Exercises

First exercise session:

A bit of programming

Connect your ESP32 to the PC using the micro USB cable and start thonny. Locate the shell window and check if you see the PEPL prompt ">>>". If not, try to press <enter>. Make sure the shell window has the input focus. If this does not work, press the stop button. If this also doesn't help, press the reset button on the ESP32 CPU board.

Playing with REPL

```
print "Hello World!"
calculate 127,9 * 157.6 * 17.2 /(3.5*2**4) # 2**4 means 2 to the
power of 4
```

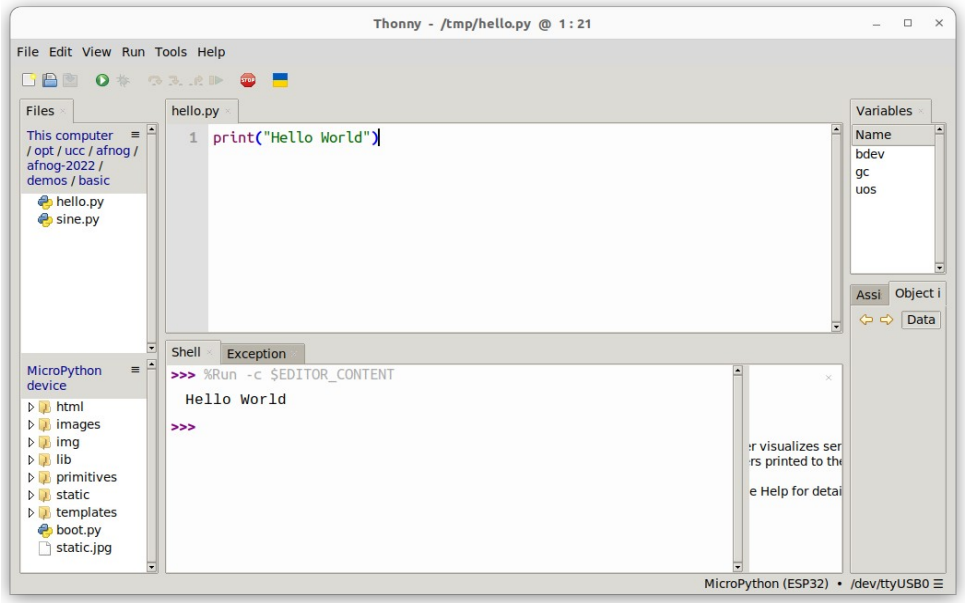
you may try to calculate e.g 2**4 separately first and check that you get the result expected
Continue experimenting with REPL

Hello World

Instead of working on the shell window, we will now write a Python script (a small program). For this to work, select File -> New. This will create an editor window named <untitled>. Save this empty program to the disk on your PC. Select File -> Save, select "This Computer" give it a file name (e.g. hello.py) and press "OK". The window title changes to hello.py.

Now enter
```
print("Hello World!")
```

Finally, run the program by pressing the green button with the white triangle. You should see "Hello World!" printed in the shell window. This is what you should get:



Some Arithmetic

Assign 2 values to 2 variables a and b. Calculate the sum, the difference, the product and the division and print the results. Start assigning integer values. Try integer and float division.

Then assign float values and try again.

Conditions

Assign again values to two variables a and b with a being bigger than b . In your program, check if a is equal, bigger or smaller than b. Modify the values you assign to a and b, with b now bigger than a and try again.

In the lectures, we used the "<" and the ">" conditions to find out if the value in the variable a is bigger, smaller or equal to the value in variable b. Can you modify the program using "<" and "!=" to find out?

What happens if you try:

```
if a:
   print("yes")
else:
   print("No")
```

Try this with a being assigned to zero and a being assigned to 7. What do you conclude?

Loops

Write a script that prints "Hello World!" forever. You can stop the program with the red Stop button.

Write a program that prints "Hello World!" 7 times. Do this using a while loop and in a second program, using a for loop, Which of the programs is more elegant?

