Feuille d’exercices

Exercice de programmation de base

Un peu de programmation

Connectez votre ESP32 au PC à l'aide du câble micro USB ou USB-C et démarrez Thonny. Localisez la fenêtre du shell et vérifiez si vous voyez le prompte du REPL ">>>". Sinon, essayez d'appuyer sur <enter>. Assurez-vous que la fenêtre shell a le focus d’entrée. Si cela ne fonctionne pas, appuyez sur le bouton d'arrêt. Si cela ne résout pas non plus le problème, appuyez sur le bouton de réinitialisation de la carte CPU ESP32.

En jouant avec REPL

```
print "Hello World!"
calculate 127,9 * 157.6 * 17.2 /(3.5*2**4) # 2**4 signifie 2 à la puissance 4
```

Vous pouvez d'abord essayer de calculer par exemple 2**4 séparément et vérifier que vous obtenez le résultat attendu.
Continuez à expérimenter avec REPL

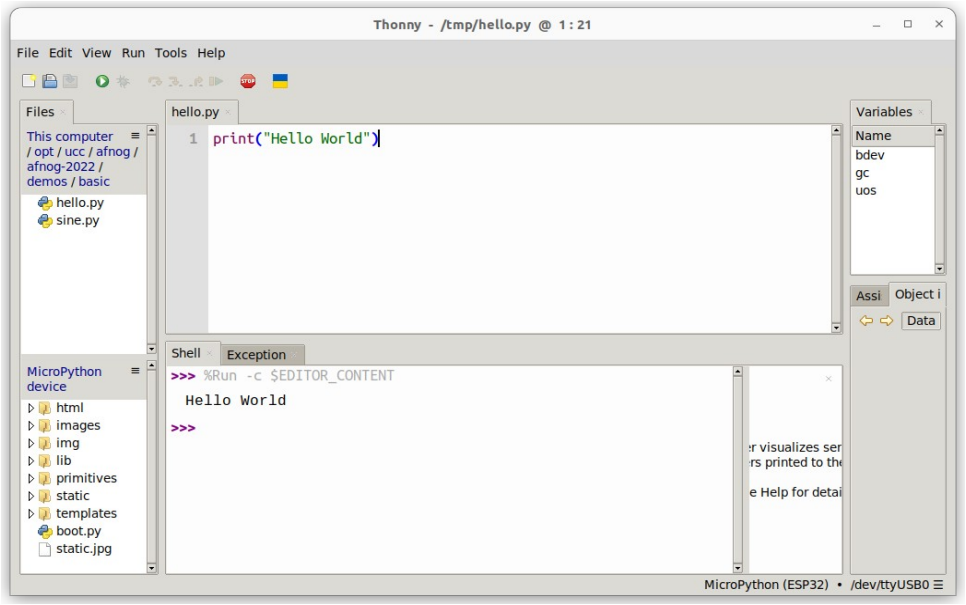
Hello World

Au lieu de travailler sur la fenêtre shell, nous allons maintenant écrire un script Python (un petit programme). Pour que cela fonctionne, sélectionnez File -> New. Cela créera une fenêtre d'éditeur nommée <untitled>. Enregistrez ce programme vide sur le disque de votre PC. Sélectionnez File -> Save, sélectionnez "This Computer", donnez-lui un nom de fichier (par exemple hello.py) et appuyez sur "OK". Le titre de la fenêtre devient hello.py.

Entrez maintenant

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

Enfin, exécutez le programme en appuyant sur le bouton vert avec le triangle blanc. Vous devriez voir « Hello World ! » imprimé dans la fenêtre du shell. Voici ce que vous devriez obtenir :



Un peu d'arithmétique

Attribuez 2 valeurs à 2 variables a et b. Calculez la somme, la différence, le produit et la division et imprimez les résultats. Commencez à attribuer des valeurs entières. Essayez la division entière et flottante.

Attribuez ensuite des valeurs flottantes et réessayez.

Conditions

Attribuez à nouveau des valeurs à deux variables a et b, a étant plus grand que b . Dans votre programme, vérifiez si a est égal, plus grand ou plus petit que b. Modifiez les valeurs que vous attribuez à a et b, avec b maintenant plus grand que a et réessayez.

