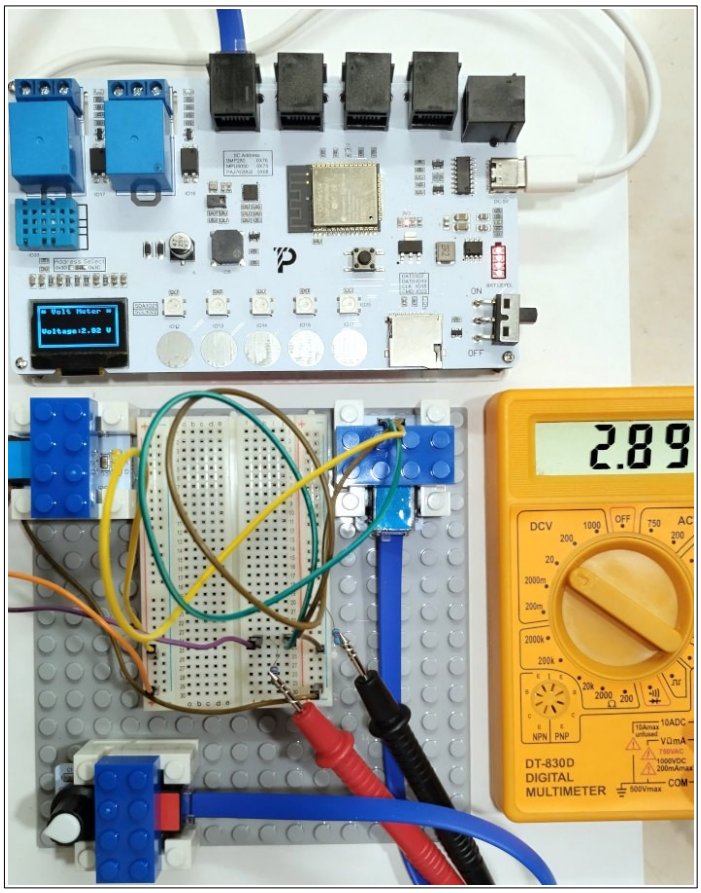
Άσκήσεις Micropython

στο σύστημα ARD:Icon II της Polytech



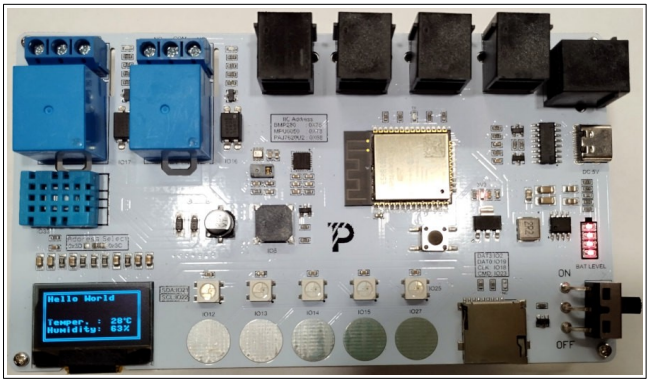
Έκδοση 1.0

Σταύρος Σ. Φώτογλου ΠΕ86
Ε.Κ. Πρέβεζας – 1ο ΕΠΑ.Λ. Πρέβεζας

Πρέβεζα 2025

Άσκηση 1

Στην πλακέτα ARD:icon II (IOT) του κιτ ρομποτικής S4T της Polytech, να γίνει πρόγραμμα σε Micropython το οποίο εμφανίζει την θερμοκρασία και την σχετική υγρασία από τον αισθητήρα DHT 11 στην ενσωματωμένη οθόνη OLED του συστήματος. Οι τιμές θα ανανεώνονται κάθε 2 δευτερόλεπτα.



Μέσα στο σύστημα αρχείων πρέπει να αντιγραφτεί η βιβλιοθήκη ssd1306.py

```
#Πρόγραμμα το οποίο παρουσιάζει μετρήσεις θερμοκρασίας και υγρασίας από DHT11 στην οθόνη OLED
import machine, time, dht
i2c = machine.I2C(0, sda = machine.Pin(21), scl = machine.Pin(22), freq = 400000)
#Ή εναλλακτικά
#i2c = machine.SoftI2C(sda=machine.Pin(21), scl=machine.Pin(22))
from ssd1306 import SSD1306_I2C
sensor = dht.DHT11(machine.Pin(33))
sensor.measure()

#display = SSD1306_I2C(128, 64, i2c) #Λίπλα μπορώ να βάλω διεύθυνση π.χ. SSD1306_I2C(128, 64, i2c, 0x3c)
display = SSD1306_I2C( width=128, height=64, i2c=i2c, addr=0x3c, external_vcc=False )
display.fill(0) #Καθαρίζει οθόνη
#Δημιούργησε στικέτες
display.text('Hello World', 5, 5, 1) #Θέση x, y, color 1/0
display.text("Temper. : ", 5, 35, 1)
display.ellipse(106, 36, 1, 1, 1)
display.text("C", 108, 35, 1)
display.text("Humidity: ", 5, 45, 1)
display.text("%", 105, 45, 1)
#Δημιούργησε διπλό περίγραμμα
display.rect(0,0,127,63,1,0) #x, y, width, heigh, color=0/1, fill=0/1
display.rect(2,2,123,59,1,0) #x, y, width, heigh, color=0/1, fill=0/1
display.show() #Παρουσίασε στην οθόνη

#Για πάντα
while True:
    sensor.measure()
    display.rect(80, 34, 25, 9, 0, True) #Σβήσε παλιά τιμή
    display.text("{: 3}".format(sensor.temperature()), 80, 35, 1) #Εμφάνισε νέα τιμή
    display.rect(80, 44, 25, 9, 0, True) #Σβήσε παλιά τιμή
    display.text("{: 3}".format(sensor.humidity()), 80, 45, 1) #Εμφάνισε νέα τιμή
    print("Θερμοκρασία:", sensor.temperature()) #Εμφάνισε στο τερματικό
    print("Σχετ. Υγρασία:", sensor.humidity())
    display.show() #Παρουσίασε στην οθόνη
    time.sleep(2) #Περίμενε 2 sec
```

