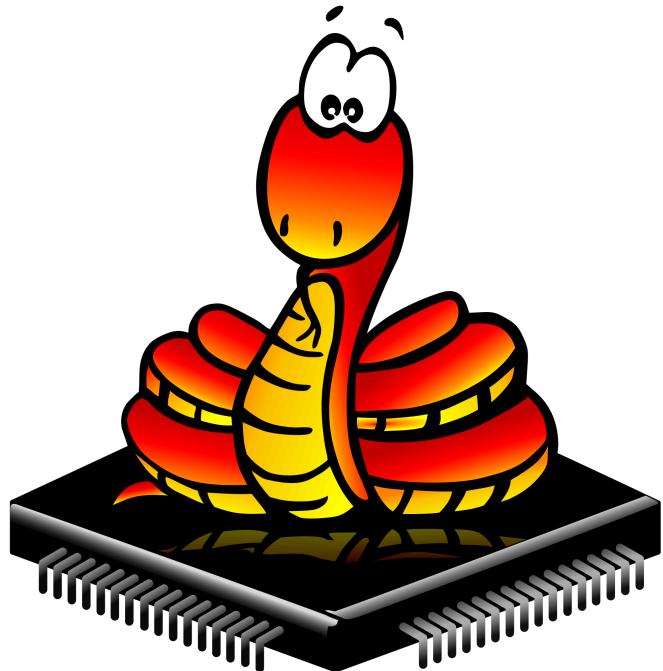


# תכנות MicroPython למורי הנדסת אלקטרוני ומחשבים



## מדריך למורה

אוקטובר 2024

גרסה 4.06

כתב: גדי הרמן

## תוכן עניינים

4	משימה 1 - התקנת <b>MicroPython</b> על גב בקר <b>ESP32</b> וסביבת הפיתוח <b>Thonny</b>
16	משימה 2 - כתיבת קוד <b>MicroPython</b> בסביבת הפיתוח <b>Visual Studio Code</b>
21	משימה 3 - כתיבת תוכנית ראשונה לביוץ פלט בשפת <b>Python</b>
26	משימה 4 - שימוש ב- <b>Timer</b> פנימי
28	משימה 5 - פסיקות חומרה
33	משימה 6 - קלט אות אנלוגי
37	משימה 7 - ממיר דיגיטלי לאנלוגי <b>DAC</b>
41	משימה 8 - הפעלת מד מרחק אולטרארו סוני מודם <b>hc-sr04</b>
47	משימה 9 - תקשורת <b>UART</b> בין 2 בקרים
51	משימה 10 - תקשורת <b>Bluetooth</b> מבוססת <b>HC-05</b>
57	משימה 11 - תקשורת <b>Bluetooth Low Energy</b> בברker <b>ESP32</b>
59	משימה 12 - אתחול קישוריות ה- <b>WiFi</b> בברker <b>ESP32</b>
67	משימה 13 - יימוש שרת אינטרנט מבוסס <b>HTTP</b>
73	משימה 14 - <b>HTTP GET</b>
78	משימה 15 - הפעלת צג גרפי דגם <b>SSD1306</b> OLED display
83	משימה 16 - שירות ענן מבוססי <b>MQTT</b>
111	משימה 17 - הפעלת רכיב השמעת קבצי <b>MP3</b> מבוסס על <b>YX5300</b>
118	משימה 18 - הפעלת צג גרפי <b>16x2 LCD</b>
123	משימה 19 - הפעלת צג <b>LCD</b> גרפי צבעוני <b>320*320</b> פיקסלים מבוסס על <b>ILI9341</b>
127	משימה 20 - קריאה וכתיבה של תגיות <b>RFID</b> תוך שימוש ב- <b>RC522</b>
138	משימה 21 - עדכון <b>RTC</b> פנימי בברker תוך שימוש ב- API מבוסס <b>JSON</b>
142	משימה 22 - תקשורת אלחוטית מבוססת מקם"ש <b>NRF24L01</b>
151	נספח א' - בדיקת הספריות האפשרות לתוכנות ב- <b>MicroPython</b> תחת בקר <b>ESP32</b>
152	נספח ב' - תוכנות אסינכרוני
154	נספח ג' - יבוא ספריות קוד תוך שימוש בכלים <b>PIP</b> ייעודיים ל- <b>MicroPython</b>
155	נספח ד' - מיפוי הדקי בברker <b>ESP32</b>
156	נספח ה' - עדכון קושחה לבקר <b>ESP32</b>
158	תנאי השימוש

מורים יקרים,

במטרה לקדם את החינוך הטכנולוגי בכלל ובמגמת הנדסת אלקטרוניתיקה ומחשבים בפרט כתיבתי מדריך למורה שיעזר לכם להיכנס לעולם מיקרו-בקרים הניתנים לתוכנות בשפת חספota MicroPython.

לרשוכם פתחתי קבוצה Whatsapp בשם "קהילת מורי MicroPython בבר" ESP32" שתשתמש כפלטפורמה לעזרה ושאלות בתחום זה. אתם מוזמנים ללמוד מהמדריך הנ"ל ובמקביל לפרסום ולשאול שאלות בקבוצה.

להלן הקישור לקבוצה:

<https://chat.whatsapp.com/LfOH8x5RSI6GKWSSePUAAb>

ניתן להוריד את כל קבצי הקוד בספר דרך הקישור הבא:

[https://github.com/GadiHerman/ESP32\\_MicroPython\\_AllBookFiles](https://github.com/GadiHerman/ESP32_MicroPython_AllBookFiles)

בברכה

גדי הרמן

## משימה 1 - התקנת Thonny על גבי בקר ESP32 וסביבת הפיתוח MicroPython

קישורים:

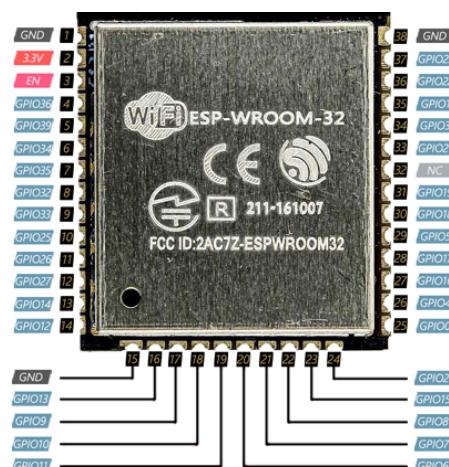
<https://www.youtube.com/watch?v=GJinEfhD4Cw>

### היכרות עם בקר ESP32

ESP32 הוא בקר רב עצמה מוגן SoC(Clomer System on Chip) ממערכת מחשב מלאה על גבי רכיב אחד. שפותה ESP32 על ידי חברת סינית בשם Espressif Systems.

<https://www.espressif.com/en/products/socs/esp32>

סוגים שונים של מיקרו-בקרים ESP32 זמינים בשוק. במדרייך זה אני מתמקד בגרסת-32 ESP-WROOM-32, הידוע גם בשם WROOM32. מיקרו-בקר זה זוכה לשבחים רבים ונפוץ בהרבה לוחות פיתוח מבוססי ESP32 כגון ESP32 DEVKIT. גם אם אתם משתמשים בלוחות פיתוח אחרים המבוססים על מיקרו-בקר ESP32, מדרייך זה עשוי להיות רלוונטי גם לכם, שכן מיקרו-בקרים אחרים של ESP32 חולקים תכונות הדקים דומות ל-WROOM32. לבקר זה 48 הדקים, כל אחד מהם משמש לפונקציות רבות. אך לא כל הפינים נגישים בכל לוח פיתוח של ESP32, ולהדקים מסוימים עשויים להיות מגבלים שימושם. להלן, תיאור בסיסי של הדקי GPIO הזמינים בברker ESP32-WROOM-32:



הברקרים הנ"ל פופולריים מאוד ומשמשים בעיקר ליישומי IoT כЛОם ליישומי "ה האינטרנט של הדברים" - .internet of things

הברker מספק ביצועים גבוהים בפרויקטים הדורשים עבודה עם מעבד (MCU) בעל 2 ליבות הכלול תקשורת רבתות ועבודה בזרמים נמוכים לאורך זמן. בנוסף, ESP32 מספק ביצועים גבוהים עם MCU ליבה כפולה העודדת בתדר של 200MHz כמו כן הברker מצוי בזיכרון הבזק בנפח של 4 מגה-בייט. הברker מצוי ברזב הממשקים הזמינים היום כולל Wi-Fi, BT, I2C, SPI ו-LCD.

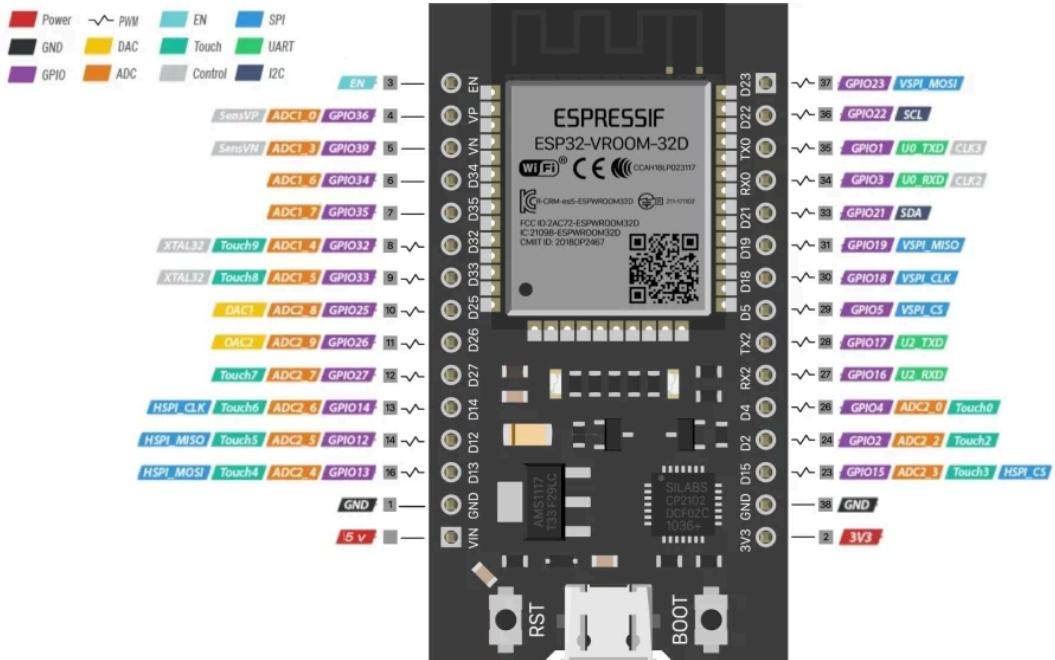
הברker מגיע עם מערכת ההתקנים הבא:

- 18 12-bit ADC pins
- 2 8-bit DAC pins
- 3 SPI interfaces
- 2 I2C interfaces
- 3 UART interfaces
- 16 PWM channels

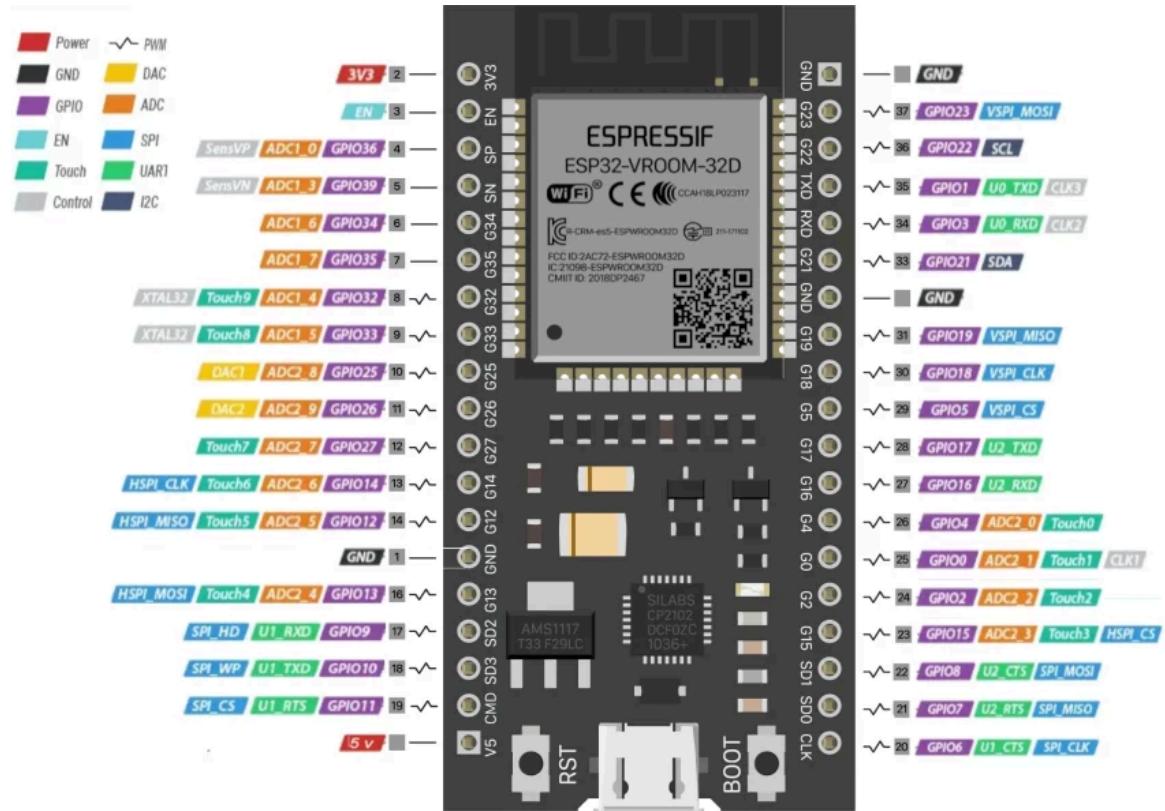
- 10 touch pins

לצורך עבודת פיתוח תוכנה תחת בקר ESP-WROOM-32 ESP-Chip חבותות שונות מספקות מספר מוצרים. ערכת הפיתוח הופולארית ביותר היא ESP32-DevkitC הנמכרת עלות של כ- 4 עד 6 דולר ליחידה.

מספר לוחות פיתוח המבוססים על ESP32-DevkitC זמינים עם הקצאות פינים שונות. לא כל פין ה-GPIO קיימם בכל הלוחות, אולם כל ה-GPIO מתנהגים אותו הדבר, לא משנה באיזה לוח פיתוח ESP32 נעשה שימוש. ניתן לראות את הקצאת הפינים של גרסת 30 הפינים של לוח הפיתוח ESP32-WROOM-32D בתמונה למטה.



האיור הבא מציג את הקצאת הפינים של גרסת 38 הפינים של לוח הפיתוח ESP32.



כרטיס הפיתוח מספק לנו את כל מה שהוא צריך לו כדי לתוכנת את הבקר, דרך ממשק USB המאפשר לחבר את הבקר לחשב PC ולעבוד ישירות אליו.

ניתן לתוכנת את הבקר במספר שפות פיתוח. במסגר זה נמקד את הדרישה שלנו בפיתוח תוכנה בסביבת Python או ליתר דיוק בסביבת MicroPython שהיא גרסה רזה של שפת Python המותאמת לעובדה על מספר בקרים בניהם ESP32.

### התקנת סביבת הפיתוח Thonny

THONNY היא סביבת פיתוח המיעדת למתחילה. הסביבה מאפשרת ריצת קוד בשפת Python, בשפת MicroPython. כמו כן סביבת הפיתוח זו תעזר לנו להתקין את הקושחה הרלוונטיית כדי להפוך את ה-ESP32 לעבודה עם MicroPython. להלן קישור לאתר התוכנה:

<https://thonny.org/>



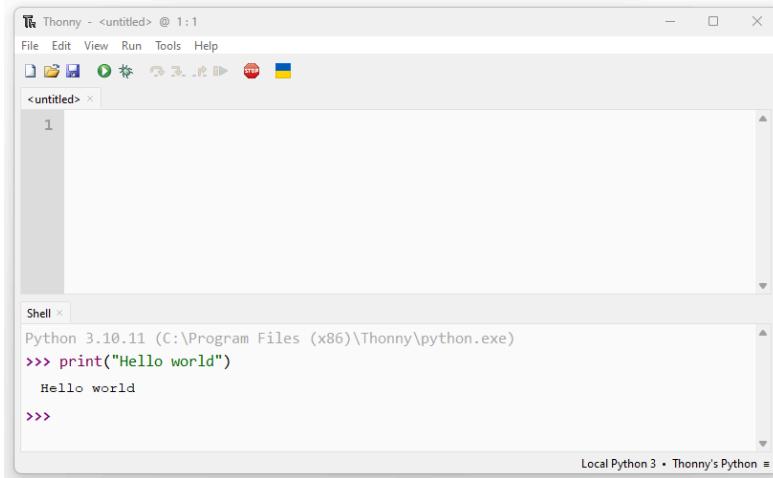
### Official downloads for Windows

**Installer with 64-bit Python 3.10**, requires 64-bit Windows 8.1 / 10 / 11  
[thonny-4.1.4.exe \(21 MB\)](#) ↗ recommended for you

**Installer with 32-bit Python 3.8**, suitable for all Windows versions since 7  
[thonny-py38-4.1.4.exe \(20 MB\)](#)


Download version **4.1.4** for  
Windows • Mac • Linux

נריץ את סביבת הפיתוח:



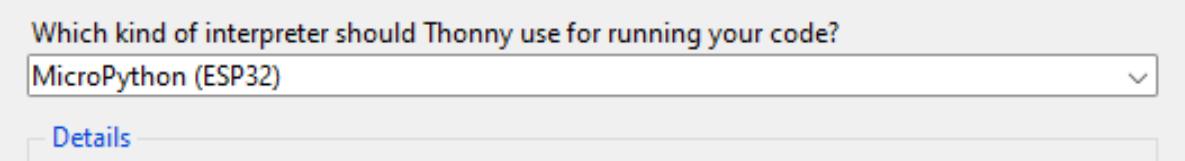
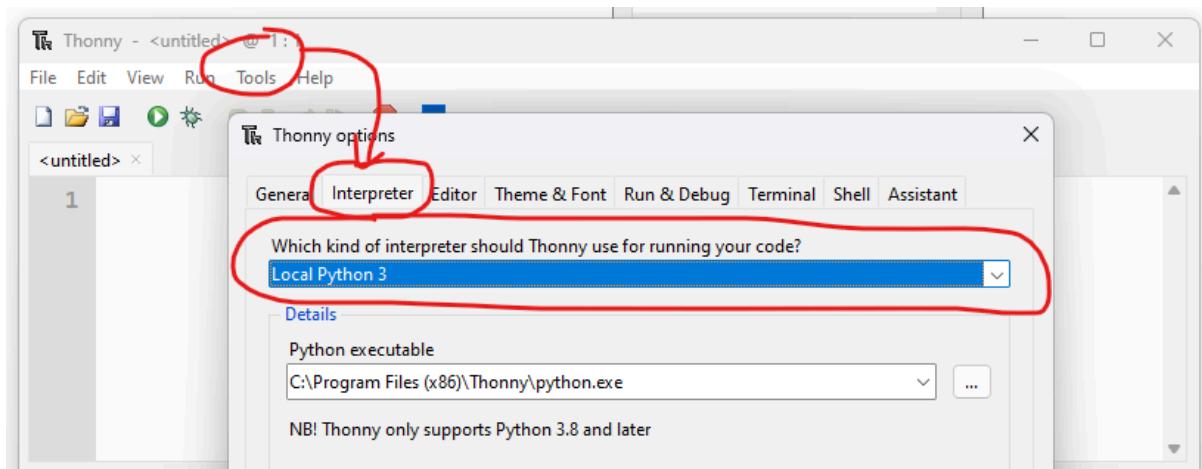
### אפשרות א': התקנת MicroPython על בקר ESP32 תוך שימוש בסביבת הפיתוח Thonny

נוריד את הגרסה האחרונה של MicroPython מהאתר הבא:

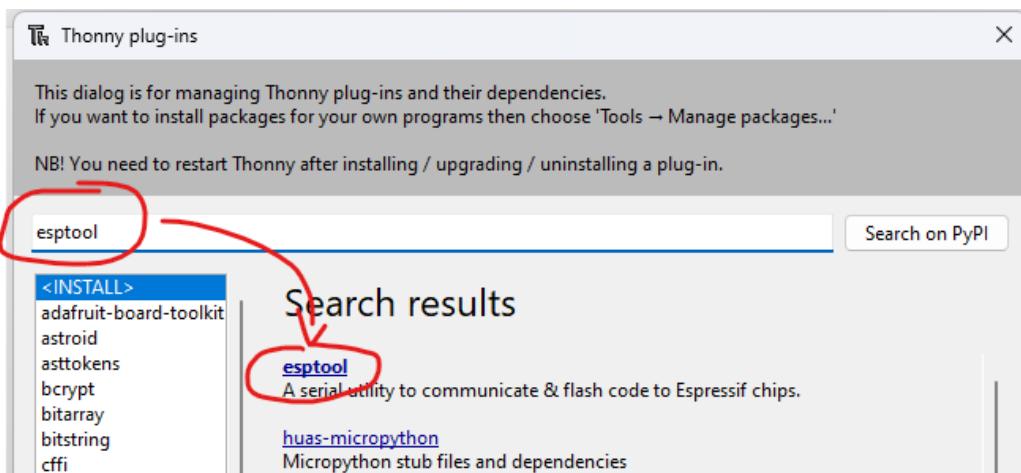
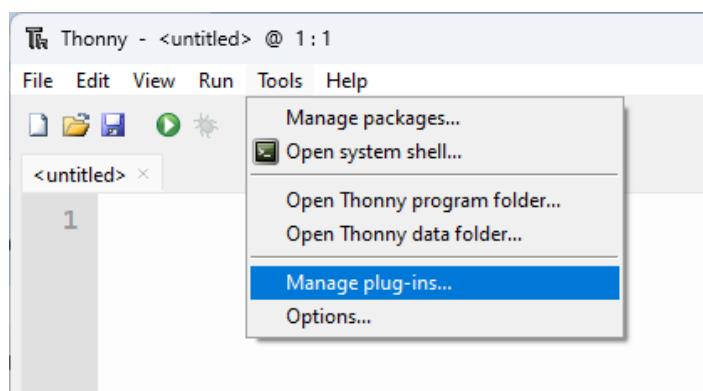
[https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT\\_Binary\\_Lists/esp\\_at\\_binaries.html](https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT_Binary_Lists/esp_at_binaries.html)

The screenshot shows the "ESP-AT User Guide" website. On the left sidebar, there are dropdown menus for "ESP32" and "master (latest)". Below them is a search bar and a "Get Started" button. Under "AT Binary Lists", there is a section for "ESP32 AT Released Firmware" which lists several series: ESP32-WROOM-32 Series, ESP32-MINI-1 Series, ESP32-WROVER-32 Series, ESP32-PICO Series, and ESP32-SOLO Series. There is also a link to "Subscribe to AT Releases" and "Brief Introduction to AT Firmware". The main content area is titled "Released Firmware" and contains a note in Chinese. It states that it is recommended to use the latest version of firmware. Currently, Espressif releases AT firmware for the following ESP32 series of modules. A "Note" section provides instructions for generating new firmware if no released firmware is available for a specific module. It lists four options: Modify UART Configuration, Modify Wi-Fi Configuration, Modify Certificate and Key Configuration, and Modify GATT Configuration. Below this is a section titled "ESP32-WROOM-32 Series" which lists three firmware versions: v3.4.0.0 ESP32-WROOM-32-AT-V3.4.0.0.zip (Recommended), v3.2.0.0 ESP32-WROOM-32-AT-V3.2.0.0.zip, and v2.4.0.0 ESP32-WROOM-32-AT-V2.4.0.0.zip. The first item is circled in red.