Too easy? Try to calculate the Fibonacci numbers

The Fibonacci numbers are defined as follows:

- F(0) = 0
- F(1) = 1
- F(n) = F(n-1) + F(n-2)

Write a program that prints the Fibonacci series of numbers for n=10, n=20, n=30.

This is the end of the first exercise session

Second exercise session

MicroPython modules

Modify the loop program in such a way that "Hello World!" is printed only every second. Import the utime module and make the program sleep for 1s after each print statement.

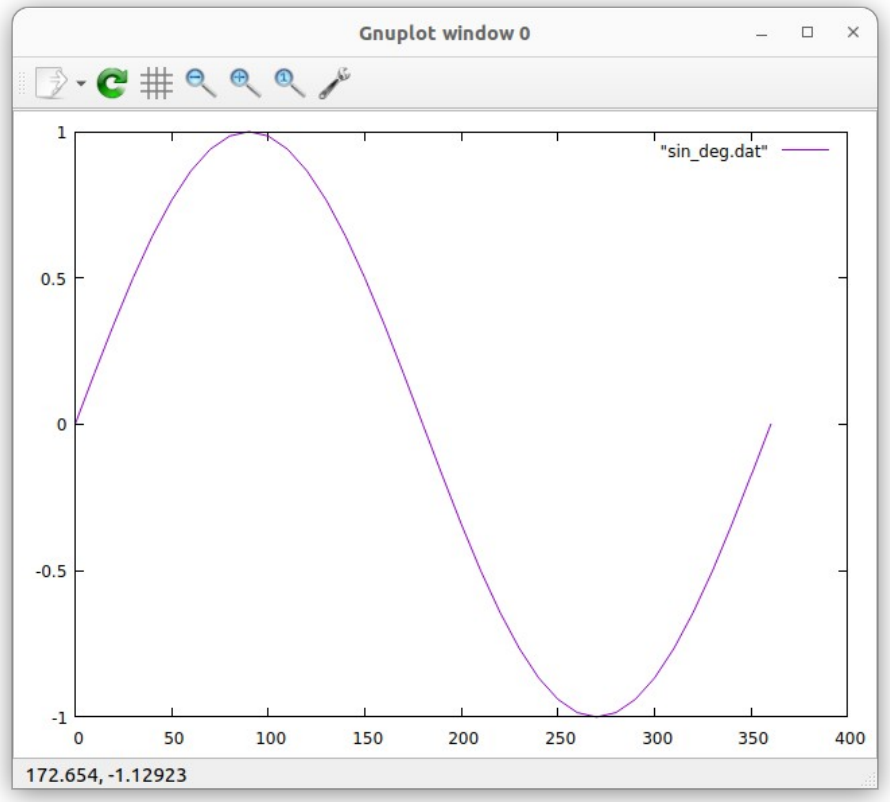
Import the sin function and the value of pi from the math module

```
from math import sin,pi
```
Remember that the sin function takes its values in units of radians. The angle values for a complete sin period will range 0 .. 2*pi. Print the angles and the sin values for a full sin period using 30 equidistant angle values. (x and sin(x)).

Creating and using your own function

Create a sin_deg function which takes angles in units of degrees instead of radians. Print the angles and the sin values for 36 equidistant angle values.

You may copy/paste the result to a new editor window in thonny and save them to a file on the PC. Then you can plot the function using gnuplot.



Too easy? Calculate the throwing parabola

If the above is too simple for you, you may try to calculate the *throwing parabola*. Let's say, a stone is thrown at a speed of 30m/s and an angle of 30° with respect to ground. Calculate the trajectory of the stone until it hits the ground. You way define a function *trajectory*, taking initial speed, the angle and the time resolution for which you want to calculate the stone positions.

v_hor = v_initial * cos(angle)