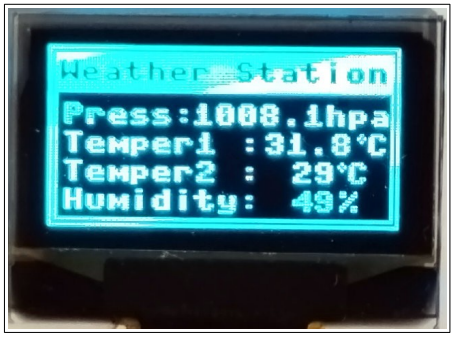
Άσκηση 2

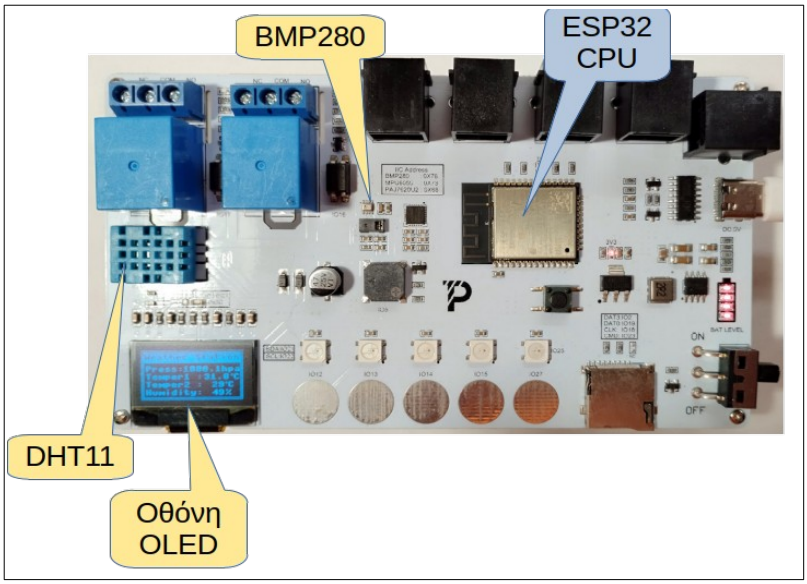
Να τροποποιηθεί το προηγούμενο ώστε να εμφανίζει και μετρήσεις του αισθητήρα BMP280. Ο αισθητήρας BMP280 μπορεί να μετρήσει ατμοσφαιρική πίεση και θερμοκρασία.

Μέσα στο σύστημα αρχείων πρέπει να αντιγραφτεί η βιβλιοθήκη ssd1306.py και η βιβλιοθήκη bmp280.py.

```
# Πρόγραμμα το οποίο παρουσιάζει μετρήσεις θερμοκρασίας και υγρασίας από DHT11
# και ατμ. πίεσης και θερμοκρασίας από το BMP280 στην οθόνη OLED
import machine, time, dht, math
i2c = machine.I2C(0, sda = machine.Pin(21), scl = machine.Pin(22), freq = 400000)
#Ή εναλλακτικά
#i2c = machine.SoftI2C(sda=machine.Pin(21), scl=machine.Pin(22))
from ssd1306 import SSD1306_I2C #Ο ελεγκτής της οθόνης OLED
sensor = dht.DHT11(machine.Pin(33)) #Ο αισθητήρας DHT11
from bmp280 import *
bmp = BMP280(i2c) #Στιγμιότυπο του BMP280
#Στιγμιότυπο της οθόνης.
display = SSD1306_I2C(width=128, height=64, i2c=i2c, addr=0x3c, external_vcc=False)
display.fill(0) #Καθαρίζει οθόνη

#Δημιούργησε διπλό περίγραμμα
display.rect(0, 0, 127, 63, 1, 0) #x, y, width, heigh, color=0/1, fill=0/1
display.rect(2, 2, 123, 59, 1, 0) #x, y, width, heigh, color=0/1, fill=0/1

```

```
#Δημιούργησε ετικέτες
display.rect(3, 2, 121, 14, 1, 1) #x, y, width, heigh, color=0/1, fill=0/1
display.text('Weather Station', 3, 5, 0) #Θέση x, y, color 1/0
display.text("Press: ", 5, 20, 1)
display.text("hpa", 100, 20, 1)
display.text("Temper1 : ", 5, 30, 1)
display.ellipse(113, 31, 1, 1, 1)
display.text("C", 115, 30, 1)
display.text("Temper2 : ", 5, 40, 1)
display.ellipse(106, 41, 1, 1, 1)
display.text("C", 108, 40, 1)
display.text("Humidity: ", 5, 50, 1)
display.text("%", 105, 50, 1)
display.show() #Παρουσίασε στην οθόνη

#Για πάντα
while True:
    bmp.force_measure() #Διάβασε τιμές από BMP280
    display.rect(52, 19, 48, 9, 0, True) #Σβήσε παλιά τιμή ατμ. πίεσης
    display.text(str(round((bmp.pressure / 100), 1)), 52, 20, 1) #Εμφάνισε νέα τιμή ατμ. πίεσης
    display.rect(77, 29, 34, 9, 0, True) #Σβήσε παλιά τιμή θερμοκρασίας BMP
    display.text(str(round(bmp.temperature, 1)), 78, 30, 1) #Εμφάνισε νέα τιμή θερμοκρασίας BMP
    print("Θερμοκρασία BMP: " + str(round(bmp.temperature, 1))) #Εμφάνισε σε τερματικό
    print("Πίεση: " + str(round((bmp.pressure / 100), 1))) #Εμφάνισε σε τερματικό

    sensor.measure() #Διάβασε τιμές από DHT11
    display.rect(80, 39, 25, 9, 0, True) #Σβήσε παλιά τιμή θερμοκρασίας DHT
    display.text("{: 3}".format(sensor.temperature()), 80, 40, 1) #Εμφάνισε νέα τιμή θερμοκρ. DHT
    display.rect(80, 49, 25, 9, 0, True) #Σβήσε παλιά τιμή σχετ. υγρασίας
    display.text("{: 3}".format(sensor.humidity()), 80, 50, 1) #Εμφάνισε νέα τιμή υγρασίας
    print("Θερμοκρασία DHT:", sensor.temperature()) #Εμφάνισε στο τερματικό
    print("Σχετ. Υγρασία:", sensor.humidity()) #Εμφάνισε στο τερματικό
    print() #Άσε μια γραμμή κενή

    display.show() #Παρουσίασε στην οθόνη
    time.sleep(2) #Περίμενε 2 sec
```