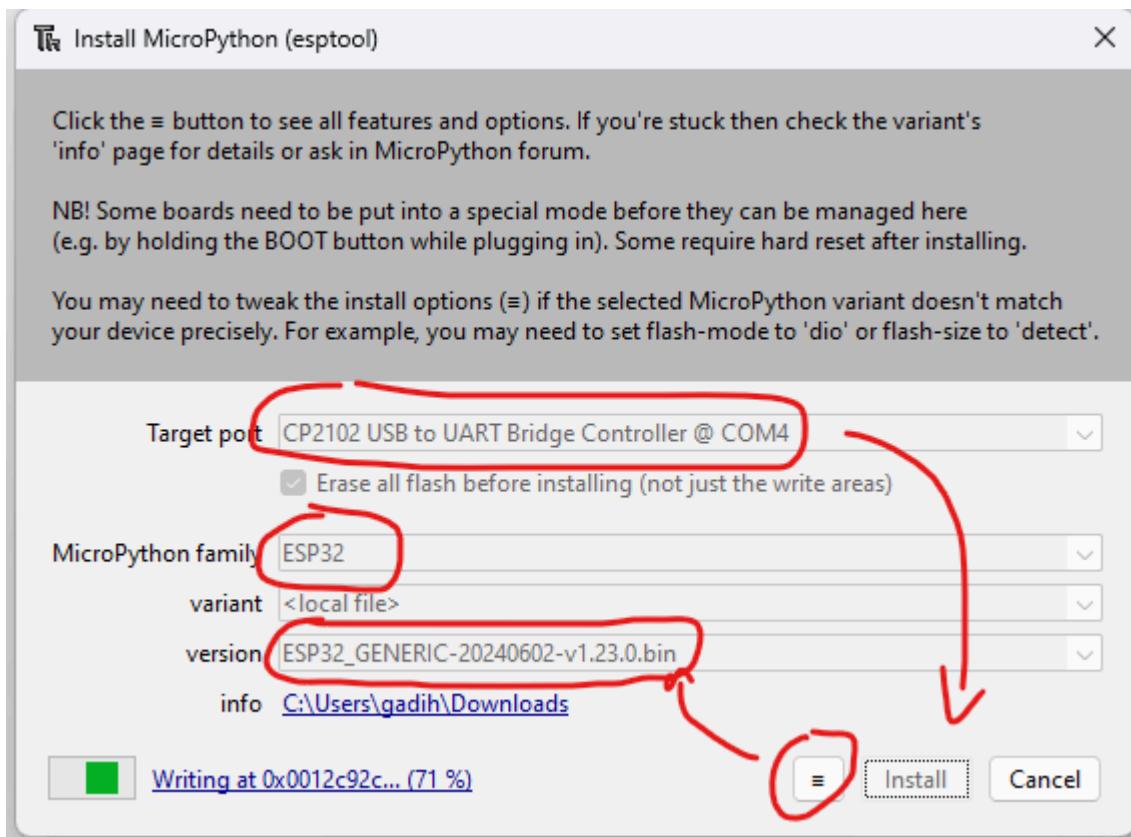
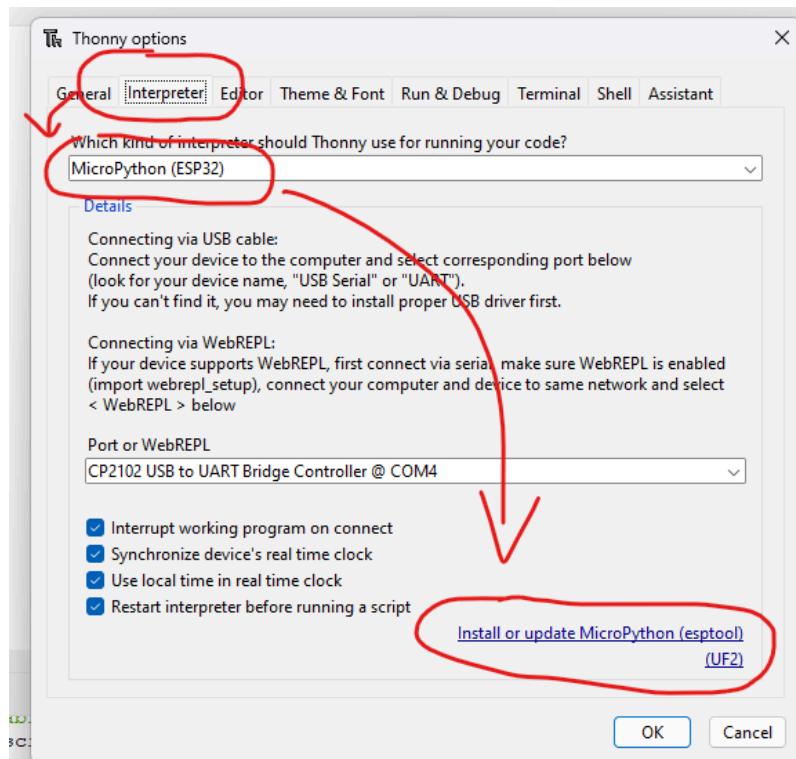
תחת התפריט Tools → Options → נחליף את שפת התוכנות מ- Python ל- MicroPython כה:



בutor סביבת העבודה נתקין את התוסף esptool.py המאפשר לנו לזרוב את הקושחה בברא:



נחזיר לתפריט Tools-> Options על הבקר: תחת לשונית את ה- MicroPythonInterpreter נתקין את ה- Tools-> Options על הבקר:



The screenshot shows the Thonny IDE interface. At the top, it says "Thonny - <untitled> @ 1:1". Below the menu bar is a toolbar with icons for file operations, run, stop, and other tools. The main area has a tab labeled "<untitled>". In the bottom left, there's a "Shell" tab. The shell window contains the following text:

```

Process ended with exit code None.

MicroPython v1.23.0 on 2024-06-02; Generic ESP32 module with ESP32
Type "help()" for more information.

MicroPython v1.23.0 on 2024-06-02; Generic ESP32 module with ESP32
Type "help()" for more information.

>>> |

```

A red circle highlights the second "MicroPython" line, and a red checkmark is placed next to it. At the bottom right of the shell window, it says "MicroPython (ESP32) • CP2102 USB to UART Bridge Controller @ COM4".

זהו... סימנו להתקין גם את סביבת העבודה וגם את הקושחה המאפשרת תכונות בסביבת MicroPython על גבי בקר!!!ESP32

#### **אפשרות ב': התקנת esptool.py על בקר ESP32 תוך שימוש ב- esptool.py**

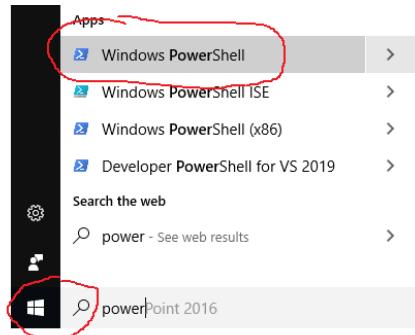
##### **התקנת esptool.py**

הכללי esptool.py מאפשר לנו לצרוב את מערכת הפעלה על הבקר.

מקור:

<https://github.com/espressif/esptool>

לשם ביצוע המשימה נפתח את התוכנה cmd או Windows PowerShell



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\gadih> pip --version
pip 23.1.2 from C:\Users\gadih\AppData\Local\Programs\Python\Python311\Lib\site-packages\p
ip (python 3.11)
PS C:\Users\gadih> \|

```

נעוץ ב- **דיק להתקין את התוכנה אך לפני נבדוק שה- דיק מותקן על ידי היקלחת ההוראה הבא**

```
pip --version
```

**דוגמיה לפולט תקין:**

```

PS C:\Users\gadih> pip --version
pip 23.1.2 from C:\Users\gadih\AppData\Local\Programs\Python\Python311\Lib\site-packages\p
ip (python 3.11)

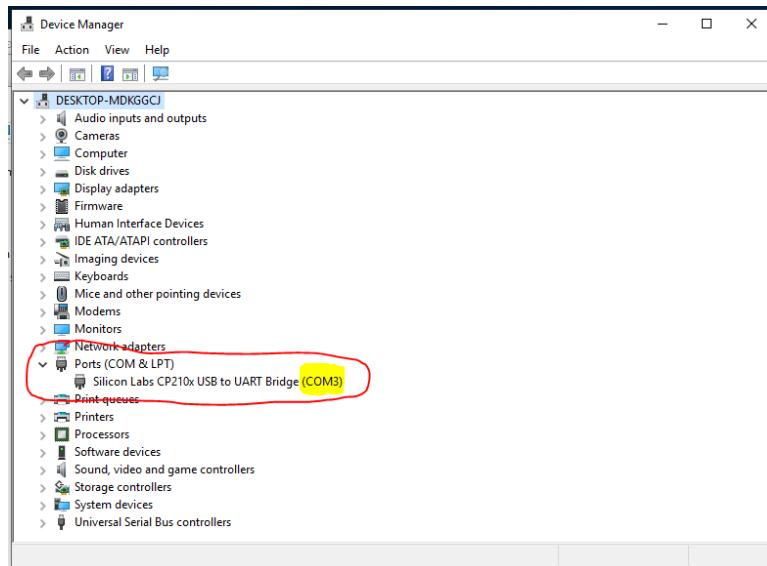
```

נעוץ ב- **דיק CDI להתקין את esptool**

```
> python -m pip install -U pip

> python -m pip install -U esptool
```

לאחר התקינה נחבר את הבקר ESP32 למחשב PC המריץ Windows 11 ונבדוק לאיזה com (מפתח מחשב) התחבר הבקר:



מערכת הפעלה איתרה את הבקר ב- COM3.

**מקור:**

[https://micropython.org/download/ESP32\\_GENERIC/](https://micropython.org/download/ESP32_GENERIC/)

בוצע אתחול של הבקר על ידי הוראה הבא

```
esptool --chip esp32 --port com3 erase_flash
```

נקבל לבקרה את הפלט הבא:

```
esptool.py v4.7.0
Serial port com4
Connecting...
Failed to get PID of a device on com4, using standard reset sequence.
.....
Chip is ESP32-D0WDQ6 (revision v1.0)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 24:0a:c4:c5:f4:10
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 8.6s
Hard resetting via RTS pin...
PS C:\Users\gadih>
```

\***שים לב** לכך שצריך להוחז על לחץ BOOT בבקר כאשר המחשב מציג על המסך פלט של נקודות זו אחריו זו.

כדי לזרוב בברker את מערכת הפעלה נוריד תחילה למחשב שלנו את הקובץ הבא:

[https://micropython.org/download/ESP32\\_GENERIC/](https://micropython.org/download/ESP32_GENERIC/)

## Firmware

### Releases

- v1.23.0 (2024-06-02) .bin / [.app-bin] / [.elf] / [.map] / [Release notes] (latest)
- v1.22.2 (2024-02-22) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
- v1.22.1 (2024-01-05) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
- v1.22.0 (2023-12-27) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
- v1.21.0 (2023-10-05) .bin / [.app-bin] / [.elf] / [.map] / [Release notes]
- v1.20.0 (2023-04-26) .bin / [.elf] / [.map] / [Release notes]
- v1.19.1 (2022-06-18) .bin / [.elf] / [.map] / [Release notes]
- v1.18 (2022-01-17) .bin / [.elf] / [.map] / [Release notes]
- v1.17 (2021-09-02) .bin / [.elf] / [.map] / [Release notes]
- v1.16 (2021-06-23) .bin / [.elf] / [.map] / [Release notes]
- v1.15 (2021-04-18) .bin / [.elf] / [.map] / [Release notes]
- v1.14 (2021-02-02) .bin / [.elf] / [.map] / [Release notes]
- v1.13 (2020-09-02) .bin / [.elf] / [.map] / [Release notes]
- v1.12 (2019-12-20) .bin / [.elf] / [.map] / [Release notes]

קחו בחשבון שיש למקם את הקובץ שהורד מהאתר בתיק'יה שבה ממוקה חלון ה - Windows PowerShell  
במחשב שלך הורדתי את הקובץ לתיק'יה:

C:\Users\gadi>

ניתן לבדוק זאת על ידי הוראה זו כר:

```

PS C:\Users\gadih>
PS C:\Users\gadih> cd .\Downloads\
PS C:\Users\gadih\Downloads> ls ESP*
Red arrow points from the command 'ls ESP*' to the output table.

Directory: C:\Users\gadih\Downloads

Mode                LastWriteTime         Length Name
----                -----        -----
-a----       6/28/2024   9:38 AM      26404705 ESP32-WROOM-32-AT-V3.4.0.0 (1).zip
-a----       6/14/2024  10:57 AM      26404705 ESP32-WROOM-32-AT-V3.4.0.0.zip
-a----       6/29/2024   6:10 AM      1734240  ESP32_GENERIC-20240602-v1.23.0 (1).bin
-a----       6/18/2024  6:20 PM      1734240  ESP32_GENERIC-20240602-v1.23.0.bin

```

לביצוע הזריבה נשתמש בהוראה הבאה:

```

esptool --chip esp32 --port com3 --baud 460800 write_flash -z 0x1000
.\ESP32_GENERIC-20240602-v1.23.0.bin

```

נקבל לבקשתו את הפלט הבא:

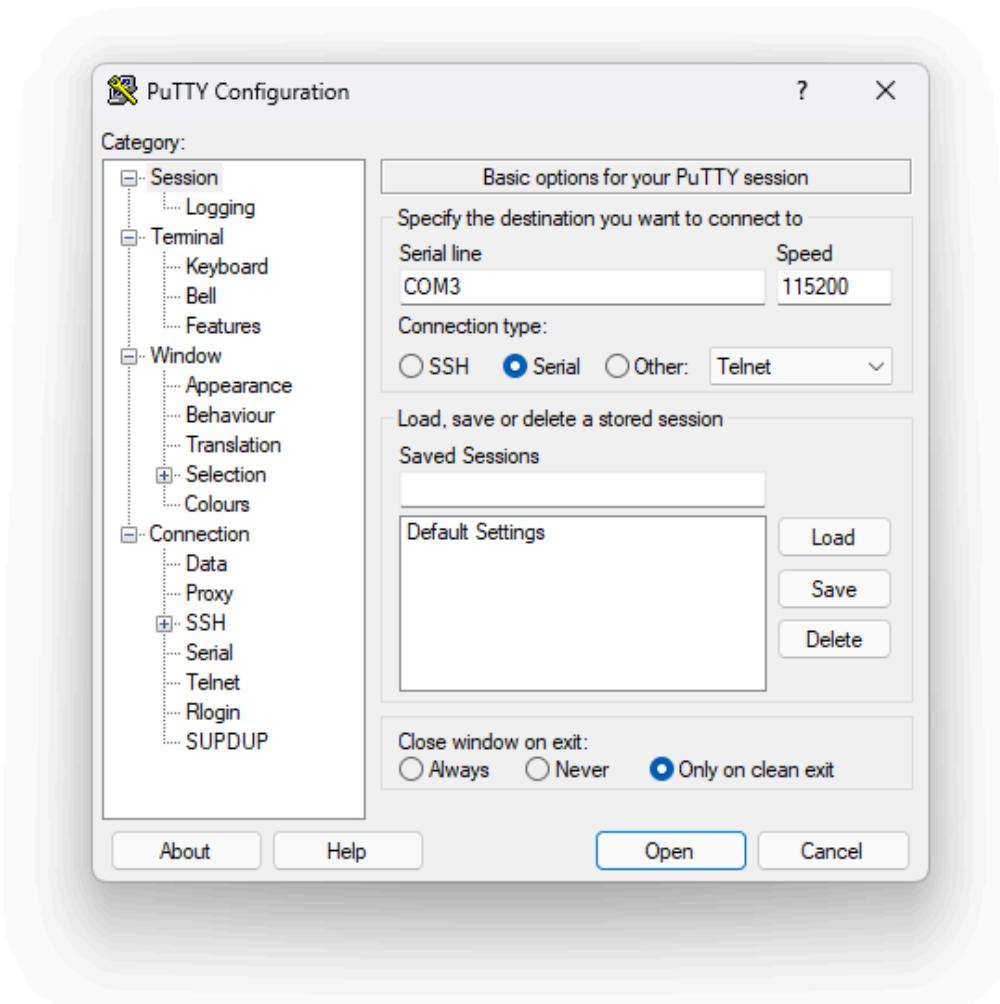
```

esptool.py v4.7.0
Serial port com4
Connecting...
Failed to get PID of a device on com4, using standard reset sequence.
.....
Chip is ESP32-D0WDQ6 (revision v1.0)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: 24:0a:c4:c5:f4:10
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x001a8fff...
Compressed 1734240 bytes to 1142447...
Wrote 1734240 bytes (1142447 compressed) at 0x00001000 in 26.5 seconds (effective 524.2 kbit/s)...
Hash of data verified.

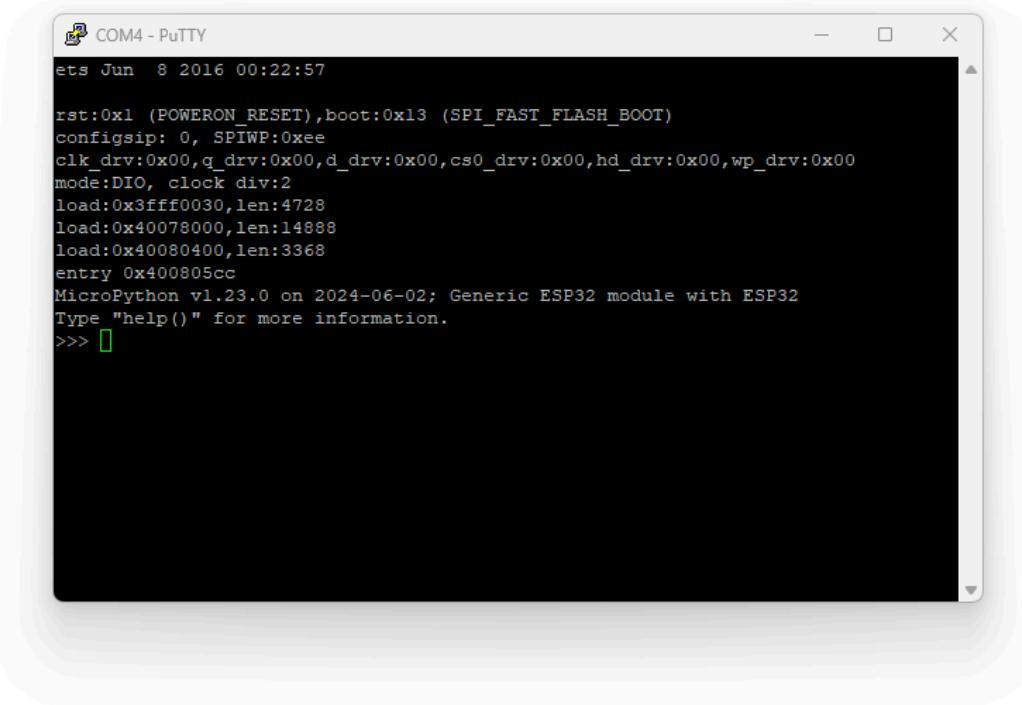
```

כדי לבדוק שהקוד נצרב בהצלחה ניתן להיתחבר לבקר דרך מפתח המחשב COM תוך שימוש בתוכנה כמו putty

[Download PuTTY: latest release \(0.81\) \(greenend.org.uk\)](https://www.greenend.org.uk/)



בהתחרות מוצלחת נקבל את הפלט הבא:



מכאן אנו יכולים לכתוב קוד ראשון ב-MicroPython לדוגמה:

```
>>> str1="Hello"
>>> str2="World"
>>> print(str1,str2)
Hello World
>>>
```

בשלב זה ניתן גם לכתוב קוד בסיס להדלקה וכיובי נורות לד. לדוגמה:

```
>>> from machine import Pin
>>> led=Pin(2,Pin.OUT)
>>> led.value(1)
>>> led.value(0)
>>>
```

## משימה 2 - כתיבת קוד MicroPython בסביבת הפיתוח Visual Studio Code

מקור:

<https://docs.pycom.io/gettingstarted/software/vscode/>

<https://www.youtube.com/watch?v=YOeV14SESIs>

אם אתם מתקננים מנוסים שמכירים את סביבת הפיתוח Visual Studio, פועלות זו מיועדת לכם. אחרת המשיכו לכתוב תוכנה תוך שימוש בסביבת הפיתוח Uhoh! כפי שלמדנו בפעולות הראשונה.

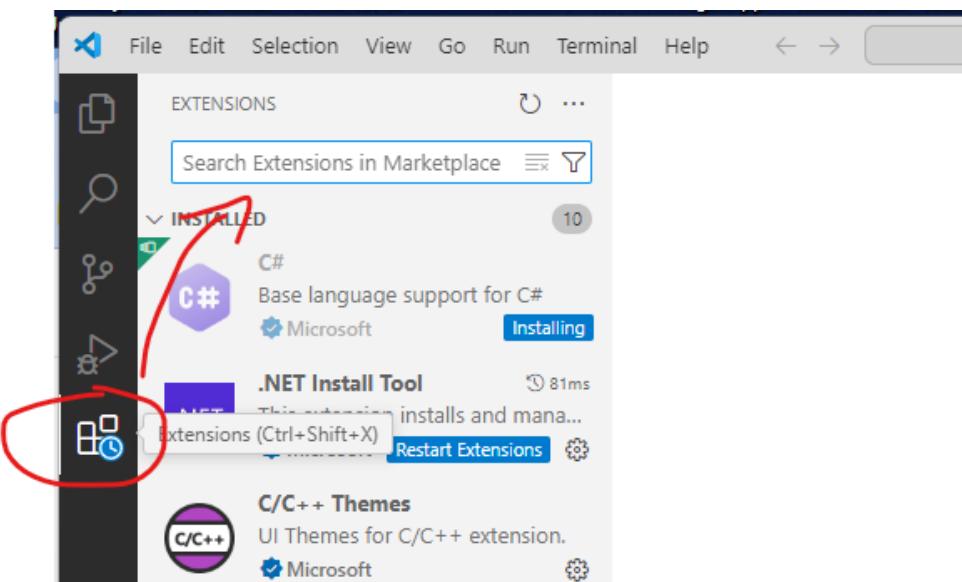
לכטוב Visual Studio Code הוא עורך קוד חינמי, קל משקל הנitin להרחבה לבניית אפליקציות ולתיקון בעיות. הוא פועל על מערכות הפעלה Windows, macOS ו-Linux. סביה זו מאפשרת להתקין לה תוספים המאפשרים לה גמישות גדולה בכל הקשור לשפות התכנות והפלטפורמות שהיא תומכת בהם.

בפעולות זו נתקין את התוסף PyMakr המאפשר לנו לכטוב קוד ב-MicroPython לבקרי ESP32.

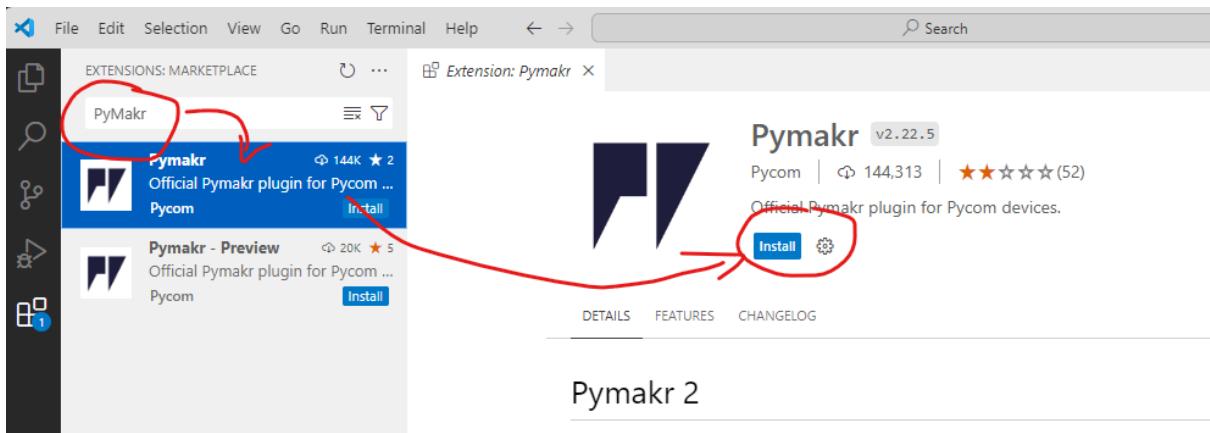
בסוף זה דרוש התקינה של SJNode על המחשב המקומי שעליו אתם עובדים. ניתן להוריד את התוסף מהאתר:

<https://nodejs.org/>

נפתח את סביבת הפיתוח Visual Studio Code ונלחץ על **תוספים** (Extensions)

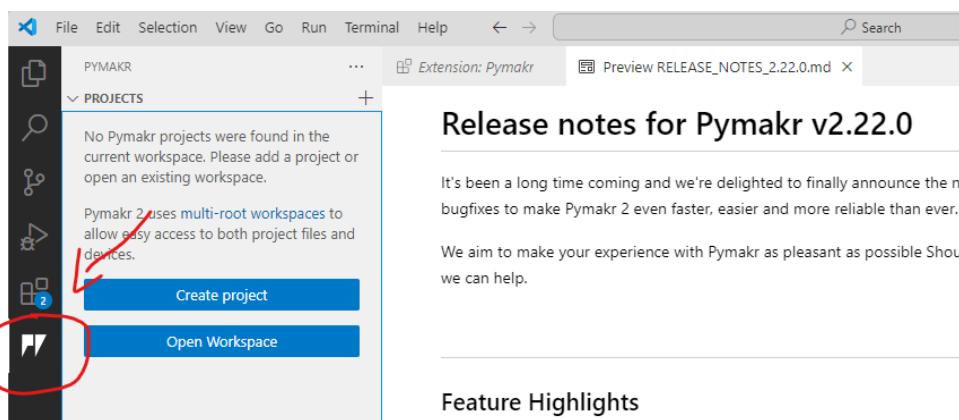


נחפש את התוסף :PyMakr



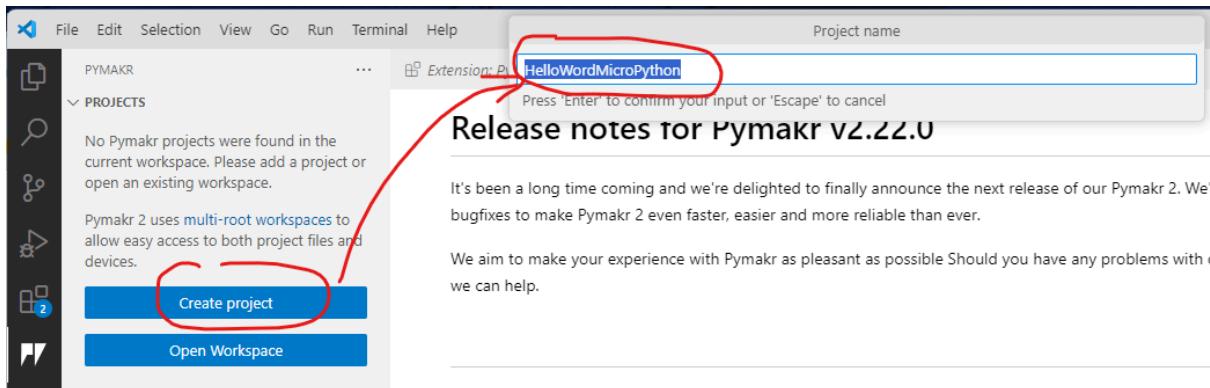
## Pymakr 2

לאחר ההתקנה נקלט לחץ חדש בסביבת העבודה:

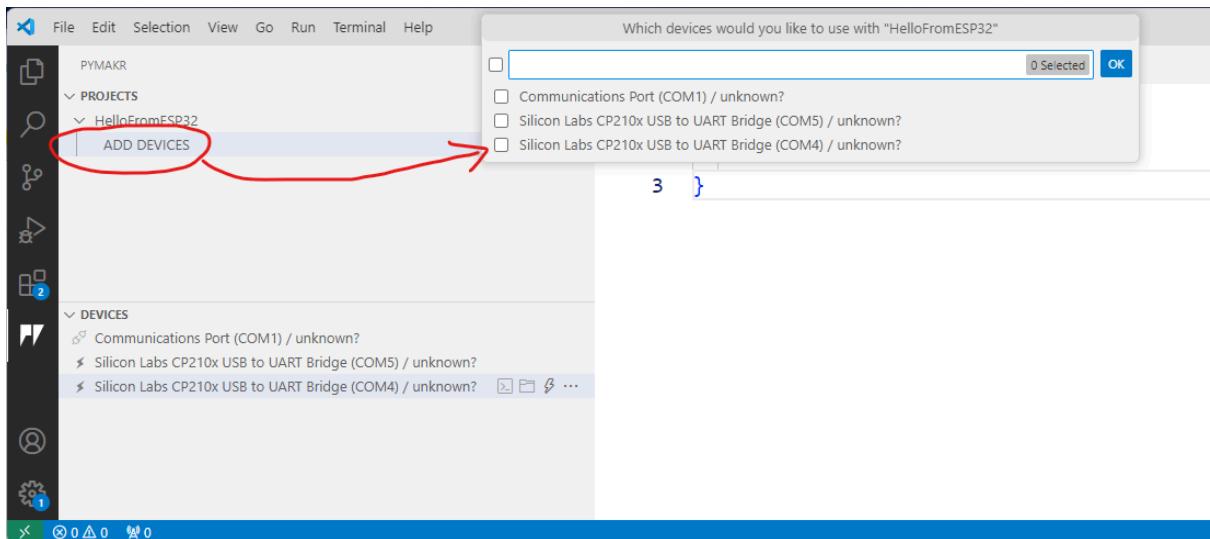


### Feature Highlights

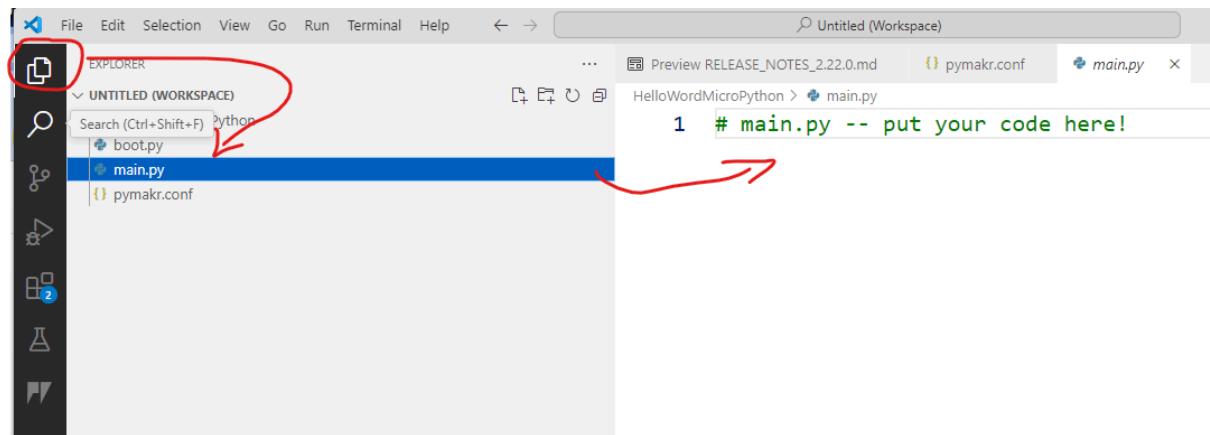
:Create project נלחץ על הלחץ



נקבל את החלון הבא וنبחר בו את הרכיב שアイתו נעובד:



לאחר בחירת הרכיב נעבור לחלון הפROYIKט:

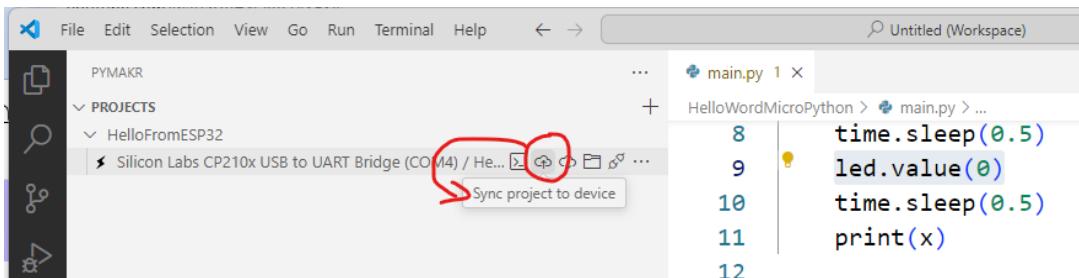


נפתח את קובץ הקוד `main.py` ונכתב בו את הקוד הבא:

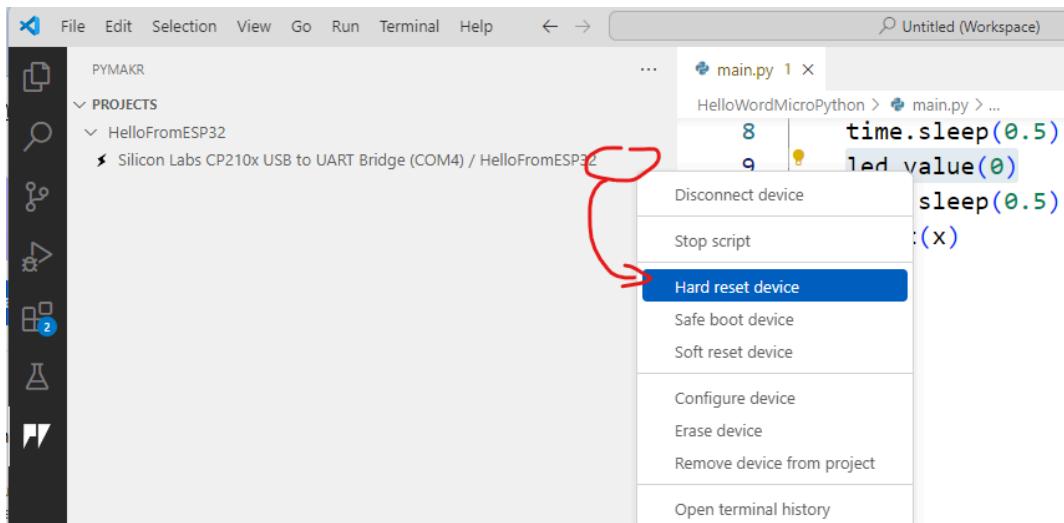
```
import time
from machine import Pin
led=Pin(32,Pin.OUT)

for x in range(10):
    led.value(1)
    time.sleep(0.5)
    led.value(0)
    time.sleep(0.5)
print(x)
```

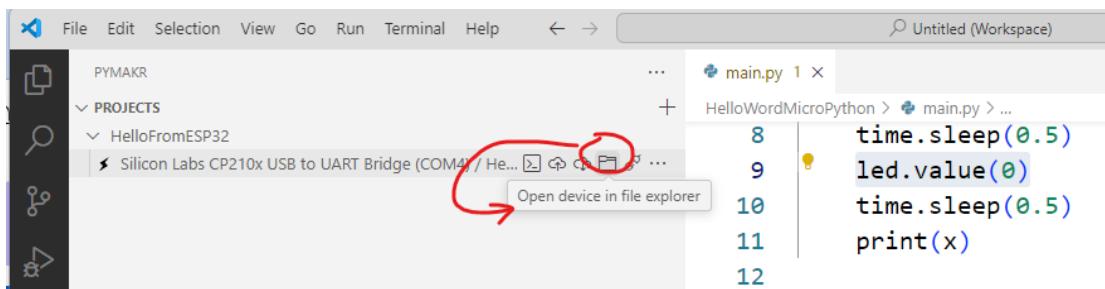
כדי להעביר את התוכנה שכתבנו לבקר נלחץ על הלץ הבא:



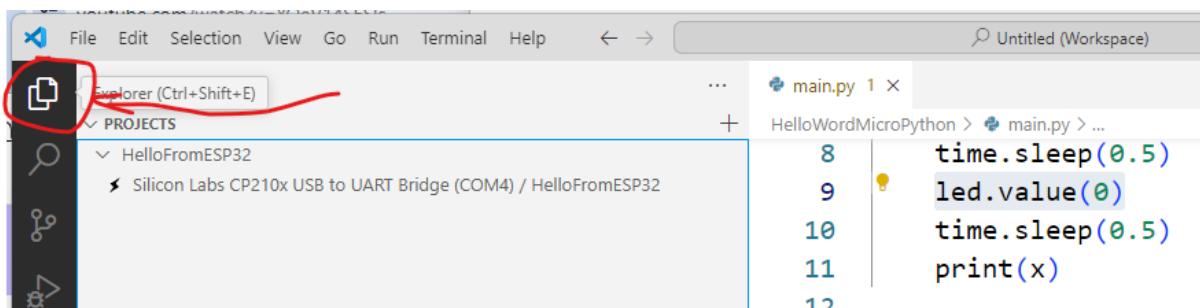
כדי להריץ את הקוד בברker נלחץ על:



ניתן לתקן ולעורך קוד ישירות על גבי הAKER. כדי לעשות זאת יש לפתח את הפרויקט ששמור בAKER בחלון ה-Explorer על ידי לחיצה על "Open device in file explorer":



ולאחר מכן נלחץ על לחץ על ה-Explorer:



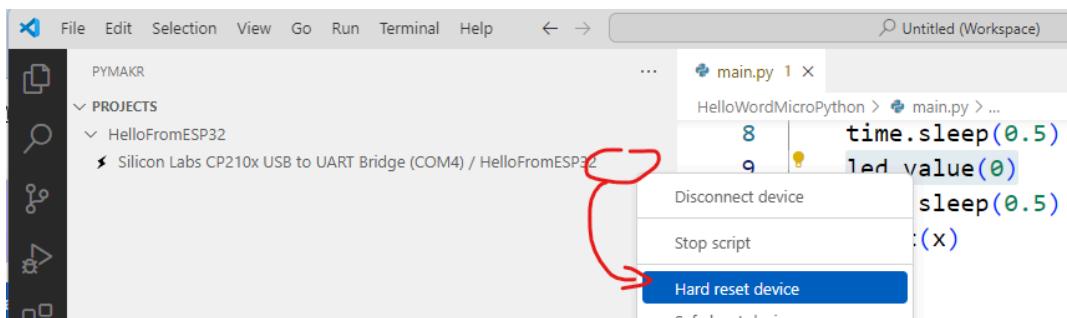
```

# main.py -- put your code here!
import time
from machine import Pin
led=Pin(32,Pin.OUT)

for x in range(10):
    led.value(1)
    time.sleep(2)
    led.value(0)
    time.sleep(1)
    print(x)

```

בשלב זה ניתן לתקן ולערוך את הקוד שבבקיר וצחריץ אותו על ידי:



## משימה 3 - כתיבת תוכנית ראשונה לביצוע פלט בשפה Python

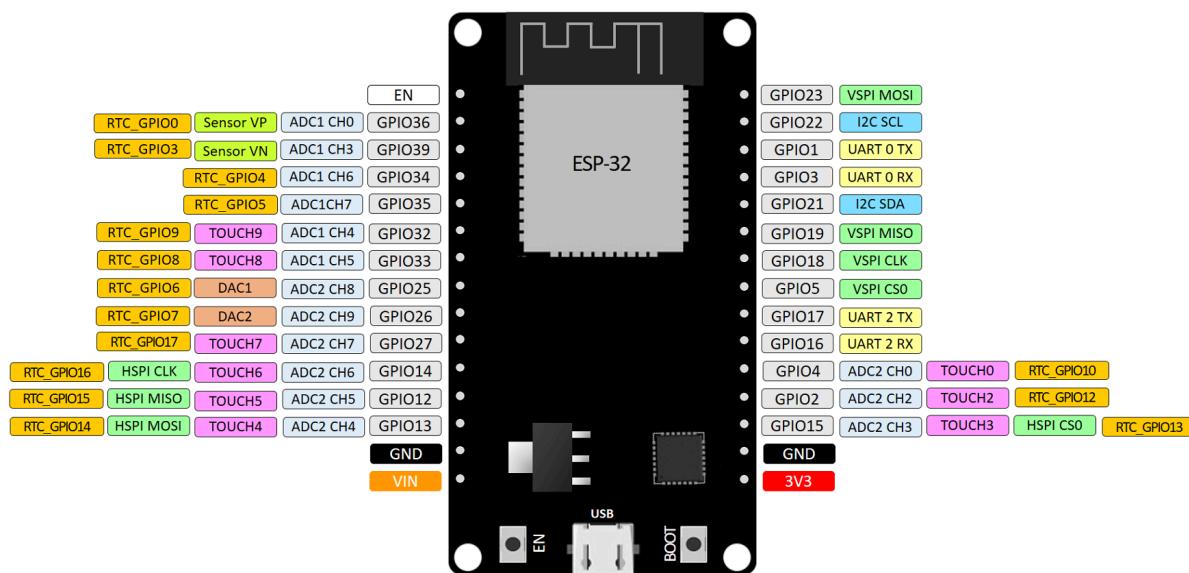
מקור:

<https://myhomethings.eu/en/esp32-pinout-which-pin-is-for-what/>

<https://electropeak.com/learn/full-guide-to-esp32-pinout-reference-what-gpio-pins-should-we-use/>

בניסוי זה נתרgal כתיבת תוכנית ראשונה בפייטון העשוה שימוש בפעולות פלט למפתח הבקר.

נחבר למחשב את הבקר ESP32 דרך מפתח ה- USB



\*קיים בשוק מוצר ESP32 שבמהלכו הרכיב ממופים באופן שונה מהמתואר כאן.

מהאויר ניתן ללמוד שכרטיס הפעיתו המתואר מספק לנו גישה ל- 25 דקקי GPIO. כמו כן ניתן לראות שככל אחד מהדקקי ה- GPIO מספק מספר יישומים נוספים כמו מבואות אנלוגיים, הדקי תקשורת מסווגים שונים ומוצאים אנלוגיים.

שליטה בהדקקי ה- GPIO של בקר ה- ESP32 נעשית על ידי שימוש במחלקה Pin המהווה חלק מהמספרייה machine.

נדרש להגדיר עצם מהמספרייה Pin עבור כל הדק GPIO שנרצה להפעיל.

הקוד הבא מגדים 2 עצמים led1 ו-led2 המהווים מופעים של המחלקה Pin מופעים אלה ממופים להדקים 25 ו- 32 של הבקר ומגדירים כמוצאים.

```
from machine import Pin

led2=Pin(25,Pin.OUT)
```

ניתן להגדיר כבר בשלב האתחול של העמם את מצבו הלוגי. לדוגמה:

```
pinLed = Pin(32, mode=Pin.OUT, value=1) # 3.3V on output
time.sleep(2)
```

```
pinLed = Pin(32, mode=Pin.OUT, value=0) # 0V on output
```

המחלקה Pin מספקת לנו מספר פעולות לשינוי מצבו הלוגי של הדק. נדגים זאת:

```
LedPin2 = Pin(2, mode=Pin.OUT)

LedPin2.on()      # Set 3.3V at the output (high logic state)
LedPin2.high()    # Set 3.3V at the output (high logic state)
LedPin2.value(1)   # Set 3.3V at the output (high logic state)

time.sleep(2)

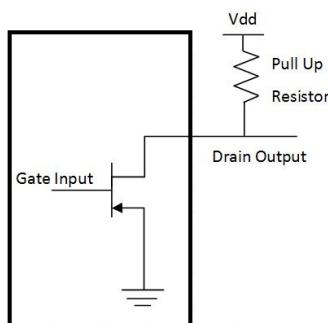
LedPin2.off()     # Set 0V at the output (low logic state)
LedPin2.low()     # Set 0V at the output (low logic state)
LedPin2.value(0)   # Set 0V at the output (low logic state)
```

### חיבור התקני קלט להדק - GPIO

ראינו שאשר אנו רוצים לעבוד עם אחד מהדקי פלט של רכיב, אנו נדרשים לאתחל עצם מהמחלקה Pin. באופן דומה כאשר אנו רוצים להגדיר הדק קלט נכתב את הקוד הבא:

```
from machine import Pin
led=Pin(15,Pin.IN)
```

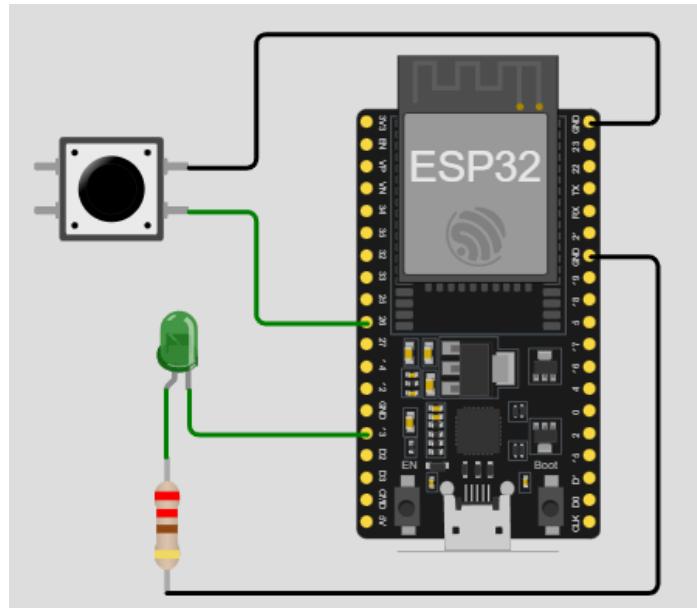
ניתן לאתחל את העצם המיצג את ההדק בדרכים נוספות כמו הגדרת ההדק כ- open-drain output (מצב שבן נקלט עבר 1 לוגי הדק מוצא מקוצר לאדמה וב- 0 לוגי נקלט נתוך).



כמו כן ניתן לאתחל את ההדק באמצעות פתרים נוספים כמו: Pin.PULL\_DOWN או Pin.PULL\_UP פרטם נוספים ניתן בקישור הבא:

<http://docs.micropython.org/en/v1.11/library/machine.Pin.html#machine-pin>

להלן שימוש בסיסי להדלקת נורת LED על ידי מפסוק:



לහן קוד התוכנית:

```
from machine import Pin

pin_button = Pin(26, mode=Pin.IN, pull=Pin.PULL_UP)
pin_led     = Pin(13, mode=Pin.OUT)

pin_led.on()

while True:
    if pin_button.value() == 1:
        pin_led.on()
    else:
        pin_led.off()
```

תרגום:

כמה פעמים ידליק ה- LED כאשר נורץ את הקוד הבא בברך:

```
from machine import Pin
import time

pin_led = Pin(13, mode=Pin.OUT)
```

```

pin_led.on()
time.sleep(0.5)
pin_led.off()
time.sleep(0.5)

for x in range(1,10,2):
    pin_led.value(1)
    time.sleep(0.5)
    pin_led.value(0)
    time.sleep(0.5)
print(x)

```

**תשובה:**

6 פעמים

**תרגום:**

כתב תוכנית לマイקו בקר ממשפחת EPS32 בשפת MicroPython המבצע את הפעולות הבאות:

1. מגדר את הדק 15 של הבקר בפתח מוצא.
2. נורית LED המתחברת לדק 15 תהבהב 10 שניות בקצב של 100 מייל. שניות (0.1 שניות)
3. בהמשך תדלק הנורית למשך 3 שניות נוספת ואז תכבה.

**תשובה:**

```

import time
from machine import Pin
led=Pin(15,Pin.OUT)

for x in range(100):
    led.value(1)
    time.sleep(0.1)
    led.value(0)
    time.sleep(0.1)

    led.value(1)
    time.sleep(3)
    led.value(0)

```

### שילוב השהוות בתוכנה.

ניתן לשלב הוראות השהיה בשלוש טווחי זמן שונים: שניות, אלפיות השניה, מיליוןיות השניה להלן דוגמה:

```
import time

time.sleep(1)          # sleep for 1 second
time.sleep_ms(500)     # sleep for 500 milliseconds
time.sleep_us(10)      # sleep for 10 microseconds
start = time.ticks_ms() # get millisecond counter
delta = time.ticks_diff(time.ticks_ms(), start) # compute time difference
```

## משימה 4 - שימוש ב-Timer פונימי

כמו בכל בקר גם ב- ESP32 קיימ Timer להלן דוגמה לעובדה עם הרכיב.

קישורים:

[https://www.youtube.com/watch?time\\_continue=42&v=Mku1Bq78nXw&feature=emb\\_logo](https://www.youtube.com/watch?time_continue=42&v=Mku1Bq78nXw&feature=emb_logo)

<http://docs.micropython.org/en/v1.11/library/machine.Timer.html#machine-timer>

בקוד הבא נגרום לנורית LED המתחוברת להדק 13 של הבקר להבהב בקצב של שנייה

```
from machine import Pin, Timer

led1 = Pin(13,Pin.OUT)

def handleTimerInt(timer):
    led1.value(not led1.value())

myTimer = Timer(0)
myTimer.init(period=1000, mode=Timer.PERIODIC, callback=handleTimerInt)
```

חשוב: שימוש לב שהבהוב נורית ה- LED על ידי ה- Timer אינה גורמת לבקר להיתקע, כלומר ניתן להמשיך להפעיל את הבקר במקביל לפעולות ה- Timer

ננצל את זה שהבקר פוני ונציג על המסך את מספר הפעמים שהייתה 90יקת Timer:

```
from machine import Pin, Timer

interruptCounter = 0
led1 = Pin(13,Pin.OUT)

def handleTimerInt(timer):
    led1.value(not led1.value())
    global interruptCounter
    interruptCounter = interruptCounter+1
    print("Interrupt has occurred: " + str(interruptCounter))

myTimer = Timer(0)
myTimer.init(period=300, mode=Timer.PERIODIC, callback=handleTimerInt)
```

נקבל את הפלט הבא בנוסף לפעולות ההבהוב של נורית ה- LED

```
Interrupt has occurred: 172
Interrupt has occurred: 173
Interrupt has occurred: 174
Interrupt has occurred: 175
Interrupt has occurred: 176
Interrupt has occurred: 177
Interrupt has occurred: 178
```

## משימה 5 - פסיקות חומרה

אפשרות חומרה מאפשרת לבקר לטפל ביעילות בשינויים המתרחשים בחומרה. הטיפול בפסיקה אינו נדרש לבדוק כל הזמן את מצב החומרה אלה כאשר מתגלה שינוי, מופעל אירוע המומש כפונקציה בתוכנה. כאשר מתרחשת הפסיקה, המעבד מפסיק את ביצוע התוכנית הראשית, מבצע פונקציה מוכנה מראש ואז חוזר לתוכנית הראשית.

בבקר ESP32 ניתן להגדיר אפשרות לכל הדקי הבקר פרט להדקים GPIO6 ו-GPIO11.

קישורים:

<https://randomnerdtutorials.com/micropython-interrupts-esp32-esp8266/>

<https://docs.micropython.org/en/latest/library/machine.Pin.html>

<https://techtutorialsx.com/2017/10/08/esp32-micropython-external-interrupts/>

אחד היישומים למנגנון הפסיקות הוא שימוש פסיקה כדי לטפל בלחיצה על לחץ. נדגים זאת על ידי הקוד הבא:

```
from machine import Pin

def on_pressed(timer):
    print('pressed')

# Setup the button input pin with a pull-up resistor.
button = Pin(33, Pin.IN, Pin.PULL_UP)

# Register an interrupt on rising button input.
button.irq(on_pressed, Pin.IRQ_RISING)
```

בקוד זה אנו מגדירים עצם בשם button הקולט לחץ שמחובר לדק 33 של הבקר וכלל נגד PULL\_UP המומש בתוך הרכיב.

ההוראה button.irq מגדירה להדק זה פיסקה שתופעל בעבר מנמוך לגבוה (IRQ). ברגע שהפסיקה תתקבל נזקן את הפעולה on\_pressed לביוץ קוד הפסיקה.

ניתן להשתמש במאפיין trigger כדי לקבוע באיזה שלב תתקיים הפסיקה להלן האפשרויות:

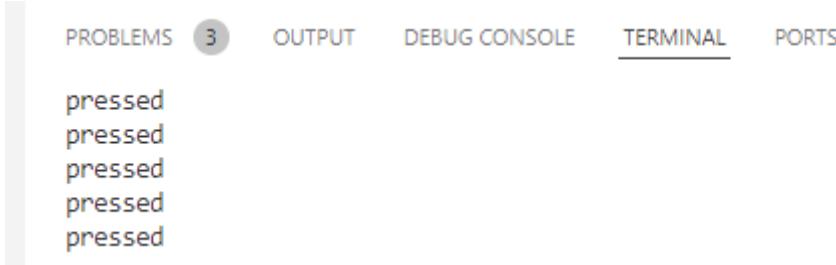
Pin.IRQ\_FALLING - interrupt on falling edge.

Pin.IRQ\_RISING - interrupt on rising edge.

Pin.IRQ\_LOW\_LEVEL - interrupt on low level.

Pin.IRQ\_HIGH\_LEVEL - interrupt on high level.

זהירות!! שימוש לב שבגלא בעית ריטוטים בכל לחיצה על הלחצן נקלט מספר הודעת על המסר. זאת למרות שהלחצנו רק פעם אחת.



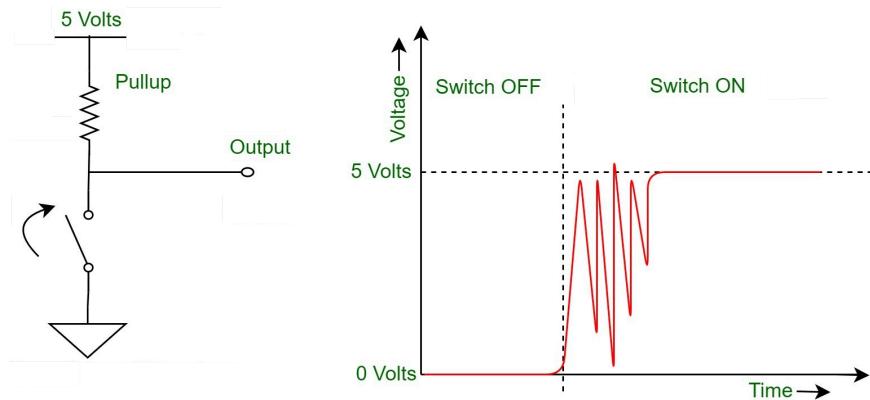
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS

pressed
pressed
pressed
pressed
pressed

```

נדגים תרשימים חשמלי ותרשימים של אות המבוא של הדק הפסיוקה:



זמן הריתוט (switch debounce) נמשך כ- 50ms עד 200ms

להלן דוגמת קוד לתוכנה הממתינה לחיצה על לחצן המחבר לדק 33 של הבקר. ברגע היליצה יופעל טימר למשך 200ms לצורך טיפול בבעית הריטוטים ולאחר מכן תופעל הפולה `on_pressed` שתציג על המסר את ההודעה "pressed"

```

from machine import Pin, Timer

def on_pressed(timer):
    print('pressed')

def debounce(pin):
    # Start or replace a timer for 200ms, and trigger on_pressed.
    timer.init(mode=Timer.ONE_SHOT, period=200, callback=on_pressed)

    # Register a new hardware timer.
    timer = Timer(0)

# Setup the button input pin with a pull-up resistor.

```

```

button = Pin(33, Pin.IN, Pin.PULL_UP)

# Register an interrupt on rising button input.

button.irq(debounce, Pin.IRQ_RISING)

```

ניתן לראות מנגנון פסיקה המפעיל טימר למשך 200ms שבסופה מופעלת הפעולה .on\_pressed

נדגים קוד, לא ממש יעיל, לטיפול במספר לחיצים:

```

from machine import Pin, Timer

def on_pressed1(timer):
    print('PIN 26 pressed')

def on_pressed2(timer):
    print('PIN 25 pressed')

def on_pressed3(timer):
    print('PIN 33 pressed')

def debounce1(pin):
    timer.init(mode=Timer.ONE_SHOT, period=200, callback=on_pressed1)

def debounce2(pin):
    timer.init(mode=Timer.ONE_SHOT, period=200, callback=on_pressed2)

def debounce3(pin):
    timer.init(mode=Timer.ONE_SHOT, period=200, callback=on_pressed3)

timer = Timer(0)

button1 = Pin(26, Pin.IN, Pin.PULL_UP)
button2 = Pin(25, Pin.IN, Pin.PULL_UP)
button3 = Pin(33, Pin.IN, Pin.PULL_UP)
button1.irq(debounce1, Pin.IRQ_RISING)
button2.irq(debounce2, Pin.IRQ_RISING)

```

```
button3.irq(debounce3, Pin.IRQ_RISING)
```

לצורך מימוש קוד ייעיל לטיפול בלחצים המוחברים לפוסיקות חומרה נממש מחלקה בשם Button לטיפול בלחץ כללי תוך שימוש בפוסיקה:

```
import time
from micropython import const
from machine import Pin, Timer

BUTTON_A_PIN = const(26)
BUTTON_B_PIN = const(25)
BUTTON_C_PIN = const(33)

led1=Pin(13,Pin.OUT)
led2=Pin(12,Pin.OUT)
led3=Pin(14,Pin.OUT)
led4=Pin(27,Pin.OUT)

class Button:
    """
        Debounced pin handler
    usage e.g.:
    def button_callback(pin):
        print("Button (%s) changed to: %r" % (pin, pin.value()))
    button_handler = Button(pin=Pin(32, mode=Pin.IN, pull=Pin.PULL_UP),
    callback=button_callback)
    """
    def __init__(self, pin, callback, trigger=Pin.IRQ_FALLING, min_ago=300):
        self.callback = callback
        self.min_ago = min_ago

        self._blocked = False
        self._next_call = time.ticks_ms() + self.min_ago

    pin.irq(trigger=trigger, handler=self.debounce_handler)

    def call_callback(self, pin):
        self.callback(pin)

    def debounce_handler(self, pin):
        if time.ticks_ms() > self._next_call:
```

```

        self._next_call = time.ticks_ms() + self.min_ago
        self.call_callback(pin)

    #else:
    #    print("debounce: %s" % (self._next_call - time.ticks_ms()))

def button_a_callback(pin):
    print("Button A (%s) changed to: %r" % (pin, pin.value()))
    led1.value(1)
    led2.value(1)
    led3.value(1)
    led4.value(1)

def button_b_callback(pin):
    print("Button B (%s) changed to: %r" % (pin, pin.value()))
    led1.value(0)
    led2.value(0)
    led3.value(0)
    led4.value(0)

def button_c_callback(pin):
    print("Button C (%s) changed to: %r" % (pin, pin.value()))

button_a = Button(pin=Pin(BUTTON_A_PIN, mode=Pin.IN, pull=Pin.PULL_UP),
callback=button_a_callback)

button_b = Button(pin=Pin(BUTTON_B_PIN, mode=Pin.IN, pull=Pin.PULL_UP),
callback=button_b_callback)

button_c = Button(pin=Pin(BUTTON_C_PIN, mode=Pin.IN, pull=Pin.PULL_UP),
callback=button_c_callback)

```

מקור:

<https://gist.github.com/jedie/8564e62b0b8349ff9051d7c5a1312ed7>

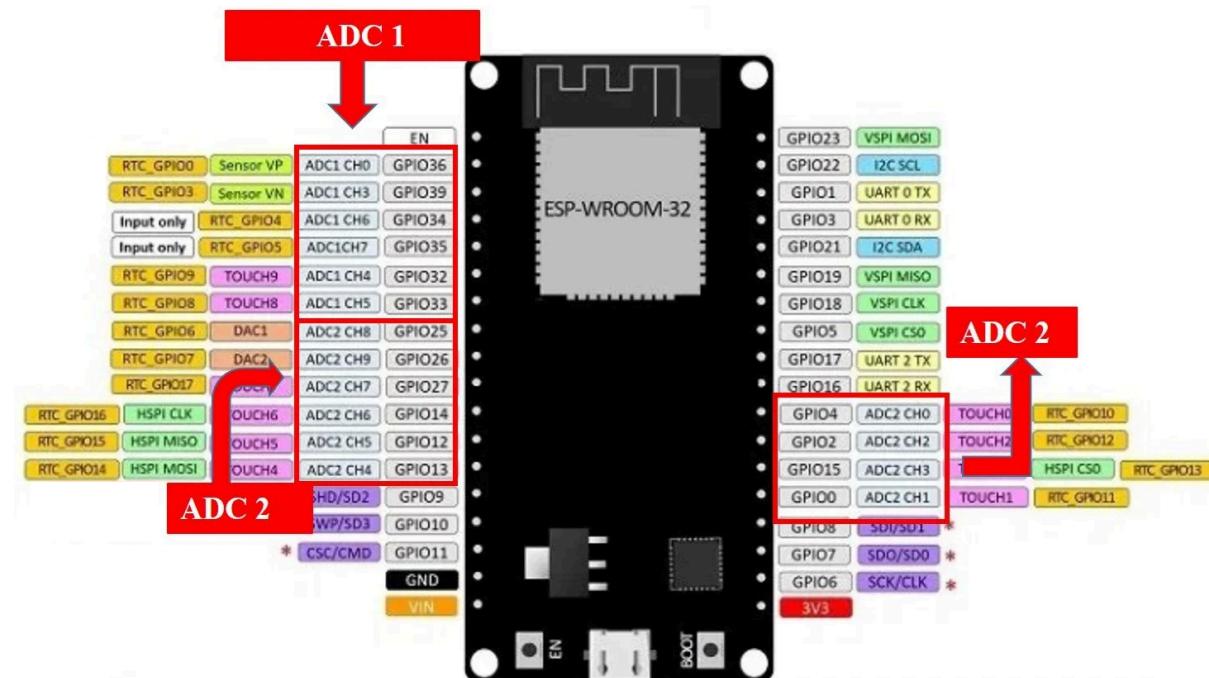
## משימה 6 - קלט אות אולוגי

קישורים:

<https://randomnerdtutorials.com/esp32-esp8266-analog-readings-micropython/>

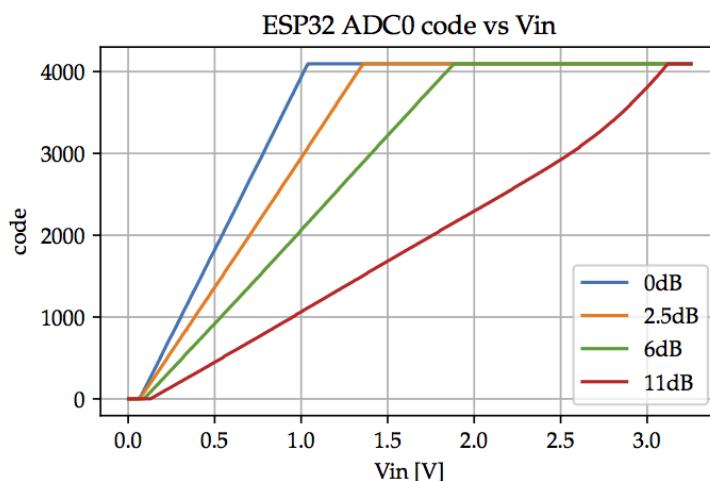
בקר ESP32 כולל שני ממירם ADC. הראשון ADC1 כולל 8 מבואות כאשר רק 6 מהם זמינים לעובודה. השני ADC2 כולל 10 מבואות. להלן רשימת המבואות:

- ADC1\_CH0 :GPIO 36
- ADC1\_CH1 :GPIO 37 (NOT AVAILABLE)
- ADC1\_CH2 :GPIO 38 (NOT AVAILABLE)
- ADC1\_CH3 :GPIO 39
- ADC1\_CH4 :GPIO 32
- ADC1\_CH5 :GPIO 33
- ADC1\_CH6 :GPIO 34
- ADC1\_CH7 :GPIO 35
- ADC2\_CH0 :GPIO 4
- ADC2\_CH1 :GPIO 0
- ADC2\_CH2 :GPIO 2
- ADC2\_CH3 :GPIO 15
- ADC2\_CH4 :GPIO 13
- ADC2\_CH5 :GPIO 12
- ADC2\_CH6 :GPIO 14
- ADC2\_CH7 :GPIO 27
- ADC2\_CH8 :GPIO 25
- ADC2\_CH9 :GPIO 26



ברירת המחדל של הבקר היא המרה של אות אנלוגי לדיגיטלי בטוויה שבין 0 ל- 71 המומר למספר בין 0 ל- 4095 (כולם מודובר ב-12bit).

לבדי ADC של בקר ESP32 אין התנהגות לינארית. סביר להניח שלא תוכל לבדוק בין 0 ל-0.1, או בין 0.1 ל-0.7. לכן צריך לזכור את זה בעת השימוש ב-ADC. ניתן לראות זאת באירוע הבא:



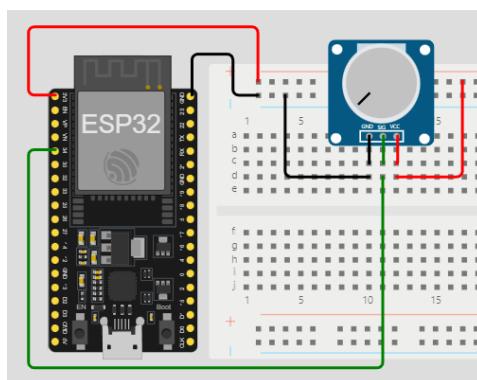
כמו כן ניתן לראות שאפשר לקבוע את טווח ההמרה של הממיר על פי הגדרה שניתן לעשوت בחומרה:

ADC.ATTN\_0DB: 0dB attenuation, gives a maximum input voltage of 1.00v

ADC.ATTN\_2\_5DB: 2.5dB attenuation, gives a maximum input voltage of approximately 1.34V

ADC.ATTN\_6DB: 6dB attenuation, gives a maximum input voltage of approximately 2.00v

ADC.ATTN\_11DB: 11dB attenuation, gives a maximum input voltage of approximately 3.6V



**להלן דוגמה לעובדה עם הממיר:**

```
from machine import Pin, ADC

adc_pin = Pin(34, mode=Pin.IN)

adc = ADC(adc_pin)

adc.atten(ADC.ATTN_11DB)

print(adc.read())
```

ההוראה השנייה בקוד יוצרת עצם מהמחלקה PIN בשם pin\_adc. עצם זה מועבר כפרמטר לפעולה הבונה של המחלקה ADC כדי ליצור את העצם adc שאיתו נעבד כדי לקרוא את הערך האנלוגי.

כמובן ניתן לכתוב אותו הקוד בכתביה מקוצרת כך:

```
from machine import Pin, ADC

adc = ADC(Pin(34)) # Create an ADC object linked to pin 34
adc.atten(ADC.ATTN_11DB)
print(adc.read())
```

דוגמת הקוד הבא מדגימה כיצד להמיר את הערך המתתקבל למתח:

```
from machine import Pin, ADC
import time

adc = ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)

while True:
    val = adc.read()
    val = val * (3.3 / 4095)
    print(round(val, 2), "V")
    time.sleep_ms(1000)
```

נקבל את הפלט הבא:

The screenshot shows the PyMakr IDE interface. On the left is a sidebar with icons for projects, devices, and other tools. The main workspace has tabs for 'main.py \ main.py' and '3\_GPIO 1 x'. Below the tabs is a code editor with the following Python script:

```
1 from machine import Pin, ADC
2 import time
3
4 adc = ADC(Pin(34))
5 adc.atten(ADC.ATTN_11DB)
6 while True:
7     val = adc.read()
8     val = val * (3.3 / 4095)
9     print(round(val, 2), "V")
10    time.sleep_ms(1000)
11
```

Below the code editor is a terminal window showing the output of the script. The output is circled in red, and a red arrow points from the bottom of the terminal window to the circled area. The terminal output is as follows:

PROBLEMS	1
1.02 V	
1.33 V	
1.63 V	
1.28 V	
3.3 V	
3.3 V	
1.59 V	
0.0 V	
0.0 V	
0.0 V	
0.23 V	
0.24 V	
0.23 V	

מהדוגמה ניתן לראות שהרזולוציה של מmir ה-ADC היא 12bit . בקר ESP32 מאפשר את שינוי רזולוציית המmir.

כבריתת מחדל, התוכניתת תפעל C-D ADC של 12 סיביות ונותנת לנו ערכים בין 0-4095 אך אנו יכולים לשנות את הרזולוציה ב-ESP32 בהתאם לצרכים שלנו. ניתן להציג זאת על ידי שימוש בפונקיה width באופן הבא:

```
adc.width(ADC.WIDTH_9BIT)      #For range between 0-511
adc.width(ADC.WIDTH_10BIT)     #For range between 0-1023
adc.width(ADC.WIDTH_11BIT)     #For range between 0-2047
adc.width(ADC.WIDTH_12BIT)     #For range between 0-4095
```

נבחן זאת דרך הדוגמה הבא:

```
from machine import Pin, ADC
import time

adc = ADC(Pin(34))
adc.atten(ADC.ATTN_11DB)
adc.width(ADC.WIDTH_9BIT) #For range between 0-511
while True:
    val = adc.read()
    val = val * (3.3 / 511)
    print(round(val, 2), "V")
    time.sleep_ms(1000)
```

נקבל את הפלט הבא:

```
1  from machine import Pin, ADC
2  import time
3
4  adc = ADC(Pin(34))
5  adc.atten(ADC.ATTN_11DB)
6  adc.width(ADC.WIDTH_9BIT) #For range between 0-511
7
8  while True:
9      val = adc.read()
10     val = val * (3.3 / 511)
11     print(round(val, 2), "V")
12     time.sleep_ms(1000)
13
```

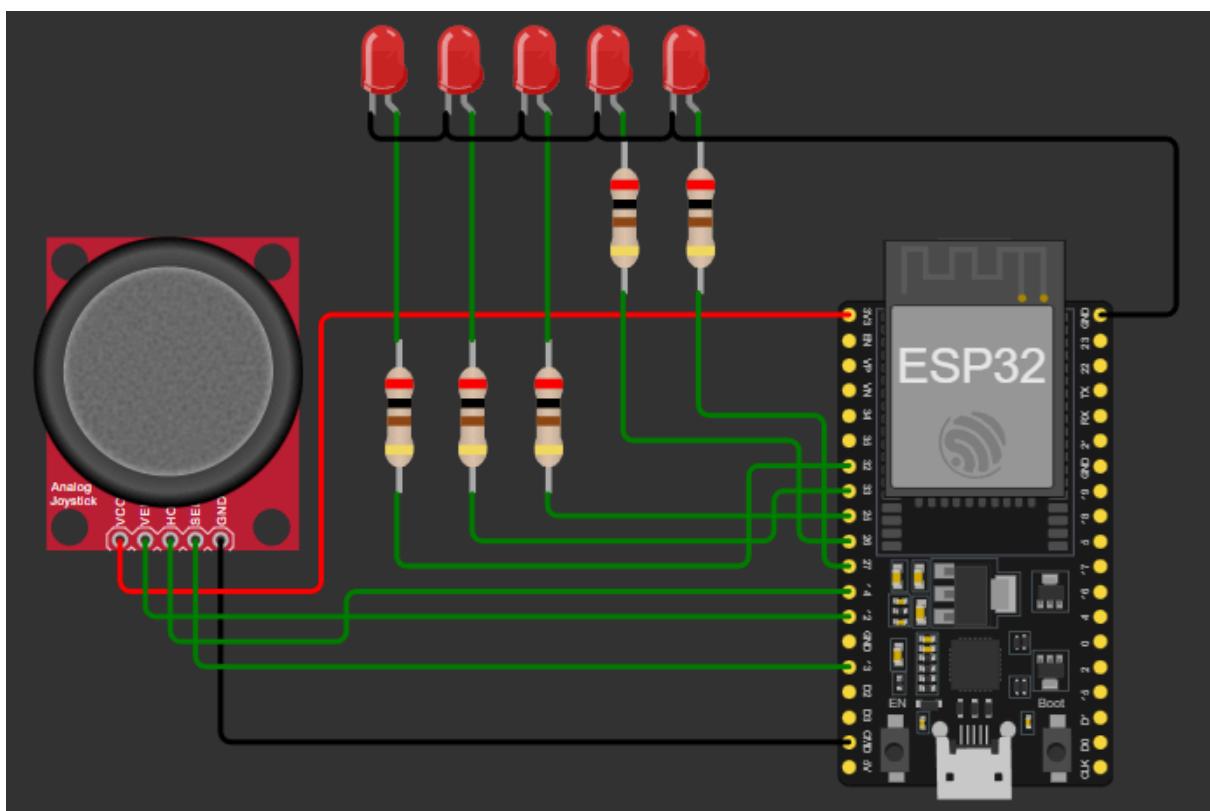
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

2.52 V  
2.03 V  
1.5 V  
1.01 V  
0.14 V  
0.14 V  
0.0 V  
0.0 V

## ישום analog-joystick



להלן חיבורו המugenל



להלן דוגמת קוד שמציצה את הנוריות ימינה ושמאליה בהתאם לתזוזת ה- joystick

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

#joystick setup-----
sw = Pin(13,Pin.IN, pull= Pin.PULL_UP)
x_pin = Pin(12,Pin.IN)
y_pin = Pin(14,Pin.IN)
```

