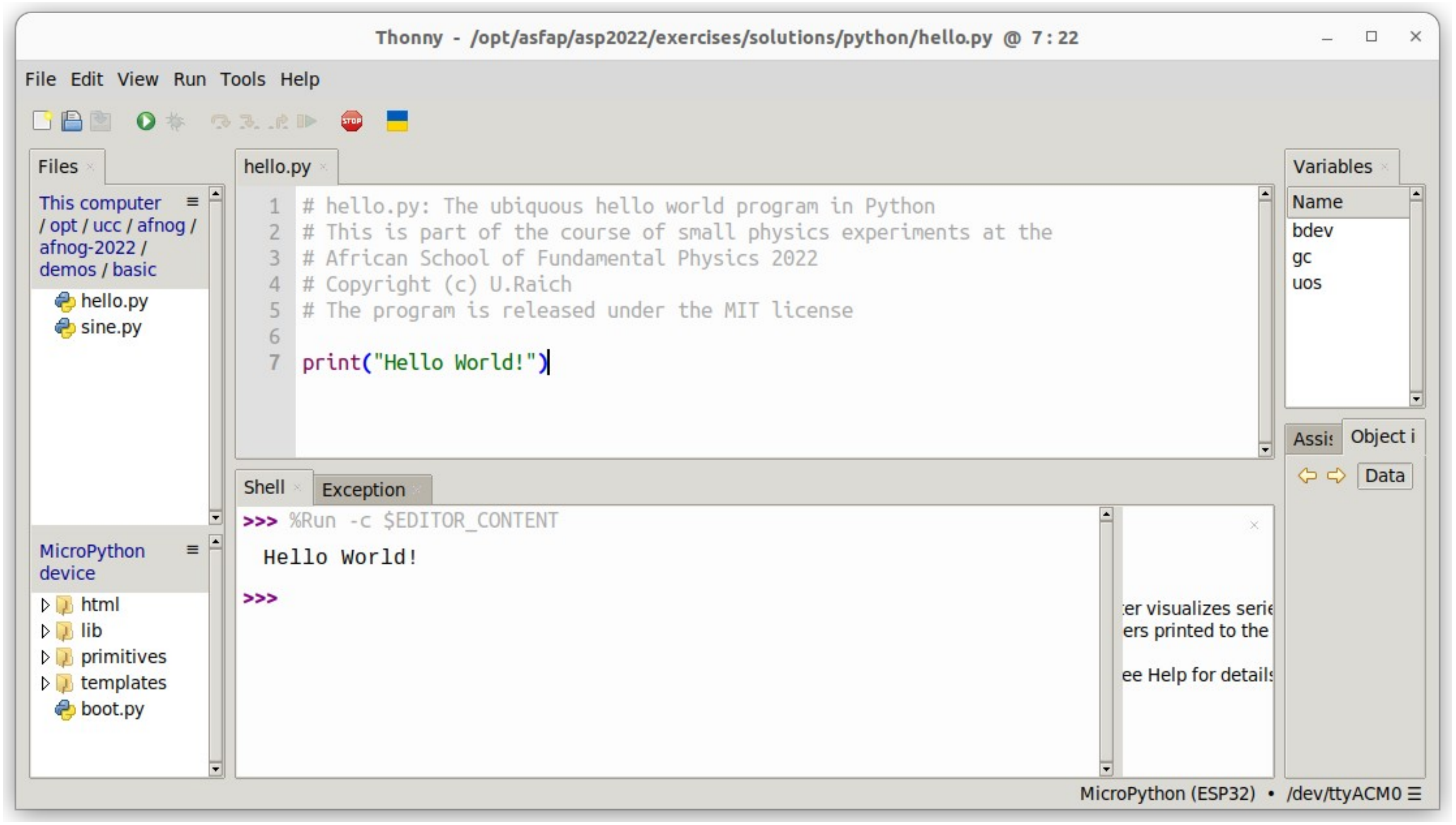
thonny is a simple Integrated Development Environment (IDE) which allows

- talking to the Read Evaluate and Print Loop (REPL) of the MicroPython interpreter
- editing scripts, saving them on files on the PC or the ESP32 and executing them with print output going to its shell window
- file transfer between the PC and the ESP32

