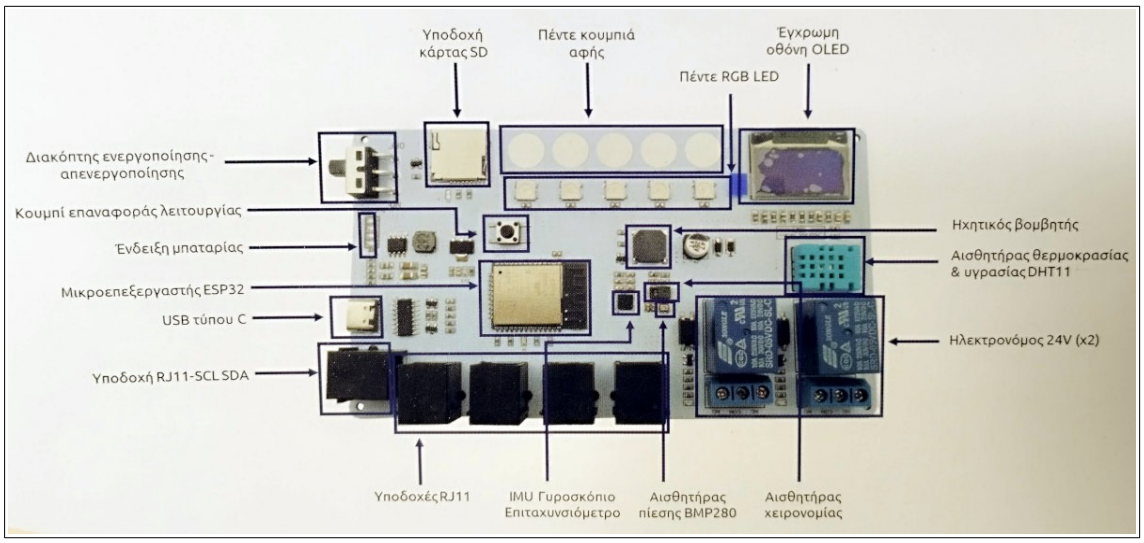
Προγραμματισμός Micropython

στο σύστημα ARD:Icon II της Polytech



Έκδοση 1.0

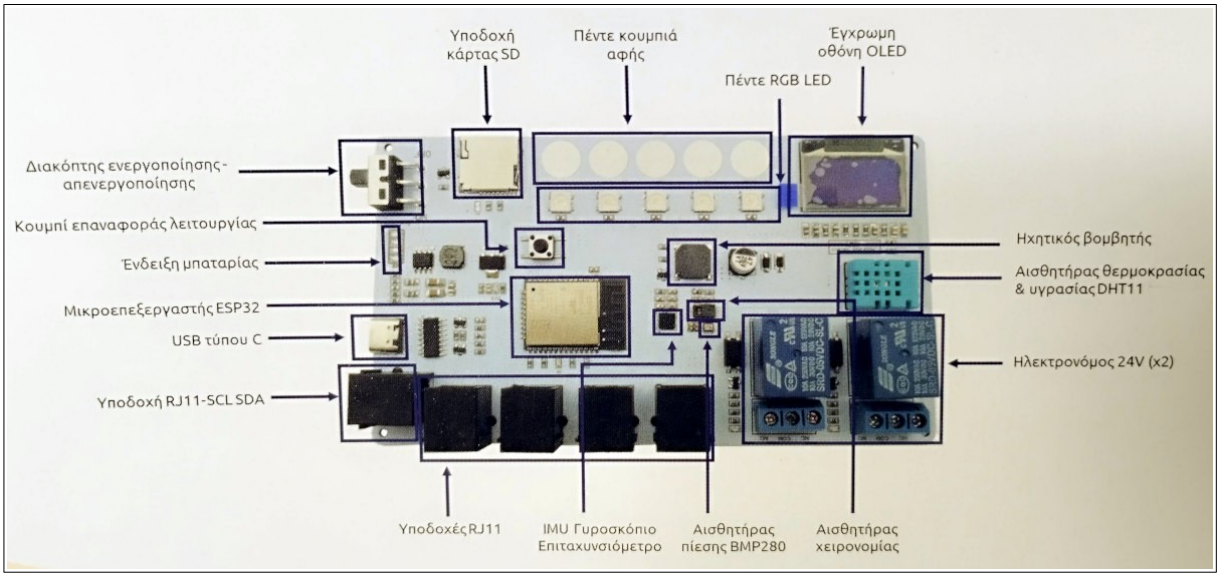
Σταύρος Σ. Φώτογλου ΠΕ86

Ε.Κ. Πρέβεζας – 1ο ΕΠΑ.Λ. Πρέβεζας

Πρέβεζα 2025

MicroPython στο ARD:Icon II

Η πλακέτα **ARD:Icon II** υπάρχει στο κιτ ρομποτικής **S4T**. Έχει πολλές δυνατότητες και εγκατεστημένους πολλούς αισθητήρες. Με μια μικρή τροποποίηση των μπλε καλωδίων με τους συνδετήρες RJ12 (παραλλαγή με το latch στην άκρη) μπορούμε να χρησιμοποιήσουμε σχεδόν το σύνολο των αισθητήρων του κιτ. Στην πλακέτα μπορούμε να εγκαταστήσουμε την Micropython για πιο εύκολο προγραμματισμό ειδικά από τους μαθητές των ΕΠΑ.Λ. οι οποίοι διδάσκονται την Python σύμφωνα με το αναλυτικό πρόγραμμα.



Τρόπος εγκατάστασης της Micropython

α. Ubuntu Linux 24.04 LTS

Αρχικά φτιάχνουμε ένα εικονικό περιβάλλον της Python 3.12.x από το τερματικό με **`python3 -m venv ~/micropython`**. Χρησιμοποιούμε το περιβάλλον με source micropython/bin/activate. Το prompt τώρα έγινε: **`(micropython) stavros@stavLinux:~$`**. Ελέγχω το περιβάλλον με **`which python`** και απαντά:

`/home/stavros/micropython/bin/python`.

Εγκατάσταση του esptool με **`pip3 install esptool`**.

Εντοπισμός σειριακής σύνδεσης με την εντολή **`ls -l /dev/serial/by-path`**. Εμφανίζει συνήθως **`ttyUSB0`** ή **`ttyUSB1`**.

Έλεγχος επικοινωνίας

Δίνω την εντολή **`python -m esptool --port /dev/ttyUSB0 flash-id`** και απαντάει:

```
esptool v5.0.1
Connected to ESP32 on /dev/ttyUSB0:
Chip type:          ESP32-D0WDQ6 (revision v1.1)
Features:           Wi-Fi, BT, Dual Core + LP Core, 240MHz, Vref calibration in eFuse, Coding Scheme None
Crystal frequency:  40MHz
MAC:                ec:64:c9:a4:1b:10

Stub flasher running.

Flash Memory Information:
=========================
Manufacturer: 5e
Device: 4016
Detected flash size: 4MB
Flash voltage set by a strapping pin: 3.3V

Hard resetting via RTS pin…
```

Σβήσιμο της flash

Για να σβήσουμε ότι υπάρχει στην μνήμη flash γράφουμε **`python -m esptool --port /dev/ttyUSB0 erase-flash`** και επιστρέφει:

```
esptool v5.0.1
Connected to ESP32 on /dev/ttyUSB0:
Chip type:          ESP32-D0WDQ6 (revision v1.1)
Features:           Wi-Fi, BT, Dual Core + LP Core, 240MHz, Vref calibration in eFuse, Coding Scheme None
Crystal frequency:  40MHz
MAC:                ec:64:c9:a4:1b:10

Stub flasher running.

Flash memory erased successfully in 12.4 seconds.

Hard resetting via RTS pin...
```

Κατεβάζουμε την τελευταία έκδοση της Micropython από εδώ: https://micropython.org/download/ESP32_GENERIC/ . Την στιγμή που γράφεται ο οδηγός η τρέχουσα έκδοση είναι η 1.25.0. Σε μερικά παραδείγματα ίσως να φαίνεται η έκδοση 1.24.1 αλλά δεν υπάρχει κανένα πρόβλημα όποια έκδοση και να εγκαταστήσετε.

Εγκατάσταση

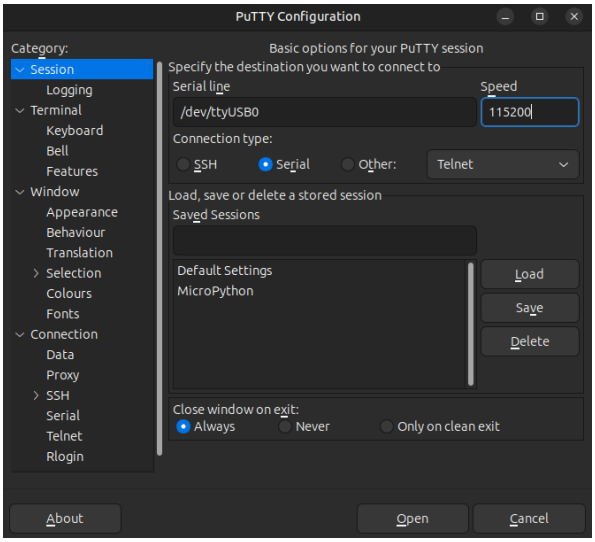
Για να γράψουμε την micropython στην μνήμη Flash του ESP32 γράφουμε:
`python3 -m esptool --port /dev/ttyUSB0 --baud 460800 write-flash -z 0x1000 ~/Λήψεις/ESP32_GENERIC-20250415-v1.25.0.bin` και επιστρέφει:

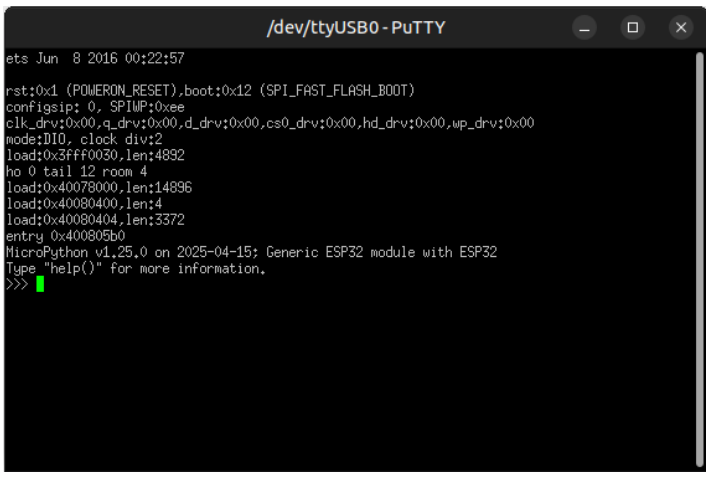
```
esptool v5.0.1
Connected to ESP32 on /dev/ttyUSB0:
Chip type:          ESP32-D0WDQ6 (revision v1.1)
Features:           Wi-Fi, BT, Dual Core + LP Core, 240MHz, Vref calibration in eFuse, Coding Scheme None
Crystal frequency:  40MHz
MAC:                ec:64:c9:a4:1b:10

Stub flasher running.
Changing baud rate to 460800...
Changed.

Configuring flash size...
Flash will be erased from 0x00001000 to 0x001a0fff...
Wrote 1702240 bytes (1117021 compressed) at 0x00001000 in 28.6 seconds (475.9 kbit/s).
Hash of data verified.

Hard resetting via RTS pin...
```

Πατάμε **`exit`** για να βγούμε από το venv και μετά κλείνουμε το τερματικό.

Δοκιμή

Ανοίγουμε το putty και επιλέγουμε Serial μετά στο Serial line βάζουμε /dev/ttyUSB0 και στο speed βάζουμε 115200. Πατάμε κάτω δεξιά το κουμπί Open.

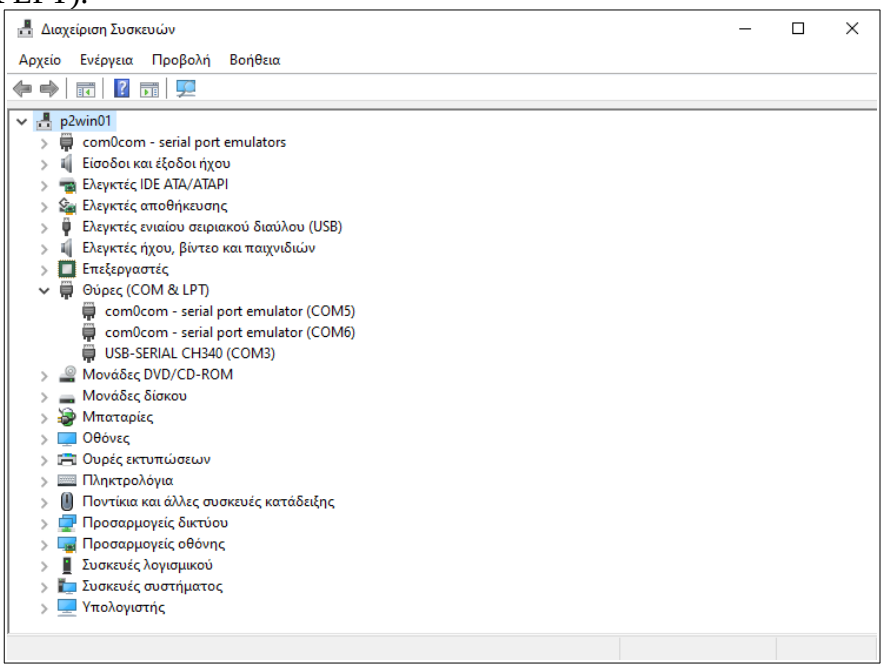




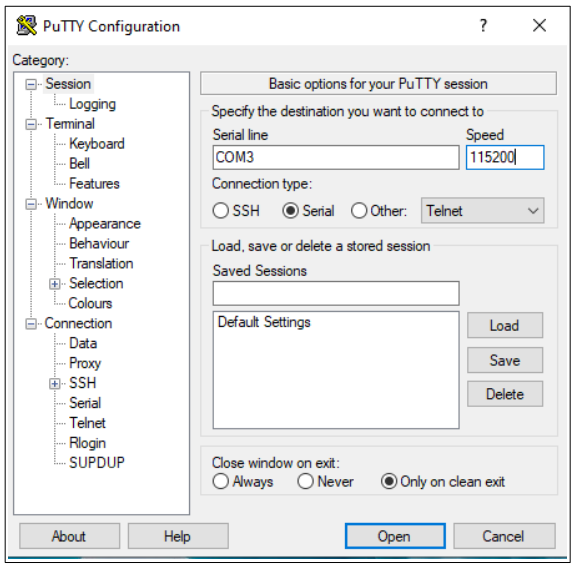
Πατάμε το button reset στην πλακέτα και αν εμφανίσει ότι στην εικόνα πάνω δεξιά σημαίνει ότι η Micropython εγκαταστάθηκε επιτυχώς.