```

x = ADC(x_pin)
y = ADC(y_pin)

x.atten(ADC.ATTN_11DB)
y.atten(ADC.ATTN_11DB)
#joystick setup-----


#leds setup-----
led1 = Pin(27,Pin.OUT)
led2 = Pin(26,Pin.OUT)
led3 = Pin(25,Pin.OUT)
led4 = Pin(33,Pin.OUT)
led5 = Pin(32,Pin.OUT)

led_arr = [led1,led2,led3,led4,led5]
active_led = [0,0,0,0,0]
#leds setup-----


one_location = int

while True :

    for i in range(len(led_arr)): #reset all leds
        led_arr[i].value(0)

    one_location = active_led.index(1) #finds the active led

    if x.read() <= 500: #checks if the joystick moved to the left
        active_led = active_led[1:] + active_led[:1]
    elif x.read() >= 2200: #checks if the joystick moved to the right
        active_led = active_led[4:] + active_led[:4]

    led_arr[one_location].value(1)

```

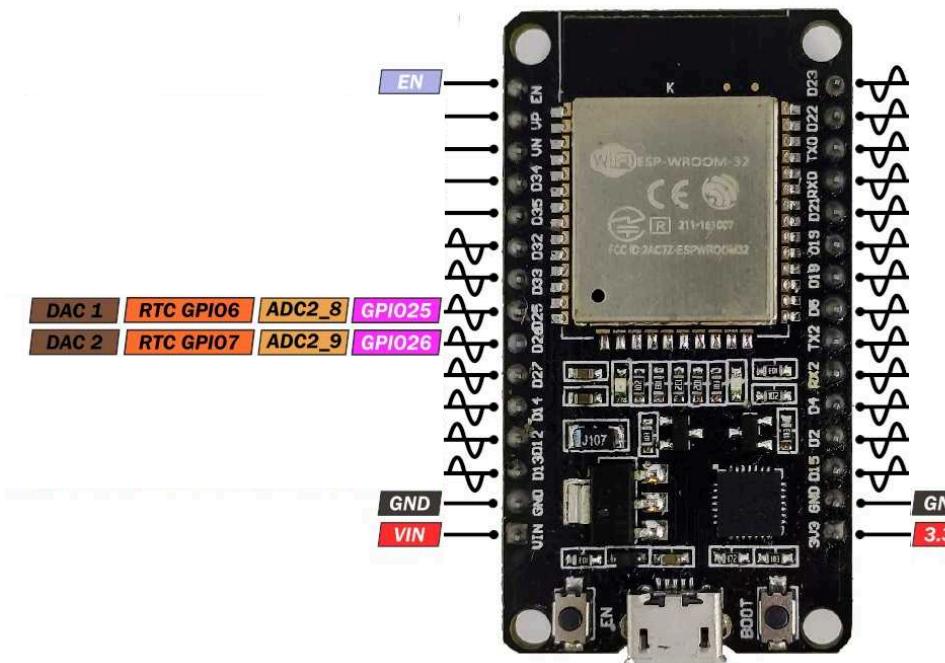
```
sleep(0.2)

# def check_joystick(): #test function to see the joystick values
#     print("x= " ,x.read())
#     print("y= " ,y.read(), "\n")
```

תודה לאריאל הרמן.

## **משימה 7 - ממיר דיגיטלי לאנלוגי DAC**

בקיר ESP32 מספק לנו 2 ממירים דיגיטליים-אלוגיים (D/A) בני 8 כל אחד לצורך המרת ערכיהם דיגיטליים למתחים אלוגיים. מתחים אלה מסופקים דרך הדק' פלט GPIO של ערוץ DAC כמפורט באירור



ה-DAC כולל בן 8 סיביות מבוא כך שהערך הקלט שלו בין 0 ל-255. המתח האנלוגי שווה ערך מ-0V למתוך הייחוס (Vref) כאשר ברירת המחדל היא 3.3V ל-DAC יש רשות נגדים פנימית שימושת ברמת מתח קרובות למתח האספקה כמתוך הייחוס המבוא (Vref). מתוך הייחוס 'Vref' מתקבל מהפין RTC\_VDD3P3 של שבב ESP32.

נדגים קוד בסיסי להפעלת ה- DAC:

```
from machine import DAC, Pin
import time

dac = DAC(Pin(25))

for i in range(10):
    dac.write(0)
    print("0V")
    time.sleep_ms(3000)

    dac.write(128)
    print("1.65V")
    time.sleep_ms(3000)

    dac.write(255)
    print("3.3V")
    time.sleep_ms(3000)
```

ברירת המחדל של ה-DAC הוא 8 סיביות. כדי לשנות את אופן העבודה שלו ל- 12 סיביות נוסיף לפערלה הבונה של המחלקה DAC את המשתנה  $\text{bits} = 12$  נדגים זאת:

```
print("3.3V")
time.sleep_ms(3000)
```

#### יצירת אות סינוס

להלן קוד המדגים יצירת אות סינוס

```
from machine import Pin, DAC
import math

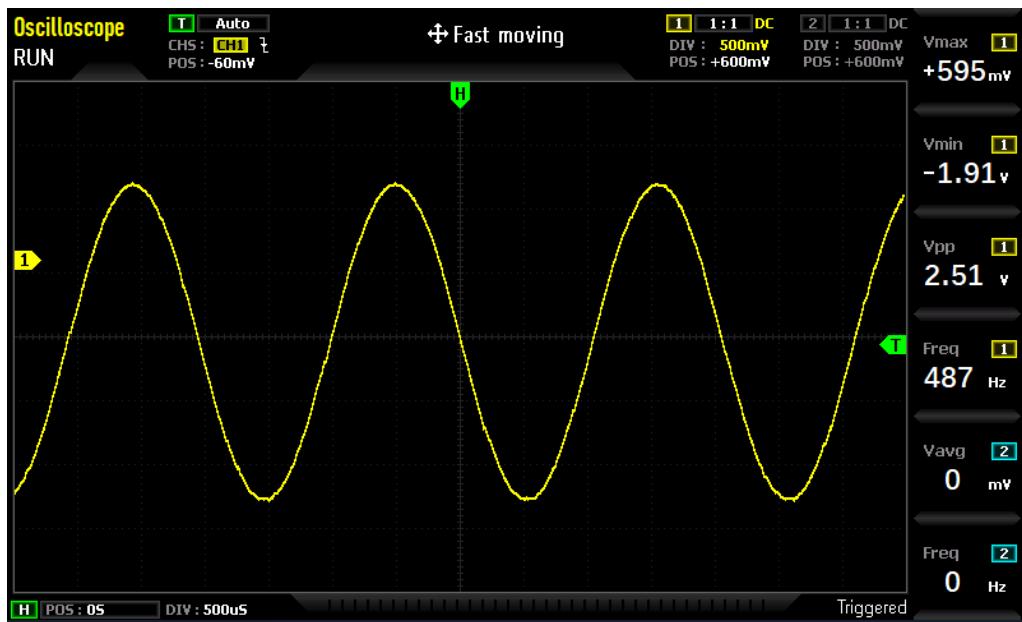
dac = DAC(Pin(25, Pin.OUT))
dac.write(0)

P = 2
N = 200
A = 100
Omega = 2*math.pi/N

sin_table = []
for i in range(N):
    arg = Omega*i
    x = A*math.sin(P*arg) + 127
    sin_table.append(int(x))

index = 0
while True:
    dac.write(sin_table[index])
    index = (index+1) % N
```

#### יצירת אות סינוס



## יצירת שני אוטות יחד

מעשה שימוש בשני ממיריו ה- DAC ליצירת שני אותו באותו תדר ובאותו וואצמה. האחד בתזוזת מופע של 90 מעלות אחד מהשני. נדגים זאת על ידי הקוד הבא:

```
from machine import Pin, DAC
import math

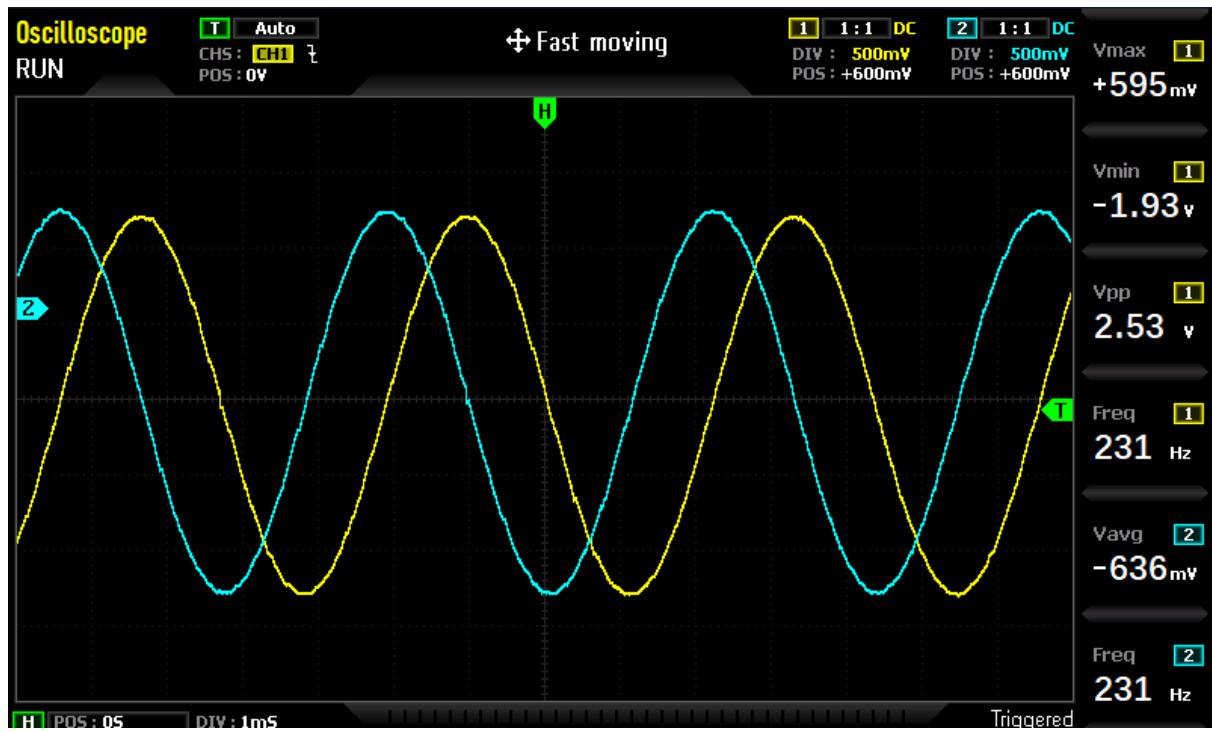
dac1 = DAC(Pin(25, Pin.OUT))
dac1.write(0)
dac2 = DAC(Pin(26, Pin.OUT))
dac2.write(0)

# global variables
P = 2
Q = 2
N = 200
A = 100
Omega = 2*math.pi/N

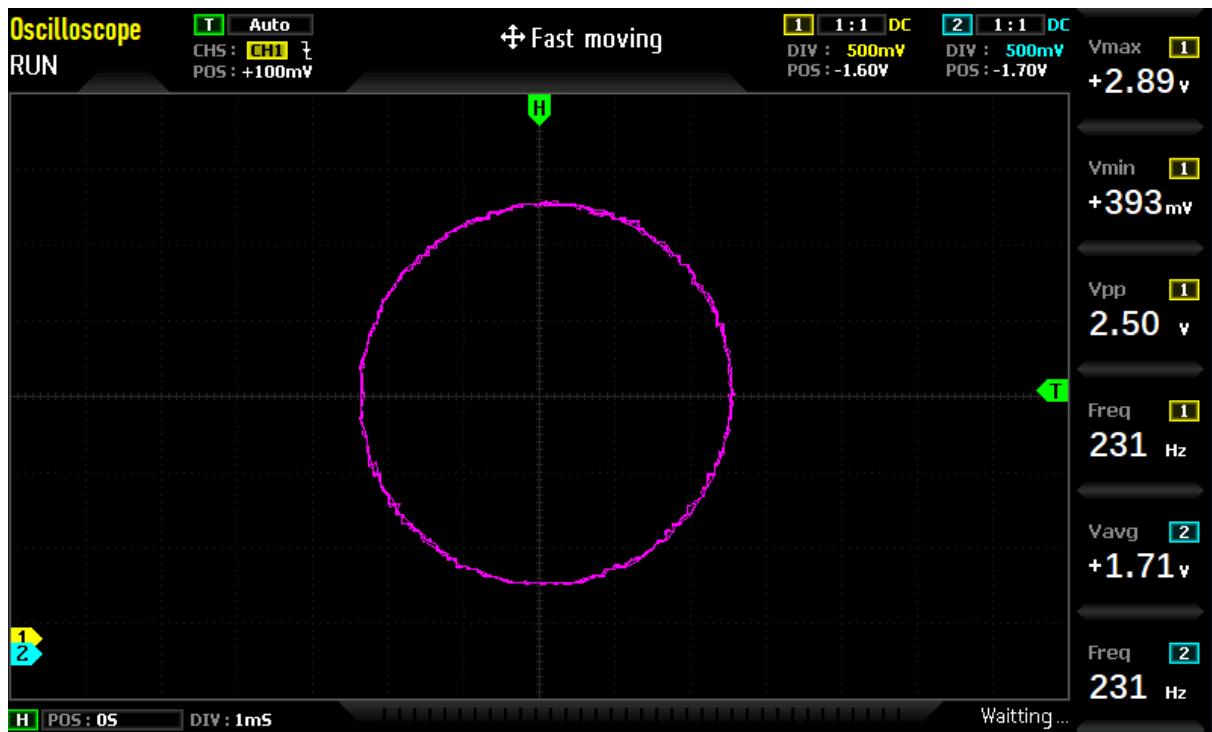
sin_table = []
cos_table = []
for i in range(N):
    arg = Omega*i
    x = A*math.sin(P*arg) + 127
    y = A*math.cos(Q*arg) + 127
    sin_table.append(int(x))
    cos_table.append(int(y))

index = 0 # index range: 0..(N-1)
while True:
    x = sin_table[index]
    y = cos_table[index]
    index = (index+1) % N
    dac1.write(x)
    dac2.write(y)
```

נציג את שני האוטות על גבי סקופ (ערוץ אחד מחובר להדק 25 והערוץ השני של הסקופ מחובר להדק 26) נקבל את האות הבא:



כמובן שאפשר לעבור למצב XY כדי לקבל את הצורה הנ"ל:



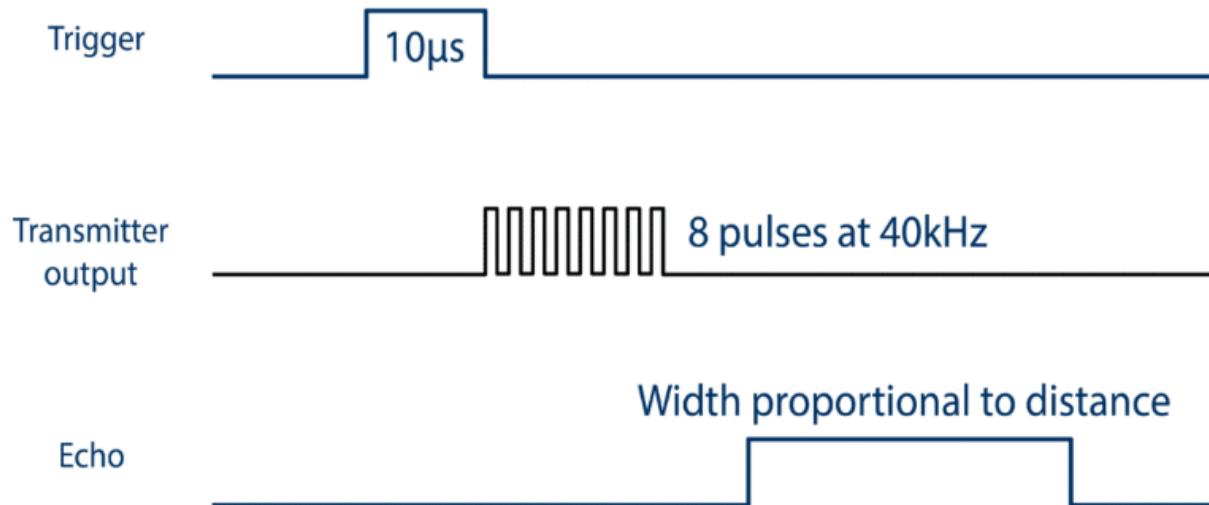
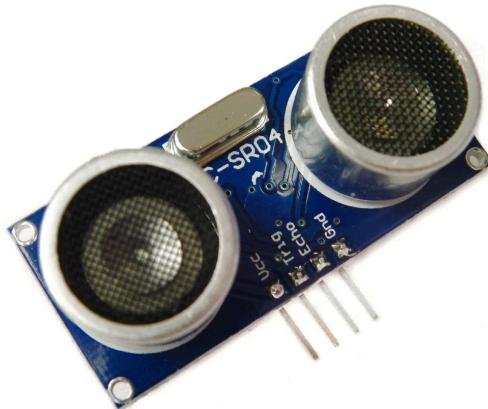
## משימה 8 - הפעלת מד מרחק אולטרו-סוני מודגמ hc-sr04

קישורים:

<https://thepihut.com/blogs/raspberry-pi-tutorials/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi>

ה-HC-SR04 הוא מד מרחק אולטרו-סוני זול ונפוץ מאוד. לרכיב ארבע הדקים:

- הדק Vcc המתקבל 5V.
- הדק אדמה - Gnd.
- הדק Trigger - המשמש להפעלת המדידה.
- הדק Echo - שדרוכה אנו מקבלים מדידת מרחק כפולו שהרכיב שלו יחסית למרחק.

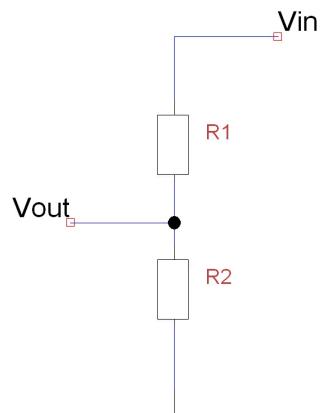


הבעיה המרכזי בחיבור מד מרחק hc-sr04 לבודר ESP-32 היא ההבדל בין מתח העבודה של 2 הרכיבים. בעוד hc-sr04 עובד במתח עבודה של 5V, מתח העבודה של ESP-32 הוא 3.3V ממשען שאנו זקוקים למתח אספקה חיצוני של 5V לעבוד החישון כמו כן חובה עלינו להתאים בין רמת המתחים של הדק Echo שמתקובל בرمות של 5V לרמת 3.3V אחרת אנו נגרום נזק לבקר. המצב ההפוך שבו דרך Trigger המסופק בرمות של 3.3V נכנס לחישון שעבוד בرمות של 5V לא יגרום נזק.

הדרך להתאים את רמות המתחים היא על ידי רכיב ייעודי לכך, או דרך מחלוקת מתח פשוט המתווסף הבאים הבא:

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2}$$

$$\frac{V_{out}}{V_{in}} = \frac{R_2}{R_1 + R_2}$$



מכאן שערך הנגדים במקרה שלנו יהיה:

$$\frac{3.3}{5} = \frac{R_2}{1000 + R_2}$$

$$0.66 = \frac{R_2}{1000 + R_2}$$

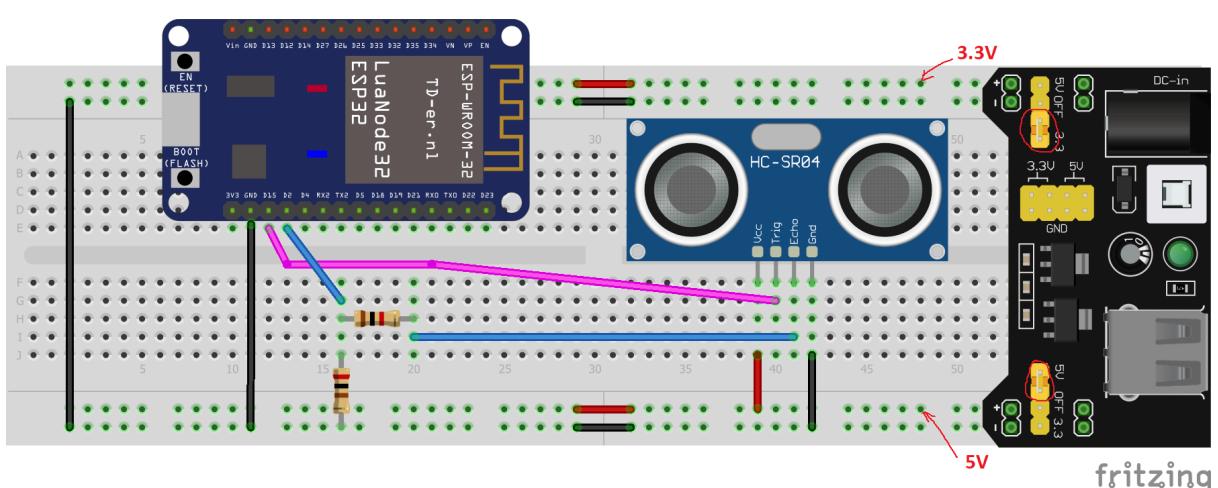
$$0.66(1000 + R_2) = R_2$$

$$660 + 0.66R_2 = R_2$$

$$660 = 0.34R_2$$

$$1941 = R_2$$

להלן שרטוט החומרה המחבר בין החישון לבקה:



נדגים תוכנה בסיסית למדידת מרחק:

```
import time
from machine import Pin, time_pulse_us
from utime import sleep_us

def Get_distance_cm(trigger_pin=32,echo_pin=34):

    # Init trigger pin (out)
    trigger = Pin(trigger_pin, mode=Pin.OUT, pull=None)
```

```
# Init echo pin (in)
echo = Pin(echo_pin, mode=Pin.IN, pull=None)

# Send a 10us pulse.
trigger.value(0)
sleep_us(5)
trigger.value(1)
sleep_us(10)
trigger.value(0)

pulse_time = time_pulse_us(echo, 1, 500*2*30)
cms = (pulse_time / 2) / 29.1    # 1cm each 29.1us
return cms

for i in range(5):
    distance = Get_distance_cm()
    print('Distance:', distance, 'cm')
    time.sleep(2)
```

**נקבל את הפלט הבא:**

```
load:0x3fff0030,len:4728
load:0x40078000,len:14888
load:0x40080400,len:3368
entry 0x400805cc
Distance: 10.7732 cm
Distance: 13.33333 cm
Distance: 13.81443 cm
Distance: 16.09966 cm
Distance: 17.1134 cm
}
} 60
MicroPython v1.23.0 on 2024-06-02; Generic ESP32 module with ESP32
Type "help()" for more information.
>>>
```

נער בדוגמת הקוד הבא:

<https://github.com/rsc1975/micropython-hcsr04>

כדי להגדיר מחלקה המטפלת במד המרחק. את הגוד הבא נחלק ל-2 חלקיים. הראשון מחלקת ברכיב:

```
from machine import Pin, time_pulse_us
from utime import sleep_us

__version__ = '0.2.1'
__author__ = 'Roberto Sánchez'
__license__ = "Apache License 2.0.
https://www.apache.org/licenses/LICENSE-2.0"

class HCSR04:
    """
        Driver to use the ultrasonic sensor HC-SR04.
        The sensor range is between 2cm and 4m.