*minicom* is a virtual serial terminal connecting to REPL on the ESP32

*ampy* allows file transfers and running of programs on the ESP32

## thonny


Thonny - /opt/asfap/asp2022/exercises/solutions/python/hello.py @ 7 : 22
File Edit View Run Tools Help
Files
This computer / opt / ucc / afnog / afnog-2022 / demos / basic
hello.py
sine.py
MicroPython device
html
lib
primitives
templates
boot.py
hello.py
Shell
Exception
Variables
Name
bdev
gc
uos
Assis
Object i
Data
er visualizes serie
ers printed to the
ee Help for detail
MicroPython (ESP32) • /dev/ttyACM0

```python
# hello.py: The ubiquous hello world program in Python
# This is part of the course of small physics experiments at the
# African School of Fundamental Physics 2022
# Copyright (c) U.Raich
# The program is released under the MIT license

print("Hello World!")
```

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

 Hello World!

>>>
```

African School of Fundamental Physics and Applications

## Assignment

In Python the “=” signal does not represent an equation but an assignment

```
a=6            # assigns the integer value 5 to the variable a
               # the “#” sign denotes a comment

f=5.8          # assigns the floating point value 5.8to the
               # variable f

b=True     # b takes the boolean value true

s=”Hello”      # s is assigned the string “Hello”

l=[1,2,3,5]    # l is a list with values 1,2,3,5

t=(1,2,3,5)    # t is a tuple with the same values

               # tuples are similar to lists by they are
               # immutable

a = a+3        # would be a illegal equation but is a perfectly legal
               # assignment

print(a,f,b,s,l,t)
```

Types are assigned automatically

## Arithmetic

The basic arithmetic operations work as expected:

```
s=5+7
a=5
b=7
sum = a+b
difference = a-b
product = a*b
division = a/b     # yields a float value
div_int  = a//b    # yields an integer division,
                   # only integer part of the division is taken
                   # into account
```

## Conditions

We can execute on part of our program or another one depending on conditions

```
a,b = 5,7
if a > b:
    print("a is bigger than b")   # note that the body of the condition
                                  # is indented
                                  # all the indented code is executed if the
                                  # condition evaluates to true
elif a < b:
    print("a is smaller than b")
else:
    print("a and b are equal")
```

Comparison operators are:

- < : smaller than , > bigger than
- == : are equal, != : are not equal

African School of Fundamental
Physics and Applications

# loops

A computer is particularly good a repeating things without making errors, something humans find extremely boring.

In Python we have 2 types of loops:

```
for i in range(5):
    print(“Hello World”)  # prints “Hello World” 5 times)
```

```
i = 5
while i > 0:
    print(“Hello World”)  # does the same thing
    i -= 1                # different syntax for i= i – I
```

```
while True:
   print(“Hello World!”)  # will print forever
```

```
i=5
while True:
   print(“Hello World”)
   i -= 1
   if i == 0:
       break              # breaks out of the loop
```

Now you should be ready to try yourself: First practical part. Please consult the exercise sheet and please ask if things are unclear!

## Modules

Libraries are arranged as modules and/or classes

The time module has functions that delay execution by a certain time

Modules (or functions from modules) must be imported:

```
import time
while True:
    print(“Hello World!”)
    time.sleep(1)  # or time.sleep_ms(1000) delay execution
                   # for 1s
```

## The math module

The math module supplies mathematical functions.

Instead of importing the whole module you may also import individual functions

```
from math import sin,radians
# The sin function takes angles in radians
# The radians() function converts degrees to radians
print(“sin(30°) = “,sin(radians(30))
```

African School of Fundamental Physics and Applications

## Creating you own function

You can also create you own function

Imagine you write to have a function sin_deg which calculates the sin function taking angles in degrees instead of radians

```
from math import sin,radians
def sin_deg(angle):    # definition of a function
                  # calculating the in function for
             # angles in degrees
   return sin(radians(angle)) # return s the sin value