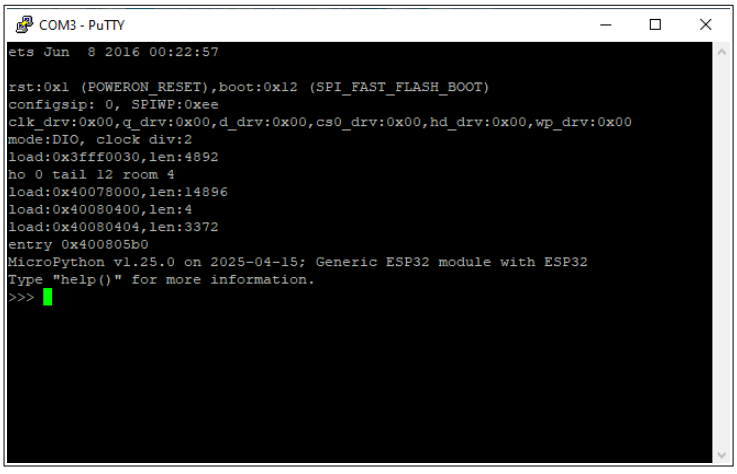
β. Windows

Στην αναζήτηση γράφουμε cmd και εκτελούμε την γραμμή εντολών.
Εκεί γράφουμε pip install esptool.
Από την διαχείριση συσκευών εντοπίζουμε την σειριακή θύρα της πλακέτας ανοίγοντας τον κλάδο Θύρες (COM & LPT).



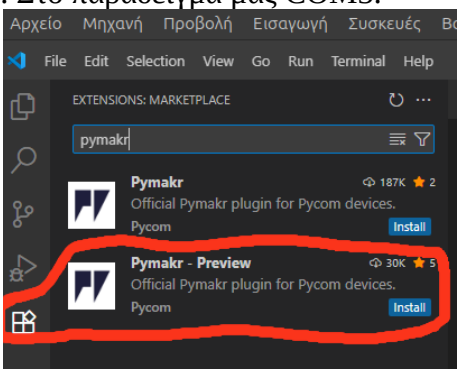
Στην περίπτωσή μας είναι η COM3.
Από εδώ και μετά ακολουθούμε την ίδια διαδικασία με το linux, απλώς μόνο αλλάζουμε το όνομα της θύρας σε com3. Για παράδειγμα για να δούμε πληροφορίες του ESP32 module γράφουμε:
`python -m esptool --port com3 flash-id`
Αφού ακολουθήσουμε την παραπάνω διαδικασία ανοίγουμε το putty και ελέγχουμε αν όλα πήγαν καλά.

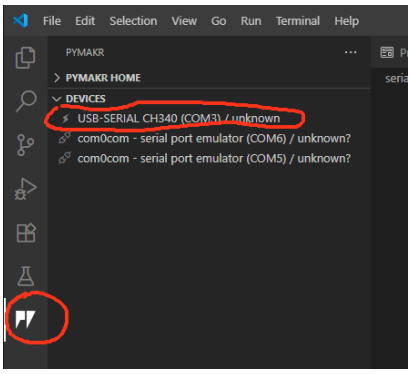




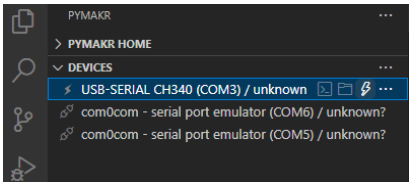
Χρήση του IDE VS Code με την Micropython

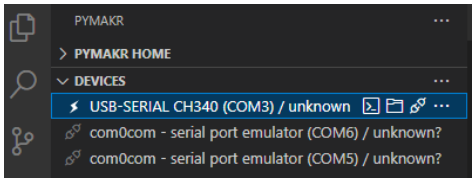
Στο vscode πάμε στα extensions και κάνουμε αναζήτηση για το pymakr το οποίο εγκαθιστούμε. Προσοχή στα windows λειτουργεί η έκδοση Pymakr – Preview 2.25.2 με ημερομηνία τελευταίας έκδοσης 7-9-22. Μόλις τελειώσει η εγκατάσταση εμφανίζει αριστερά ένα νέο εργαλείο σαν διπλά εισαγωγικά. Πατάμε το εργαλείο και μας εμφανίζει τις σειριακές θύρες που είναι συνδεδεμένο το ESP32. Στο παράδειγμά μας COM3.





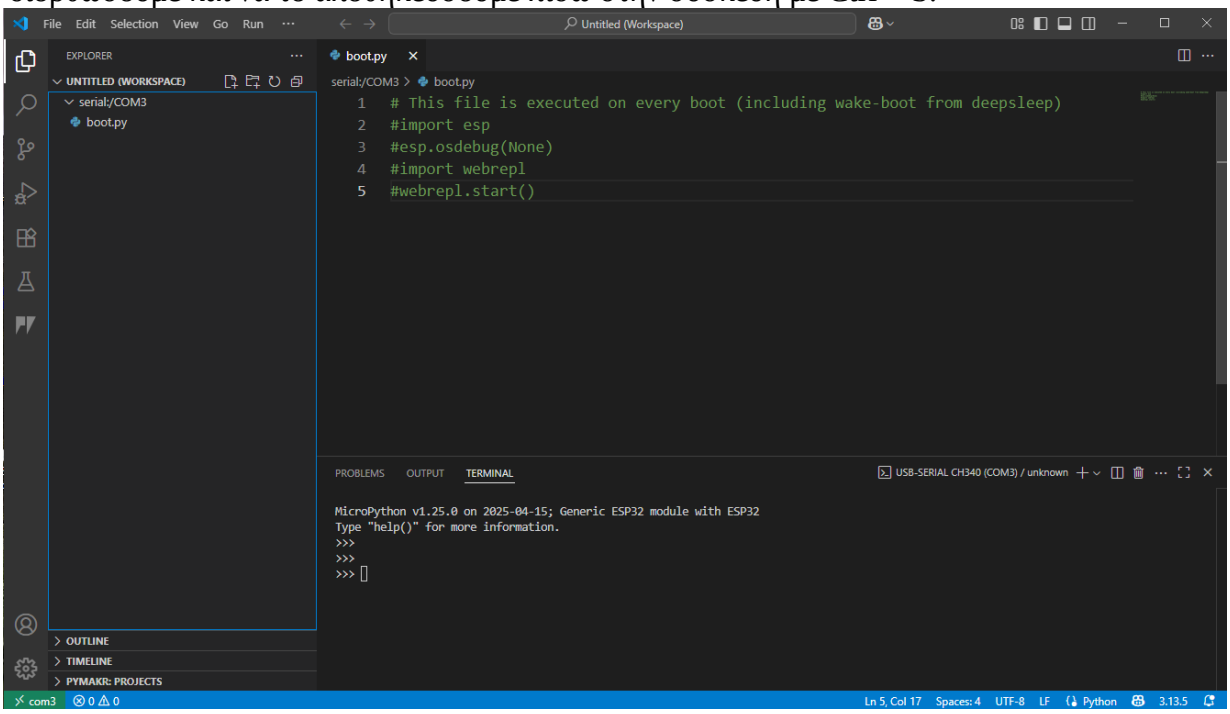
Επιλέγουμε την COM3 και πατάμε το σύμβολο του κεραυνού (connect device). Τότε ενεργοποιούνται τα εικονίδια του τερματικού του διαχειριστή αρχείων και της αποσύνδεσης.





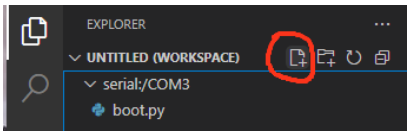
Αν πατήσουμε το τερματικό ανοίγει κάτω στο κέντρο το τερματικό της micropython και αν πατήσουμε τον διαχειριστή αρχείων ανοίγει τα περιεχόμενα της συσκευής στον διαχειριστή αρχείων.

Αν πατήσουμε το εργαλείο δεξιά πάνω που είναι ο Explorer, βλέπουμε τα αρχεία στο σύστημα αρχείων της συσκευής που αρχικά είναι μόνο το boot.py. Αν πατήσουμε πάνω σ' αυτό μπορούμε να το διορθώσουμε και να το αποθηκεύσουμε πίσω στην συσκευή με Ctrl + S.



Ο κύκλος ανάπτυξης

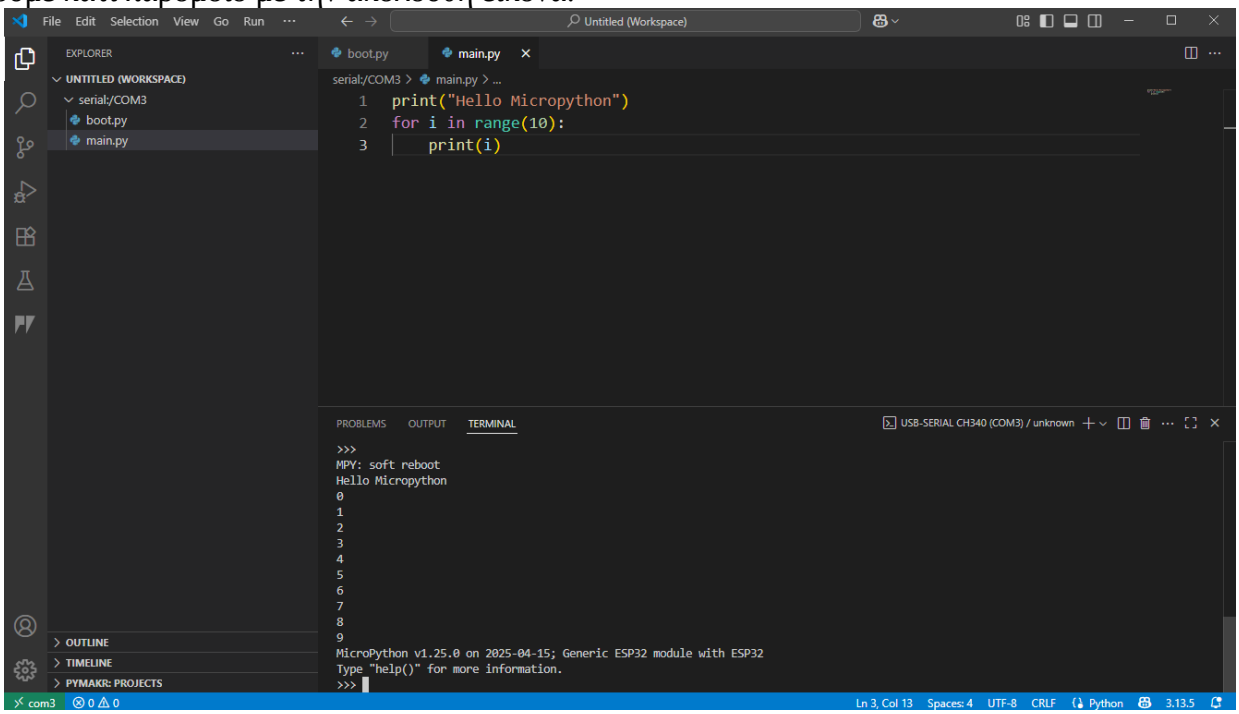
Πατάμε το εργαλείο με την σελίδα και το '+' ώστε να φτιάξουμε νέο αρχείο. Εκεί ζητάει όνομα και γράφουμε main.py. Τώρα στο κεντρικό τμήμα της οθόνης ανοίγει νέο tab στο οποίο μπορούμε να γράψουμε κώδικα σε python.



Γράφουμε για δοκιμή το παρακάτω πρόγραμμα και πατάμε Ctrl + S ώστε να αποθηκευτεί στην συσκευή.

```
print("Hello Micropython")
for i in range(10):
    print(i)
```

Στη συνέχεια κάνουμε κλικ στο κάτω μέρος που είναι το τερματικό και πατάμε Ctrl + D ώστε να εκτελεστεί ο κώδικας στην συσκευή. Αν δεν υπάρχουν συντακτικά λάθη θα εκτελεστεί και θα δούμε κάτι παρόμοιο με την ακόλουθη εικόνα:

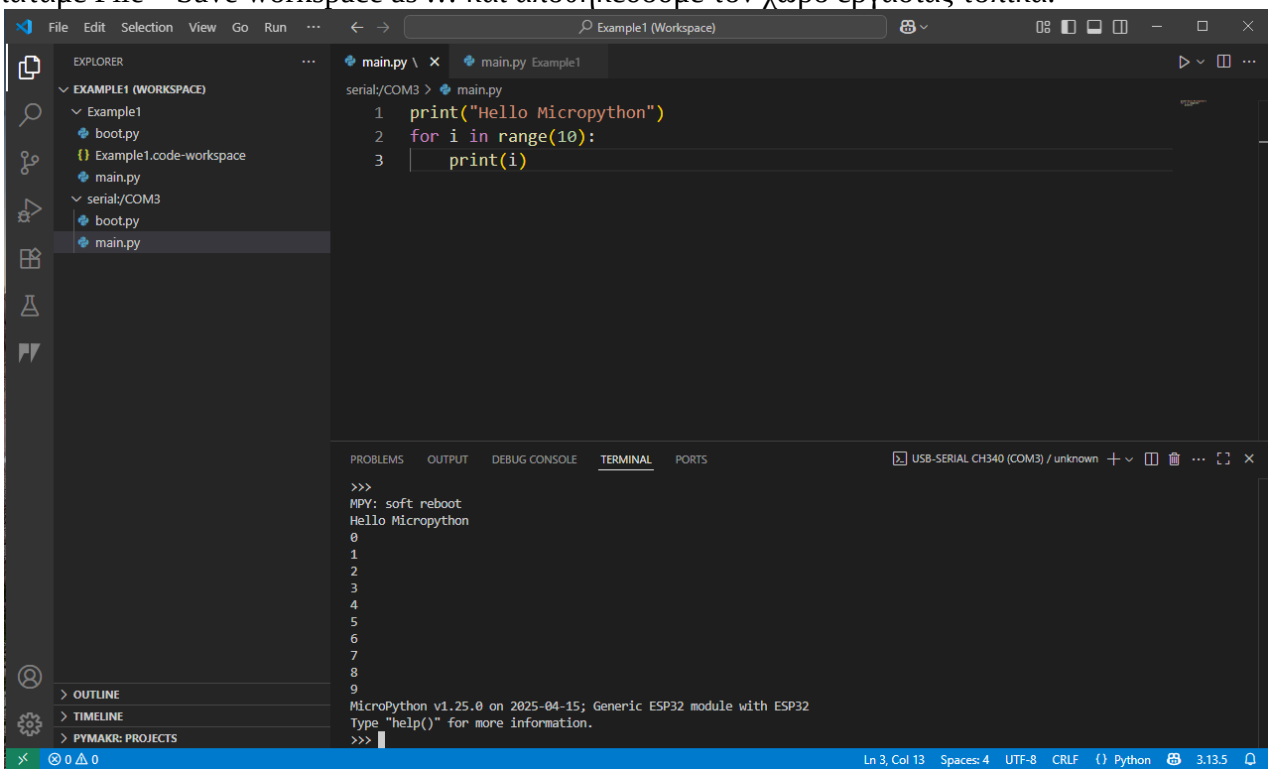


Αν θέλουμε να διορθώσουμε κάνουμε κλικ στο τμήμα του κώδικα, αποθηκεύουμε τις αλλαγές με CTRL + S, κάνουμε κλικ στο κάτω τμήμα που είναι το τερματικό, εκτελούμε με CTRL + D κ.ο.κ. Αν το πρόγραμμα σκόπιμα ή από λογικό λάθος μπαίνει σε ατέρμονα βρόχο μπορούμε να σταματήσουμε την εκτέλεση με CTRL + C. Προσοχή αν το πρόγραμμα βρίσκεται σε ατέρμονα βρόχο δεν μπορεί να αποθηκεύσει αλλαγές που γίνονται στον κώδικα. Δηλαδή για να αποθηκεύσουμε αλλαγές στην συσκευή το πρόγραμμα πρέπει να είναι σταματημένο.