Άσκηση 3

Να γίνει πρόγραμμα σε Micropython το οποίο θα ανιχνεύει το άγγιγμα των κουμπιών IO12 και IO13. Όταν πατιέται τι IO12 θα ανάβει το RGB Led πάνω απ' αυτό σε λευκό χρώμα και όταν πατιέται το IO13 θα σβήνει. Κατά το πάτημα των κουμπιών θα ακούγεται και χαρακτηριστικός ήχος από τον βομβητή.
Για την υλοποίηση δεν χρειάζονται επιπλέον βιβλιοθήκες.

```
import machine, time, neopixel

#Ορισμοί
threshold = 200 # Threshold to be adjusted
Debounce = 50 #msec

#Στιγμιότυπο RGB Leds
rgb = neopixel.NeoPixel(machine.Pin(25), 5) #IO25, 5 x RGB Leds

#Ορισμός ακροδεκτών κουμπιών αφής
t_pin1 = machine.TouchPad(machine.Pin(12)) #1ο κουμπί αφής
t_pin2 = machine.TouchPad(machine.Pin(13)) #2ο κουμπί αφής
```

```

#Ορισμός ακροδέκτη Beeper
beeper = machine.Pin(5, machine.Pin.OUT) #Ο βομβητής συνδέεται στο pin IO5
#Συνάρτηση παραγωγής τόνου συχνότητας freq σε Hz και διάρκειας dur σε msecs
def beep(freq = 500, dur = 500): #Προκαθορισμένο 500Hz, 500msec
    #Υπολογισμοί
    period = 1.0 / freq #Περίοδος
    half_per = int((period / 2) * 1000000) #Ημιπερίοδος σε μsecs
    times = int(dur / 1000 / period) #Αριθμός κύκλων
    for i in range(times):
        beeper.value(1)
        time.sleep_us(half_per)
        beeper.value(0)
        time.sleep_us(half_per)

#----- Ορισμός κλάσης ελέγχου κουμπιών αφής (Μόνο απλό κλικ) ----------------------------------------
class Touch:
    #Κατασκευαστής
    def __init__(self, pin):
        self.pin = pin  #Το GPIO του κουμπιού αφής
        self.validclick = False #Έγκυρο πάτημα
        self.timer = 0  #Ο timer που κρατάει τα ticks σε msec
        self.downtime = 0 #Πόσο χρόνο είναι πατημένο (Δάχτυλο πάνω στο κουμπί αφής)

    #Συμβάν απλό κλικ
    def onClick(self):
        self()

    #Καλείται περιδικά και ελέγχει το κάθε κουμπί
    def checkButton(self):
        self.timer = time.ticks_ms() #Κράτησε τον χρόνο συστήματος
        cVal = self.pin.read() #Διάβασε τιμή του αισθητήρα touch
        state = False #Δεν πατήθηκε
        if cVal < threshold: #Αν είναι κάτω από το κατώφλι (Δάχτυλο πάνω στην νησίδα)
            state = True #Πατήθηκε το κουμπί
        #------ Πατημένο ------------------------------------------------------------------------------
        #Πατήθηκε τώρα για πρώτη φορά και ο χρόνος δεν μετράει
        if state and self.downtime == 0:
            self.downtime = self.timer #Άρχισε να μετράς τον χρόνο που είναι πατημένο - downtime
        #Παραμένει πατημένο και έλεγξε αν πέρασε ο χρόνος debounce και δεν έχει ενεργοποιηθεί valid click
        elif state and not self.validclick and (self.timer - self.downtime) > Debounce:
            self.validclick = True #Να μην ξαναμπείς εδώ
            self.onClick() #Κάλεσε συνάρτηση εξυπηρέτησης
        #------ Ελεύθερο ------------------------------------------------------------------------------
        #Αλλιώς αν ήταν πατημένο και τώρα το άφησε
        elif not state:
            self.downtime = 0 #Επαναφορά
            self.validclick = False #Επαναφορά για την επόμενη φορά
#----- Τέλος ορισμού κλάσης ------------------------------------------------------------------------

#----- Συναρτήσεις εξυπηρέτησης συμβάντων ----------------------------------------------------------
def key1_click(): #Άναψε
    print("Led is ON")
    #--------- R ,  G ,  B
    rgb[4] = (255, 255, 255) #Τιμές από 0 (σβηστό) έως 255 (μέγιστη ένταση)
    rgb.write()
    beep(2000, 50)

def key2_click(): #Σβήσε
    print("Led is OFF")
    rgb[4] = (0, 0, 0)
    rgb.write()
    beep(400, 50)

#===== Κυρίως πρόγραμμα ===================================================================================
#----- Δημιουργία στιγμιοτύπων και ορισμός συναρτήσεων εξυπηρέτησης ---------------------------------
ts1 = Touch(t_pin1)
ts1.onClick = key1_click

ts2 = Touch(t_pin2)
ts2.onClick = key2_click

print("\nESP32 Touch Buttons & RGB Leds Demo")

#----- Για πάντα -----------------------------------------------------------------------------------
while True:
    ts1.checkButton() #Έλεγχος 1ου κουμπιού
    ts2.checkButton() #Έλεγχος 2ου κουμπιού
    time.sleep_ms(5) #Περίμενε 5 - 10msec
```