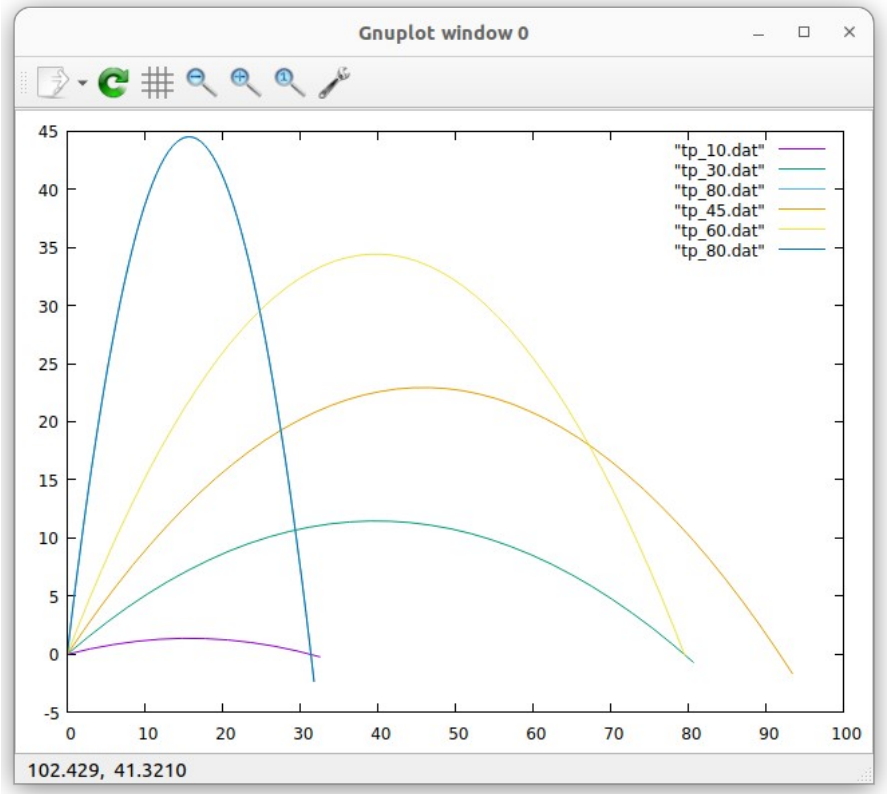
v_ver = v_initial * sin(angle)

x(t) = v_hor * t

y(t) = v_ver * t - 1/2 * g * t**2

Like this, you can easily calculate the trajectories for different angles and different initial speeds.

In the plot below an initial speed on 30 m/s is assumed. Tp_10 shows the trajectory for an angle of 10 degrees.



End of the second exercise session

Third exercise session

Programming the user LED on the CPU card

Write a program that blinks the user programmable LED on the CPU card at a frequency of 1 Hz (500 ms on, 500 ms off)

Write a function that blinks the LED once with a delay that is passed as a parameter:

```
def blink_once(delay):
```

Test the function with a main program calling is with different delays.

Use the function to implement a program that blinks an SOS: 3 short pulses, followed by 3 long pulses, followed by 3 short pulses, followed by a 2s pause. You may use a 200 ms delay for the short pulses and 700 ms delay for the long ones.

Reading the pushbutton

Plug the pushbutton shield into the triple base.
Write a program that reads the push button state ever 100 ms and prints its state (pressed or released)

Improve the program printing the state only when there was a state change.

Too easy? Change the light intensity on the LED using PWM

Write a program that linearly increases the light intensity of the user programmable LED using **P**ulse **W**idth **M**odulation (PWM).

End of the third exercise session

Forth exercise session

Reading an analogue signal level from the slider potentiometer

Read the documentation of the ADC driver in the MicroPython manual
Connect the potentiometer to the triple base according to this table:

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

Create an ADC object and make sure you set the attenuator to 11 dB
Read the slider value every 100 ms and print its raw 12 bit value and print it.
Move the slider and observe the changes in the value

Controlling the rgb LED ring

The LED ring consists of 7 WS2812B addressable and cascadable LEDs. Each rgb LED consists of 3 tiny single color LEDs emitting red, green and blue light. The color you finally see is the mixture between these color components. The MicroPython driver for this type of LED is called NeoPixel. The color is defined as a tuple with three 8 bit color component intensities. 8 bits means that you can have values in the range 0..255. Since the LEDs are extremely bright, please restrict the values to 0..31.