Τρόπος προγραμματισμού των δραστηριοτήτων

Αρχικά στον κατάλογο του χρήστη δημιουργούμε φάκελο με όνομα Micropython. Μέσα σ' αυτόν νέο φάκελο με όνομα π.χ. Example1. Από το vscode πάμε στον Explorer ή File – Open folder και επιλέγουμε τον φάκελο Example1. Από το pymakr ανοίγουμε την σύνδεση όπως πριν και πατάμε το εικονίδιο του file manager και του τερματικού. Τώρα στον Explorer έχουμε δύο τοποθεσίες, την τοπική του χρήστη Example1 και την απομακρυσμένη π.χ. COM3. Τώρα πατώντας δεξί κλικ σε οποιοδήποτε αρχείο και μετά pymakr μπορούμε να αντιγράφουμε αρχεία μεταξύ των δύο περιοχών. Αυτό είναι χρήσιμο αν έχουμε βιβλιοθήκες και πρέπει να τις αντιγράψουμε στο σύστημα. Επίσης

μπορούμε να έχουμε ένα αντίγραφο ασφαλείας των αρχείων τοπικά στον υπολογιστή μας. Τέλος πατάμε File – Save workspace as … και αποθηκεύουμε τον χώρο εργασίας τοπικά.



Το περιβάλλον της Micropython

Με το πρόγραμμα τερματικού putty ή μέσα από το extension **PyMakr** του vscode ανοίγουμε ένα τερματικό. Η ταχύτητα επικοινωνίας είναι 115200bps.

Με την εντολή help() βλέπουμε οδηγίες για την χρήση του shell της Micropython.

Βασικά εμάς μας ενδιαφέρουν οι συντομεύσεις **CTRL-C** για να διακόπτουμε την εκτέλεση ενός προγράμματος και **CTRL-D** για να κάνουμε reset την πλακέτα.

Με την εντολή **`help('modules')`** βλέπουμε τα εγκατεστημένα modules της Micropython.

```
MicroPython v1.24.1 on 2024-11-29; Generic ESP32 module with ESP32
Type "help()" for more information.
>>>
>>> help('modules')
__main__          bluetooth         heapq             select
_asyncio          btree             inisetup          socket
_boot             builtins          io                ssl
_espnow           cmath             json              struct
_onewire          collections       machine           sys
_thread           cryptolib         math              time
_webrepl          deflate           micropython       tls
aioespnow         dht               mip/__init__      uasyncio
apa106            ds18x20           neopixel          uctypes
array             errno             network           umqtt/robust
asyncio/__init__  esp               ntptime           umqtt/simple
asyncio/core      esp32             onewire           upysh
asyncio/event     espnow            os                urequests
asyncio/funcs     flashbdev         platform          vfs
asyncio/lock      framebuf          random            webrepl
asyncio/stream    gc                re                webrepl_setup
binascii          hashlib           requests/__init__ websocket
Plus any modules on the filesystem
```

Αν θέλουμε να χρησιμοποιήσουμε ένα module γράφουμε **`import <όνομα>`** π.χ.

`import machine`

Αν γράψουμε **`dir(machine)`** θα εμφανιστούν όλες οι κλάσεις του module.

```
>>> dir(machine)
['__class__', '__name__', 'ADC', 'ADCBlock', 'DAC', 'DEEPSLEEP', 'DEEPSLEEP_RESET', 'EXT0_WAKE', 'EXT1_WAKE',
'HARD_RESET', 'I2C', 'I2S', 'PIN_WAKE', 'PWM', 'PWRON_RESET', 'Pin', 'RTC', 'SDCard', 'SLEEP', 'SOFT_RESET',
'SPI', 'Signal', 'SoftI2C', 'SoftSPI', 'TIMER_WAKE', 'TOUCHPAD_WAKE', 'Timer', 'TouchPad', 'UART', 'ULP_WAKE',
'WDT', 'WDT_RESET', '__dict__', 'bitstream', 'deepsleep', 'dht_readinto', 'disable_irq', 'enable_irq', 'freq',
'idle', 'lightsleep', 'mem16', 'mem32', 'mem8', 'reset', 'reset_cause', 'sleep', 'soft_reset', 'time_pulse_us',
'unique_id', 'wake_reason']
```

Αν επιλέξουμε μια κλάση π.χ. την Pin και γράψουμε **`dir(machine.Pin)`** θα δούμε όλες τις μεθόδους της κλάσης.

```
>>> dir(machine.Pin)
['__class__', '__name__', 'value', 'DRIVE_0', 'DRIVE_1', 'DRIVE_2', 'DRIVE_3', 'IN', 'IRQ_FALLING', 'IRQ_RISING',
'OPEN_DRAIN', 'OUT', 'PULL_DOWN', 'PULL_UP', 'WAKE_HIGH', 'WAKE_LOW', '__bases__', '__dict__', 'board', 'init',
'irq', 'off', 'on']
```

Έλεγχος ελεύθερου χώρου στο File system

Κάνουμε εισαγωγή το module os με **`import os`** και μετά καλούμε την μέθοδο **`os.statvfs('/')`**. Αυτή επιστρέφει μια πλειάδα με το πρώτο στοιχείο να είναι το μέγεθος του block, το τρίτο είναι ο συνολικός χώρος και το τέταρτο ο ελεύθερος χώρος.

```
>>> import os
>>>
>>> os.statvfs('/')
(4096, 4096, 512, 483, 483, 0, 0, 0, 0, 255)
```

Δηλαδή ο συνολικός χώρος είναι 4096 x 512 = 2097152 = 2 MB και ο ελεύθερος χώρος είναι 4096 x 483 = 1978368 bytes.

Αν γράψω **`stat = os.statvfs('/')`** και μετά

`print((stat[2] - stat[3]) * stat[0])` εμφανίζει την δεσμευμένη μνήμη του συστήματος αρχείων (Flash).

```
>>> stat = os.statvfs('/')
>>> print((stat[2] - stat[3]) * stat[0])
118784
```

Εμφάνιση αρχείων στο σύστημα αρχείων

Με **`os.listdir()`** βλέπουμε τα αρχεία που υπάρχουν στο σύστημα αρχείων την μνήμης flash.

```
>>> os.listdir()
['OLED-ESP32.code-workspace', 'bmp280.py', 'boot.py', 'courier20.py', 'font10.py', 'font6.py', 'freesans20.py',
'main.py', 'ssd1306.py', 'writer.py']
>>>
```

Έλεγχος μνήμης RAM

Κάνουμε εισαγωγή το module gc (garbage collector) με **`import gc`** και μετά γράφουμε **`gc.mem_free()`**. Αν θέλουμε χειροκίνητα να κάνουμε εξοικονόμηση μνήμης γράφουμε **`gc.collect()`** και ελέγχουμε πάλι την ελεύθερη μνήμη. Παρακάτω βλέπουμε ότι έχουμε 142Kbytes ελεύθερης μνήμης. Τέλος η εντολή **`gc.mem_alloc()`** επιστρέφει την δεσμευμένη μνήμη Ram από τον κώδικα της Python.

```
>>> import gc
>>> gc.mem_free()
138944
>>> gc.collect()
>>> gc.mem_free()
142512
>>>
>>> gc.mem_alloc()
37552
```

Συχνότητα λειτουργίας

Με την εντολή **`print(machine.freq())`** εμφανίζει την συχνότητα λειτουργίας της CPU που εξ' ορισμού είναι 160000000 (160MHz). Η μέγιστη είναι 240MHz και αλλάζει με την εντολή:

`machine.freq(240000000)`

Παράδειγμα 1

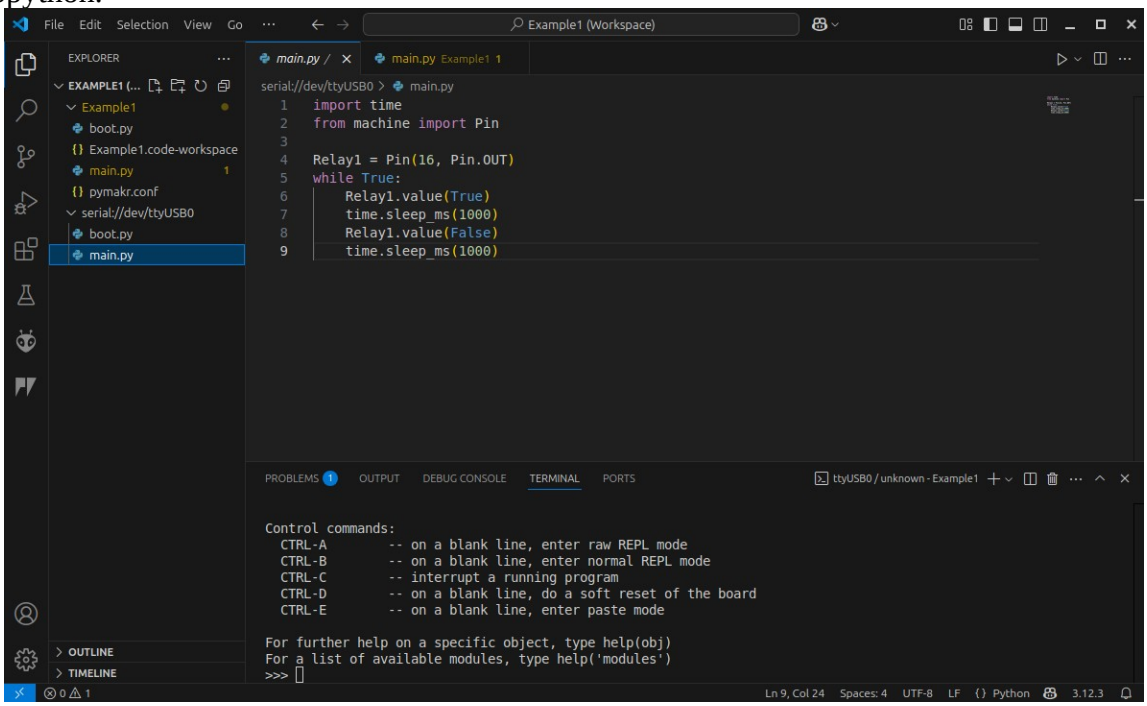
Το πρώτο μας πρόγραμμα – blink

Θα γράψουμε ένα πολύ απλό πρόγραμμα που δεν είναι άλλο από το blink του Arduino αλλά σε γλώσσα Python. Φτιάχνουμε ένα νέο Project με όνομα Example1 και το αποθηκεύουμε σε κάποιο φάκελο. Επιλέγουμε συσκευή com της πλακέτας και γράφουμε στο αρχείο main.py τον παρακάτω κώδικα:

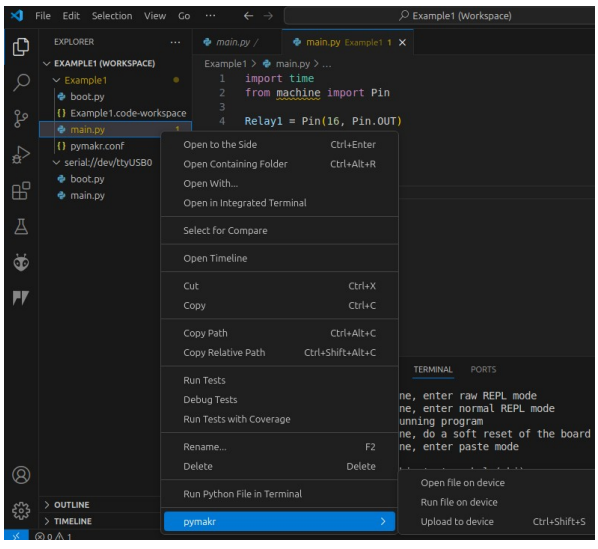
```
1 import time
2 from machine import Pin
3
4 Relay1 = Pin(16, Pin.OUT)
5 while True:
```

```
    Relay1.value(True)
    time.sleep_ms(1000)
    Relay1.value(False)
    time.sleep_ms(1000)
```

Το πρώτο Relay συνδέεται στον ακροδέκτη IO16. Στην 4η γραμμή γίνεται έξοδος και μέσα στην while η οποία εκτελείται για πάντα αρχικά το ενεργοποιεί, περιμένει 1sec, το απενεργοποιεί και περιμένει άλλο ένα sec.
Όπως βλέπουμε στον Explorer αριστερά έχουμε τον τοπικό φάκελο του έργου και το σύστημα αρχείων της συσκευής. Μπορούμε να διορθώνουμε το αρχείο είτε στον φάκελο του υπολογιστή είτε στην συσκευή απευθείας. Στον υπολογιστή φαίνονται και μερικά λάθη γιατί δεν γνωρίζει την ύπαρξη της βιβλιοθήκης machine. Πατάμε στην περιοχή τερματικού CTRL-D για να ξεκινήσει η εκτέλεση. Με CTRL-C σταματάμε την εκτέλεση. Για να λειτουργήσει ένα πρόγραμμα σε micropython είναι απαραίτητο το αρχείο **main.py** με τον κυρίως κώδικα, το **boot.py** το οποίο εκτελείται πρώτο κατά το reset και σ' αυτό συνήθως φορτώνουμε modules ή αρχικοποιούμε το υλικό. Στα παραδείγματά μας αυτό το αρχείο είναι κενό (μόνο σχόλια). Επίσης βάζουμε όσα αρχεία γίνονται import μέσα στον κώδικα του main και δεν υπάρχουν στα ενσωματωμένα modules της micropython.

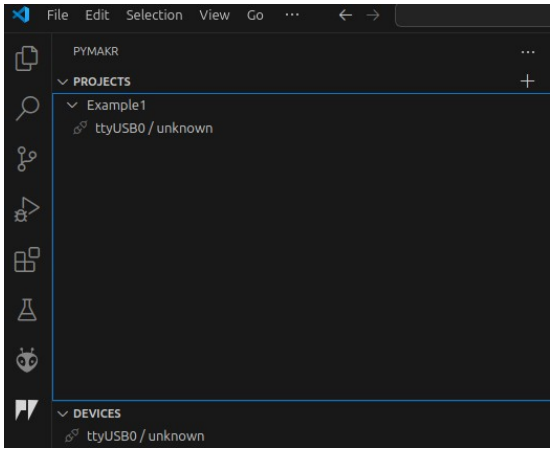


Αν κάνουμε αλλαγές στον υπολογιστή πρέπει να συγχρονίσουμε τα αρχεία πατώντας δεξί κλικ – pymakr – Upload to device.



Καλύτερα είναι να γράφουμε απευθείας στην συσκευή και στο τέλος να συγχρονίζουμε το φάκελο του έργου με αυτόν της συσκευής πατώντας το εικονίδιο του pymakr, κάνουμε focus την συσκευή (εδώ είναι Linux /dev/ttyUSB0) και πατάμε το τρίτο εικονίδιο Download project from device.