Άσκηση 4

Να κατασκευαστεί κλάση για τα κουμπιά αφής της πλακέτας η οποία ανιχνεύει μονό κλικ, διπλό κλικ και παρατεταμένο κλικ. Για κάθε συμβάν καλείται ξεχωριστή συνάρτηση εξυπηρέτησης. Επίσης να γραφεί κώδικας ο οποίος δημιουργεί στιγμιότυπα για δύο κουμπιά και εξυπηρετεί και τις τρεις λειτουργίες για το καθένα.

```
import machine, time
#---- Ορισμοί ------------------------------------------------------------------
threshold = 200 # Threshold to be adjusted
Debounce = 50 #msec
DblClickDelay = 300 #msec
LongPressDelay = 1000 #msec
t_pin1 = machine.TouchPad(machine.Pin(12)) #1ο κουμπί αφής
t_pin2 = machine.TouchPad(machine.Pin(13)) #2ο κουμπί αφής

#----- Ορισμός κλάσης ----------------------------------------------------------
class Touch:
    #Κατασκευαστής
    def __init__(self, pin):
        self.pin = pin
        self.laststate = True
        self.timer = 0
        self.downtime = -1
```

```
        self.uptime = -1
        self.ignoreUP = False
        self.singleClickOK = False
        self.dblClickWaiting = False
        self.dblClickOnNextUp = False
        self.longPressHappened = False

    #Συμβάντα
    #Απλό κλικ
    def onClick(self):
        self()

    #Διπλό κλικ
    def onDblClick(self):
        self()

    #Παρατεταμένο κλικ
    def onLongClick(self):
        self()

    #Καλείται περιοδικά και ελέγχει το κάθε κουμπί
    def checkButton(self):
        resultEvent = 0
        self.timer = time.ticks_ms() #Κράτησε τον χρόνο συστήματος
        cVal = self.pin.read() #Διάβασε τιμή του αισθητήρα touch
        state = False #Δεν πατήθηκε
        if cVal < threshold: #Αν είναι κάτω από το κατώφλι
            state = True #Πατήθηκε το κουμπί
        #------ Πατημένο ---------------------------------------------------------------------
        #Πατήθηκε τώρα και πριν δεν είχε πατηθεί και ο uptime έχει ξεπεράσει τον χρόνο debounce.
        #Αρχικά θα μπει εδώ αμέσως όταν πατηθεί γιατί δεν έχει κρατηθεί ο uptime
        if state == True and self.laststate == False and (self.timer - self.uptime) > Debounce:
            self.downtime = self.timer #Άρχισε να μετράς τον χρόνο που είναι πατημένο - downtime
            self.ignoreUP = False
            self.singleClickOK = True
            self.longPressHappened = False
            #Αν δεν έχει περάσει ο χρόνος uptime τον χρόνο double click και δεν έχει ενεργοποιηθεί το double click στην επόμενη επαναφορά
            #και περιμένει για double click. Εδώ θα μπει κατά το 2ο πάτημα
            if (self.timer - self.uptime) < DblClickDelay and self.dblClickOnNextUp == False and self.dblClickWaiting == True:
                self.dblClickOnNextUp = True #Στο άφημα να το θεωρήσει διπλό κλικ
            else: #Εδώ θα μπει στο 1ο κλικ του διπλού κλικ
                self.dblClickOnNextUp = False
            self.dblClickWaiting = False #Επαναφορά για την επόμενη φορά
        #------ Ελεύθερο ---------------------------------------------------------------------
        #Αλλιώς αν ήταν πατημένο και τώρα το άφησε και πέρασε ο downtime έχει ξεπεράσει τον χρόνο debounce
        #Εδώ θα μπει αν κρατήθηκε πατημένο για χρόνο > debounce
        elif state == False and self.laststate == True and (self.timer - self.downtime) > Debounce:
            #Αν δεν θέλει να αγνοήσει το Up. Εδώ δεν μπαίνει κατά το άφημα του long click
            if self.ignoreUP == False:
                self.uptime = self.timer #Άρχισε να μετράς τον χρόνο που είναι πάνω - uptime χρήσιμος για double click
                if self.dblClickOnNextUp == False: #Έχει γίνει το 1ο κλικ και τώρα το άφησε
                    self.dblClickWaiting = True #Θα περιμένει για δεύτερο πάτημα
                #Εδώ έχει συμβεί διπλό κλικ. Είναι πάνω μετά από διπλό κλικ
                else:
                    resultEvent = 2 #Αποτέλεσμα διπλό κλικ
                    self.dblClickOnNextUp = False #Επαναφορά
                    self.dblClickWaiting = False
                    self.singleClickOK = False

        #Έλεγχος για μονό κλικ. Ο χρόνος του διπλού κλικ έχει λήξει
        if state == False and (self.timer - self.uptime) >= DblClickDelay and self.dblClickWaiting == True and self.dblClickOnNextUp == False and \
            self.singleClickOK == True and resultEvent != 2:
            resultEvent = 1
            self.dblClickWaiting = False

        #Έλεγχος για παρατεταμένο κλικ
        if state == True and (self.timer - self.downtime) >= LongPressDelay:
            #Πυροδότησε το παρατεταμένο πάτημα
            #Αν δεν πατήθηκε πριν παρατεταμένα
            if self.longPressHappened == False:
                resultEvent = 3
                self.ignoreUP = True #Αγνόησε το άφημα
                self.dblClickOnNextUp = False
                self.dblClickWaiting = False
                self.longPressHappened = True

        self.laststate = state #Κράτα την προηγούμενη κατάσταση
        #Κλήσεις ανάλογα με την ενέργεια
        if resultEvent == 1:
            self.onClick()
        if resultEvent == 2:
            self.onDblClick()
        if resultEvent == 3:
            self.onLongClick()
#----- Τέλος ορισμού κλάσης ------------------------------------------------------------

#----- Συναρτήσεις εξυπηρέτησης συμβάντων ----------------------------------------------
def key1_click():
    print("key1 Click")

def key2_click():
    print("key2 Click")

def key1_dblclick():
    print("key1 Double Click")

def key2_dblclick():
    print("key2 Double Click")

def key1_longclick():
    print("key1 Long Click")

def key2_longclick():
    print("key2 Long Click")
#----- Τέλος συναρτήσεων εξυπηρέτησης συμβάντων ----------------------------------------

#===== Κυρίως πρόγραμμα ============================================================
#----- Δημιουργία στιγμιοτύπων και ορισμός συναρτήσεων εξυπηρέτησης -------------------
ts1 = Touch(t_pin1)
ts1.onClick = key1_click
ts1.onDblClick = key1_dblclick
ts1.onLongClick = key1_longclick

ts2 = Touch(t_pin2)
ts2.onClick = key2_click
ts2.onDblClick = key2_dblclick
ts2.onLongClick = key2_longclick

print("\nESP32 Touch Demo")

#----- Για πάντα -----------------------------------------------------------------------
while True:
    ts1.checkButton() #Έλεγχος 1ου κουμπιού
    ts2.checkButton() #Έλεγχος 2ου κουμπιού
    time.sleep_ms(10) #Περίμενε 5 - 10msec
```