# calling the function
print(sin_deg(30))     # print sin(30 degrees)
                       # angle in sin_deg becomes 30
```

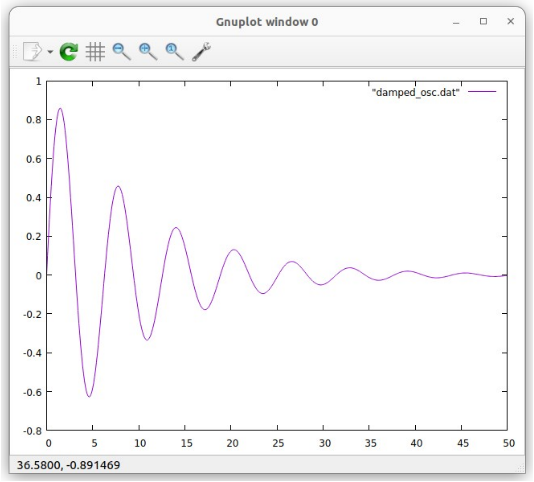
African School of Fundamental Physics and Applications

# Plotting a function

$e^{-x/10} * \sin(x)$ is the formula of a damped oscillator

Can we plot its shape?

```
from math import exp,sin

def damped_osc(x):                 # calculates the damped
                                 # oscillator function
    return exp(-1/10)*sin(x)

for i in range(500):               # calculate 500 values
    print(i/10,damped_osc(i/10)    # and print them
```



You can run the program on CPython (on the PC) redirecting its output to damped_osc.dat
python3 damped_osc.py > damped _osc.dat

Or on the ESP32 with:

ampy run damped_osc.py > damped_osc.dat

Then plot it in gnuplot with plot “damped_osc.dat” with lines

Let’s try all this in another exercises session

## Hardware access

Up to now, all programs could be run in Cpython (python3) on the PC or on MicroPython on the ESP32. (*sleep_ms* does not exist on Cpython but can be replaced by *sleep*)

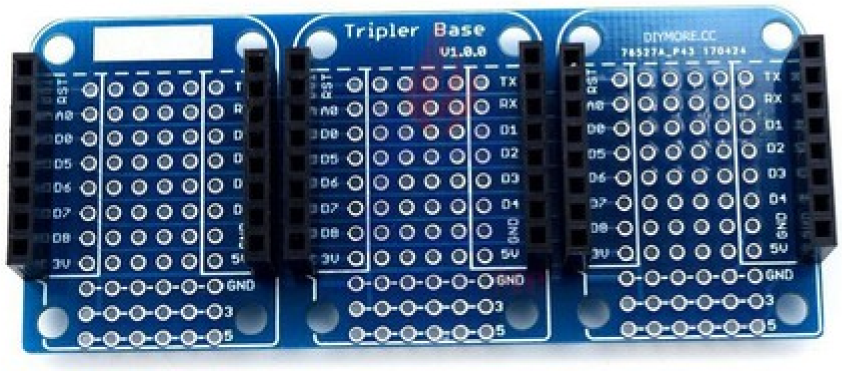
The ESP32 has hardware interfaces for

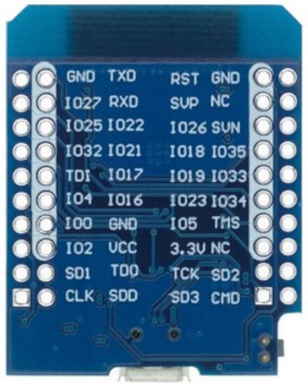
- General Purpose IO lines (GPIO, we will only use these)
- The I2C bus and SPI, the Serial Peripheral Interface
- I2S for audio signals
- WiFi and BlueTooth

African School of Fundamental Physics and Applications

## The triple board

The triple board acts as a bus system. The pins of the individual slots are connected. We put the CPU board into one of the slots and the sensor or actuator shield onto another and the triple board connects the pins.





African School of Fundamental Physics and Applications

## ESP32 CPU pinout

IOxx represents the GPIO pin numbers. IO26 corresponds to GPIO 26

| Left Column on triple base | Left Column on CPU board (GPIO) | Right Column on triple base | Right Column on CPU board (GPIO) |
|---|---|---|---|
| RST (Reset) | RST | Tx | TXD |
| A0 (ADC) | SVP (IO36) | Rx | RXD |
| D0 | IO26 | D1 | IO22 |
| D5 | IO18 | D2 | IO21 |
| D5 | IO19 | D3 | IO17 |
| D7 | IO23 | D4 | IO16 |
| D8 | IO5 | GND | GND |
| 3V3 | 3.3V | 5V | VCC |

African School of Fundamental Physics and Applications

# Sensor and actuator shields

For this short course we only use 4 different devices:

The user LED on the CPU board

The push button

The RGB LED ring
With 7 LEDs

The slider potentiometer

African School of Fundamental Physics and Applications

## Interfacing a LED

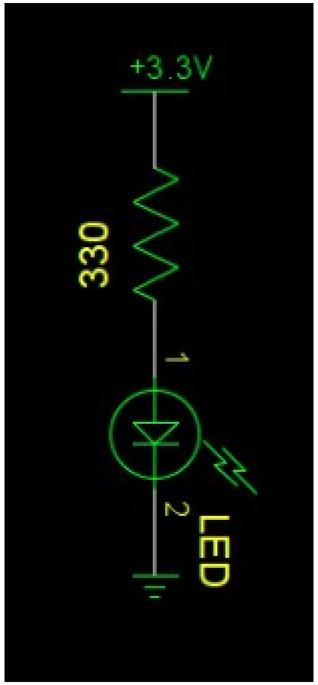
In order to drive an LED you just need a digital signal level:

- Vcc to switch the LED on
- GND to switch it off

Micropython has a driver for GPIO (General Purpose I/O) in the *machine* module name Pin