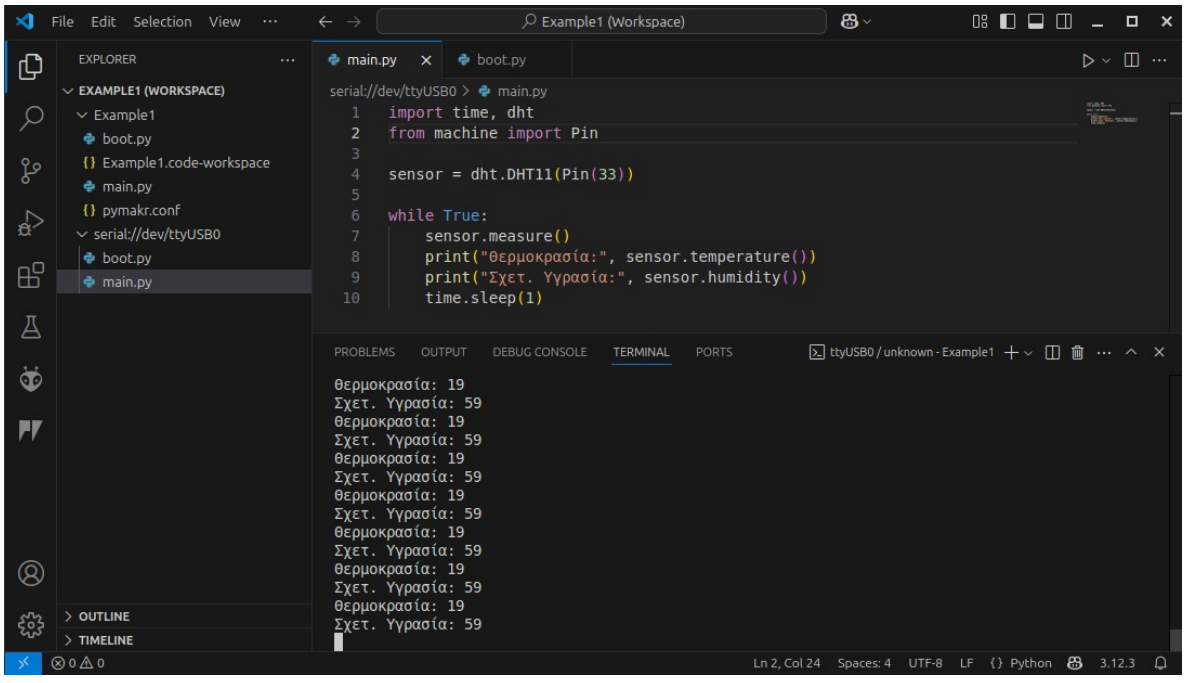
Προσοχή για να αποθηκευτούν οι αλλαγές στην συσκευή πρέπει να σταματάμε την εκτέλεση στο τερματικό με CTRL-C.



Παράδειγμα 2 – Μέτρηση υγρασίας και θερμοκρασίας με το DHT11

Γράφουμε τον παρακάτω κώδικα στο main.py.

```
import time, dht
from machine import Pin

sensor = dht.DHT11(Pin(33))

while True:
    sensor.measure()
    print("Θερμοκρασία:", sensor.temperature())
    print("Σχετ. Υγρασία:", sensor.humidity())
    time.sleep(1)
```



Παράδειγμα 3 – Έξοδος στα RGB Leds

Η πλακέτα διαθέτει 5 addressable RGB Leds με το chip WS2812B τα οποία συνδέονται στο pin IO25. Το module είναι το neopixel και είναι ήδη ενσωματωμένο μέσα στην micropython.

```
import machine, neopixel

np = neopixel.NeoPixel(machine.Pin(25), 5) #IO25, 5 x RGB Leds
#         R , G , B
```

```
np[0] = (255, 0, 0) #Κόκκινο 100%
np[1] = (125, 204, 223)
np[2] = (120, 153, 23)
np[3] = (255, 0, 153)
np[4] = (0, 0, 64) #Μπλε 25%
np.write()
```

Τροποποίηση

Να τροποποιηθεί ο παραπάνω κώδικας ώστε να κάνει διάφορα οπτικά effects όπως fade in, fade out, Kit (ιππότης της ασφάλτου) κλπ.

```
import machine, neopixel
import time

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

def blank():
    np[0] = np[1] = np[2] = np[3] = np[4] = (0, 0, 0)
    np.write()

def fade(R = 0, G = 0, B = 0):
    #Fade in
    for i in range(0, 256, 5):
        r = i * R; g = i * G; b = i * B
        for j in range(5):
            np[j] = (r, g, b)
        np.write()
        time.sleep_ms(50)
    #Fade out
    for i in range(255, -1, -5):
        r = i * R; g = i * G; b = i * B
        for j in range(5):
            np[j] = (r, g, b)
        np.write()
        time.sleep_ms(50)

def kit(color, times):
    for k in range(times):
        for i in range(5):
            np[0] = np[1] = np[2] = np[3] = np[4] = (0, 0, 0)
            np[i] = color
            np.write()
            time.sleep_ms(100)
        for i in range(4, -1, -1):
            np[0] = np[1] = np[2] = np[3] = np[4] = (0, 0, 0)
            np[i] = color
            np.write()
            time.sleep_ms(100)

while(True):
    fade(1, 0, 0)
    fade(0, 1, 0)
    fade(0, 0, 1)
    kit((0, 255, 128), 5)
    kit((255, 0, 128), 5)
    kit((255, 0, 0), 5)
    blank()
    time.sleep(1)
```

Παράδειγμα 4 – Ο βομβητής

Ο βομβητής (buzzer) συνδέεται στο pin IO5. Για να παραχθεί τόνος πρέπει να στείλουμε τετραγωνικό παλμό κάποιας ακουστικής συχνότητας στον βομβητή.

```
import time
from machine import Pin

beeper = Pin(5, Pin.OUT) #Ο βομβητής συνδέεται στο pin IO5

def beep():
    for i in range(200):
        beeper.value(1)
        time.sleep_us(1000)
        beeper.value(0)
        time.sleep_us(1000)

beep()
```

Στην συνάρτηση beep() παράγουμε τόνο συχνότητας 500Hz για 200 κύκλους. Ο κάθε κύκλος διαρκεί 2msec (2000μsec), 1msec High και 1msec Low δηλαδή 1 / (2 * 10^{-3}) = 500Hz. Εφόσον ο κάθε κύκλος διαρκεί 2msec και εμείς κάνουμε 200 επαναλήψεις τότε το beep διαρκεί 2msec x 200 = 400msec ή 0,4 δευτερόλεπτα.

Τροποποίηση

Τροποποιήστε το παραπάνω πρόγραμμα ώστε στην συνάρτηση beep να δίνουμε παραμέτρους συχνότητας και διάρκειας και να παράγει τον αντίστοιχο τόνο.

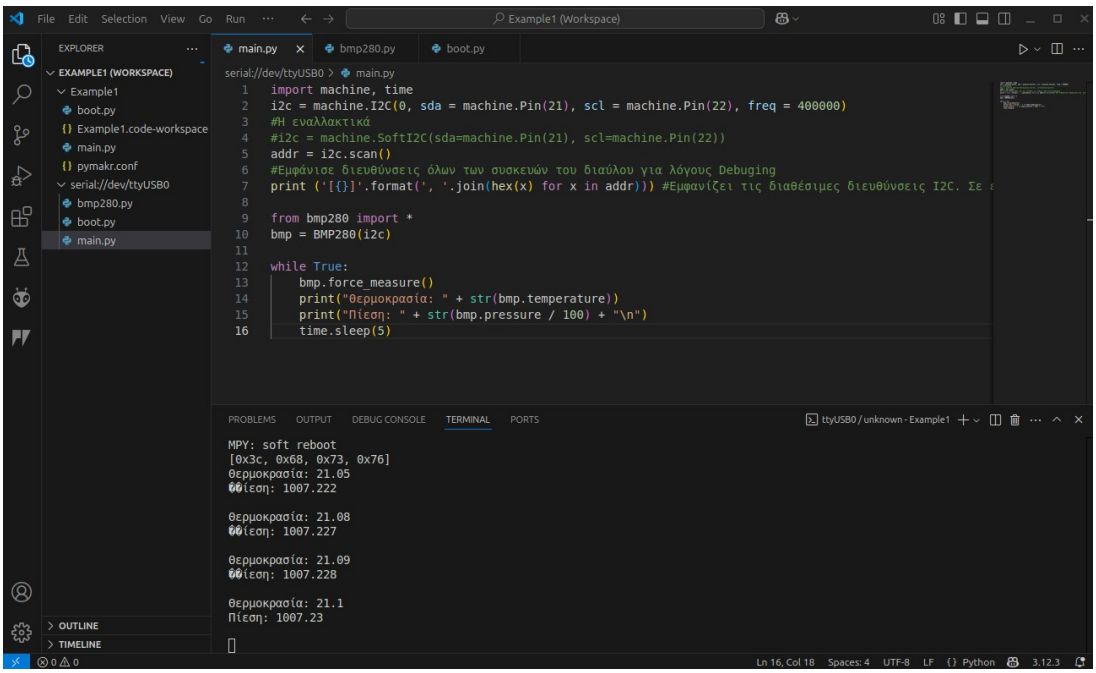
```python
import time
from machine import Pin

beeper = Pin(5, Pin.OUT) #Ο βομβητής συνδέεται στο pin IO5

def beep(freq = 500, dur = .5): #Default values
    #Υπολογισμοί
    period = 1.0 / freq #Περίοδος
    half_per = int((period / 2) * 1000000) #Ημιπερίοδος
    times = int(dur / period) #Αριθμός κύκλων
    for i in range(times):
        beeper.value(1)
        time.sleep_us(half_per)
        beeper.value(0)
        time.sleep_us(half_per)

beep() #Θα παραχθεί τόνος 500Hz για 0,5sec
beep(1000, .5) #1000Hz
beep(2000, .5) #2000Hz

for i in range(200, 4001, 10): #Σταδιακό ανέβασμα από 200 – 4000
    beep(i, .003)
for i in range(4000, 195, -10): #Σταδιακό κατέβασμα από 4000 – 200
    beep(i, .003)
```

Παράδειγμα 5 – Ο αισθητήρας BMP280

Ο αισθητήρας BMP280 της BOSCH είναι ενσωματωμένος στην πλακέτα και χρησιμοποιείται για μέτρηση ατμοσφαιρικής πίεσης και θερμοκρασίας. Συνδέεται με τον ESP32 μέσω του σειριακού διαύλου IIC ή I^2C. Επειδή δεν είναι ενσωματωμένο το module στην Micropython θα πρέπει να το συμπεριλάβουμε στο σύστημα αρχείων ως ξεχωριστό αρχείο σε γλώσσα Python. Δηλαδή μέσα στο σύστημα αρχείων θα έχουμε το boot.py, το main.py και το bmp280.py το οποίο υπάρχει στο αρχείο zip στο Example5.

Η πλακέτα μας χρησιμοποιεί το pin IO21 για το σήμα SDA που είναι δικατευθυντήριο και το pin IO22 για το σήμα SCL που είναι το σήμα χρονισμού από το ESP32 (master) προς τις συσκευές slaves του διαύλου IIC. Το BMP280 έχει διεύθυνση 0x76.

```python
import machine, time
i2c = machine.I2C(0, sda = machine.Pin(21), scl = machine.Pin(22), freq = 400000)
#Η εναλλακτικά
#i2c = machine.SoftI2C(sda=machine.Pin(21), scl=machine.Pin(22))
addr = i2c.scan()
#Εμφάνισε διευθύνσεις όλων των συσκευών του διαύλου για λόγους Debuging
print ('[{}]'.format(', '.join(hex(x) for x in addr))) #Εμφανίζει τις διαθέσιμες διευθύνσεις I2C. Σε εμάς είναι η 0x76

from bmp280 import *
bmp = BMP280(i2c)

while True:
    bmp.force_measure()
    print("Θερμοκρασία: " + str(bmp.temperature))
    print("Πίεση: " + str(bmp.pressure / 100) + "\n")
    time.sleep(5)
```



Πριν τις μετρήσεις μπορούμε να τροποποιήσουμε κάποιες παραμέτρους της βιβλιοθήκης όπως φαίνεται παρακάτω:

```
bmp.use_case(BMP280_CASE_INDOOR)
bmp.oversample(BMP280_OS_HIGH)

bmp.temp_os = BMP280_TEMP_OS_8
bmp.press_os = BMP280_PRES_OS_4

bmp.standby = BMP280_STANDBY_250
bmp.iir = BMP280_IIR_FILTER_2

bmp.spi3w = BMP280_SPI3W_ON
```

Παράδειγμα 6 – Η μονόχρωμη οθόνη γραφικών OLED

Η πλακέτα διαθέτει οθόνη OLED 0,96’’ και ανάλυσης 128x64 pixels. Η οθόνη συνδέεται στον ίδιο δίαυλο IIC με το BMP280 και έχει διεύθυνση 0x3c. Για να λειτουργήσει θα χρειαστούμε το module ssd1306.py το οποίο θα το βάλουμε μέσα στο σύστημα αρχείων. Το module βρίσκεται στο αρχείο zip στον φάκελο Example6.

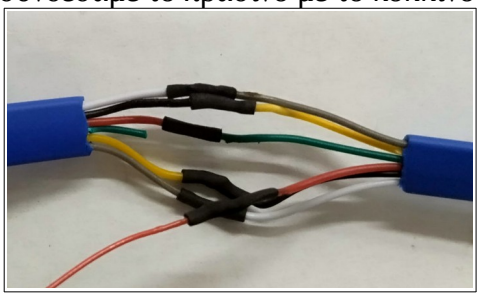
```
import machine, time
i2c = machine.I2C(0, sda = machine.Pin(21), scl = machine.Pin(22), freq = 400000)
#Η εναλλακτικά
#i2c = machine.SoftI2C(sda=machine.Pin(21), scl=machine.Pin(22))
addr = i2c.scan()
#Εμφάνισε διευθύνσεις όλων των συσκευών του διαύλου για λόγους Debuging
print ('[{}]'.format(', '.join(hex(x) for x in addr))) #Εμφανίζει τις διαθέσιμες διευθύνσεις I2C. Σε εμάς είναι η 0x76

from ssd1306 import SSD1306_I2C
#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.text('Hello', 5, 5)  #Θέση x, y
display.text('World', 5, 15, 1) #color 1/0
display.rect(0,0,127,63,1,0) #x, y, width, heigh, color=0/1, fill=0/1
display.show()

#display.fill(0) #Καθαρίζει οθόνη
#display.show()
```

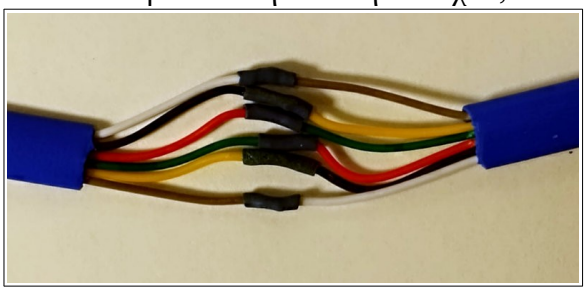
Παράδειγμα 7 – Η οθόνη χαρακτήρων 2x16

Μέσα στο κιτ υπάρχει και μία οθόνη χαρακτήρων 2 γραμμών και 16 στηλών. Τα περιφερειακά αυτά έχουν σχεδιαστεί για τον ελεγκτή arduino ARD:icon I και υπάρχει ένα πρόβλημα με τα καλώδια σύνδεσης τα οποία είναι αντεστραμμένα και την τάση λειτουργίας των περισσοτέρων η οποία είναι 5V ενώ η πλακέτα του ESP32 λειτουργεί στα 3,3V.