Άσκηση 5

Να γραφεί πρόγραμμα σε micropython το οποίο χρησιμοποιεί την ενσωματωμένη βιβλιοθήκη asyncio και αναβοσβήνει τα τρία από τα πέντε RGB Led, με σταθερά χρώματα το καθένα (κόκκινο, πράσινο, μπλε), με διαφορετική συχνότητα αναλαμπών. Συγκεκριμένα το πράσινο αναβοσβήνει με περίοδο 4 sec, το κόκκινο με περίοδο 1 sec και το μπλε με περίοδο 0,2 sec. Η asyncio επιτρέπει την ασύγχρονη εκτέλεση των προγραμμάτων χωρίς την χρήση εντολών blocking όπως η time.sleep().

```
import machine, neopixel, asyncio

# Προετοιμασία RGB Led
np = neopixel.NeoPixel(machine.Pin(25), 5) #IO25, 5 x RGB Leds
led_status = [0, 0, 0, 0, 0] #Λίστα με την κατάσταση των Led (0=σβηστό, 1=αναμμένο)
RED = (255, 0, 0) #Κόκκινο
GREEN = (0, 255, 0) #Πράσινο
BLUE = (0, 0, 255) #Μπλε

# Η συνάρτηση αναβοσβήνει το Led με αριθμό num (0-4) με το χρώμα color
def toggle_Led(num, color):
    if led_status[num] == 0: #Είναι σβηστό
        np[num] = color #Γράψε χρώμα
        led_status[num] = 1 #Σημείωσε ότι τώρα είναι αναμμένο
    else: #Διαφορετικά είναι αναμμένο
        np[num] = (0, 0, 0) #Σβήσε (0,0,0) = σβηστό (μαύρο)
        led_status[num] = 0 #Σημείωσε ότι τώρα είναι σβηστό
    np.write() #Στείλε εντολή στα LED

# Ασύγχρονη λειτουργία (coroutine) για Πράσινο
async def blink_green_led():
    while True: #Για πάντα
        toggle_Led(4, GREEN) #Αναβόσβησε το 1ο στην σειρά με χρώμα πράσινο
        await asyncio.sleep(2) #Περίμενε 2 sec και δώσε την δυνατότητα εκτέλεσης άλλων λειτουργιών

# Ασύγχρονη λειτουργία (coroutine) για Κόκκινο
async def blink_red_led():
    while True:
        toggle_Led(0, RED) #Αναβόσβησε το 5ο στην σειρά με χρώμα κόκκινο
        await asyncio.sleep(0.5) #Περίμενε .5 sec και δώσε την δυνατότητα εκτέλεσης άλλων λειτουργιών

# Ασύγχρονη λειτουργία (coroutine) για Μπλε
async def blink_blue_led():
    while True: #Για πάντα
        toggle_Led(2, BLUE) #Αναβόσβησε το μεσαίο με χρώμα μπλε
        await asyncio.sleep(0.1) #Περίμενε 100 msec και δώσε την δυνατότητα εκτέλεσης άλλων λειτουργιών

# Ορισμός της συνάρτησης main που περιέχει το event loop
async def main():
    # Δημιουργία εργασιών (tasks) ώστε να αναβοσβήνουν τα 3 Led ταυτόχρονα
    asyncio.create_task(blink_green_led())
    asyncio.create_task(blink_red_led())
    asyncio.create_task(blink_blue_led())

# Δημιουργία και εκτέλεση του event loop
loop = asyncio.get_event_loop()
loop.create_task(main())  # Δημιουργία εργασίας (task) που εκτελεί το event loop
loop.run_forever()  # Τρέξε για πάντα
```

Άσκηση 6

Να γραφεί πρόγραμμα σε micropython το οποίο διαβάζει την θέση του ποτενσιομέτρου του αρθρώματος επέκτασης AJS06.

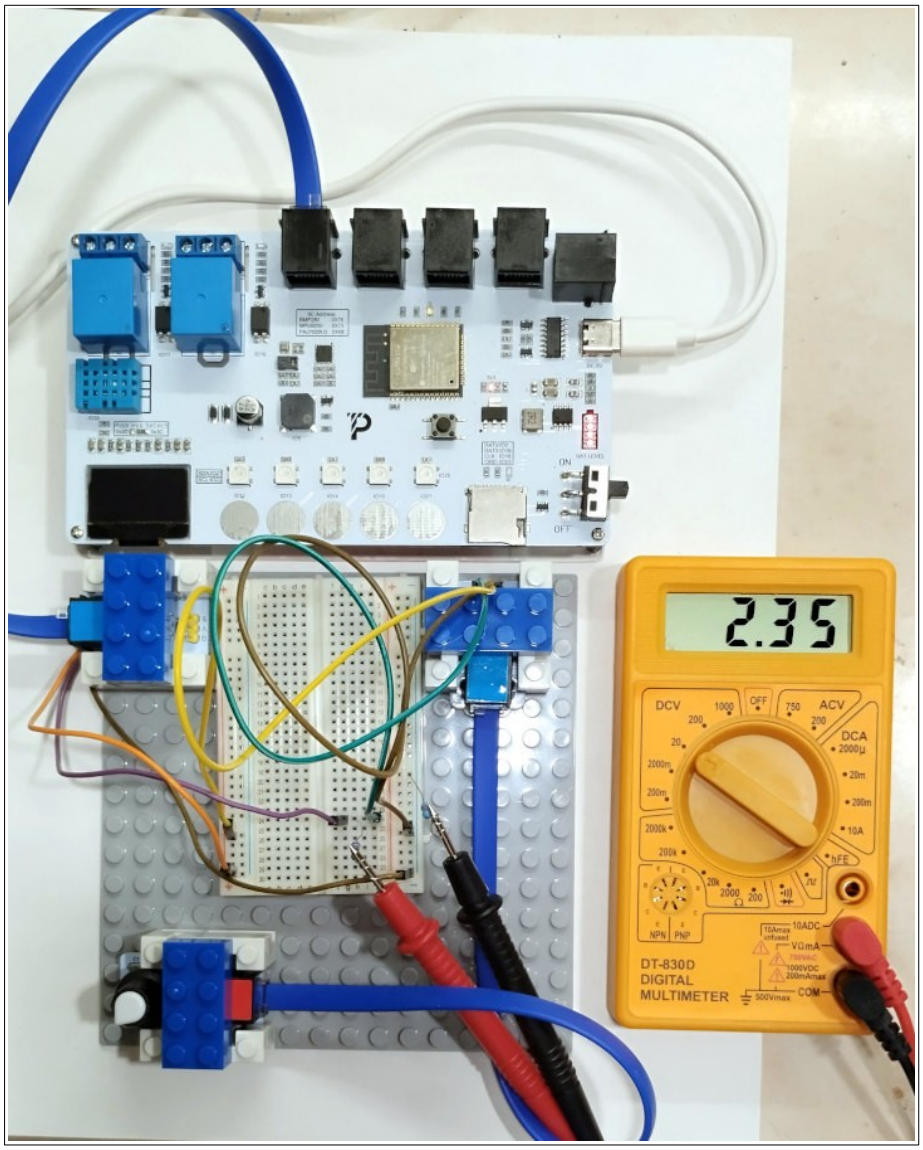
Συνδέουμε το ποτενσιόμετρο AJS06 σε οποιαδήποτε από τις τέσσερις πάνω θύρες της πλακέτας (IO4, IO26, IO32, IO34) με το τροποποιημένο καλώδιο ή δύο EXP-AJ11 όπου το ένα έχει τροποποιηθεί και την breadboard. Ο μικροελεγκτής ESP32 διαθέτει μετατροπέα από αναλογικό σε ψηφιακό (Analog to Digital Converter **ADC**) εύρους 12bit δηλαδή το εύρος τιμών είναι από 0-4095. Μπορούμε να μειώσουμε την ευκρίνεια σε 11, 10 ή 9 bit. Επίσης διαθέτει στην είσοδο μεταβλητό εξασθενητή με τον οποίο αλλάζουμε το εύρος τάσεων που μπορεί να μετατρέψει. Δηλαδή αν επιλέξουμε την σταθερά ADC.ATTN_6DB τότε για τάση εισόδου 2 V ο ADC θα επιστρέψει τιμή 4095 στα 12bit.