```
from machine import Pin
from utime import sleep_ms
led = Pin(19,Pin.OUT)
led.on()
sleep_ms(500)
led.off()
```

Using a loop you can easily create a blinking program

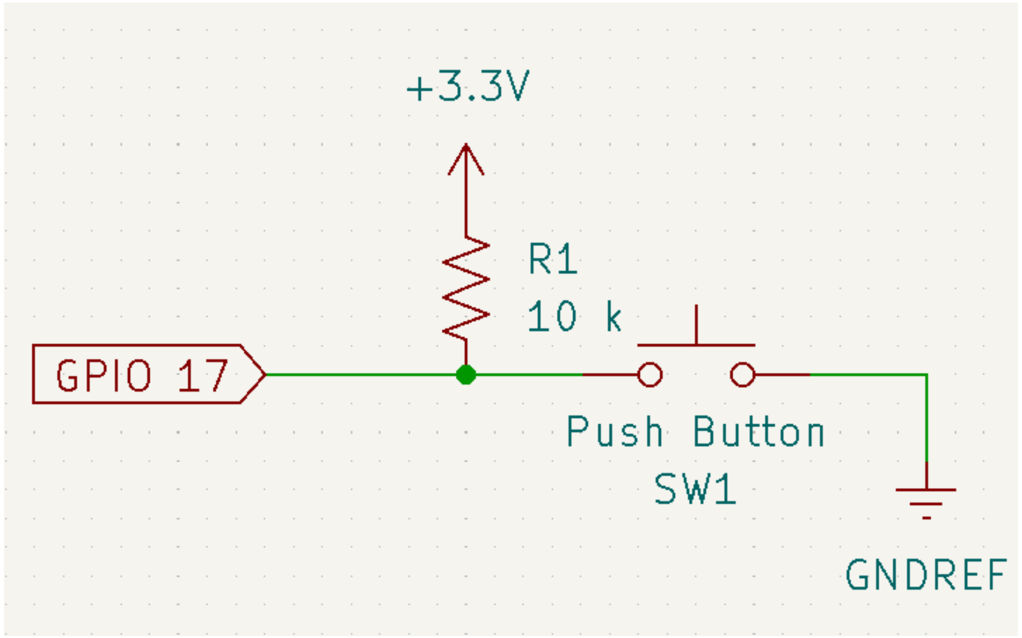


## Switch hardware

The pushbutton is connected to GND on one side and to Vcc via a pull-up resistor on the other.

On the pushbutton shield you find no resistor!

The ESP32 GPIO interface can add this resistor using a programmed command.



African School of Fundamental Physics and Applications

## Reading the switch

The switch is connected to GPIO 17