Dans les cours, nous avons utilisé les conditions "<" et ">" pour savoir si la valeur de la variable a est plus grande, plus petite ou égale à la valeur de la variable b. Pouvez-vous modifier le programme en utilisant "<" et "!=" pour le savoir ?

Que se passe-t-il si vous essayez :

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

Essayez ceci avec a affecté à zéro et a affecté à 7. Qu'en concluez-vous ?

Boucles

Écrivez un script qui imprime « Hello World ! » pour toujours. Vous pouvez arrêter le programme avec le bouton rouge Stop.

Écrivez un programme qui affiche « Hello World ! » 7 fois. Faites cela en utilisant une boucle while et dans un deuxième programme, en utilisant une boucle for. Lequel des programmes est le plus élégant ?

Trop facile? Essayez de calculer les nombres de Fibonacci

Les nombres de Fibonacci sont définis comme suit :

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

Écrivez un programme qui imprime la série de nombres de Fibonacci pour n=10, n=20, n=30.

C'est la fin de la première séance d'exercices

Exercices sur les fonctions et modules

Modules MicroPython

Modifiez le programme de boucle de telle sorte que "Hello World!" est imprimé seulement toutes les secondes. Importez le module utime et mettez le programme en veille pendant 1 seconde après chaque instruction d'impression.

Importez la fonction sin et la valeur de pi depuis le module mathématique

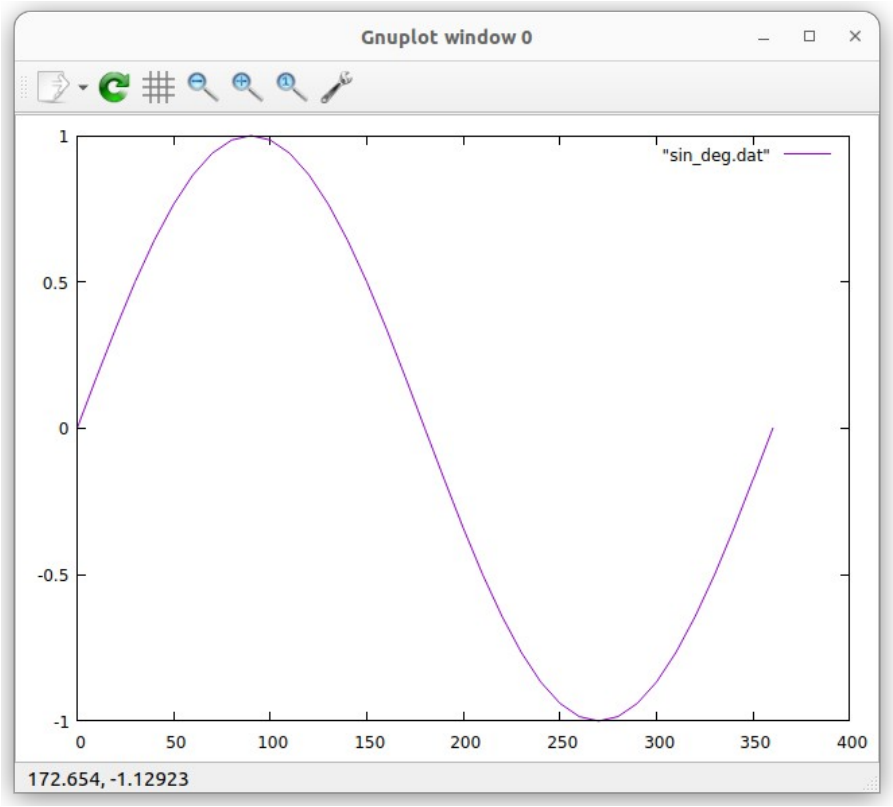
```
from math import sin,pi
```

N'oubliez pas que la fonction sin prend ses valeurs en unités de radians. Les valeurs d'angle pour une période de sin complète seront comprises entre 0 .. 2*pi. Imprimez les angles et les valeurs de sin pour une période de sin complète en utilisant 30 valeurs d'angle équidistantes. (x et sin(x)).