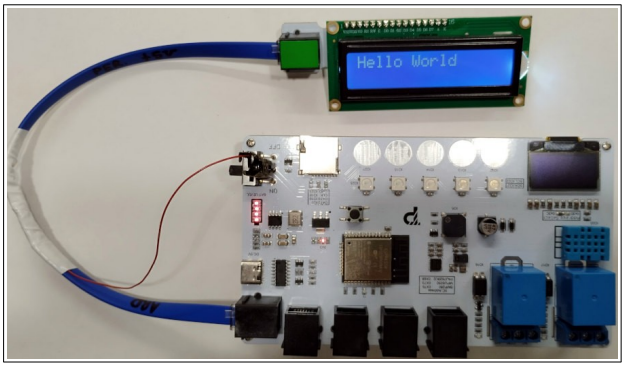
Το πρόβλημα λύνεται με δύο τρόπους:

1. **Μετατροπή καλωδίων RJ12.** Κόβουμε στην μέση ένα καλώδιο και αντιστρέψουμε τα χρώματα των εσωτερικών καλωδίων. Στην αριστερή εικόνα αριστερά είναι η πλακέτα ESP32 Icon II και δεξιά οποιοδήποτε περιφερειακό (αισθητήρας ή ενεργοποιητής) του Icon I. Στην αριστερή εικόνα δεν συνδέσαμε το πράσινο με το κόκκινο ώστε να δώσουμε 5V στην οθόνη και όχι 3,3V.

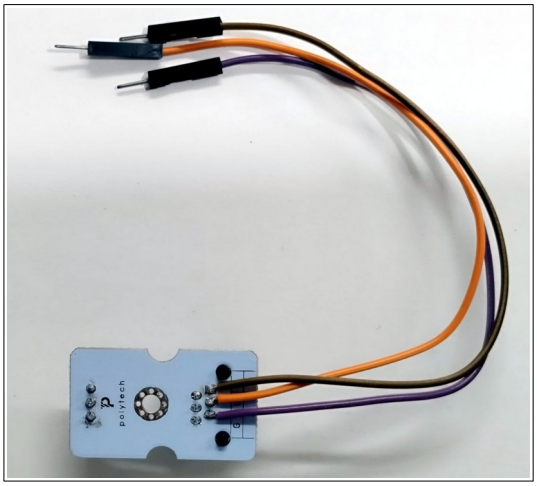


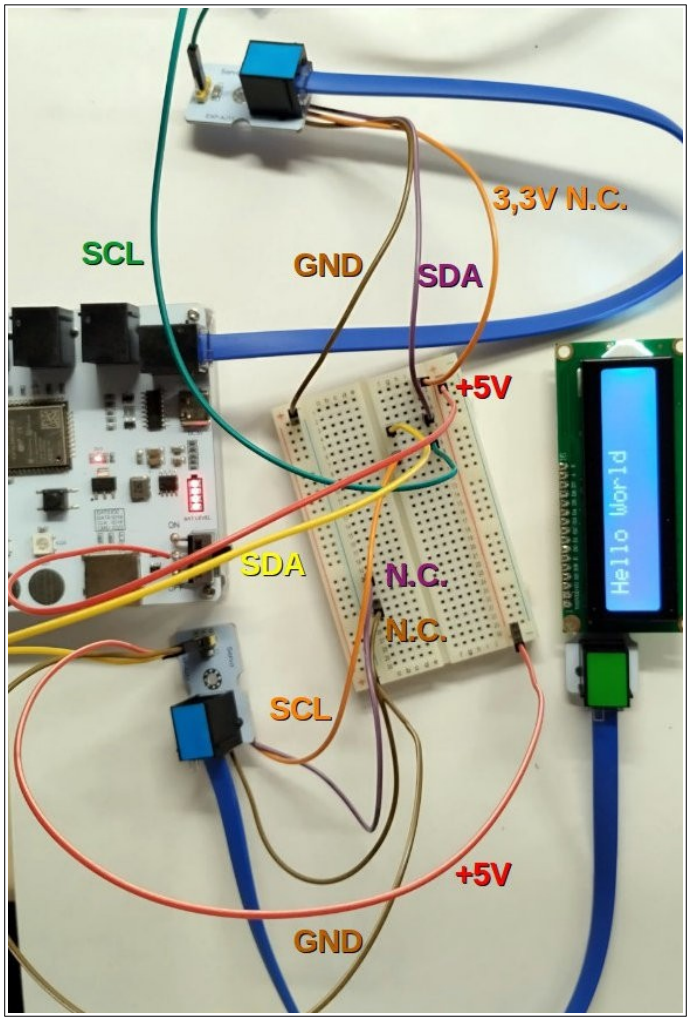


Στην διπλανή εικόνα φαίνεται η διασύνδεση της πλακέτας με την οθόνη χαρακτήρων 2x16. Η οθόνη λειτουργεί σωστά με τάση 5V την οποία την δίνουμε συνδέοντας ένα κροκοδειλάκι στην μεσαία επαφή του διακόπτη.



2. Χρήση Breadboard χωρίς μετατροπή καλωδίων. Θα χρειαστούμε δύο **EXP-AJ11** (Προσαρμογέας RJ11) από τους τρεις που διαθέτει το κιτ. Επίσης θα χρειαστούμε τρία χρωματιστά καλώδια breadboard test (M/M) τα οποία τα κόβουμε στην μέση και τα κολλάμε στις EXP-AJ11 σύμφωνα με την παρακάτω εικόνα αριστερά. Για το icon II το πορτοκαλί είναι **+3,3V**, το καφέ **GND** και το μωβ είναι το **GPIO** αναλόγως με την θέση σύνδεσης. Στο παράδειγμά μας είναι το σήμα **SDA**. Ακόμη θα χρειαστούμε τέσσερα καλώδια dupont (F/M) όπως είναι (χωρίς να τα κόψουμε) και την breadboard.

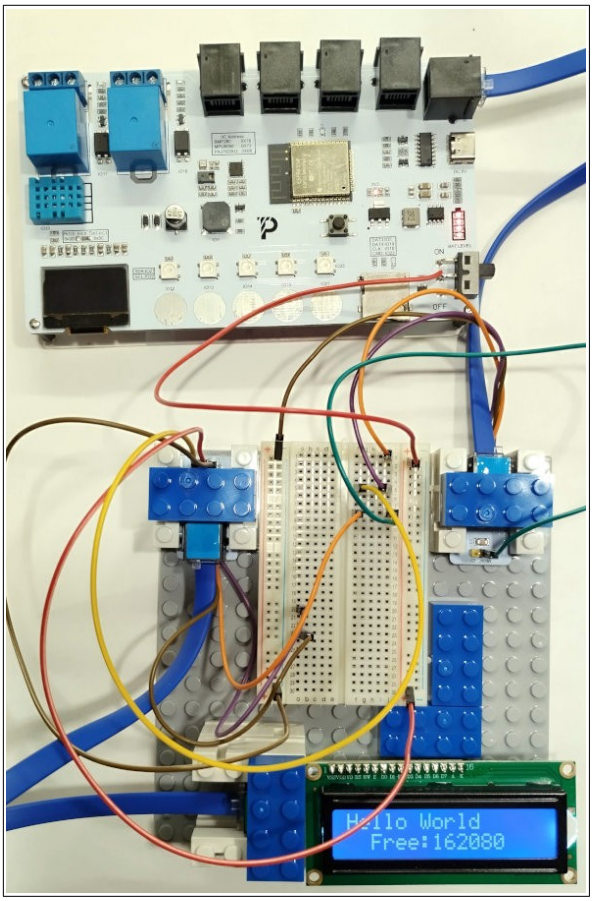




Στην διπλανή εικόνα φαίνεται ο τρόπος διασύνδεσης των δύο τροποποιημένων EXP-AJ11, των καλωδίων RJ12, του icon II, της breadboard και της οθόνης χαρακτήρων. Προσοχή τα +5V τα παίρνουμε από την μεσαία επαφή του διακόπτη. Στο πάνω EXP-AJ11 το πράσινο καλώδιο (SCL) συνδέεται στο μεσαίο pin με ένδειξη V. Στο κάτω EXP-AJ11 το καφέ πάει στο G, το κόκκινο στο V και το κίτρινο (SDA) στο S. Τα καλώδια με σήμανση N.C. (Not Connected) δεν συνδέονται κάπου.

Για να λειτουργήσει πρέπει να συμπεριληφθεί το αρχείο lcd_i2c8574.py μέσα στο σύστημα αρχείων.

```
# ------- Διεύθυνση I2C -----
I2C_Addr = 0x27 #  Η διεύθυνση μπορεί να αλλάξει στο πίσω πλακετάκι του PCF8574

# ------- Διαστάσεις LCD οθόνης ----
LCD_Dim = (16, 2)

from machine import I2C,  Pin
i2c = I2C(0, sda=Pin(21), scl=Pin(22), freq=100000)

import gc
from time import sleep
from lcd_i2c8574 import I2cLcd

lcd = I2cLcd(i2c, I2C_Addr, LCD_Dim)
gc.collect() #Συλλογή σκουπιδιών
mfree1 = gc.mem_free() #Ελεύθερη μνήμη RAM

lcd.clear() #Καθαρισμός οθόνης και τοποθέτηση cursor στην θέση 0, 0
lcd.write('Hello World', end='')
lcd.move_to(2, 1) #Τοποθέτησε cursor στην 3η στήλη, 2η γραμμή
lcd.write("Free:" + str(mfree1), end='')
```



Παράδειγμα 8 – Τα κουμπιά αφής

Η πλακέτα διαθέτει πέντε κουμπιά αφής τα οποία ακουμπώντας τα με το δάχτυλό μας μπορούμε να τα προγραμματίσουμε ώστε να κάνουν όποια ενέργεια θέλουμε. Ακολουθεί ο κώδικας δοκιμής για το 1ο κουμπί. Όταν είναι πατημένο εμφανίζει στο τερματικό το μήνυμα ‘Με πάτησες’. Δεν απαιτούνται επιπλέον αρχεία εκτός από το boot.py και το main.py.

```
import machine, time
threshold = 150 #Κατώφλι. Τιμές κάτω από εδώ σημαίνουν ότι πατήθηκε
touch_pin = machine.TouchPad(machine.Pin(12)) #Το πρώτο κουμπί

print("\nESP32 Touch Demo")
while True: # Για πάντα
  capacitiveValue = touch_pin.read() #Διάβασε τιμές
  if capacitiveValue < threshold: #Είναι κάτω από το κατώφλι
    print("Με πάτησες")
    time.sleep_ms(500)
```

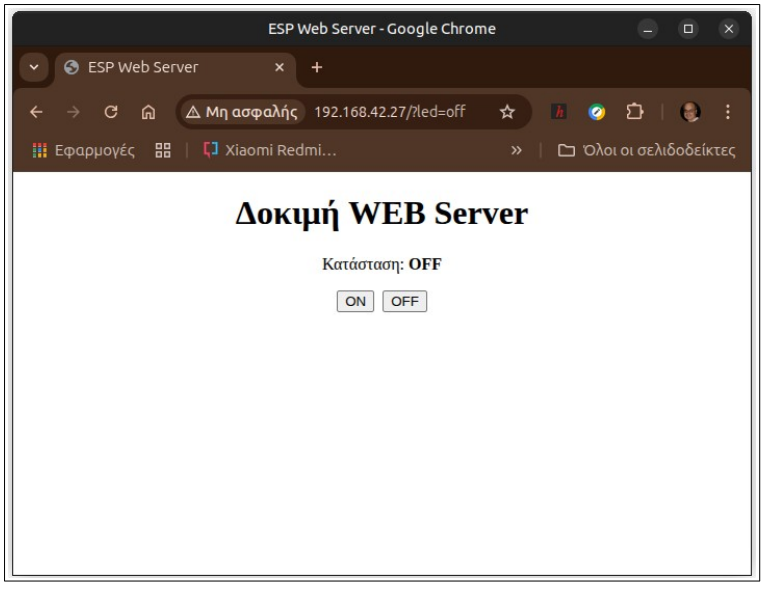
Παράδειγμα 9 – Σύνδεση WiFi και Web Server

Το άρθρωμα του ESP32 όπως και το ESP8266 διαθέτει υποσύστημα διασύνδεσης WiFi με υποστήριξη πλήρους TCP/IP stack. Στο παράδειγμά μας θα συνδεθεί η πλακέτα με το οικείο σημείο πρόσβασης (Access Point) και με το διαδίκτυο. Θα εκτελέσει έναν μικρό διακομιστή ιστού (WEB) και από τον browser θα αναβοσβήνουμε τα 5 RGB Leds της πλακέτας. Στις γραμμές 4 και 5 βάζουμε το SSID και τον κωδικό του δικού μας WiFi. Δεν απαιτούνται επιπλέον βιβλιοθήκες.

```
import machine, time, socket, network, gc, esp, neopixel
esp.osdebug(None)
gc.collect()
ssid = 'SSID' #Το SSID του τοπικού δικτύου WiFi
password = 'WiFiPassword' #Ο κωδικός του WiFi
station = network.WLAN(network.STA_IF) #Θα λειτουργήσει σαν σταθμός ώστε να συνδεθεί στο AP

station.active(True) #Ενεργοποίηση
station.connect(ssid, password) #Σύνδεση

while station.isconnected() == False: #Περίμενε μέχρι να συνδεθεί
    pass

print('Connection successful')
print(station.ifconfig())

```

```
17 np = neopixel.NeoPixel(machine.Pin(25), 5) #IO25, 5 x RGB Leds
18 led_state = False
19
20 #Ανάβει άσπρα τα 5 Led της πλακέτας
21 def led_ON():
22     global led_state
23     for i in range(5):
24         np[i] = (255, 255, 255) #Άσπρο
25     led_state = True
26     np.write()
27
28 #Σβήνει τα 5 Led της πλακέτας
29 def led_OFF():
30     global led_state
31     for i in range(5):
32         np[i] = (0, 0, 0) #Σβηστό
33     led_state = False
34     np.write()
35
36 #Η ιστοσελίδα που επιστρέφει στον browser
37 def web_page():
38     if led_state == True:
39         cur_state="ON"
40     else:
41         cur_state="OFF"
42     html = """
43 <!DOCTYPE HTML>
44 <html>
45  <head>
46   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
47   <title>ESP Web Server</title>
48   <meta name="viewport" content="width=device-width, initial-scale=1">
49  </head>
50  <body>
51   <center><h1>Δοκιμή WEB Server</h1></center>
52   <p><center>Κατάσταση: <strong>""" + cur_state + """</strong></center></p>
53   <p><center><a href="/?led=on"><button>ON</button></a> 
54   <a href="/?led=off"><button>OFF</button></a></center></p>
55  </body>
56 </html>"""
57     return html
58
59 s = socket.socket() #Δημιουργία υποδοχής TCP (Socket)
60 s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) #Options
61 s.bind(('', 80)) #Ακούει από οποιαδήποτε διεύθυνση IP - θύρα 80 HTTP
62 s.listen(5) #Ακούει μέχρι 5 ταυτόχρονες συνδέσεις
63
64 while True: #Για πάντα
65     conn, addr = s.accept()
66     print('Got a connection from %s' % str(addr))
67     request = conn.recv(1024) #Το αίτημα μέχρι 1024 bytes
68     request = str(request) #Να γίνει string από byte array
69     #print('Content = %s' % request) #Debug
70     request = request.split()[1] #Χώρισέ το στα κενά και πάρε το 2ο στοιχείο της λίστας
71     if request == '/?led=on': #Αν υπάρχει αίτημα on
72         print('LED ON')
73         led_ON() #Άναψε τα Led
74     elif request == '/?led=off': #Διαφορετικά αν υπάρχει αίτημα off
75         print('LED OFF')
76         led_OFF() #Σβήσε τα Led
77     response = web_page() #Ετοίμασε σελίδα για απάντηση
78     conn.send('HTTP/1.0 200 OK\r\nContent-type: text/html\r\n\r\n') #Επικεφαλίδα HTTP
79     conn.send(response) #Στείλε πίσω στον browser την σελίδα
80     conn.close() #Κλείσε σύνδεση
```

Στις γραμμές 43 έως 56 είναι η σελίδα που εμφανίζεται σε HTML. Στην γραμμή 52 παρεμβάλλεται η μεταβλητή cur_state ώστε στο σημείο αυτό η ιστοσελίδα να έχει δυναμική συμπεριφορά.
Η σελίδα όπως φαίνεται στον Chrome.