        The timeouts received listening to echo pin are converted to
        OSError('Out of range')
```

```

"""
# echo_timeout_us is based in chip range limit (400cm)
def __init__(self, trigger_pin, echo_pin, echo_timeout_us=500*2*30):
    """
        trigger_pin: Output pin to send pulses
        echo_pin: Readonly pin to measure the distance. The pin should be
protected with 1k resistor
        echo_timeout_us: Timeout in microseconds to listen to echo pin.
        By default is based in sensor limit range (4m)
    """
    self.echo_timeout_us = echo_timeout_us
    # Init trigger pin (out)
    self.trigger = Pin(trigger_pin, mode=Pin.OUT, pull=None)
    self.trigger.value(0)

    # Init echo pin (in)
    self.echo = Pin(echo_pin, mode=Pin.IN, pull=None)

def _send_pulse_and_wait(self):
    """
        Send the pulse to trigger and listen on echo pin.
        We use the method `machine.time_pulse_us()` to get the microseconds
until the echo is received.
    """
    self.trigger.value(0) # Stabilize the sensor
    sleep_us(5)
    self.trigger.value(1)
    # Send a 10us pulse.
    sleep_us(10)
    self.trigger.value(0)
    try:
        pulse_time = time_pulse_us(self.echo, 1, self.echo_timeout_us)
        if pulse_time < 0:
            MAX_RANGE_IN_CM = const(500)
            pulse_time = int(MAX_RANGE_IN_CM * 29.1) # 1cm each 29.1us
        return pulse_time
    except OSError as ex:
        if ex.args[0] == 110: # 110 = ETIMEDOUT
            raise OSError('Out of range')
        raise ex

def distance_mm(self):
    """
        Get the distance in milimeters without floating point operations.
    """
    pulse_time = self._send_pulse_and_wait()

    # To calculate the distance we get the pulse_time and divide it by
2
    # (the pulse walk the distance twice) and by 29.1 because
    # the sound speed on air (343.2 m/s), that It's equivalent to
    # 0.34320 mm/us that is 1mm each 2.91us

```

```

    # pulse_time // 2 // 2.91 -> pulse_time // 5.82 -> pulse_time * 100
// 582
    mm = pulse_time * 100 // 582
    return mm

def distance_cm(self):
    """
    Get the distance in centimeters with floating point operations.
    It returns a float
    """
    pulse_time = self._send_pulse_and_wait()

    # To calculate the distance we get the pulse_time and divide it by
2
    # (the pulse walk the distance twice) and by 29.1 because
    # the sound speed on air (343.2 m/s), that It's equivalent to
    # 0.034320 cm/us that is 1cm each 29.1us
    cms = (pulse_time / 2) / 29.1
    return cms

```

קוד השני מזמן את המחלקה ומציג את המרחק:

```

import time
from hcsr04 import HCSR04

sensor = HCSR04(trigger_pin=32, echo_pin=34)
for i in range(10):
    distance = sensor.distance_cm()
    print('Distance:', distance, 'cm')
    time.sleep(1.5)

```

פלט התוכנית צפוי להיראות כך:

```
import time
from hcsr04 import HCSR04

sensor = HCSR04(trigger_pin=32, echo_pin=34)
for i in range(10):
    distance = sensor.distance_cm()
    print('Distance:', distance, 'cm')
    time.sleep(1.5)
```

rst:0xc (SW\_CPU\_RESET),boot:0x13 (SPI\_FAST\_FLASH\_BOOT)  
configsip: 0, SPIWP:0xee  
clk\_drv:0x00,q\_drv:0x00,d\_drv:0x00,cs0\_drv:0x00,hd\_drv:0x00,wp\_drv:0x00  
mode:DIO, clock div:2  
load:0x3fff0030,len:4728  
load:0x40078000,len:14888  
load:0x40080400,len:3368  
entry 0x400805cc  
Distance: 31.9244 cm  
Distance: 120.2577 cm  
Distance: 26.03093 cm  
Distance: 14.12371 cm  
Distance: 11.90722 cm  
Distance: 108.2646 cm  
Distance: 12.13058 cm  
Distance: 9.621993 cm  
Distance: 9.725086 cm  
Distance: 9.75945 cm

## משימה 9 - תקשורת UART בין 2 בקרים

קישורים:

<https://docs.micropython.org/en/latest/library/machine.UART.html>

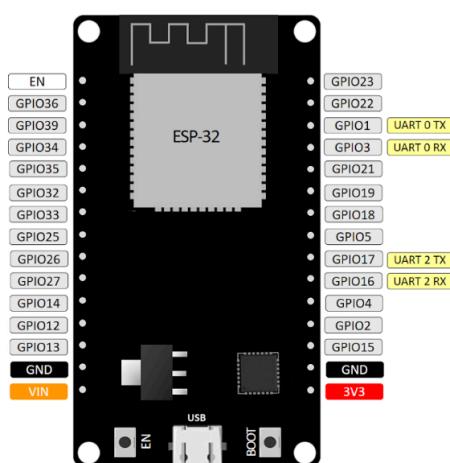
לבקר ESP32 יש 3 ממשיки תקשורת UART (ש רק אחד זמין לשימוש) על פי המפרט הבא:

UART0: (GPIO 1 and GPIO3) משמש לתקשורת מול המחשב -

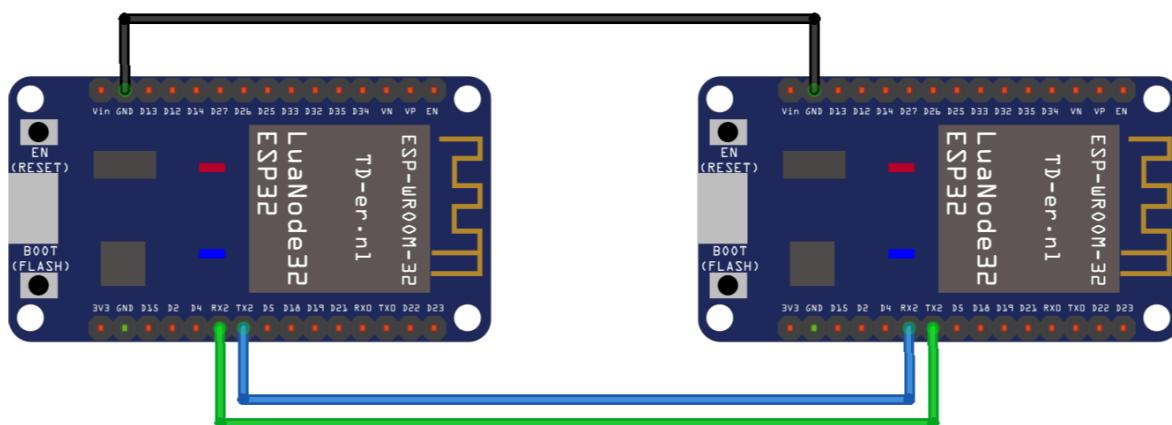
UART1: (GPIO 9 and GPIO10) – connected to the ESP32 SPI flash memory, so you can't use them.

UART2: (GPIO 17 and GPIO 16) זמין לשימוש -

להלן מיקום הדקי ה- UART בברker:



כלומר UART0 לא זמין - UART2 זמין דרך הדקים tx=17 ו rx=16rx שלוש נחבר את התקשרות בין 2 הבקרים כמפורט באירור הבא:



נתען את הקוד הבא באל אחד ממשי הבקרים:

```
import uasyncio as asyncio
from machine import UART
uart = UART(2, 115200)
```

```

async def sender():
    i=0
    while True:
        s = 'Hello uart' + str(i) + '\n'
        uart.write(s)
        print(s)
        await asyncio.sleep(2)
        i=i+1

async def receiver():
    sreader = asyncio.StreamReader(uart)
    while True:
        res = await sreader.readline()
        print('Received', res)

loop = asyncio.get_event_loop()
loop.create_task(sender())
loop.create_task(receiver())
loop.run_forever()

```

להלן דוגמה לפלט התקשרות:

The screenshot shows two terminal windows side-by-side. The left terminal window displays the Python code for the Arduino Uno, which includes an `async def` function for sending data over UART and another for receiving data. The right terminal window shows the actual serial communication between the Arduino and the computer, with the received data being printed to the screen. Handwritten red annotations with numbers 1 through 13 are overlaid on both windows, pointing to specific lines of code or data points.

דוגמים תקשורת בין ESP32 לArduino UNO בckettן

בקר ESP32 מכיל את הקוד הבא:

טען את הקוד הבא באל אחד משני הבקרים:

```
import uasyncio as asyncio
from machine import UART
uart = UART(2, 115200)

async def sender():
    i=0
    while True:
        s = 'Hello uart' + str(i) + '\n'
        uart.write(s)
        print(s)
        await asyncio.sleep(2)
        i=i+1

async def receiver():
    sreader = asyncio.StreamReader(uart)
    while True:
        res = await sreader.readline()
        print('Recieved', res)

loop = asyncio.get_event_loop()
loop.create_task(sender())
loop.create_task(receiver())
loop.run_forever()
```

בקר UNO arduino מכיל את הקוד הבא:

```
#include <SoftwareSerial.h>

// RX is digital pin 2 (connect to TX of other device)
// TX is digital pin 3 (connect to RX of other device)
SoftwareSerial ESP32Serial(2, 3); // RX, TX

void setup()
{
    // Open serial communications and wait for port to open:
    Serial.begin(115200);

    // set the data rate for the SoftwareSerial port
    ESP32Serial.begin(115200);
}

void loop() // run over and over
{
    if (ESP32Serial.available())
        Serial.write(ESP32Serial.read());
    if (Serial.available())
        ESP32Serial.write(Serial.read());
}
```

נקבל את הפלט הבא:

The screenshot shows two terminal windows side-by-side. The left window is a Python terminal (main.py) displaying a sequence of "Hello uart" messages from index 202 to 211. The right window is an Arduino Serial Monitor showing a sequence of "Hello uart" messages from index 206 to 211, along with a received message "Recieved b'Hello from arduino\n'". Red arrows and a brace highlight the matching messages between the two terminals.

Left Terminal (Python):

```
1 import uasyncio as asyncio
2 from machine import UART
3 uart = UART(2, 115200)
4
5 async def sender():
6     # swriter = asyncio.StreamWriter(uart, {})
7     i=0
8     while True:
9         s = 'Hello uart' + str(i) + '\n'
10        # await swriter.awrite(s)
11        uart.write(s)
12        print(s)
13        await asyncio.sleep(2)
```

Right Terminal (Arduino):

```
3 // RX is digital pin 2 (connect to TX
4 // TX is digital pin 3 (connect to RX
5 SoftwareSerial ESP32Serial(2, 3); //
6
7 void setup()
8 {
9     // Open serial communications and w
10    Serial.begin(115200);
11
12    // set the data rate for the Softwa
13    ESP32Serial.begin(115200);
14 }
15
16 void loop() // run over and over
17 {
18     if (ESP32Serial.available())
19         Serial.write(ESP32Serial.read());
```

Output (Arduino Serial Monitor):

```
Hello uart202
Hello uart203
Hello uart204
Hello uart205
Hello uart206
Hello uart207
Hello uart208
Hello uart209
Recieved b'Hello from arduino\n'
Hello uart210
Hello uart211
```

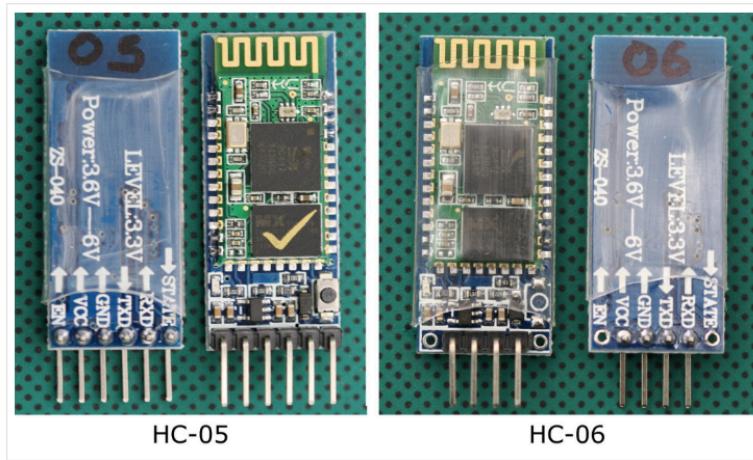
## משימה 10 - תקשורת Bluetooth מבוססת HC-05

קישורים:

<http://www.martyncurrey.com/hc-05-and-hc-06-zs-040-bluetooth-modules-first-look/>

<https://heeed.net/micro-bit-and-the-blue-micropython/>

ההבדל בין HC-05 ל-HC-06:



רכיב HC-05 יכול לשמש גם כ-Master ו גם כ-Slave בעוד שהרכיב HC-06 יכול לשמש רק כ-Slave. במלחים אחרים המשמעות היא שהיא HC-05 יכולה לחבר אחריה ו HC-06 יכולה לחבר בלבד ממחסיר אחר. לאחר יצירת הקשר בשני הכוונים התקשרות יכולה להיות דו-כיוונית.

מאפיינים טכניים וחשמליים של הרכיב:

Radio Chip: CSR BC417

Memory: External 8Mbit Flash

Output Power: -4 to +6dbm Class 2

Sensitivity: -80dbm Typical

Bit Rate: EDR, up to 3Mbps

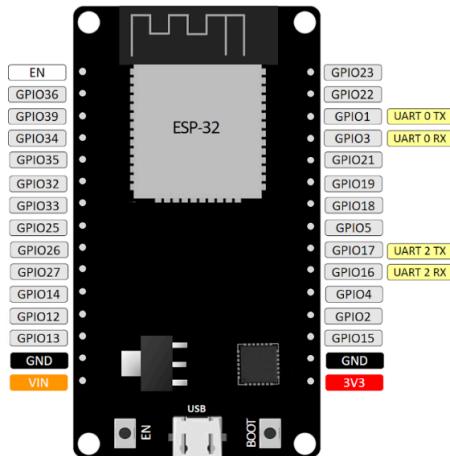
Interface: UART

Antenna: Built-in

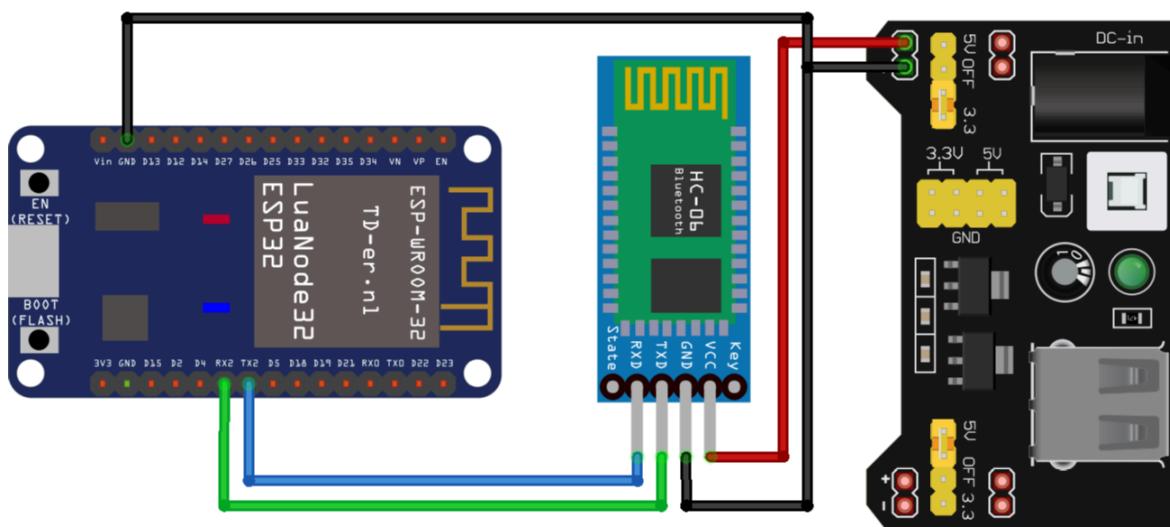
Voltage: 3.1 to 4.2VDC

Current: 40mA max

כפי שראינו במשימה הקודמת לבקר ESP32 יש 3 ממתקי תקשורת UART כאשר רק UART2 המתפרק בין הדקים (16 GPIO 17 and GPIO 2) זמין באופן מלא לשימוש. להלן מיקום הדק ה-UART בבלוק:



נחבר את רכיב ה- BT ל-2 UART הדריך הדקים :rx=16 - i tx=17



### אתחול הגדרות של hc-06:

נכתב את הקוד הבא לבקר במטרה לאתחול את הגדרות התחברות בין הטלפון הנייד לבין ה- HC-06. זאת בມידא ואנו לא יודעים את הגדרות האתחול של הרכיב, כמו שם הרכיב, הסיסמה של הרכיב וקצב התחברות שלו.

```
from machine import UART
import utime

uart = UART(2, 9600)
uart.init(9600, bits=8, parity=None, stop=1)
print(uart)

print("-----")
print("|\ Module HC-06 configuration |")
print("|\ enter AT           -- To test serial communication |")
print("|\ enter AT+NAME?????? -- To modify the module name |")
print("|\ enter AT+PIN1234    -- To modify the module PIN code |")
print("|\ enter AT+BAUD4      -- To modify the module communication speed|")
```

```

print("|\ Note: 1 for 1200, 2 for 2400, 3 for 4800, 4 for 9600 |")
print("|\ 5 for 19200, 6 for 38400, 7 for 57600, 8 for 115200 |")
print("-----|")

while True:
    print("ENTER AT Commands: ")
    try:
        str = input()
        uart.write(str)
        utime.sleep_ms(100)
    except OSError:
        pass

    # wait for response
    start_time = utime.ticks_ms()
    timeout = False
    while not uart.any() and not timeout:
        if utime.ticks_diff(utime.ticks_ms(), start_time) > 500:
            timeout = True
    if timeout:
        print('Failed, response timed out')
    else:
        buf = uart.read()
        print("received:",buf)
    utime.sleep_ms(600)

```

להלן דוגמה לפט התוכנית:

```

;32mI (2910) uart: ALREADY NULLm
UART(2, baudrate=9600, bits=8, parity=None, stop=1, tx=17, rx=16, rts=-1, cts=-1, txbuf=256, rxbuf=256, timeout=0, timeout_char=2)
-----|
| Module HC-06 configuration
| enter AT      -- To test serial communication
| enter AT+NAME?????  -- To modify the module name
| enter AT+PIN1234   -- To modify the module PIN code
| enter AT+BAUD4     -- To modify the module communication speed
| Note: 1 for 1200, 2 for 2400, 3 for 4800, 4 for 9600
|      5 for 19200, 6 for 38400, 7 for 57600, 8 for 115200
-----|
ENTER AT Commands:
AT
2
received: b'OK'
ENTER AT Commands:
AT+NAMEESP32_BT
15
received: b'OKsetname'
ENTER AT Commands:
AT+PIN1234
10
received: b'OKsetPIN'
ENTER AT Commands:
AT+BAUD4
8
received: b'OK9600'
ENTER AT Commands:

```

במקרה זה יתקבל תוצאה דומה זו:

```

Module HC-06 configuration
enter AT          -- To test serial communication
enter AT+NAME?????  -- To modify the module name
enter AT+PIN1234   -- To modify the module PIN code
enter AT+BAUD4     -- To modify the module communication speed
Note: 1 for 1200, 2 for 2400, 3 for 4800, 4 for 9600
      5 for 19200, 6 for 38400, 7 for 57600, 8 for 115200

ENTER AT Commands:
AT
2
Failed, response timed out
ENTER AT Commands:
AT
2
received: b'OK'
ENTER AT Commands:

```

לאחר אתחול הרכיב נעבור לתוכנית המבוצעת תקשורת טקסט זו כיוון בין הבקר לטלפון נייד

קוד התוכנית:

```

from machine import UART
import utime

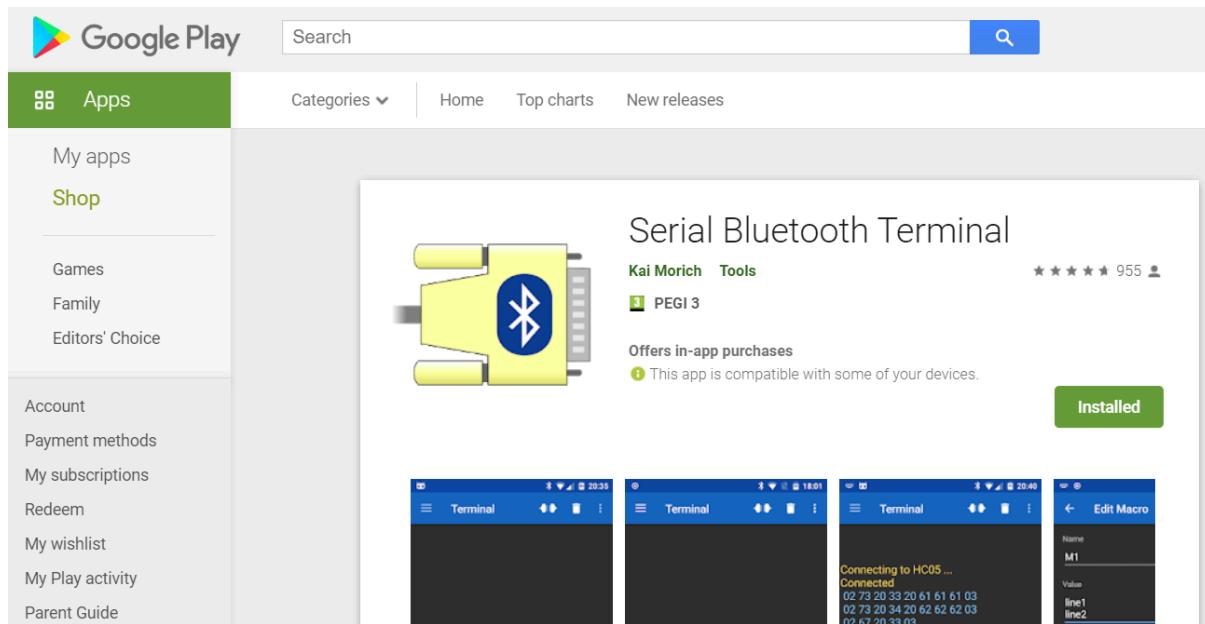
uart = UART(2, 9600)
uart.init(9600, bits=8, parity=None, stop=1)
print(uart)

while True:
    if uart.any():
        while uart.any():
            buf = uart.read()
            print('received:',buf)
            utime.sleep_ms(15)

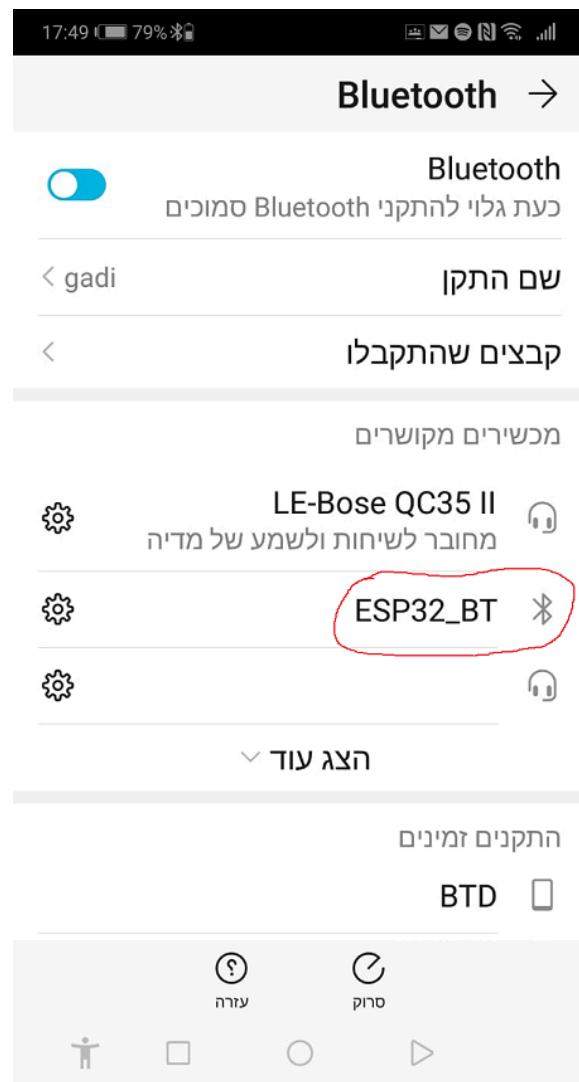
        utime.sleep_ms(10)
        try:
            uart.write("OK")
            print('sent response')
        except OSError:
            pass

```

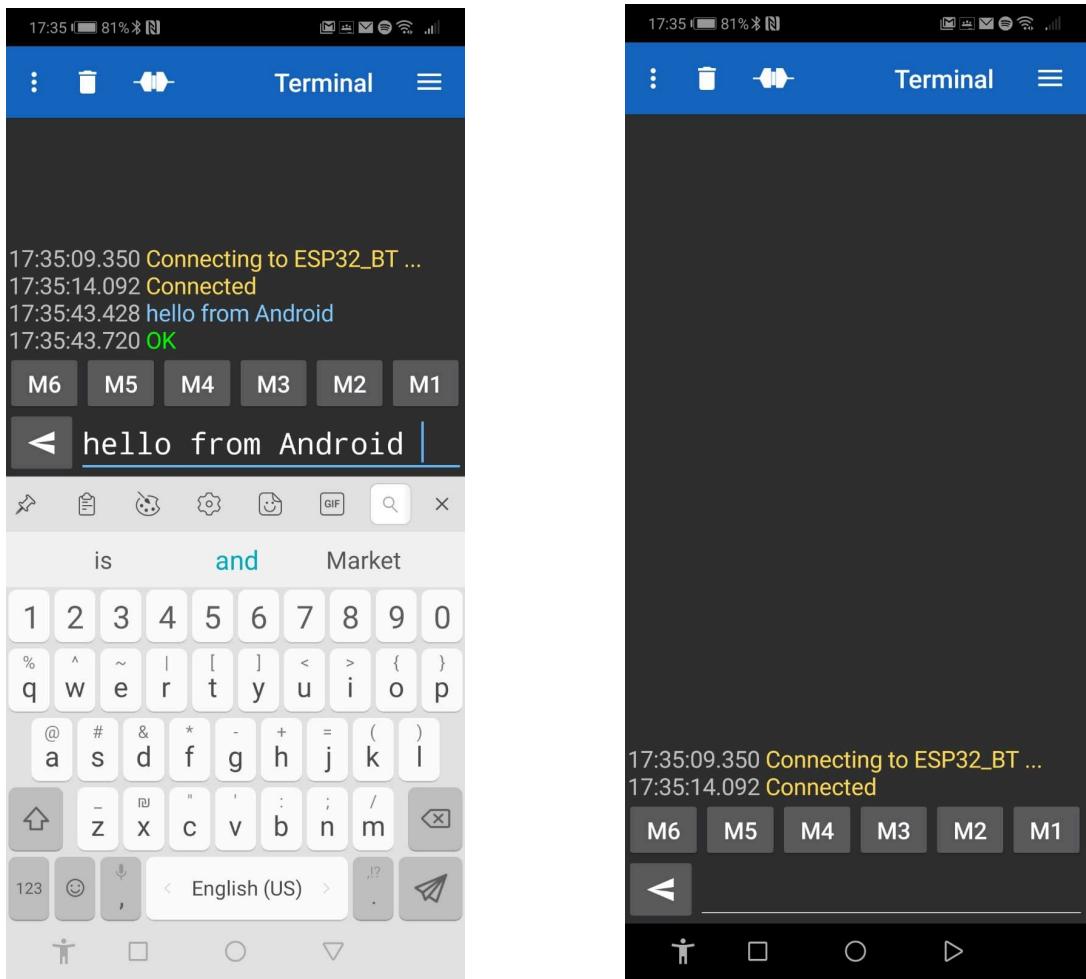
נפתח תוכנה ייעודית לתקשורת BT דרך הטלפון הנייד, כמו התוכנה כמו בתמונה



נחבר את הטלפון נייד למכשיר ה-Bluetooth על ידי כך שנסורך את התקני ה-Bluetooth הזמינים:



לבסוף נפעיל את היישום שהתקנו ונשלח לבקר טקסט:



על מסך המחשב נקלט הפלט הבא:

```
import time
import utime

uart = UART(2, 9600)
uart.init(9600, bits=8, parity=None, stop=1)
print(uart)

while True:
    if uart.any():
        while uart.any():
            buf = uart.read()
            print('received:',buf)
            utime.sleep_ms(15)

        utime.sleep_ms(10)
        try:
            uart.write("OK")
            print('sent response')
        except OSError:
            pass
```

605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634

sent response  
received: b'hello from Android \r\n'  
2 sent response

## משימה 11 - תקשורת Bluetooth Low Energy בבקר ESP32

היא טכנולוגיה להעברת מידע אלחוטית בין התקנים. לעומת Bluetooth Low Energy, היא גרסה לצריכת חשמל נמוכה המיעדת להתקנים שצריכים ח"י סוללה ארוכים, כמו צמיד כושר ושעונים חכמים.

- Bluetooth: מתאים להעברת מידע מהיר, כמו העברת שמע לאוזניות או רמקולים. פרוטוקול זה תומך בהעברה בקצבים גבוהים, אך דורש צריכת חשמל גבוהה יחסית.
- Bluetooth Low Energy: פועל בצריכת חשמל נמוכה ויכול לפעול זמן רב. מתאים להתקנים שצריכים ח"י סוללה ארוכים אך סובל מקצב העברת מידע מוגבל.

בקר ESP32 כולל רכיב רדיו Bluetooth Low Energy מובנה שלו נטמון בתחום הפעילות זו

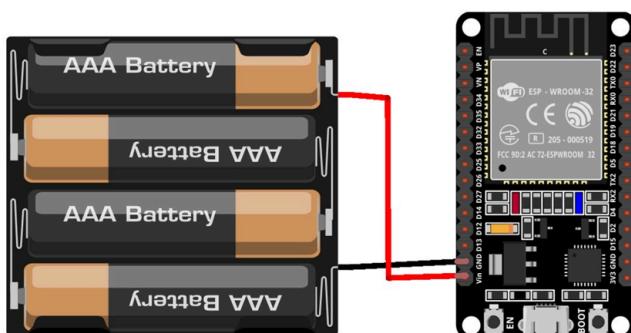
קישורים:

<https://randomnerdtutorials.com/micropython-esp32-bluetooth-low-energy-ble/>

<https://docs.micropython.org/en/latest/library/bluetooth.html>

### חשיבות: שימוש במקור מתוך חיצוני

העבודה עם רכיב Bluetooth-low או רכיב Wi-Fi המובנים בבקר ESP32 דורשת זרם עבודה גדול מ-500mA המספק מחברת USB. על כן יש צורך לחבר מקור מתוך חיצוני מתאים באיזור



הדק Vin ב-ESP32 מחובר למייצב מתוך פנימי. על כן ניתן לחבר בהדק Vin מתח בין 5V ל-12V. כל מתח בטוח זהה בהדק Vin עובר למייצב המפחת אותו ל-3.3V ולאחר מכן מזון לציר ההיקפי של לוח ה-ESP32.

באמצעות סוללה חיצונית של 7V או 9V נוכל להפעיל את ESP32 דרך פין Vin על ידי חיבור GND של ESP32 עם GND של הסוללה. ניתן לחבר כל מתח בין 5V ל-12V לפין Vin ESP32 אולם מומלץ לא להשתמש ביותר מסוללה חיצונית של 7V. מכיוון ש-ESP32 צריך רק 3.3V כדי לפעול, שאר המתחים מתפזרים על ידי מייצב המתח חום.

### בדיקה:

להלן דוגמת קוד:

```
print("Scanning for WiFi networks, please wait...")
print("")
```

בלה..

בלה..

בלה..

בלה..

בלה..

בלה..

## משימה 12 - אתחול קישוריות ה-WiFi בבקר ESP32

בקר ESP32 מספק ביצועים גבוהים על בסיס מעבד הכלול 2 ליבוט. כמו כן הבקר כולל קישוריות ישירה לרשת האינטרנט דרך רכיב WiFi מובנה. כמו כן בהמשך משימה זו נעשה היכרות על הקבצים `uf2` ו- `main.py`.

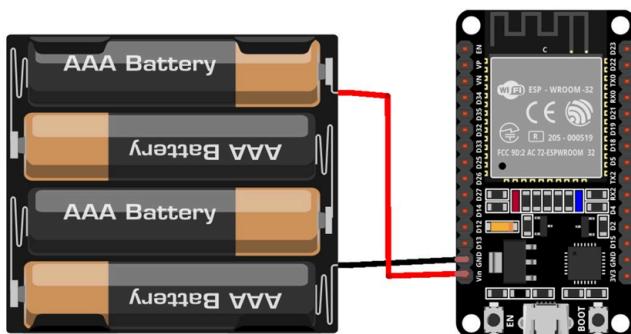
ונלמד כיצד לחבר את הבקר לרשת מיד לאחר האתחול.

קישורים:

<https://randomnerdtutorials.com/micropython-wi-fi-manager-esp32-esp8266/>

חשיבות: שימוש במקור מתוך חיצוני

העובדת עם רכיב `htc-ws0a` או רכיב ה-WiFi המובנים בבקר ESP32 דורש זרם עזובה גדול מ-`500mA` המספק מחבר ה-USB. על כן יש צורך לחבר מקור מתח חיצוני מתאים באוויר



הדק `Vin` ב-ESP32 מחובר למיצב מתח פנימי. על כן ניתן לחבר בהדק `Vin` מתח בין `5V` ל-`12V`. כל מתח בטח זהה בהדק `Vin` עובר למיצב המפחית אותו ל-`3.3V` ולאחר מכן מזון לצירוד ההיקי של לוח ה-ESP32.

באמצעות סוללה חיצונית של `7V` או `9V` נוכל להפעיל את ESP32 דרך פין `Vin` על ידי חיבור `GND` של ESP32 עם `GND` של הסוללה. ניתן לחבר כל מתח בין `5V` ל-`12V` לפין `Vin` ESP32 ואולם מומלץ לא להשתמש בו יותר מסוללה חיצונית של `7V`. מכיוון ש-ESP32 צריך רק `3.3V` כדי לפעול, שאר המתחרים מתפזרים על ידי מיצב המתח כחום.

בדיקות רשתות WiFi זמניות:

להלן דוגמת קוד שבודק מהם רשתות WiFi הזמינים לשימוש:

```
print("Scanning for WiFi networks, please wait...")
print("")

import network
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)

authmodes = [ 'Open' , 'WEP' , 'WPA-PSK' , 'WPA2-PSK4' , 'WPA/WPA2-PSK' ]
for (ssid, bssid, channel, RSSI, authmode, hidden) in sta_if.scan():
    print("* {:s}".format(ssid))
    print("  - Auth: {}".format(authmodes[authmode], '(hidden)' if hidden else ''))
    print("  - Channel: {}".format(channel))
    print("  - RSSI: {}".format(RSSI))
    print("  - BSSID: ")
    print("{:02x}:{:02x}:{:02x}:{:02x}:{:02x}:{:02x}.".format(*bssid))
    print()
```

נקבל את הפלט הבא:

```
Scanning for WiFi networks, please wait...
* [REDACTED]
- Auth: WPA/WPA2-PSK
- Channel: 11
- RSSI: -30
- BSSID: 54:db:a2:0f:48:81

* [REDACTED]
- Auth: WPA/WPA2-PSK
- Channel: 11
- RSSI: -44
- BSSID: 6e:56:97:d0:b6:8e

* [REDACTED]
- Auth: WPA/WPA2-PSK
- Channel: 1
- RSSI: -65
- BSSID: 18:a6:f7:fe:93:ea
```

### חיבור הבקר לרשת ה-Wi-Fi

לאחר שקיבלנו את רשימת הרשנות הזמין את הקוד הבא כדי לחבר את הבקר לאחת הרשנות שברשימה:

```
import network

def connect():
    ssid = "שם הרשת"
    password = "סיסמת החיבור לרשת"

    station = network.WLAN(network.STA_IF)

    if station.isconnected() == True:
        print("Already connected")
        print(station.ifconfig())
        return

    station.active(True)
    station.connect(ssid, password)

    while station.isconnected() == False:
        pass

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

connect()
```

לאחר הריצת הקוד נקבל את הקוד הבא:

```
load:0x40078000,len:14888
load:0x40080400,len:3368
entry 0x400805cc
Connection successful
('10.0.0.10', '255.255.255.0', '10.0.0.138', '10.0.0.138')
MicroPython v1.23.0 on 2024-06-02; Generic ESP32 module with ESP32
Type "help()" for more information.
>>>
>>>
```

ניתן לראות שהבקר הת לחבר לרשת וקיבל את כתובת ה- IP המפורטת.

### הקבצים boot.py ו main.py

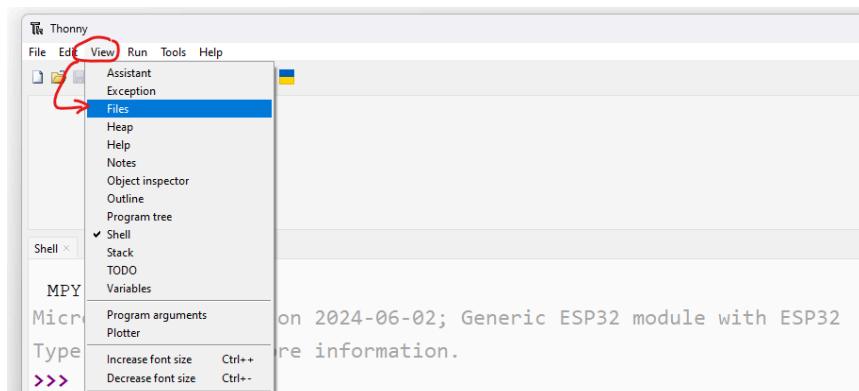
לכל בקר מוגדרים 2 קבצים ייחודיים הנשמרים בו. הראשון קובץ בשם boot.py והשני קובץ בשם main.py. בקובץ boot משמש אותנו להקצת קוד עבור הוראות אתחול של הבקר, קוד זה רץ פעם אחת בלבד בזמן האתחול. בקובץ זה מקובל ליבא ספריות רלוונטיות, הגדרת קבועים כמו שמות ויסודות. במשימה זו נעשו שימוש בקובץ זה כדי ליצר לבקר את הקישורית לרשת האינטרנט.

הקובץ main עתיד להכיל את קוד התוכנית שתעבד באוטומטי לאחר הרמת הקובץ boot.py. קובץ זה ישימוש להפעלת היישום שכתבנו לאחר שלב הפיתוח.

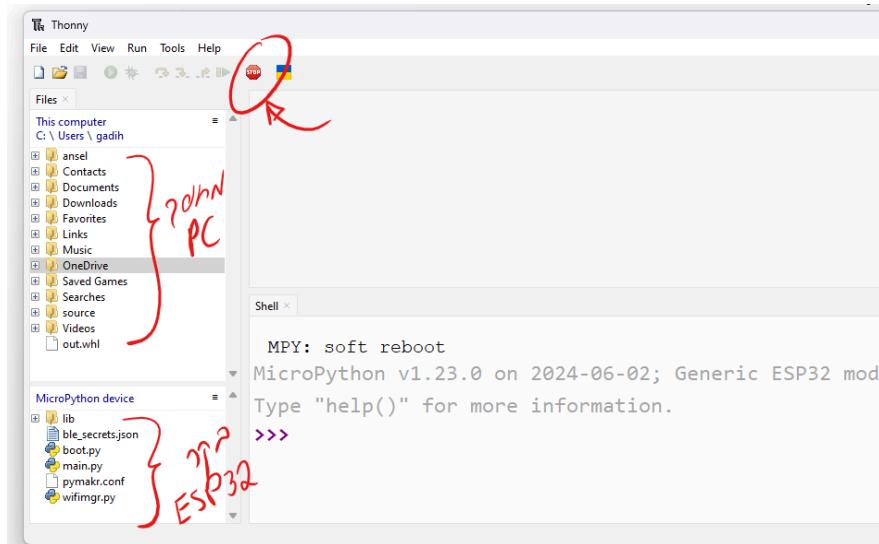
הערה: ברוב הפעמים כאשר כותבים קוד תוכנה עבור בקר בשפת MicroPython תוכלו למצוא שני קבצים המאוחסנים בברcker עצמו: הקובץ הראשון נקרא: boot.py והשני main.py. ברגע שהבקר מקבל מתח עבודה או מיד לאחר איפוס יוזם, הבקר מפעיל אוטומטית את הקוד השמור בקובץ boot.py לאחר מכן הוא יפעיל את תוכן הקובץ main.py. שימוש הדבר היא שלאחר שמירת הקובץ main.py בברcker. לא ניתן היה לבצע עדכוני תוכנה בו ללא מחיקת הקובץ הנ"ל ככל מדובר באתחול בקר חדש. لكن השימוש באפשרות זו רק כאשר אתם מעבירים את הקוד ממצב פיתוח למצב שימוש.

### כתיבת הקובץ boot.py לתחול תקשורת WiFi

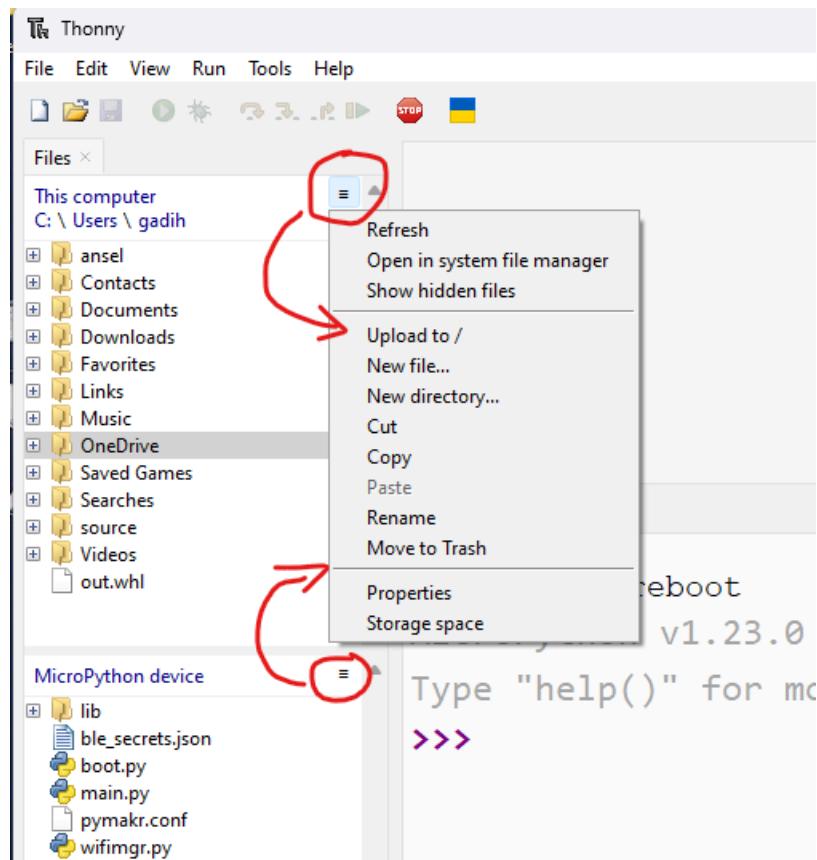
בקר ESP32 מצוי בזיכרון הבזק בנפח של 4 מגה-בייט שבו ניתן לשמר קבצי קוד ונתונים. נעזר בסביבת הפיתוח Thonny כדי לבדוק את הקבצים שבזיכרון הבקר מיד לאחר אתחול בברcker בקורסיה חדשה. נעשה זאת על ידי לחיצה על View -> File כדי לראות את הקבצים שבברcker:



נפתח לנו המסר הבא:



בעזרתחלון שנפתח אנו יכולים לראות אילו קבצים שמררים בזיכרון של הבקר. כמו כן ניתן לבצע את כל הפעולות הבסיסיות על הקבצים כמו הוספה קבצים, מחיקת קבצים, יצרה ומחיקה של תיקיות ושינוי של של קובץ.



נעזר במשק שפתחנו כדי להעתיק את הקובץ `boot.py` מהזיכרון של הבקר למחשב על ידי לחיצה על Upload to

להלן קוד התוכנית **עבור הקובץ `boot.py`**:

```
# This file is executed on every boot (including wake-boot from deepsleep)
import network

def connect():
    ssid = "yourNetworkName"
    password = "yourNetworkPassword"

    station = network.WLAN(network.STA_IF)

    if station.isconnected() == True:
        print("Already connected")
        print(station.ifconfig())
        return

    station.active(True)
    station.connect(ssid, password)

    while station.isconnected() == False:
        pass

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

connect()
```

חשוב: עדכנו בקובץ את שם רשת ה-Wifi הזמין ואות הסיסמה במקום השורות הבאות:

```
ssid = "yourNetworkName"
password = "yourNetworkPassword"
```

לאחר כתיבת הקובץ `boot.py` נעלם אותו לבקר.

כדי לבדוק שאכן הבקר התחבר לאינטרנט נבצע אתחול ונקלט את הפלט הבא:

The screenshot shows the Thonny IDE interface. On the left, there's a file browser with 'This computer' and 'MicroPython device' sections. In the 'MicroPython device' section, 'boot.py' is selected. The main window has two tabs: 'boot.py' (code editor) and 'Shell'. The code in 'boot.py' is:

```

station = network.WLAN(network.STA_IF)

if station.isconnected() == True:
    print("Already connected")
    return

```

The 'Shell' tab shows the output of running the code:

```

>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Connection successful
('10.0.0.34', '255.255.255.0', '10.0.0.138', '10.0.0.138')
Already connected
>>>

```

A red circle highlights the line 'Already connected'.

בעקבות חיבורית מוצלחת נקלט בחלון ה- Terminal את מאפייני החיבור מהם:

- כתובת IP של הבקר
- ערך ה- subnet mask
- כתובת ה- gateway
- כתובת ה- DNS

#### בדיקת כתובת ה- MAC של הבקר

כתובת MAC היא מספר המורכבת מ-48 סיביות שמשמעותו זהות באופן ייחודי רכיב חומרה המחבר לרשת. כתובת MAC מוטבעת בדרך כלל על כרטיס הרשת של המחשב כМОן שבמקרה שלנו כתובת ה- MAC מוטבעת על הבקר עצמו.

כדי לבדוק מה כתובת ה- MAC של הבקר שלנו ניתן להריץ את הקוד הבא:

```

import network
import ubinascii
mac = ubinascii.hexlify(network.WLAN().config('mac'), ':').decode()
print (mac)

```

נקבל פלט הדומה לזה:

```

MPY: soft reboot
Already connected('10.0.0.10', '255.255.255.0', '10.0.0.138', '10.0.0.138')
08:3a:f2:50:ed:d4
>>>

```

כדי לבדוק את התקשרות לאינטרנט ננעזר בקוד הבא:

מקור:

[https://docs.micropython.org/en/latest/esp8266/tutorial/network\\_tcp.html](https://docs.micropython.org/en/latest/esp8266/tutorial/network_tcp.html)

```

def http_get(url):
    import socket
    _, _, host, path = url.split('/', 3)
    addr = socket.getaddrinfo(host, 80)[0][-1]
    s = socket.socket()
    s.connect(addr)
    s.send(bytes('GET /%s HTTP/1.0\r\nHost: %s\r\n\r\n' % (path, host),
    'utf8'))
    while True:
        data = s.recv(100)
        if data:
            print(str(data, 'utf8'), end='')
        else:
            break
    s.close()

http_get('http://micropython.org/ks/test.html')

```

נקבל את הפלט הבא:

The screenshot shows a development interface with three tabs: 'Files', '[boot.py] x', and '[main.py] \* x'. The '[main.py]' tab contains the Python code for the 'http\_get' function. A red arrow points from the 'boot.py' tab to the '[main.py]' tab, indicating the file being run. The 'Shell' tab at the bottom displays the executed code and its output. The output shows the HTTP response headers and the HTML content of the test page, which includes the text 'It's working if you can read this!' circled in red.

```

def http_get(url):
    import socket
    _, _, host, path = url.split('/', 3)
    addr = socket.getaddrinfo(host, 80)[0][-1]
    s = socket.socket()
    s.connect(addr)
    s.send(bytes('GET /%s HTTP/1.0\r\nHost: %s\r\n\r\n' % (path, host),
    'utf8'))
    while True:
        data = s.recv(100)
        if data:
            print(str(data, 'utf8'), end='')
        else:
            break
    s.close()

http_get('http://micropython.org/ks/test.html')

```

```

Connection: close
Vary: Accept-Encoding
ETag: "529d22da-b4"
Strict-Transport-Security: max-age=15768000
Accept-Ranges: bytes

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Test</title>
  </head>
  <body>
    <h1>Test</h1>
    It's working if you can read this!
  </body>
</html>

```

ניתן לראות שהתחברות מוחזקת לאינטרנט לכתובת <http://micropython.org/ks/test.html> מוחזירה למשוך HTML הכלול את הקוד הבא:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Test</title>
  </head>
  <body>
    <h1>Test</h1>
    It's working if you can read this!
  </body>
</html>
```

## משימה 13 - מימוש שרת אינטרנט מבוסס HTTP

בפעולות זו ניצור שרת אינטרנט בסיסי היושב על גבי הבקר. הבקר יקבל גישה לרשת האינטרנט דרך חיבור

WiFi מקומי ויתחיל להאזין לבקשת GET ב프וטוקול HTTP דרך מפתח 80.