Créer et utiliser votre propre fonction

Créez une fonction sin_deg qui prend les angles en degrés au lieu de radians. Imprimez les angles et les valeurs de sin pour 36 valeurs d'angle équidistantes.

Vous pouvez copier/coller le résultat dans une nouvelle fenêtre de l'éditeur dans Thonny et les enregistrer dans un fichier sur le PC. Ensuite, vous pouvez tracer la fonction en utilisant gnuplot.



Trop facile? Calculer la parabole de lancement

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.

Si ce qui précède est trop simple pour vous, vous pouvez essayer de calculer *la parabole de lancement*. Disons qu'une pierre est lancée à une vitesse de 30 m/s et un angle de 30° par rapport au sol. Calculez la trajectoire de la pierre jusqu'à ce qu'elle touche le sol. Vous définissez ainsi une fonction *trajectoire*, en prenant la vitesse initiale, l'angle et la résolution temporelle pour laquelle vous souhaitez calculer les positions des pierres.

v_hor = v_initial * cos(angle)

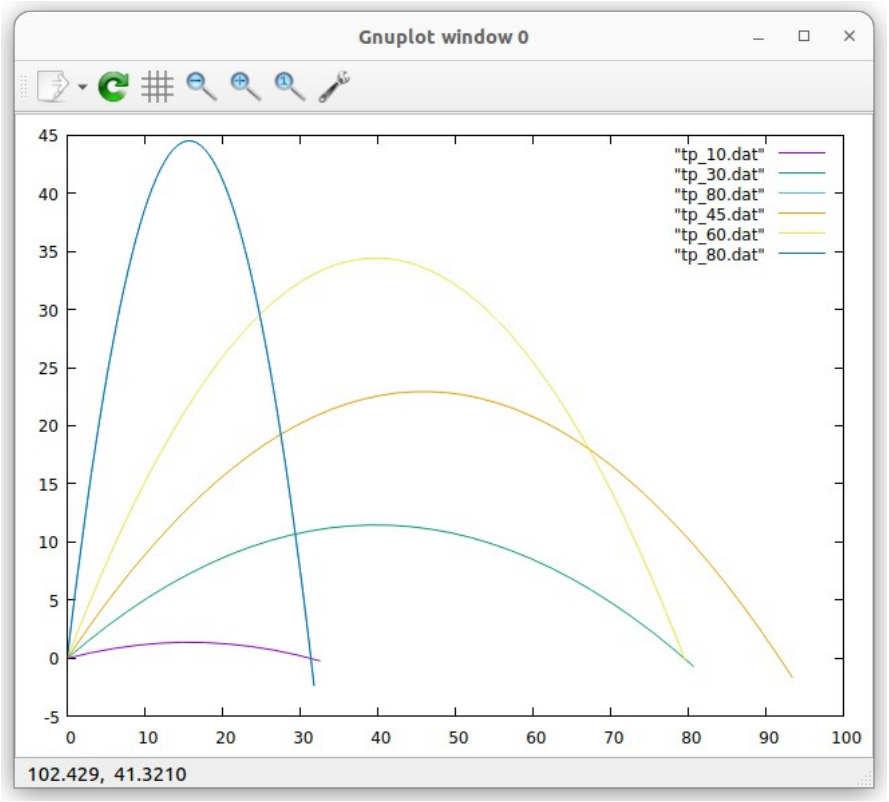
v_ver = v_initial * sin(angle)

x(t) = v_hor * t

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

De cette façon, vous pouvez facilement calculer les trajectoires pour différents angles et différentes vitesses initiales.

Dans le graphique ci-dessous, une vitesse initiale de 30 m/s est supposée. Tp_10 montre la trajectoire pour un angle de 10 degrés. Une résolution en temps de 0.1 s est utilisé.