Χρήση ενδιάμεσου κώδικα Byte Code αντί πηγαίου

Η micropython μας δίνει την δυνατότητα εκτέλεσης προμεταγλωττισμένου κώδικα ώστε να επιτύχουμε ταχύτερη εκτέλεση και προστασία του πηγαίου προγράμματος από αντιγραφές. Για να μεταγλωττίσουμε ένα αρχείο χρειάζεται το **mpy_cross** το οποίο το εγκαθιστούμε στον υπολογιστή μας με **`pip install mpy_cross`**.
Αν θέλουμε να μεταγλωττίσουμε το αρχείο main.py πάμε στον φάκελο όπου βρίσκεται και γράφουμε **`python -m mpy_cross main.py`** και μας δημιουργεί το αρχείο main.mpy.
Το ανεβάζουμε από το vscode – pymakr στην πλακέτα και σβήνουμε το πηγαίο main.py. Τέλος στο **boot.py** γράφουμε σε μια γραμμή **`import main`**.

Παράδειγμα 10 – Ο αισθητήρας χειρονομίας

Η πλακέτα διαθέτει τον αισθητήρα PAJ7620U2 ο οποίος ανιχνεύει έως 9 χειρονομίες. Κουνώντας το χέρι μας σε απόσταση μέχρι 10 εκατοστά πάνω από τον αισθητήρα, αυτός καταλαβαίνει την χειρονομία που κάναμε. Με αυτό τον τρόπο μπορούμε να δώσουμε εντολές σε ένα σύστημα μόνο με την κίνηση του χεριού μας. Η σωστή διεύθυνση του αισθητήρα είναι η 0x73 και όχι η 0x68 που αναφέρεται πάνω στην πλακέτα. Μέσα στο σύστημα αρχείων πρέπει να συμπεριλάβουμε και το module gesture.py.

```
import machine, time
i2c = machine.I2C(0, sda = machine.Pin(21), scl = machine.Pin(22), freq = 400000)
#Η εναλλακτικά
#i2c = machine.SoftI2C(sda=machine.Pin(21), scl=machine.Pin(22))

from gesture import Gesture
g=Gesture(i2c)
#        0: Τίποτα
#        1: Μπροστά
#        2: Πίσω
#        3: Δεξιά
#        4: Αριστερά
#        5: Πάνω
#        6: Κάτω
#        7: Περιστροφή σύμφωνα με τους δείκτες του ρολογιού
#        8: Περιστροφή αντίθετα με τους δείκτες του ρολογιού
#        9: Κυματισμός

while 1:
    #g.print_gesture() #Εμφανίζει την χειρονομία
    value=g.return_gesture()
    #print(value) #Debug
    #if value==0:
    #    print("Αβράνεια")  # nothing
    if value==1:
        print("Μπροστά") #Το χέρι κινείται από πάνω προς τον αισθητήρα (άξονας Z)
    if value==2:
        print("Πίσω") #Το χέρι κινείται από τον αισθητήρα προς τα πάνω (άξονας Z)
    if value==3:
        print("Δεξιά") #Δεξιά (άξονας X)
    if value==4:
        print("Αριστερά") #Αριστερά (άξονας X)
    if value==5:
        print("Πάνω") #Πάνω (άξονας Y)
    if value==6:
        print("Κάτω") #Κάτω (άξονας Y)
    if value==7:
        print("Δείκτες ρολογιού") #Περιστροφή (άξονας X-Y)
    if value==8:
        print("Αντίθετα από δείκτες ρολογιού") #Αντίθετη περιστροφή (άξονας X-Y)
    if value==9:
        print("Κυματισμός") #Γρήγορα πάνω κάτω άξονα Y
    time.sleep(1)
```

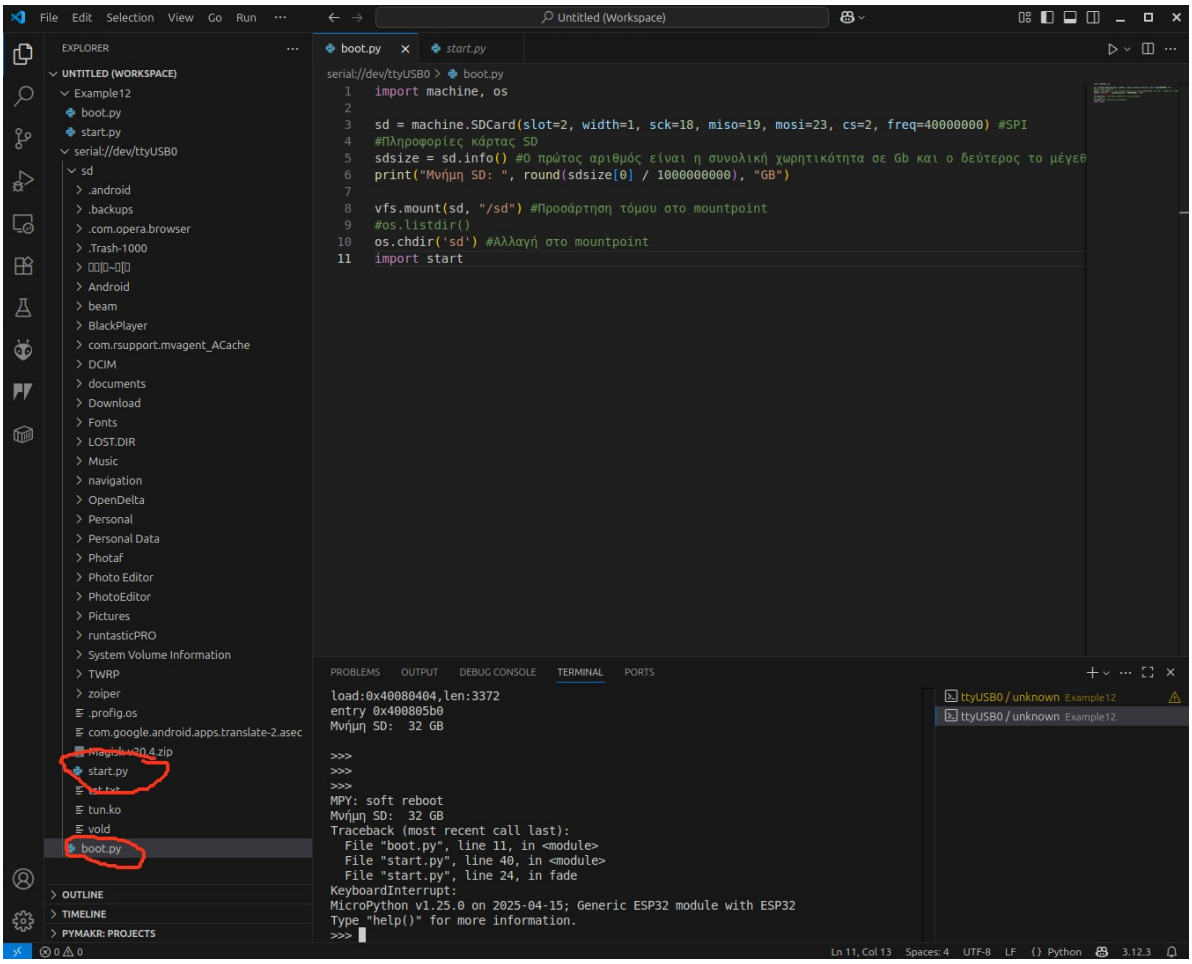
Παράδειγμα 11 – Η κάρτα μνήμης SD

Μπορούμε να αποθηκεύουμε ή να διαβάζουμε μεγάλα αρχεία όπως εικόνες, μουσική, αρχεία καταγραφής κλπ. στην εξωτερική κάρτα μνήμης SD. Θα χρειαστούμε μια καρτούλα microSD χωρητικότητας έως 32GB (ίσως υποστηρίζει και μεγαλύτερες). Το σύστημα αρχείων μπορεί να είναι FAT32 ή LitleFS. Για συμβατότητα ώστε να διαβάζεται από PC θα χρησιμοποιήσουμε σύστημα FAT32. Σε μνήμη 32GB η μονάδα εκχώρησης (allocation unit) είναι 8192 bytes. Η κάρτα συνδέεται με την διεπαφή SPI. Δεν θα χρειαστούμε επιπλέον αρχεία modules στο σύστημα αρχείων.

```
1  import machine, os
2
3  sd = machine.SDCard(slot=2, width=1, sck=18, miso=19, mosi=23, cs=2, freq=40000000) #SPI
4  #Πληροφορίες κάρτας SD
5  sdsize = sd.info() #Ο πρώτος αριθμός είναι η συνολική χωρητικότητα σε Gb και ο δεύτερος το μέγεθος τομέα (512)
6  print("Μνήμη SD: ", round(sdsize[0] / 1000000000), "GB")
7
8  vfs.mount(sd, "/sd") #Προσάρτηση τόμου στο mountpoint
9  stat = os.statvfs('/sd') #Πληροφορίες κατάτμησης
10 print("Συνολικός χώρος κατάτμησης:", round(stat[0] * stat[2] / 1000000000, 1), "GB")
11 print("Σε χρήση:", round(stat[0] * (stat[2] - stat[3]) / 1000000000, 1), "GB")
12
13 os.chdir('sd') #Αλλαγή στο mountpoint
14 print(os.listdir()) #Εμφάνιση αρχείων καταλόγου
15 #vfs.umount('/sd') #Αποπροσάρτηση τόμου
```

Παράδειγμα 12 – Εκτέλεση προγράμματος από την κάρτα SD

Υπάρχει η δυνατότητα εκτέλεσης προγραμμάτων Python τα οποία είναι αποθηκευμένα στην κάρτα μνήμης SD. Στον κατάλογο root της εσωτερικής μνήμης Flash αποθηκεύουμε μόνο το αρχείο **boot.py** το οποίο στην γραμμή 3 αρχικοποιεί την επικοινωνία της SD μέσω του διαύλου SPI και στην γραμμή 8 κάνει προσάρτηση του διαμερίσματος της SD στον κατάλογο /sd. Στην 10 μπαίνουμε στον κατάλογο sd και τέλος στην 11 κάνουμε import το αρχείο start.py ώστε να εκτελεστεί. Το αρχείο **start.py** έχει αποθηκευτεί στην μνήμη SD και είναι αυτό του παραδείγματος 3 (Τροποποίηση).



Στην παραπάνω εικόνα φαίνεται ένα διαμέρισμα FAT32 από κάρτα SD που είχε Android και δεν έχει γίνει format. Το αρχείο start.py αντιγράφεται στο root του /sd.
Ακολουθεί το αρχείο boot.py

```
import machine, os

sd = machine.SDCard(slot=2, width=1, sck=18, miso=19, mosi=23, cs=2, freq=40000000) #SPI
#Πληροφορίες κάρτας SD
sdsize = sd.info() #Ο πρώτος αριθμός είναι η συνολική χωρητικότητα σε Gb και ο δεύτερος το μέγεθος τομέα (512)
print("Μνήμη SD: ", round(sdsize[0] / 1000000000), "GB")

vfs.mount(sd, "/sd") #Προσάρτηση τόμου στο mountpoint

os.chdir('sd') #Αλλαγή στο mountpoint
import start
```

Παράδειγμα 13 – Ο αισθητήρας κίνησης 6 αξόνων (MEMS) MPU6050

Η πλακέτα διαθέτει και το chip **MPU6050** το οποίο συνδέεται μέσω του διαύλου I2C στην διεύθυνση 0x68. Ο αισθητήρας MPU6050 είναι ένα chip με μια μονάδα **MEMS** (Micro Electro-Mechanical System). Αυτό το τσιπ περιέχει γυροσκόπιο τριών αξόνων, επιταχυνσιόμετρο τριών αξόνων και ψηφιακό επεξεργαστή ελέγχου κίνησης. Επιπλέον, περιέχει έναν ενσωματωμένο αισθητήρα θερμοκρασίας. Μπορούμε να χρησιμοποιήσουμε αυτήν τη μονάδα για τη μέτρηση της ταχύτητας, της επιτάχυνσης, του προσανατολισμού, της μετατόπισης και άλλων παραμέτρων που σχετίζονται με την κίνηση. Σήμερα, όλα τα σύγχρονα smartphones διαθέτουν ενσωματωμένο αισθητήρα αδρανειακής κίνησης. Αυτός ο αισθητήρας παρέχει μια ολοκληρωμένη λύση για οποιοδήποτε σύστημα παρακολούθησης κίνησης έξι αξόνων.
Για το παράδειγμα θα χρειαστούμε και το αρχείο MPU6050.py να βρίσκεται μέσα στο root του συστήματος αρχείων.

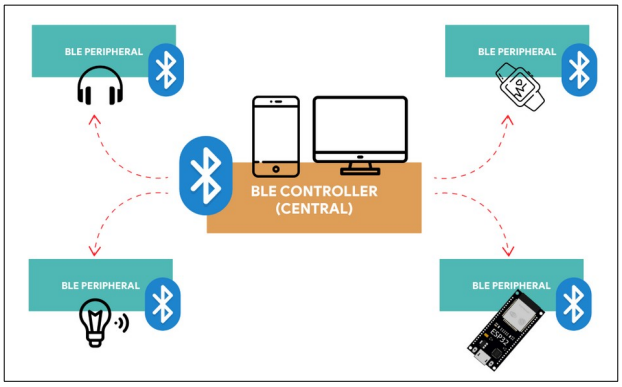
```
from MPU6050 import MPU6050

from machine import Pin
from time import sleep_ms

mpu = MPU6050()

# Για πάντα
while True:
    # Επιταχυνσιόμετρο
    #'''
    accel = mpu.read_accel_data() # Διάβασε το επιταχυνσιόμετρο [m / s^2]
    aX = accel["x"]
    aY = accel["y"]
    aZ = accel["z"]
    print("x: " + str(aX) + " y: " + str(aY) + " z: " + str(aZ))
    #'''

    # Γυροσκόπιο
    '''
    gyro = mpu.read_gyro_data()   # Διάβασε το γυροσκόπιο [deg/s]
    gX = gyro["x"]
    gY = gyro["y"]
    gZ = gyro["z"]
    print("x:" + str(gX) + " y:" + str(gY) + " z:" + str(gZ))
    '''

    # Θερμοκρασία
    #temp = mpu.read_temperature()   # Διάβασε την θερμοκρασία της συσκευής [degC]
    #print("Θερμοκρασία: " + str(temp) + "°C")

    # G-Force
    #gforce = mpu.read_accel_abs(g=True) # Διάβασε το μέτρο της επιτάχυνσης
    #print("G-Force: " + str(gforce))

    # Χρόνος παύσης σε (ms)
    sleep_ms(100)
```

Στο παραπάνω παράδειγμα δοκιμάζουμε το επιταχυνσιόμετρο, γυροσκόπιο, θερμόμετρο και την δύναμη G του αισθητήρα. Τα δεδομένα εμφανίζονται στο τερματικό. Σε επόμενο παράδειγμα θα εμφανίζουμε την κλίση της πλακέτας τρισδιάστατα με γραφικό τρόπο.