```
from machine import Pin
from time import sleep_ms

# Define an input pin on GPIO 17 and add a pull-up resistor
switch = Pin(17,Pin.IN,Pin.PULL_UP)
while True:
    # if the button is released then there is no connection to
    # GND. The pull-up resistor pulls the level to 3.3V
    if switch.value():
       print(“push button is released”)
    else:
       print(“push button is pressed”)
    sleep_ms(100)
```

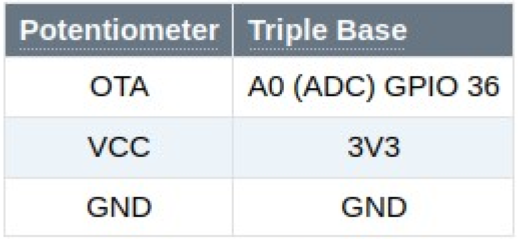
Let’s try all this. Remove the ESP32 from power, insert the push button shield and re-power the boards. Check the exercises on the user programmable LED and the switch.

African School of Fundamental Physics and Applications

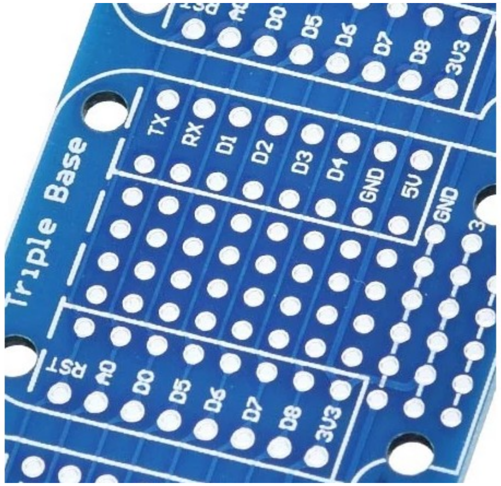
# Connecting the potentiometer

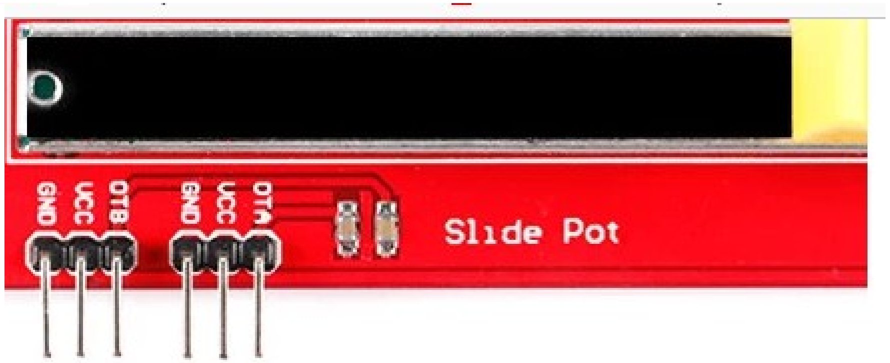
While the switch and the LED shield can simply be plugged onto the triple base, this is not the case for the potentiometer.

The potentiometer must be connected using 3 Dupont wires:



| Potentiometer | Triple Base |
|---|---|
| OTA | A0 (ADC) GPIO 36 |
| VCC | 3V3 |
| GND | GND |





## Reading the Potentiometer

The 12 bit Analogue to Digital Converter (ADC) splits the signal level range into 4096 slices. If the signal falls into a slice then the corresponding number is returned.

The level range for the ESP32 ADC is 0..1V. However, an attenuator in front of the ADC allows adapting the range to 0.. 3.3V. We will use 11 dB attenuation.

MicroPython supplies the driver necessary to read out the ADC.

African School of Fundamental Physics and Applications

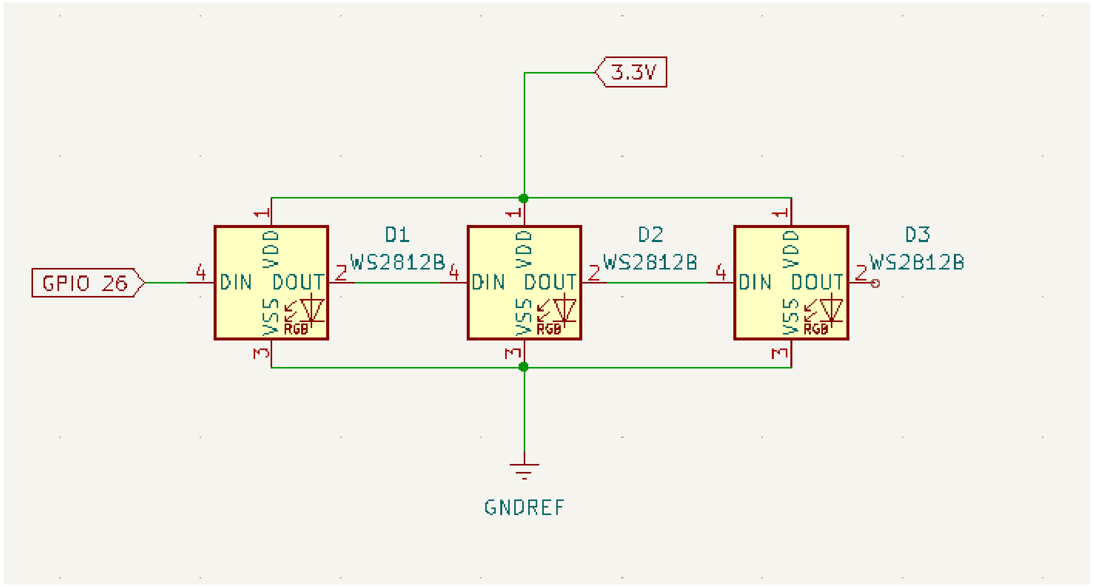
## ADC readout program