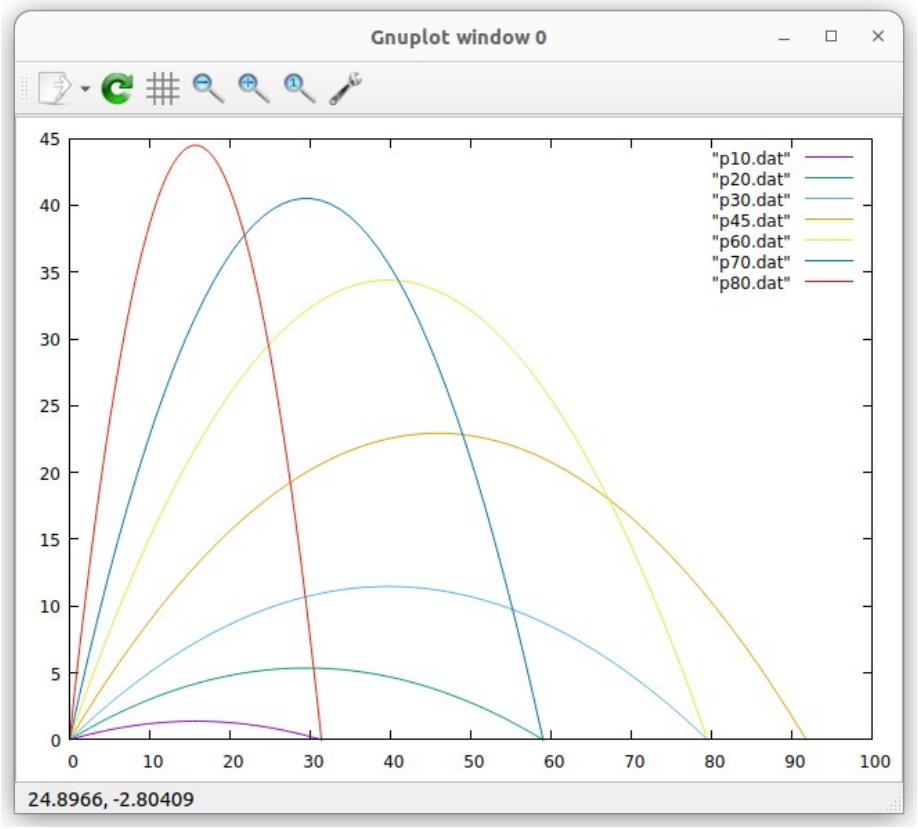
Améliorer le programme

Ce programme peut encore être amélioré en traçant le même nombre de points pour chaque parabole. Pour y parvenir, nous pouvons d'abord calculer la durée totale du vol jusqu'à ce que l'objet touche à nouveau le sol :

```
vt = 1/2*g*t**2, v => 1/2*g*t, t= 2*v/g
```

Ensuite, nous calculons un nombre fixe de points (par exemple 50) pour chaque parabole dans une boucle for.

Ainsi, on voit facilement que pour un angle de 30° et 60° la distance parcourue est la même. Cela est également vrai pour 10° et 80° ou 20° et 70°.



Fin de la deuxième séance d'exercices

Exercices avec GPIO, le bouton poussoir et les LED

Programmation de la LED utilisateur sur la carte CPU

Écrivez un programme qui fait clignoter la LED programmable par l'utilisateur sur la carte CPU à une fréquence de 1 Hz (500 ms allumée, 500 ms éteinte)
La LED utilisateur de la carte CPU est connectée au GPIO 2.

Écrivez une fonction qui fait clignoter la LED une fois avec un délai passé en paramètre :

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

Testez la fonction avec un programme principal appelant avec des délais différents.

Utilisez la fonction pour mettre en œuvre un programme qui fait clignoter un SOS : 3 impulsions courtes, suivies de 3 impulsions longues, suivies de 3 impulsions courtes, suivies d'une pause de 2 s. Vous pouvez utiliser un délai de 200 ms pour les impulsions courtes et de 700 ms pour les impulsions longues.

Lecture du bouton poussoir

Branchez le protège-bouton-poussoir dans la base triple.
Écrivez un programme qui lit l'état du bouton-poussoir toutes les 100 ms et imprime son état (enfoncé ou relâché). Le bouton poussoir est connecté au GPIO 17.

Améliorez le programme en imprimant l'état uniquement en cas de changement d'état.