Παράδειγμα 14 – Bluetooth (B.L.E.)

Ο μικροελεγκτής ESP32 διαθέτει μια μονάδα BLE (Bluetooth Low Energy). Το BLE είναι σαν το κλασικό Bluetooth αλλά δεν λειτουργεί μέσω σειριακής θύρας αλλά πολύ πιο έξυπνα, έχει πολύ χαμηλότερη κατανάλωση ενέργειας αλλά και χαμηλότερες ταχύτητες επικοινωνίας. Χρησιμοποιείται συνήθως σε εφαρμογές IOT (Internet Of Things).

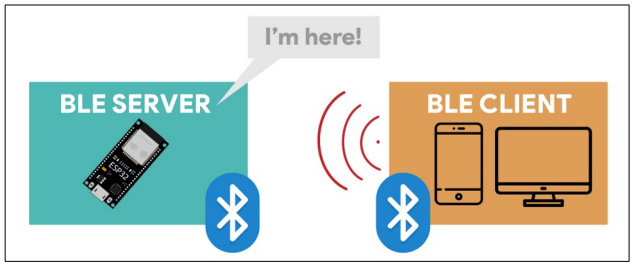
Περιφερειακό και Ελεγκτής BLE (Κεντρική Συσκευή)
Όταν χρησιμοποιείτε Bluetooth Low Energy (BLE), είναι σημαντικό να κατανοήσουμε τους ρόλους του BLE Peripheral και του BLE Controller (που αναφέρεται επίσης ως Κεντρική Συσκευή).



Το ESP32 μπορεί να λειτουργήσει είτε ως Περιφερειακό είτε ως κεντρική συσκευή. Όταν λειτουργεί ως περιφερειακό, δημιουργεί ένα προφίλ GATT και διαφημίζει την υπηρεσία του με χαρακτηριστικά που οι κεντρικές συσκευές μπορούν να διαβάσουν ή να αλληλεπιδράσουν με αυτά. Από την άλλη πλευρά, όταν έχει οριστεί ως κεντρική συσκευή, μπορεί να συνδεθεί με άλλες συσκευές BLE για να διαβάσει ή να αλληλεπιδράσει με τα προφίλ τους και να διαβάσει τα χαρακτηριστικά τους.

Στο παραπάνω διάγραμμα, το ESP32 αναλαμβάνει τον ρόλο του περιφερειακού BLE, λειτουργώντας ως η συσκευή που παρέχει δεδομένα ή υπηρεσίες. Το smartphone ή ο υπολογιστής μας λειτουργεί ως ελεγκτής BLE, διαχειριζόμενος τη σύνδεση και την επικοινωνία με το ESP32.

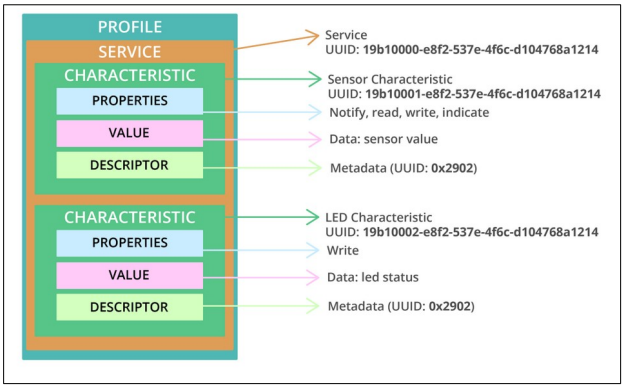
Διακομιστής και πελάτης BLE
Με το Bluetooth Low Energy, υπάρχουν δύο τύποι συσκευών: ο διακομιστής και ο πελάτης. Το ESP32 μπορεί να λειτουργήσει είτε ως πελάτης είτε ως διακομιστής. Στην παρακάτω εικόνα λειτουργεί ως διακομιστής, εκθέτοντας τη δομή GATT που περιέχει δεδομένα. Ο διακομιστής BLE λειτουργεί ως πάροχος δεδομένων ή υπηρεσιών, ενώ ο πελάτης BLE χρησιμοποιεί αυτές τις υπηρεσίες.



Ο διακομιστής **διαφημίζει** (εκθέτει) την ύπαρξή του, ώστε να μπορεί να βρεθεί από άλλες συσκευές και περιέχει δεδομένα που ο πελάτης μπορεί να διαβάσει ή να αλληλεπιδράσει με αυτά. Ο πελάτης σαρώνει τις κοντινές συσκευές και όταν βρει τον διακομιστή που αναζητά, δημιουργεί μια σύνδεση και μπορεί να αλληλεπιδράσει με αυτήν τη συσκευή διαβάζοντας ή γράφοντας τα χαρακτηριστικά της.
Ο διακομιστής BLE είναι ουσιαστικά το περιφερειακό BLE πριν από την εδραίωση μιας σύνδεσης. Ο πελάτης BLE είναι ο ελεγκτής BLE πριν από την εδραίωση μιας σύνδεσης. Πολλές φορές, αυτοί οι όροι χρησιμοποιούνται εναλλακτικά.

GATT

Το GATT, που σημαίνει Γενικό Προφίλ Χαρακτηριστικών (Generic Attribute Profile), είναι μια θεμελιώδης έννοια στην τεχνολογία Bluetooth Low Energy (BLE). Ουσιαστικά, χρησιμεύει ως ένα σχέδιο για το πώς οι συσκευές BLE επικοινωνούν μεταξύ τους. Σκεφτείτε το ως μια δομημένη γλώσσα που χρησιμοποιούν δύο συσκευές BLE για την απρόσκοπτη ανταλλαγή πληροφοριών.



- **Προφίλ:** τυπική συλλογή υπηρεσιών για μια συγκεκριμένη περίπτωση χρήσης.
- **Υπηρεσία:** συλλογή σχετικών πληροφοριών, όπως μετρήσεις αισθητήρων, στάθμη μπαταρίας, καρδιακός ρυθμός κ.λπ.
- **Χαρακτηριστικό:** είναι το σημείο όπου αποθηκεύονται τα πραγματικά δεδομένα στην ιεραρχία (τιμή).
- **Περιγραφέας:** μεταδεδομένα σχετικά με τα δεδομένα.
- **Ιδιότητες:** περιγράφουν πώς μπορεί να αλληλεπιδράσει η χαρακτηριστική τιμή. Για παράδειγμα: ανάγνωση, εγγραφή, ειδοποίηση, μετάδοση, υπόδειξη, κ.λπ.

Ας ρίξουμε μια πιο εις βάθος ματιά στην υπηρεσία και τα χαρακτηριστικά του BLE.

Υπηρεσία BLE

Το ανώτατο επίπεδο της ιεραρχίας είναι ένα προφίλ και αποτελείται από μία ή περισσότερες υπηρεσίες. Συνήθως, μια συσκευή BLE περιέχει περισσότερες από μία υπηρεσίες, όπως υπηρεσία μπαταρίας και υπηρεσία καρδιακών παλμών κλπ.

Κάθε υπηρεσία περιέχει τουλάχιστον ένα χαρακτηριστικό. Υπάρχουν προκαθορισμένες υπηρεσίες για διάφορους τύπους δεδομένων που ορίζονται από την SIG (Ομάδα Ειδικού Ενδιαφέροντος Bluetooth), όπως: Επίπεδο Μπαταρίας, Αρτηριακή Πίεση, Καρδιακός Ρυθμός, Ζυγαριά Βάρους, Περιβαλλοντική Ανίχνευση κ.λπ. Μπορείτε να δείτε στον παρακάτω σύνδεσμο τις προκαθορισμένες υπηρεσίες: https://www.bluetooth.com/specifications/assigned-numbers/

UUID

Ένα UUID είναι ένα μοναδικό ψηφιακό αναγνωριστικό που χρησιμοποιείται στο BLE και το GATT για τη διάκριση και τον εντοπισμό υπηρεσιών, χαρακτηριστικών και περιγραφών. Είναι σαν μια ξεχωριστή ετικέτα που διασφαλίζει ότι κάθε στοιχείο σε μια συσκευή Bluetooth έχει ένα μοναδικό όνομα.

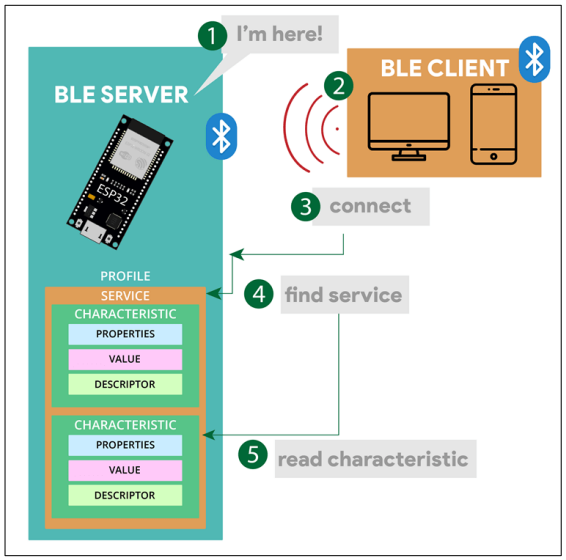
Κάθε υπηρεσία, χαρακτηριστικό και περιγραφέας έχει ένα UUID (Universally Unique Identifier). Ένα UUID είναι ένας μοναδικός αριθμός 128-bit (16 byte). Για παράδειγμα:

```
55072829-bc9e-4c53-938a-74a6d4c78776
```

Υπάρχουν συντομευμένα και προεπιλεγμένα UUID για υπηρεσίες και χαρακτηριστικά που καθορίζονται στο SIG (Bluetooth Special Interest Group) . Αυτό σημαίνει ότι εάν έχουμε μια συσκευή BLE που χρησιμοποιεί τα προεπιλεγμένα UUID για τις υπηρεσίες και τα χαρακτηριστικά της, θα γνωρίζουμε ακριβώς πώς να αλληλεπιδράσουμε με αυτήν τη συσκευή για να λάβουμε ή να αλληλεπιδράσουμε με τις πληροφορίες που αναζητάμε.

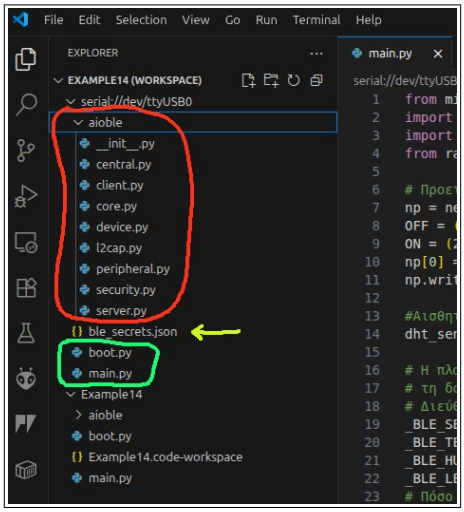
Μπορούμε επίσης να δημιουργήσουμε τα δικά σας προσαρμοσμένα UUID, αν δεν θέλουμε να μείνουμε σε προκαθορισμένες τιμές ή αν τα δεδομένα που ανταλλάσσουμε δεν εμπίπτουν σε καμία από τις κατηγορίες. Μπορούμε να δημιουργήσουμε προσαρμοσμένα UUID χρησιμοποιώντας αυτόν τον ιστότοπο δημιουργίας UUID: https://www.uuidgenerator.net/

Επικοινωνία μεταξύ συσκευών BLE
Ακολουθούν τα συνήθη βήματα που περιγράφουν την επικοινωνία μεταξύ συσκευών BLE.



1. Το BLE Peripheral (server) διαφημίζει την ύπαρξή του (ESP32).
2. Η κεντρική συσκευή BLE (client) σαρώνει για συσκευές BLE.
3. Όταν η κεντρική συσκευή βρει το περιφερειακό που αναζητά, συνδέεται με αυτό.
4. Μετά τη σύνδεση, διαβάζει το προφίλ GATT της περιφερειακής συσκευής και αναζητά την υπηρεσία που αναζητά (για παράδειγμα: αισθητήρας περιβάλλοντος).
5. Εάν εντοπίσει την υπηρεσία, μπορεί πλέον να αλληλεπιδράσει με τα χαρακτηριστικά. Για παράδειγμα, να διαβάσει την τιμή της θερμοκρασίας.

Ακολουθεί ο κώδικας σε micropython για την επικοινωνία με μια συσκευή που διαθέτει διασύνδεση Bluetooth π.χ. smartphone ή tablet. Το πρόγραμμα στέλνει κάθε 2 δευτερόλεπτα τις τιμές του αισθητήρα DHT11 και επίσης περιμένει για εντολή που ανάβει ή σβήνει το τελευταίο RGB LED. Για την υλοποίηση χρειάζεται να ανεβάσουμε την βιβλιοθήκη **aioble** που είναι ολόκληρος φάκελος όπως φαίνεται δίπλα στο κόκκινο περίγραμμα. Κατά την πρώτη εκτέλεση δημιουργείται αυτόματα το αρχείο ble_secrets.json. Εμείς γράφουμε μόνο το αρχείο main.py. Το πρόγραμμα χρησιμοποιεί επίσης το ενσωματωμένο module asyncio για ασύγχρονη εκτέλεση του προγράμματος.