ADC.ATTN_0DB - εύρος τιμών έως 1.2V
ADC.ATTN_2_5DB - εύρος τιμών έως 1.5V
ADC.ATTN_6DB - εύρος τιμών έως 2.0V
ADC.ATTN_11DB - εύρος τιμών έως 3.3V

Επειδή το ένα άκρο του ποτενσιομέτρου συνδέεται στα 3,3V και το άλλο στην γείωση (GND), επιλέγουμε την σταθερά ADC.ATTN_11DB. Ο δρομέας οδηγείται στην είσοδο του ADC.

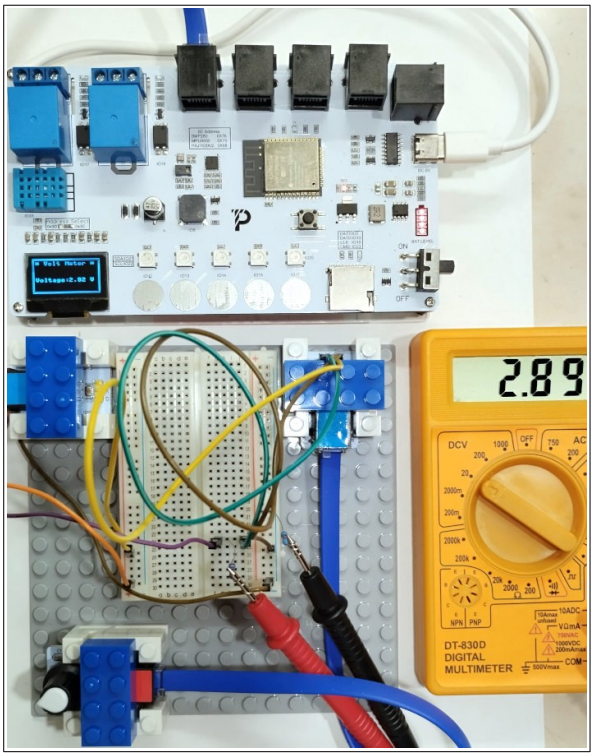
```
from machine import Pin, ADC
from time import sleep

pot = ADC(Pin(4)) #4, 26, 32, 34
pot.width(ADC.WIDTH_12BIT) #Ευκρίνεια 12BIT προκαθορισμένο, 9BIT, 10BIT, 11BIT
pot.atten(ADC.ATTN_11DB) # πλήρης κλίμακα έως 3.3V

while True:
  pot_value = pot.read()
  voltage = pot_value * (3.3 / 4096) #12bit
  print("τιμή:", pot_value, "τάση:", round(voltage, 2), "V")
  sleep(0.5)
```

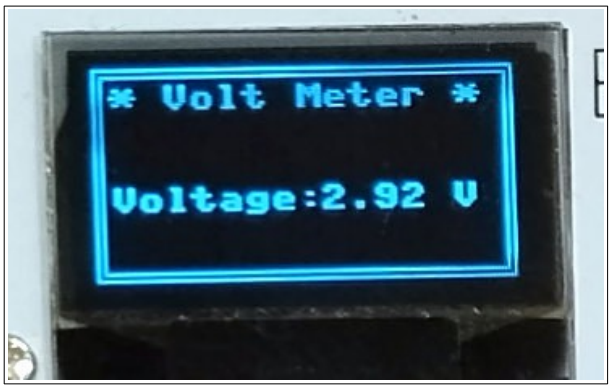
Στην breadboard μπορούμε να συνδέσουμε και το πολύμετρο σε λειτουργία βολτομέτρου (μαύρος ακροδέκτης στο '-' και κόκκινος στον δρομέα ο οποίος συνδέεται με την είσοδο ADC. Για ευκολία μπορούμε να τυλίξουμε στον κάθε ακροδέκτη έναν αντιστάτη 220Ω και το άλλο άκρο να καρφωθεί στην breadboard. Εκτελούμε τον κώδικα και περιστρέφοντας αργά το ποτενσιόμετρο παρατηρούμε αν οι τιμές στο βολτόμετρο συμφωνούν με αυτές στο τερματικό του vscode. Φυσιολογικά πρέπει να υπάρχουν αποκλίσεις της τάξεως των 10 – 20mV οι οποίες οφείλονται σε σφάλμα του πολυμέτρου αλλά κυρίως στη μη γραμμική συμπεριφορά του ADC. Αυτό μπορεί να διορθωθεί με πίνακες διόρθωσης στον κώδικα και βαθμονόμιση (καλιμπράρισμα) με κάποιο πιο αξιόπιστο βολτόμετρο.



Άσκηση 7

Να τροποποιηθεί το παραπάνω ώστε να εμφανίζεται στην ενσωματωμένη μονόχρωμη οθόνη OLED η τίμη της τάσης σε Volts.