Trop facile? Changer l'intensité lumineuse de la LED à l'aide de PWM

Écrivez un programme qui augmente linéairement l'intensité lumineuse de la LED programmable par l'utilisateur à l'aide de la modulation de largeur d'impulsion (PWM).**P**ulse **W**idth **M**odulation (PWM).

Fin de la troisième séance d'exercices

Exercises with Analogue Signals and NeoPixels

Lecture d'un niveau de signal analogique à partir du potentiomètre lineaire

Lire la documentation du pilote ADC dans le manuel MicroPython
Connectez le potentiomètre à l'embase triple selon ce tableau :

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

Créez un objet ADC et assurez-vous de régler l'atténuateur sur 11 dB
Lisez la valeur du curseur toutes les 100 ms et imprimez sa valeur brute de 12 bits.
Déplacez le curseur et observez les changements de valeur

Fin de la quatrième séance d'exercices

Exercices additionnels

Contrôler l'anneau LED RGB

L'anneau LED se compose de 7 LED adressables et cascadables WS2812B. Chaque LED RGB se compose de 3 minuscules LED de couleur unique émettant une lumière rouge, verte et bleue. La couleur que vous voyez finalement est le mélange de ces composants de couleur. Le pilote MicroPython pour ce type de LED s'appelle NeoPixel. La couleur est définie comme un tuple avec trois intensités de composantes de couleur de 8 bits. 8 bits signifie que vous pouvez avoir des valeurs comprises entre 0 et 255. Étant donné que les LED sont extrêmement lumineuses, veuillez limiter les valeurs à 0..31.

Écrivez un programme essayant de savoir à quelle LED correspond un numéro de LED. Où sur l'anneau LED se trouve LED0, LED1... Faites cela en allumant les LED une par une avec seul le composant de couleur rouge allumé. Le vert et le bleu sont mis à zéro.
Créez une table de mappage telle que la première LED soit mappée sur celle du haut, les LED suivantes soient adressées dans le sens des aiguilles d'une montre et la dernière LED soit celle du milieu.

Changez ensuite la couleur. Essayez toutes les combinaisons de couleurs (31,0,0), (0,31,0), (0,0,31) mais aussi (31,31,0), (31,0,31)... Quelles couleurs font vous obtenez?

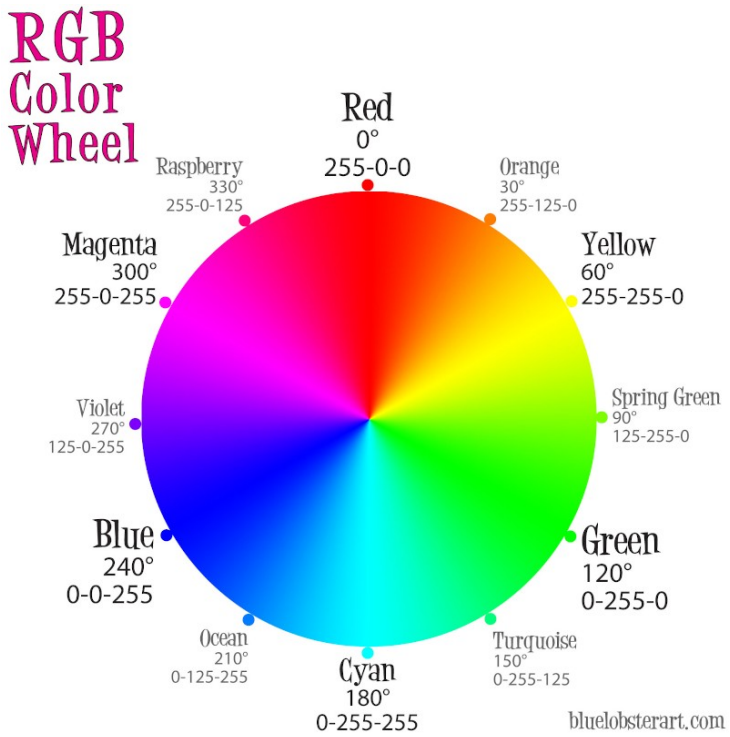
Vous pouvez modifier davantage les couleurs en changeant les valeurs d'intensité. Par exemple : (12,27,5) Jouez en essayant de générer autant de couleurs différentes que vous le souhaitez.