Write a program trying to find out to which LED an LED number corresponds to. Where on the LED ring is LED0, LED1 ... Do this by lighting the LEDs one by one with only the red color component switched on. The green and blue are set to zero.

Then change the color. Try all color combinations (31,0,0), (0,31,0), (0,0,31) but also (31,31,0), (31,0,31) ... Which colors do you get?

You can further modify the colors by changing the intensity values. For example: (12,27,5) Play around trying to generate as many different colors as you like.

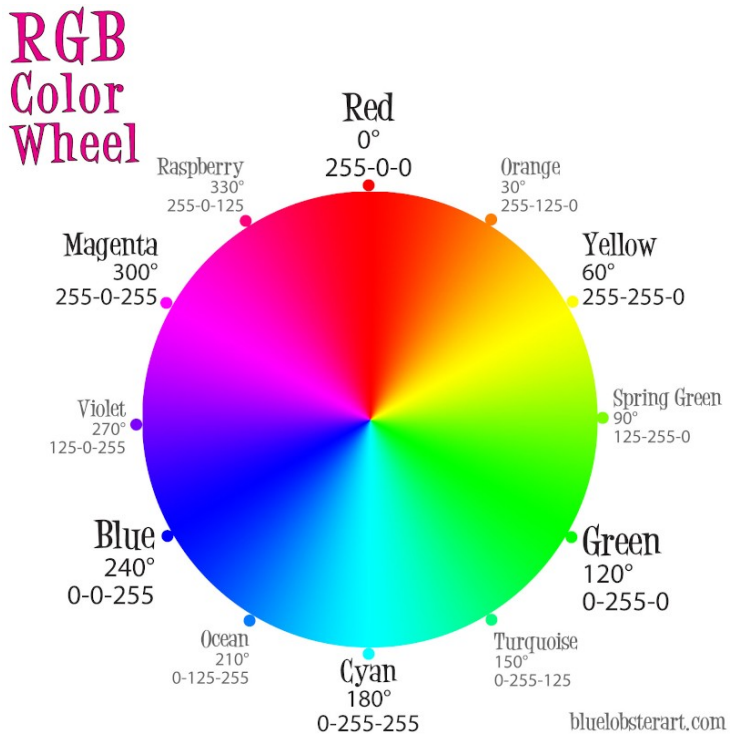
Write a program that lights the top LED in blue and then lights the following LEDs in clockwise direction to :

- blue (top LED)
- cyan
- green
- yelllow

- magenta
- red
- the middle LED becomes white

Too easy? Program the Color Wheel

Write a program that shows all the colors of the color wheel on the LEDs of the rgb LED ring. The color on the LEDs will slowly change from red to yellow to green … to magenta to red.



A small project

As a small project, we are going to combine the reading of the ADC and the control of the rgb LED ring. As we saw, the ADC delivers values between 0 and 4095 (12 bits). We can divide this range in 7 slices, corresponding to the 7 LEDs in the ring. For the lowest slice, the top LED will be lit. Then we go clockwise around the ring: the next LED lit will be to the right of the first one. For the highest slice, the middle LED will be lit.
First, print the led number for the signal level read from the slider. Move the slider and make sure that you get the correct LED number.

Then light the LED you found with a single color, e,g, red.

Very often, the color indicates the signal level: For low signal levels, we use "cold" colors (blue, cyan) for higher levels we use "hot" colors (magenta, red, white). The color for the top LED will therefore be blue, the color for the middle LED will be white.

We can then attribute a different color to each LED, as the lighting of the LED depends on the signal level reached. We only use the basic colors:

LED number	color	
1	blue	(0,0,31)
2	cyan	(0,31,31)

3	green	… you find the others
4	yellow	
5	magenta	
6	red	
0	white	

The top LED (lowest signal level) will therefore have the color blue, while the middle LED (highest signal level) will become white.

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

That’s all folks!