```
from machine import ADC,Pin
from time import sleep_ms

# create an ADC object on pin 36 with 11dB attenuation
slider = ADC(Pin(36, atten=ADC.ATTN_11DB)

while True:
     # Read the ADC every 100 ms and print the result
     print("Raw 12 bit value from slider: ",slider.read())
     sleep_ms(100)
```

African School of Fundamental Physics and Applications

## The rgb LED ring

In contrast to the user programmable LED on the CPU card the rgb LED ring uses WS2812B cascadable, addressable rgb (red,green,blue) LEDs.



On the rgb LED shield there are 7 of these LEDs but you can find LED chains with tens or even hundreds of these LEDs

## The NeoPixel driver

Programming the WS2812B LEDs can be quite complex because of the communication protocol between the CPU and the LEDs, whose timing must be strictly respected.

MicroPython supplies a NeoPixel driver which implements this protocol and makes programming the rgb LED shield a child’s game.

African School of Fundamental Physics and Applications

## Programming the WS2812B

```
from machine import Pin
from neopixel import NeoPixel

NO_OF_LEDS = 7
pin = Pin(26,Pin.OUT) # define the communication pin as output
# create a NeoPixel object on GPIO 26 for 7 LEDs
ws2812b = NeoPixel(pin,NO_OF_PINS)
red = (31,0,0)          # a tuple with r,g,b values, here pure red
ws2812b[0]=red          # set LED no 0 to red
ws2812b.write()         # communicate it to the LEDs
```

The color components are defined as 8 bit values (0..255)

**With the maximum intensity the WS2812B is extremely bright and might even damage your eyes if you look directly into it.**
**I therefore recommend to limit color intensities to 31**

## LED number

How do we know, which LED number corresponds to which physical LED?

We cannot know, because it depends on how the LED shield is cabled.

We can easily find out however:

- Light each LED number (0..6) for 1 s (e.g. in red, setting the colors to 31,0,0) and make sure all other LEDs are switched of (color value: 0,0,0)
- Observe the sequence at which the LEDs light up

## A little project

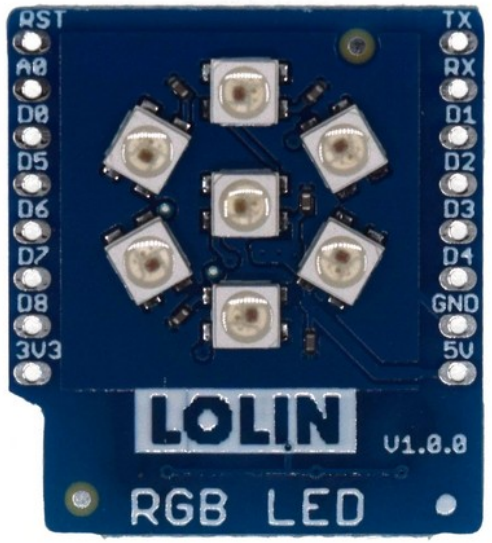
As a small project, let us read the slider and set a LED on the rgb LED ring depending on the signal level. At the smallest signal, the top LED will be lit.

At the next level it will be the LED next to it, going in clockwise direction until we reach the LED left to the top one.

Light the middle LED for the highest signal level.

Signal levels are often represented in colors:

- Blue (cold) for the lowest level
- Red for a very high level
- White for the highest level.



## Colors

For the project we only use the basic colors:

- Blue (0,0,31)
- Cyan (0,31,31)
- Green you find the others yourself...
- Yellow
- Magenta
- Red
- White

Light the LED with the color corresponding to the signal level.