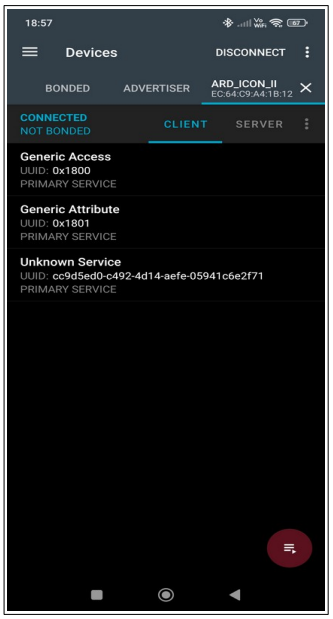


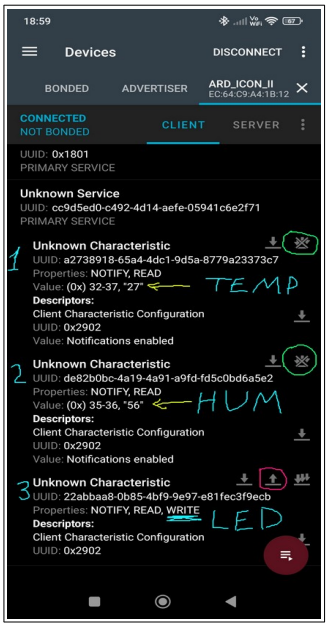
```python
from micropython import const
import bluetooth, aioble, struct, asyncio
import machine, neopixel, dht
from random import randint

# Προετοιμασία RGB Led
np = neopixel.NeoPixel(machine.Pin(25), 5) #IO25, 5 x RGB Leds
OFF = (0, 0, 0)
ON = (255, 255, 255)
np[0] = OFF
np.write()

#Αισθητήρας DHT11
dht_sensor = dht.DHT11(machine.Pin(33))

# Η πλακέτα λειτουργεί ως περιφερειακό και ο τύπος συσκευής είναι διακομιστής εκθέτοντας
# τη δομή GATT που περιέχει δεδομένα. Το κινητό είναι controller με τύπο client
# Διεύθυνση για παραγωγή UUIDs: https://www.uuidgenerator.net/
_BLE_SERVICE_UUID = bluetooth.UUID('cc9d5ed0-c492-4d14-aefe-05941c6e2f71')
_BLE_TEMP_SENSOR_CHAR_UUID = bluetooth.UUID('a2738918-65a4-4dc1-9d5a-8779a23373c7')
_BLE_HUM_SENSOR_CHAR_UUID = bluetooth.UUID('de82b0bc-4a19-4a91-a9fd-fd5c0bd6a5e2')
_BLE_LED_UUID = bluetooth.UUID('22abbaa8-0b85-4bf9-9e97-e81fec3f9ecb')
# Πόσο συχνά εκπέμπει advertising beacons.
_ADV_INTERVAL_MS = 250_000 #250000 msecs

# Καταχώρηση διακομιστή δομής GATT, της υπηρεσίας και των χαρακτηριστικών
ble_service = aioble.Service(_BLE_SERVICE_UUID)
sensorT_characteristic = aioble.Characteristic(ble_service, _BLE_TEMP_SENSOR_CHAR_UUID,
read = True, notify = True)
sensorH_characteristic = aioble.Characteristic(ble_service, _BLE_HUM_SENSOR_CHAR_UUID,
read = True, notify = True)
led_characteristic = aioble.Characteristic(ble_service, _BLE_LED_UUID,
read = True, write = True, notify = True, capture = True)

# Καταχώρηση υπηρεσίας BLE
aioble.register_services(ble_service)

# Κωδικοποίηση των δεδομένων των χαρακτηριστικών σε UTF-8 για αποστολή
def _encode_data(data):
    return str(data).encode('utf-8')

# Αποκωδικοποίηση του χαρακτηριστικού του LED από bytes σε αριθμό κατά την λήψη
def _decode_data(data):
    if data is not None:
        # Αποκωδικοποίηση από bytes σε ακέραιο
        number = int.from_bytes(data, 'big')
        return number

# Διάβασε δεδομένα από DHT11
def get_sensor_value():
    dht_sensor.measure() # Διάβασε τιμή αισθητήρα
    return dht_sensor.temperature(), dht_sensor.humidity() # Επιστροφή τιμών

# Διάβασε νέα τιμή και στείλε στο χαρακτηριστικό sensor
async def sensor_task():
    while True:
        value1, value2 = get_sensor_value()
        sensorT_characteristic.write(_encode_data(value1), send_update = True)
        sensorH_characteristic.write(_encode_data(value2), send_update = True)
        print('Θερμοκρασία: ', value1, 'Υγρασία: ', value2)
        await asyncio.sleep_ms(2000) # Περίμενε 2sec για την επόμενη αποστολή

# Περίμενε για συνδέσεις. Μην διαφημίζεις αν υπάρχει σύνδεση με controller
async def peripheral_task():
    while True:
        try:
            async with await aioble.advertise(
                _ADV_INTERVAL_MS,
                name = "ARD_ICON_II",
                services = [_BLE_SERVICE_UUID],
                ) as connection:
                    print("Σύνδεση από:", connection.device)
                    await connection.disconnected()
        except asyncio.CancelledError:
            # Catch the CancelledError
            print("Peripheral task cancelled")
        except Exception as e:
            print("Error in peripheral_task:", e)
        finally:
            # Ensure the loop continues to the next iteration
            await asyncio.sleep_ms(100)

# Περίμενε για εντολή στο χαρακτηριστικό του LED από controller
async def wait_for_write():
    while True:
        try:
            connection, data = await led_characteristic.written()
            print(data) #Debug
            print(type(data)) #Debug
            data = _decode_data(data)
            print('Connection: ', connection)
            print('Data: ', data)
            if data == 1:
                print('Το LED άναψε')
                np[0] = ON
                np.write()
            elif data == 0:
                print('Το LED έσβησε')
                np[0] = OFF
                np.write()
            else:
                print('Άγνωστη εντολή')
        except asyncio.CancelledError:
            # Catch the CancelledError
            print("Peripheral task cancelled")
        except Exception as e:
            print("Error in peripheral_task:", e)
        finally:
            # Ensure the loop continues to the next iteration
            await asyncio.sleep_ms(100)

# Εκτέλεση των διεργασιών
async def main():
    t1 = asyncio.create_task(sensor_task())
    t2 = asyncio.create_task(peripheral_task())
```

```
116     t3 = asyncio.create_task(wait_for_write())
117     await asyncio.gather(t1, t2)
118
119 asyncio.run(main())
```

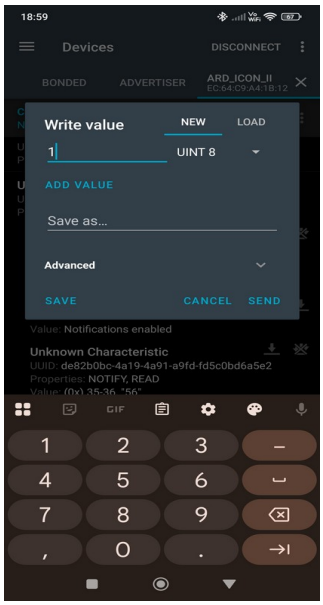
Δοκιμή

Αρχικά για δοκιμή κατεβάζουμε από το Google Play Store την εφαρμογή **nRF Connect** της Nordic και την εκτελούμε. Δίνουμε άδειες για ανίχνευση κοντινών συσκευών και αν ζητήσει τοποθεσίας. Πατάμε το κουμπί **SCAN** πάνω δεξιά και αν όλα πήγαν καλά εμφανίζει την συσκευή με όνομα ARD_ICON_II, την ένταση του σήματος σε dBm και την περίοδο του Beacon που δηλώσαμε στο πρόγραμμά μας. Πατάμε το κουμπί **CONNECT** και εμφανίζεται τρίτη στη σειρά η υπηρεσία με το UUID που δηλώσαμε. Πατάμε πάνω στην υπηρεσία Unknown Service και εμφανίζονται τα χαρακτηριστικά.





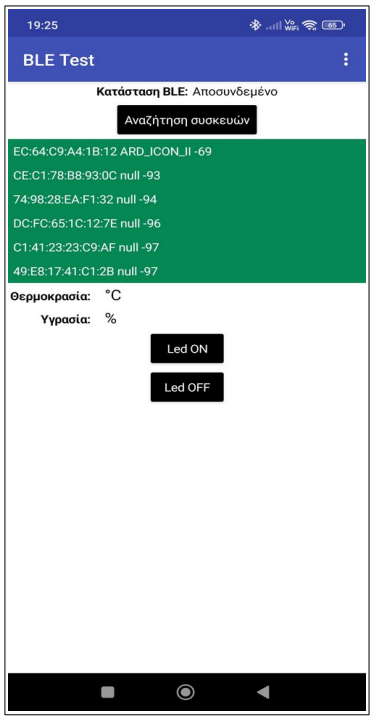
Το χαρακτηριστικό με **αριθμό 1** είναι η θερμοκρασία και αν πατήσουμε τα τρία βελάκια που είναι μέσα στον πράσινο κύκλο θα βλέπουμε τις τιμές που αλλάζουν κάθε 2 δευτερόλεπτα. Το **2** είναι η υγρασία και το **3** είναι το LED όπου παρατηρούμε ότι έχει ενεργό το attribute **Write** και σε αντίθεση με τα άλλα δύο, εμφανίζει το βελάκι στον κόκκινο κύκλο που δείχνει προς τα πάνω. Αν το πατήσουμε μπορούμε να στείλουμε δεδομένα στην πλακέτα μας.

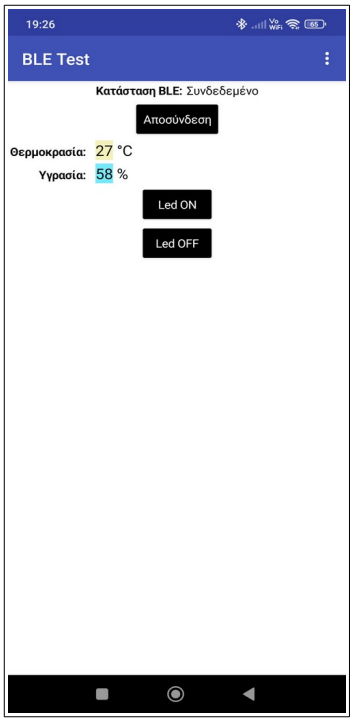


Επιλέγουμε τύπο δεδομένων **UINT8** και γράφουμε την τιμή 1 και πατάμε **SEND**. Βλέπουμε ότι το LED ανάβει. Αν στείλουμε 0 τότε το LED σβήνει.

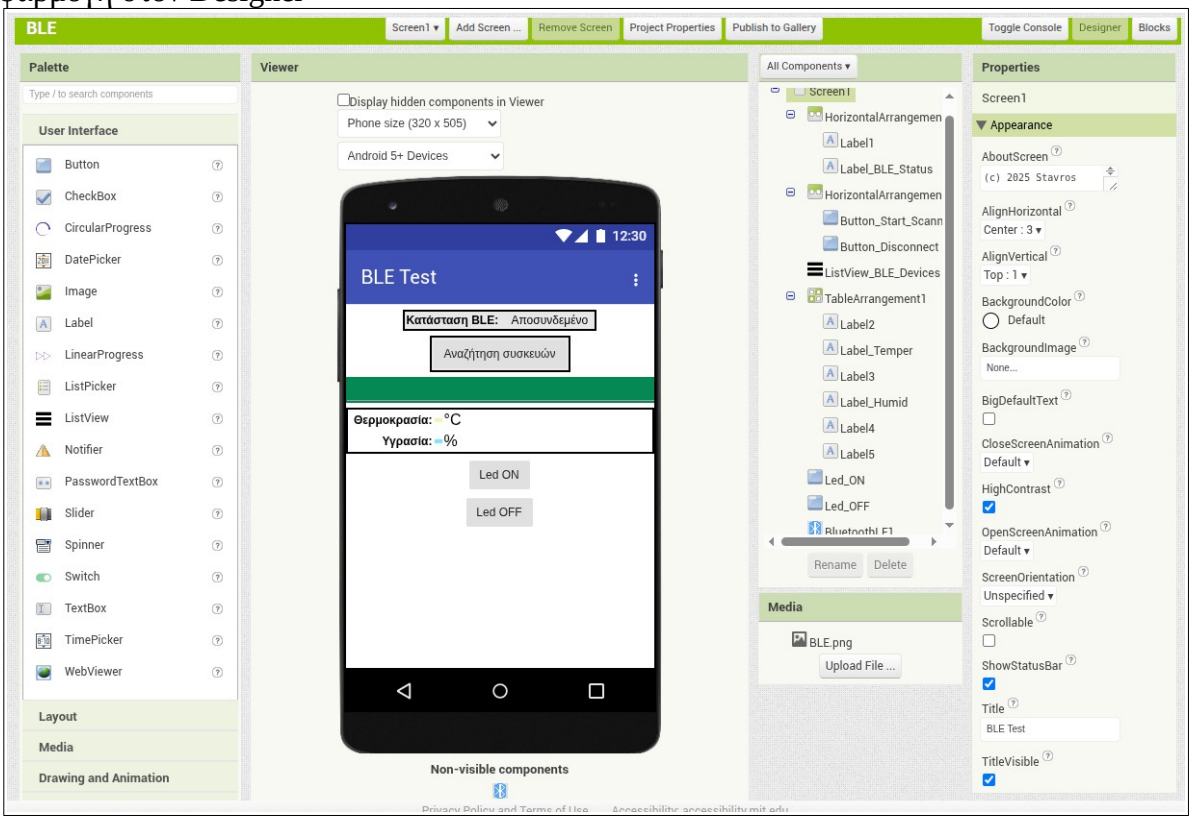
Εφαρμογή στο App Inventor

Μπορούμε να φτιάξουμε την δική μας εφαρμογή στο MIT App Inventor ώστε να διαχειριζόμαστε την πλακέτα μας μέσω BLE.

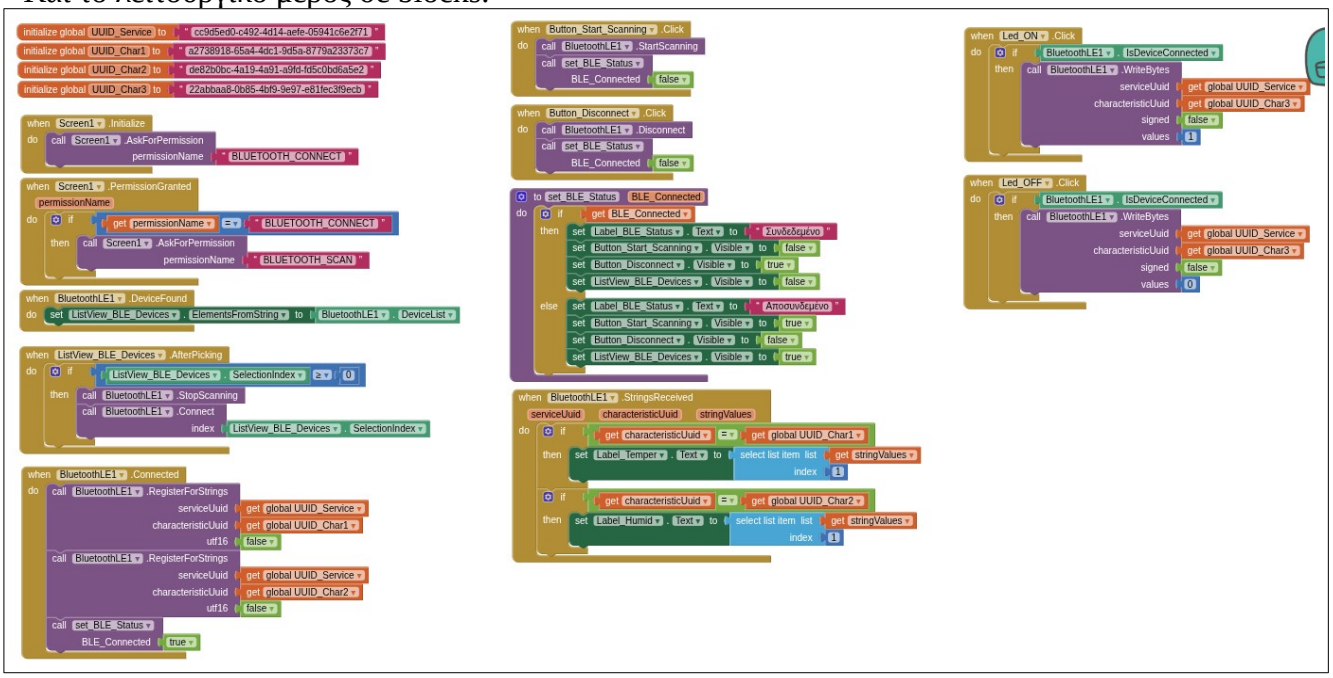




Η εφαρμογή στον Designer



Και το λειτουργικό μέρος σε blocks.



Σε επόμενα παραδείγματα θα εμβαθύνουμε στις εφαρμογές Android.