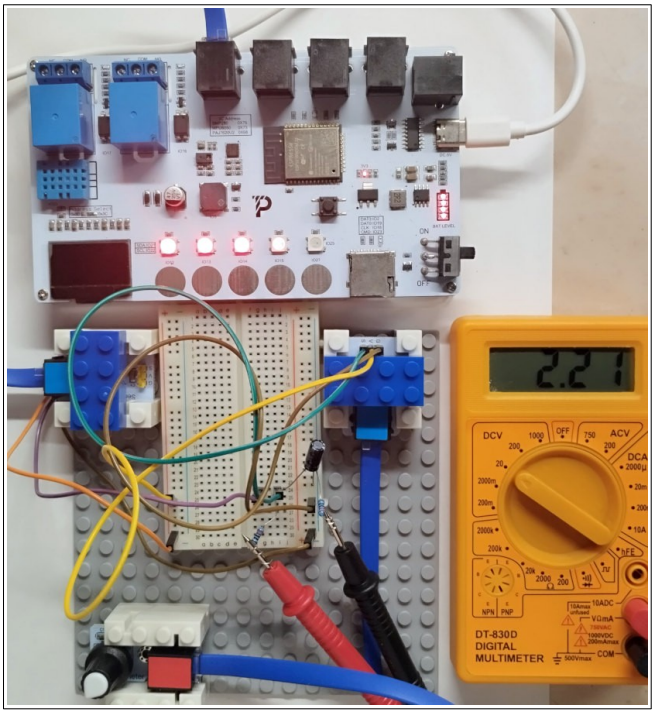


Στον κώδικα παρατηρούμε ότι έχει προστεθεί η μεταβλητή **avg_voltage** στην γραμμή 7. Αυτή είναι η μέση τιμή των μετρήσεων του ADC. Με την χρήση μέσης τιμής αποφεύγουμε την συνεχόμενη μεταβολή σε κάθε μέτρηση. Δηλαδή έχουμε μια εξομάλυνση στις ενδείξεις σαν να είχαμε συνδέσει παράλληλα στην είσοδο του ADC έναν πυκνωτή. Επίσης προσθέσαμε μια σταθερά **OFFS_COEF** που σαν σκοπό έχει να βελτιώσει την απόκλιση των μετρήσεων σε σχέση με το φυσικό ψηφιακό βολτόμετρο. Η βελτίωση είναι μερική γιατί με τον συντελεστή (συνάρτηση 1ου βαθμού) δεν διορθώνονται μη γραμμικές συμπεριφορές στα άκρα. Στις παραπάνω εικόνες βλέπουμε την τιμή που εμφανίζει η οθόνη μας σε σχέση με αυτή του βολτομέτρου.

```
1  from machine import Pin, ADC, I2C
2  from time import sleep
3  from ssd1306 import SSD1306_I2C
4
5  AVG_FACTOR = 2.0 #Συντελεστής μέσης τιμής όσο μεγαλύτερος τόσο πιο αργή μεταβολή
6  OFFS_COEF = .95 #Συντελεστής διόρθωσης
7  avg_voltage = 0
8
9  pot = ADC(Pin(4)) #4, 26, 32, 34
10 pot.width(ADC.WIDTH_12BIT) #Ευκρίνεια 12BIT προκαθορισμένο, 9BIT, 10BIT, 11BIT
11 pot.atten(ADC.ATTN_11DB) # πλήρης κλίμακα έως 3.3V
12
13 i2c = I2C(0, sda = Pin(21), scl = Pin(22), freq = 400000)
14 display = SSD1306_I2C( width=128, height=64, i2c=i2c, addr=0x3c, external_vcc=False )
15 display.fill(0) #Καθαρίζει οθόνη
16 #Δημιούργησε ετικέτες
17 display.text('* Volt Meter *', 5, 5, 1) #Θέση x, y, color 1/0
18 display.text("Voltage: ", 5, 35, 1)
19 display.text("V", 108, 35, 1)
20 #Δημιούργησε διπλό περίγραμμα
21 display.rect(0,0,127,63,1,0) #x, y, width, heigh, color=0/1, fill=0/1
22 display.rect(2,2,123,59,1,0) #x, y, width, heigh, color=0/1, fill=0/1
23 display.show() #Παρουσίασε στην οθόνη
24
25 while True:
26   pot_value = pot.read()
27   voltage = pot_value * (3.3 / 4096) / OFFS_COEF #12bit Στιγμιαία τιμή
28   avg_voltage = (AVG_FACTOR * avg_voltage + voltage) / (AVG_FACTOR + 1) #Μέση τιμή
29   print("τιμή:", pot_value, "τάση:", round(voltage, 2), "V")
30   display.rect(68, 34, 40, 9, 0, True) #Σβήσε παλιά τιμή
31   display.text(str(round(avg_voltage, 2)), 68, 35, 1) #Εμφάνισε νέα τιμή
32   display.show() #Παρουσίασε στην οθόνη
33   sleep(0.5)
```

Άσκηση 8

Να τροποποιηθεί η άσκηση 6 ώστε με την περιστροφή του ποτενσιομέτρου να ανάβουν διαδοχικά τα 5 RGB Leds της πλακέτας σε κόκκινο χρώμα. Το τελευταίο αναμμένο LED θα ανάβει αναλογικά δηλαδή αν έχουν ανάψει πλήρως δύο LED και η τιμή του ADC δεν επαρκεί ώστε να ανάψει πλήρως και τρίτο LED τότε αυτό θα ανάψει με περιορισμένη ένταση π.χ. 30%.



Παρατηρήστε ότι το τέταρτο LED ανάβει με χαμηλότερη ένταση από τα τρία αριστερά του. Η ευκρίνεια του ADC έχει μειωθεί στα 10 bits.