Écrivez un programme qui allume la LED supérieure en bleu, puis allume les LED suivantes dans le sens des aiguilles d'une montre pour :

- bleu (top LED)
- cyan
- green
- jaune
- magenta
- rouge
- la LED devient blanc

Trop facile? Programmer la roue chromatique

Écrivez un programme qui affiche toutes les couleurs de la roue chromatique sur les LED de l'anneau LED RGB. La couleur des LED passera lentement du rouge au jaune en passant par le vert… du magenta au rouge.



Un petit projet

Dans le cadre d'un petit projet, nous allons combiner la lecture de l'ADC et le contrôle de l'anneau LED RGB. Comme nous l'avons vu, l'ADC délivre des valeurs comprises entre 0 et 4095 (12 bits). On peut diviser cette plage en 7 tranches, correspondant aux 7 LED de l'anneau. Pour la tranche la plus basse, la LED supérieure sera allumée. Puis on fait le tour de l'anneau dans le sens des aiguilles d'une montre : la prochaine LED allumée sera à droite de la première. Pour la tranche la plus élevée, la LED du milieu sera allumée.
Tout d'abord, imprimez le numéro de LED pour le niveau de signal lu à partir du curseur. Déplacez le curseur et assurez-vous que vous obtenez le bon numéro de LED.
Allumez ensuite la LED que vous avez trouvée avec une seule couleur, par exemple le rouge.
Très souvent, la couleur indique le niveau du signal : Pour les niveaux de signal faibles, on utilise des couleurs « froides » (bleu, cyan) pour les niveaux plus élevés, nous utilisons des couleurs « chaudes » (magenta, rouge, blanc). La couleur de la LED du haut sera donc bleue, la couleur de la LED du milieu sera blanche.

On peut alors attribuer une couleur différente à chaque LED, car l'éclairage de la LED dépend du niveau de signal atteint. Nous utilisons uniquement les couleurs de base :

LED number	color	
1	bleu	(0,0,31)
2	cyan	(0,31,31)
3	vert	… trouvez les autres couleurs
4	jaune	
5	magenta	
6	rouge	
0	blanc	

La LED du haut (niveau de signal le plus bas) aura donc la couleur bleue, tandis que la LED du milieu (niveau de signal le plus élevé) deviendra blanche.

Allumez la LED avec la couleur correspondant au niveau du signal.

Exercises of the I2C bus

Finding I2C addresses

Connect the SHT30 temperature and humidity sensor to the WeMos D1 mini bus.

Find the addresses of all I2C slaves connected to the bus and print them. Use the ESP32 hardware interface of the I2C bus connected to bus no. 1. The I2C interface uses the following pins:

	Bus number	**SCL**	**SDA**
	1		
GPIO		22	21

You can find the MicroPython functions to be used with the I2C bus on https://docs.micropython.org/en/latest/esp32/quickref.html#hardware-i2c-bus.

i2c.scan returns an address list of all I2C slaves found on the bus. With just the SHT30 connected you should see:

```
Scanning the I2C bus
Program written for the workshop on IoT at the
African Internet Summit 2019
Copyright: U.Raich
Released under the Gnu Public License
Running on ESP32
I2C slaves found on the bus:
0x45

>>>
```

I found this program very useful because each time an I2C interface is tested we need to make sure that the device is actually seen by our program. Try to improve the formatting of the program to print all I2C addresses in a table:

```
Scanning the I2C bus
Program written for the workshop on IoT at the
African Internet Summit 2019
Copyright: U.Raich
Released under the Gnu Public License
Running on ESP32
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- 45 -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

>>>
```

The SHT30

Sensors can be quite complex devices with a big number of registers and programming them can be a challenge. You may check this out for yourself, having a look at the SHT30 data sheet.

To make things easier for you, a driver is included in the MicroPython interpreter installed on your ESP32 CPU.

Write a program to read the serial number of your SHT30 and then repeatedly read the current temperature and humidity at a frequency of 1 Hz and print the results.