Για να μειωθεί το ‘παίξιμο’ των τιμών κατά την δειγματοληψία του ADC βάλαμε έναν ηλεκτρολυτικό πυκνωτή 100μF από τον δρομέα στην γείωση (GND). Ο δρομέας συνδέεται στην είσοδο του ADC (IO4). Το μακρύ ποδαράκι του πυκνωτή είναι το (+) και συνδέεται στον δρομέα, ενώ το κοντό είναι το (-) και συνδέεται στην γείωση (- ή GND).

```
from machine import Pin, ADC
from time import sleep
from neopixel import NeoPixel

RED = (255, 0, 0); GREEN = (0, 255, 0); BLUE = (0, 0, 255); BLANK = (0, 0, 0)
AVG_FACTOR = 3 #Συντελεστής μέσης τιμής όσο μεγαλύτερος τόσο πιο αργή μεταβολή
avg_val = 0

pot = ADC(Pin(4)) #4, 26, 32, 34
pot.width(ADC.WIDTH_10BIT) #Ευκρίνεια 10BIT, 9BIT, 10BIT, 11BIT, 12BIT προκαθορισμένο
pot.atten(ADC.ATTN_11DB) # πλήρης κλίμακα έως 3.3V

np = NeoPixel(Pin(25), 5) #IO25, 5 x RGB Leds

#Συνάρτηση παρόμοια με την map του Arduino
#Δέχεται τρέχουσα τιμή, ελάχιστη τιμή, μέγιστη τιμή, έξοδος από, έως
def map(value, in_min, in_max, out_min, out_max):
  scaled_value = (value - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
  return int(scaled_value) #Επιστρέψει ακέραιο

#Συνάρτηση για να ανάβει τα Leds ανάλογα με την τιμή του ADC
def out_leds(x):
  n = x // 205 #Γιατί 1024 / 5 leds = 204,8 ακέραια διαίρεση
  y = x % 205 #Υπόποιπο για το τελευταίο Led
  i = 0
  while i < n: #Ανάβει πλήρως όσα Led υπολογίστηκαν στο n
    np[4 - i] = RED #Ανάβουν ανάποδα
    i += 1
  a = i #Κράτησε την τρέχουσα τιμή του i
  while i < 5: #Αν δεν είναι να ανάψουν όλα τότε σβήσε τα υπόλοιπα
    np[4 - i] = BLANK #Σβήνουν ανάποδα
    i += 1
  print(n, y, map(y, 0, 204, 2, 255))
  i = a #Επαναφορά του i
  if i <= 4: #Αν είναι από 0 έως 4 τότε άναψε μερικώς αυτό το Led
    np[4 - i] = (map(y, 0, 204, 2, 250), 0, 0) #Αλλαγή κλίμακας από 0-204 σε 2-255 για το κόκκινο (255 πλήρης ένταση)
  np.write()

#Για πάντα
while True:
  pot_value = pot.read() #Διάβασε τιμή ποτενσιομέτρου
  avg_val = int((AVG_FACTOR * avg_val + pot_value) / (AVG_FACTOR + 1)) #Μέση τιμή
  out_leds(avg_val) #Άναψε τα Leds
  print("τιμή:", avg_val)
  sleep(.1)
```

Άσκηση 9

Να τροποποιηθεί ο παραπάνω κώδικας ώστε να χρησιμοποιηθεί το ποτενσιόμετρο ως συσκευή εισόδου εντολών. Συγκεκριμένα ας υποθέσουμε ότι θέλουμε να επιλέγουμε πέντε προγράμματα πλυντηρίου περιστρέφοντας το ποτενσιόμετρο και ανάλογα την θέση να εμφανίζεται το πρόγραμμα στο τερματικό π.χ. 'Θέση-2' και να ανάβει ένα από τα 5 RGB Leds με διαφορετικό χρώμα. Για τον συγκεκριμένο σκοπό υπάρχουν άλλα εξαρτήματα, όπως περιστροφικοί διακόπτες 16 θέσεων που βγάζουν 4bit ανάλογα την θέση ή R.I.E. (Rotary Impulse Encoders) που καταλαβαίνουν πόσα βήματα έχουν περιστραφεί και προς ποια κατεύθυνση. Επειδή το kit δεν διαθέτει κάτι αντίστοιχο εμείς θα χρησιμοποιήσουμε το ποτενσιόμετρο.

```
from machine import Pin, ADC
from time import sleep
from neopixel import NeoPixel

#Σταθερές χρωμάτων
RED = (255, 0, 0); GREEN = (0, 255, 0); BLUE = (0, 0, 255); YELLOW = (255, 255, 0)
MAGENTA = (255, 0, 255); CYAN = (0, 255, 255); BLANK = (0, 0, 0)
AVG_FACTOR = 1.5 #Συντελεστής μέσης τιμής όσο μεγαλύτερος τόσο πιο αργή μεταβολή
avg_val = 0 #Μέση τιμή του ADC

pot = ADC(Pin(4)) #4, 26, 32, 34
pot.width(ADC.WIDTH_10BIT) #Ευκρίνεια 10BIT, 9BIT, 10BIT, 11BIT, 12BIT προκαθορισμένο
pot.atten(ADC.ATTN_11DB) # πλήρης κλίμακα έως 3.3V

np = NeoPixel(Pin(25), 5) #IO25, 5 x RGB Leds

#Λίστα με τις εντολές
C = ['Θέση - 1', 'Θέση - 2', 'Θέση - 3', 'Θέση - 4', 'Θέση - 5']
#Παράλληλη λίστα με τα χρώματα για κάθε θέση
COLORS = [RED, GREEN, BLUE, YELLOW, MAGENTA]
PLACES = len(C) #Υπολόγισε αριθμό θέσεων
THRESS = 5 #Κατώφλι υστέρησης
pos = 0 #Θέση στην λίστα με τις εντολές

#Η συνάρτηση σβήνει και τα 5 RGB Leds
def blank_leds():
  for i in range(5):
    np[i] = BLANK

#Συνάρτηση για να ανάβει τα Leds ανάλογα με την τιμή του ADC
def out_leds(n):
  blank_leds()
  np[4 - n] = COLORS[n]
  np.write()

#Η συνάρτηση επιλέγει εντολή ανάλογα με την τιμή του x δηλαδή την θέση του ποτενσιομέτρου
#Υπάρχει πρόβλεψη υστέρησης τιμής THRESS ώστε να μην παίζει η απόφαση της εντολής σε γειτονικές
#τιμές. π.χ. αν είναι 127 και 128 τότε θα μεταβάλλεται από Θέση-1 σε Θέση-2. Με υστέρηση π.χ. 5
#για να αλλάξει σε Θέση-1 πρέπει να πέσει κάτω από 123 και για να ανέβει σε Θέση-2 πρέπει να
#ξεπεράσει το 133.
def get_command(x):
  global pos
  stp = 1024 // PLACES #Βήμα για κάθε θέση
  s = 0 #Άθροισμα βημάτων
  i = 0
  for i in range(8):
    if x > s + THRESS and x < s + stp - THRESS: #Αν ξέφυγε από την περιοχή υστέρησης τότε
      pos = i #Άλλαξε την τιμή του pos, διαφορετικά ισχύει η προηγούμενη τιμή
    s += stp
  return pos

#Για πάντα
while True:
  pot_value = pot.read() #Διάβασε τιμή ποτενσιομέτρου
  avg_val = int((AVG_FACTOR * avg_val + pot_value) / (AVG_FACTOR + 1)) #Μέση τιμή
  tmp = get_command(avg_val)
  print(C[tmp]) #Εμφάνισε εντολή
  out_leds(tmp)
  #print("τιμή:", avg_val) #Debug
  sleep(.1)
```