להעמקה בנושא פרוטוקול HTTP ניתן לקבל דרך הקישור הבא:

[https://he.wikipedia.org/wiki/HTTP\\_POST](https://he.wikipedia.org/wiki/HTTP_POST)

קישורים:

<https://randomnerdtutorials.com/esp32-esp8266-micropython-web-server/>

<https://techtutorialsx.com/2017/06/11/esp32-esp8266-micropython-http-get-requests/>

<https://github.com/micropython/micropython/tree/master/examples/network>

[https://github.com/micropython/micropython-esp32/tree/esp32/tests/net\\_inet](https://github.com/micropython/micropython-esp32/tree/esp32/tests/net_inet)

[https://github.com/micropython/micropython/blob/master/examples/network/http\\_server\\_simplistic.py](https://github.com/micropython/micropython/blob/master/examples/network/http_server_simplistic.py)

לפני ביצוע משימה זו יש לוודא שהබקר מחובר לרשת האינטרנט דרך WiFi כמפורט במשימה 12.

כדי לבדוק שהබкар מחובר לאינטרנט יש לẤתחל את הבקר ולבזק שהשורה הבאה מופיעה:

A screenshot of a MicroPython REPL window titled "Shell". The text output shows the results of a command execution. It includes the command ">>> %Run -c \$EDITOR\_CONTENT", followed by several lines of Python code and their output. The output includes the text "MPY: soft reboot", "Connection successful", and two lines of IP addresses: "('10.0.0.34', '255.255.255.0', '10.0.0.138', '10.0.0.138')". The line "Already connected" is circled in red with a checkmark above it. The command ">>>" appears at the bottom.

ישנו מספר גרסאות לישם שרת HTTP בסיסי. נתחיל להדגים זאת על ידי הרשת הći בסיסי שניתן לכתוב בקוד.

**חשוב!** דוגמה זו מראה כיצד נכתב את שרת-HTTP הקטן ביותר האפשרי ב-MicroPython. אך יש לזכור בחשבון שמדובר בשרת לא מאובטח !!! נdagים זאת:

```
ai = socket.getaddrinfo("0.0.0.0", 8080)
```

משמעות ההוראה היא שהשרת יהיה נגיש למארחים אחרים ברשת המקומית שלך, ואם לשרת שלך יש חיבור ישיר (ללא חומרת אש) לאינטרנט, אז לכל אחד באינטרנט. עם זאת, היזהרו בעת הפעלת פעללה זו על מחשבך המחבר לאינטרנט! החלפו את "0.0.0.0" ב-"127.0.0.1", כדי להפוך את השרת שלכם לנגיש רק למחשב שאתה הוא פועל.

במה שקדם הראה כיצד לישם שרת מיועד לשימוש בפרויקטים.

להלן קוד התוכנית עבור שרת HTTP בסיסי (גרסה 1):

```
import socket
```

```

CONTENT = b"""
HTTP/1.0 200 OK

Hello #%d from MicroPython!
"""

def main():
    s = socket.socket()
    ai = socket.getaddrinfo("0.0.0.0", 8080)
    addr = ai[0][-1]

    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    s.bind(addr)
    s.listen(5)
    print("Listening, connect your browser to http://<this_host>:8080/")

    counter = 0
    while True:
        res = s.accept()
        client_s = res[0]
        req = client_s.recv(4096)
        print("Request:")
        print(req)
        client_s.send(CONTENT % counter)
        client_s.close()
        counter += 1
        print()

main()

```

נקבל את הפלט הבא:

```

>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Already connected('10.0.0.10', '255.255.255.0', '10.0.0.138', '10.0.0.138')
Listening, connect your browser to http://<this_host>:8080/

```

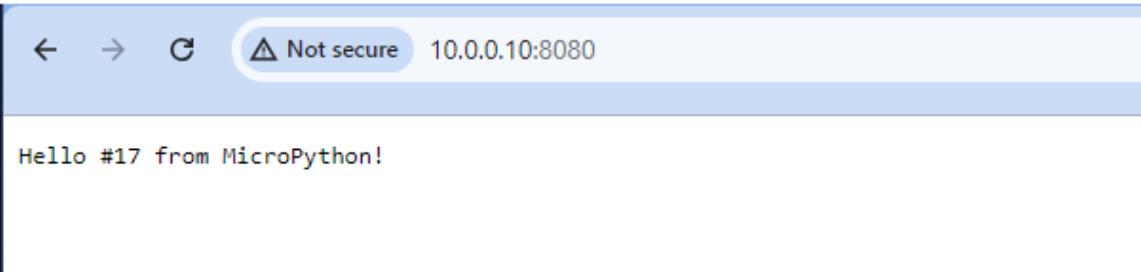
כדי להתחבר לשרת נפתח דף אינטרנט ונכתב את הכתובת

[http://<this\\_host>:8080/](http://<this_host>:8080/)

כasher במקום <this\_host> נכתבת את כתובת ה- IP כפי שרשומה במסך הפלט:

<http://10.0.0.10:8080/>

נקבל את הפלט הבא:



להלן קוד התוכנית עבור שרת HTTP בסיסי (גרסה 2):

```

import socket
def html_page():
    html = """
        <!DOCTYPE html>
        <html>
            <head>
                <meta content="width=device-width, initial-scale=1">
            </head>
            <body>
                <h1>Hello world!</h1>
                <p>I am ESP32 in a web server mode</p>
            </body>
        </html>
    """
    return html

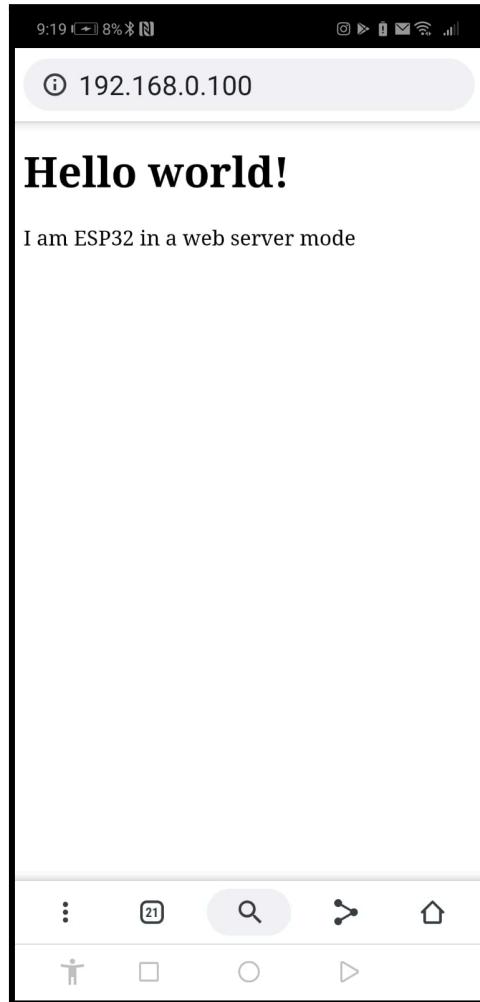
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.listen(5)
print("Listening, connect your browser to http://<this_host>")
while True:
    conn, addr = s.accept()
    print("Got a connection from %s" % str(addr))
    request = conn.recv(1024)
    print("Content = %s" % str(request))
    response = html_page()
    conn.send("HTTP/1.1 200 OK")
    conn.send("Content-Type: text/html; encoding=utf8\nContent-Length: ")
    conn.send(str(len(response)))
    conn.send("\nConnection: close\n")
    conn.send("\n")
    conn.send(response)
    conn.close()

```

נגלוש לבקר דרך כתובות ה- IP שלו ונקבל על מסך הדפסן את הפלט הבא:

# Hello world!

I am ESP32 in a web server mode



שילוב חומרה בבקר המפעיל שרת WEB.

נפתח תרגיל המפעיל נורית LED המחברות להדק 15 של הבקר תוך כדי קליטת הוראות דרך הדף.

להלן קוד התוכנית עברו הקובץ תוכנת השרת להפעלת נורית דרך האינטרנט:

```
from machine import Pin
import socket

led = Pin(2, Pin.OUT)

def web_page():
    if led.value() == 1:
        gpio_state="ON"
    else:
        gpio_state="OFF"
```

```

gpio_state="OFF"

html = """
<html>
    <head>
        <title>ESP32 HTTP Server</title>
        <meta name="viewport" content="width=device-width,
initial-scale=1">
        <style>
            html{
                display:inline-block;
                margin: 0px auto;
                text-align:
                center;}
            h1{
                color: #0F3376;
                padding: 2vh;}
            p{
                font-size: 1.5rem;}
            button{
                display: inline-block;
                background-color: #3668b8;
                border: none;
                border-radius: 4px;
                color: white;
                padding: 20px 30px;
                font-size: 25px;
            }
        </style>
    </head>
    <body>
        <h1>ESP32 HTTP Server</h1>
        <p>GPIO state: """ + gpio_state + """</p>
        <p><a href="/?led=on"><button>LED ON</button></a></p>
        <p><a href="/?led=off"><button>LED OFF</button></a></p>
    </body>
</html>
"""
return html

s = socket.socket()
ai = socket.getaddrinfo("0.0.0.0", 8080)
addr = ai[0][-1]

s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

s.bind(addr)
s.listen(5)
print("Listening, connect your browser to http://<this_host>:8080/")

while True:
    conn, addr = s.accept()
    print('Got a connection from %s' % str(addr))

```

```

request = conn.recv(1024)
request = str(request)
print('Content = %s' % request)
led_on = request.find('/?led=on')
led_off = request.find('/?led=off')
if led_on == 6:
    print('LED ON')
    led.value(1)
if led_off == 6:
    print('LED OFF')
    led.value(0)
response = web_page()
conn.send('HTTP/1.1 200 OK\n')
conn.send('Content-Type: text/html\n')
conn.send('Connection: close\n\n')
conn.sendall(response)
conn.close()

```

נקבל את הפלט הבא:

```

>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Already connected('10.0.0.10', '255.255.255.0', '10.0.0.138', '10.0.0.138')
Listening, connect your browser to http://<this_host>:8080/

```

כדי להתחבר לשרת נפתח דף אינטרנט ונכתב את הכתובת

[http://<this\\_host>:8080/](http://<this_host>:8080/)

כאשר במקום <this\_host> נכתבת את כתובת ה- IP כפוי שרשומה במסך הפלט:

<http://10.0.0.10:8080/>

נקבל את הפלט הבא:

## ESP32 HTTP Server

GPIO state: OFF

LED ON

LED OFF

## משימה 14 - HTTP GET

בפעילות זו נלמד כיצד להשתמש בבקר קלוקו WiFi. הבקר יוכל גישה לרשת האינטרנט דרך חיבור WiFi מקומי ובכך הוא יוכל לקבל ולשלוח נתונים לאינטרנט ממש כמו שדרפן במחשב או טלפון נייד לעשוות. כל התקשרות תעבור תחת בקשות GET בפרוטוקול HTTP דרך מפתח 80.

להעמקה בנושא פרוטוקול HTTP ניתן לקבל דרך הקישור הבא:

[https://he.wikipedia.org/wiki/HTTP\\_POST](https://he.wikipedia.org/wiki/HTTP_POST)

קישורים:

<https://randomnerdtutorials.com/esp32-esp8266-micropython-web-server/>

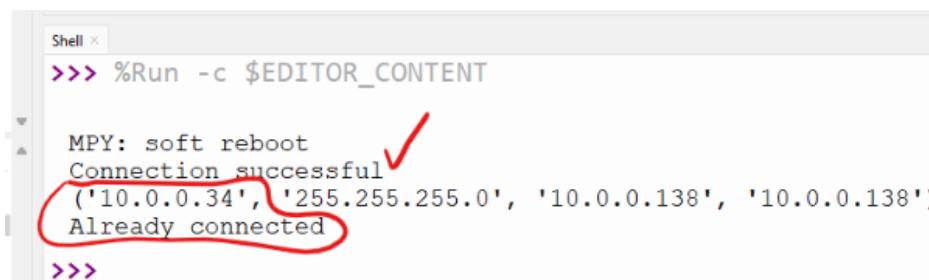
<https://techtutorialsx.com/2017/06/11/esp32-esp8266-micropython-http-get-requests/>

<https://github.com/micropython/micropython/tree/master/examples/network>

[https://github.com/micropython/micropython-esp32/tree/esp32/tests/net\\_inet](https://github.com/micropython/micropython-esp32/tree/esp32/tests/net_inet)

לפני ביצוע משימה זו יש לוודא שהבקר מחובר לרשת האינטרנט דרך WiFi כמפורט במשימה 12.

כדי לבדוק שהבקר מחובר לאינטרנט יש לאותחל את הבקר ולבדק שהשורה הבאה מופיע:

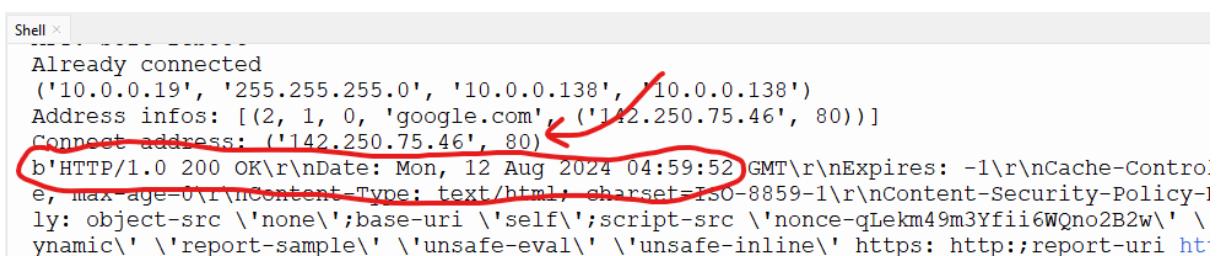


```
Shell >>> %Run -c $EDITOR_CONTENT
>>> MPY: soft reboot
Connection successful ✓
('10.0.0.34', '255.255.255.0', '10.0.0.138', '10.0.0.138')
Already connected
>>>
```

להלן קוד התוכנית התחבר לאתר של גוגל ומציג את המידע המתקבל כמחרוזת טקסט על מסך הטרמינל:

```
import socket
s = socket.socket()
ai = socket.getaddrinfo("google.com", 80)
print("Address infos:", ai)
addr = ai[0][-1]
print("Connect address:", addr)
s.connect(addr)
s.send(b"GET / HTTP/1.0\r\n\r\n")
print(s.recv(4096))
s.close()
```

פלט התוכנית יהיה כך:



```
Shell >>>
Already connected
('10.0.0.19', '255.255.255.0', '10.0.0.138', '10.0.0.138')
Address infos: [(2, 1, 0, 'google.com', ('142.250.75.46', 80))]
Connect address: ('142.250.75.46', 80) ←
b'HTTP/1.0 200 OK\r\nDate: Mon, 12 Aug 2024 04:59:52 GMT\r\nExpires: -1\r\nCache-Control: max-age=0\r\nContent-Type: text/html; charset=ISO-8859-1\r\nContent-Security-Policy: object-src \'none\';base-uri \'self\';script-src \'nonce-qLekm49m3Yfii6WQno2B2w\' \\'dynamic\' \\'report-sample\' \\'unsafe-eval\' \\'unsafe-inline\' https: http://report-uri ht...
```

מצד אחד ניתן לראות שבקר ESP32 מצליח להתחבר לכל אתר אינטרנט מצד שני לפי מבנה התשובה לא נראה שנית להשתמש בתנונים שהתקבלו באופן שימושי.

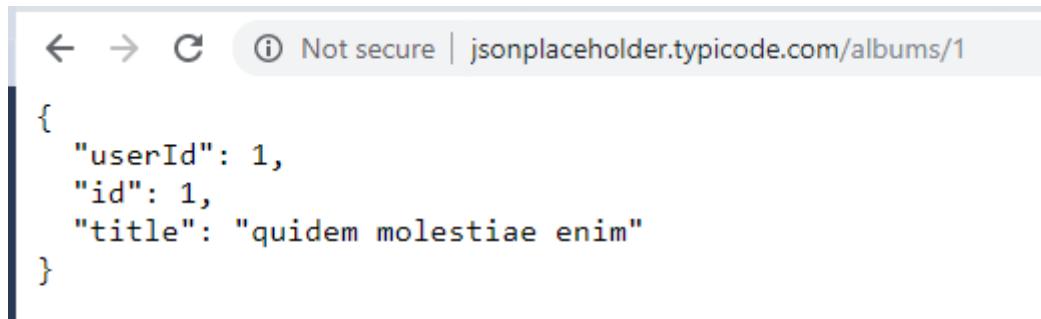
במטרה לנצל את יכולת הקישוריות לאינטרנט בצורה טוביה יותר נעשה שימוש בקבלת מבנה תנונים מסווג JSON הנראה כך:

```
{name:"Gadi", age:50, city:"Migdal HaEmek"}
```

ගלשו לכתובת האינטרנט הבא:

<http://jsonplaceholder.typicode.com/albums/1>

תקבלו את המסר הבא:



The screenshot shows a browser window with the URL "jsonplaceholder.typicode.com/albums/1". The page displays a single JSON object:

```
{  
  "userId": 1,  
  "id": 1,  
  "title": "quidem molestiae enim"  
}
```

בקר ESP32 מסוגל להתחבר לאתר כדי לקבל את מבנה הנתונים בפורמט JSON ולפרק אותו.

### שילוב JSON עם MicroPython

נדגים קוד העושה שימוש במחלקה JSON כדי לקרוא קובץ טקסט בשם `json.data` הכיל את תוכן הבא:

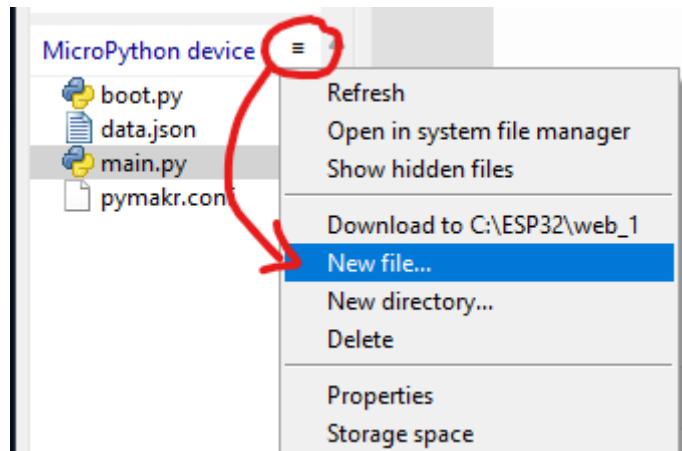
```
{"is_led_on": 1}
```

דוגמת הקוד מראה כיצד לקרוא את הקובץ. להמיר את תוכן שלו למבנה נתונים ולהשתמש מיד עם LED בהתאם לערך השמור בקובץ.

```
import json  
from machine import Pin  
  
# load the config file from flash  
f=open("data.json")  

```

נדגש שיש ליצור קובץ בשם `json.data` השמור בתוך הAKER. כדי לעשות זאת יש ללחוץ כמתואר בתמונה:



נשלב עכשו את הקוד המאפשר להתחבר לאתר ייחד עם הקוד שפענה JSON כדי להתחבר לאטר אינטרנט הכלול מידע בפורמט JSON לקרוא אותו ולהשתמש במידע שרשום בו. אך הפעם ניעיל את הקוד ונעשה שימוש במחלקה `urequests` המאפשרת גם להתחבר לאתר וגם לפענה את המידע בפורמט JSON. להלן הקוד:

```
import urequests
response = urequests.get('http://jsonplaceholder.typicode.com/albums/1')
print(type(response))
print(response.text)
print(type(response.text))
parsed = response.json()
print(type(parsed))
print(parsed["userId"])
print(parsed["id"])
print(parsed["title"])
```

ניתן לראות שהבוקר קלט את הנתונים והצליח לבדוק כל אחד מהם להמשך טיפול

```
MPY: soft reboot
Already connected
('10.0.0.19', '255.255.255.0', '10.0.0.138', '10.0.0.138')
<class 'Response'>
{
    "userId": 1,
    "id": 1,
    "title": "quidem molestiae enim"
}
<class 'str'>
<class 'dict'>
1
1
quidem molestiae enim
```

#### ישום שירות API לצורך לקבלת נתונים מזג אוויר

דמיינו שאתה במסעדה ואתם רוצח להזמין אוכל. במקומות למכת למטבח ולנסות להcin את האוכל בעצמך, אתה פשוט קוראים למלצר ומקשימים ממנו את מה שאתה רוצה. המלצר הוא כמו ה-`API`. הוא מקבל את הזמנה שלכם (הבקשה), מעביר אותה למטבח (השירות) ומחזיר לכם את האוכל (התשובה).

## از מה זה API בעצם?

API זה כמו שליח שמאפשר לישומים שונים לתקשר ביניהם ולהחליף מידע. במקומות שבהם ישום יבנה את כל התכונות שלו מ一封, הוא יכול להשתמש ב-API של ישום אחר כדי לקבל גישה לתוכנות האלה.

דוגמאות:

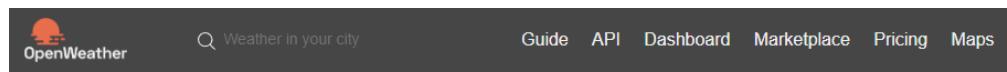
مפות גוגל: כשאתם משתמשים באפליקציה של מונית, האפליקציה משתמשת ב-API של מפות גוגל כדי להציג לכם את המיקום שלכם ואת המידע.

רשומות חברותיות: כשאתם נכנסים לאתר באמצעות חשבון פייסבוק, האתר משתמש ב-API של פייסבוק כדי לאמת את הזהות שלכם.

נדגים שירות API של <https://openweathermap.org> כדי לקבל נתונים מזג אוויר עכשוויים במקום מסוים.

כדי לקבל גישה לשירותי ה-API אנו זקוקים להירשם באתר ולקבל קוד ID ייחודי שב Zukunft נקבל גישה לנתוני מזג האוויר. שירות זה בחינם לשימושים בסיסיים כמו שאנו צריכים. נתחבר לאתר הבא:

<https://openweathermap.org/appid/>



### Best way to start and continue calling OpenWeather APIs

OpenWeather platform is a set of elegant and widely recognisable APIs. Powered by convolutional machine learning solutions, it is capable of delivering all the weather information necessary for decision making for any location on the globe. To start using our APIs, please sign up [here](#).

Why our Free Weather API is so good yet free

### How to call OpenWeather APIs with a freemium plan

The API key is all you need to call any of our weather APIs. Once you [sign up](#) using your email, the API key (APPID) will be sent to you in a confirmation email. Your API keys can always be found on your [account page](#), where you can also generate additional API keys if needed. Check our [documentation page](#) to find all technical information for each product. Documentation is an essential guide with actual examples and comprehensive description of API calls, responses and parameters.

לאחר ההרשמה נקבל קוד ID כמתואר כאן:

לאחר קבלת הקוד אנו יכולים לשלב אותו בכתובת אינטרנט URL ולקבל JSON הכלל את נתוני מזג האוויר, באופן הבא:

<https://api.openweathermap.org/data/2.5/weather?lat=32.811&lon=35.012&appid=99e5XXXXX6142>

כasher הרכים lat ו- lon מצינים את קו האורך והרוחב של הנקודה שבה אתם רוצים לקבל את נתוני מזג האוויר. ו- appid הוא הקוד ה- ID שלכם.

נדגים זאת על ידי גישה ישירה לכתובת תוך שימוש בדף:

נשלב עכשו את כל מה שלמדנו בפרק זה כדי לבדוק את נתוני מזג האוויר מתוך ה- JSON שקיבלנו:

```
import urequests
response =
urequests.get('https://api.openweathermap.org/data/2.5/weather?lat=32.000&lon=35.000&appid=99eXXXXXX4142')
#print(type(response))
#print(response.text)
#print(type(response.text))
parsed = response.json()
#print(type(parsed))
#print(parsed["main"])
#print(parsed["main"]["temp_max"] - 273.15)
print("temperature: ", parsed["main"]["temp"] - 273.15)
#print(parsed["main"]["temp_min"] - 273.15)
print("humidity: ", parsed["main"]["humidity"])
```

נקבל את הפלט הבא:

## משימה 15 - הפעלת צג גרפי דגם SSD1306 OLED display

במשימה זה תלמד כיצד להשתמש בתצוגת OLED SSD1306 בגודל 0.96 אינץ' המתחברת לבקר ESP32. במהלך המשימה נלמד להציג הודעות טקסט למרוחם שמדובר במסך גרפי.

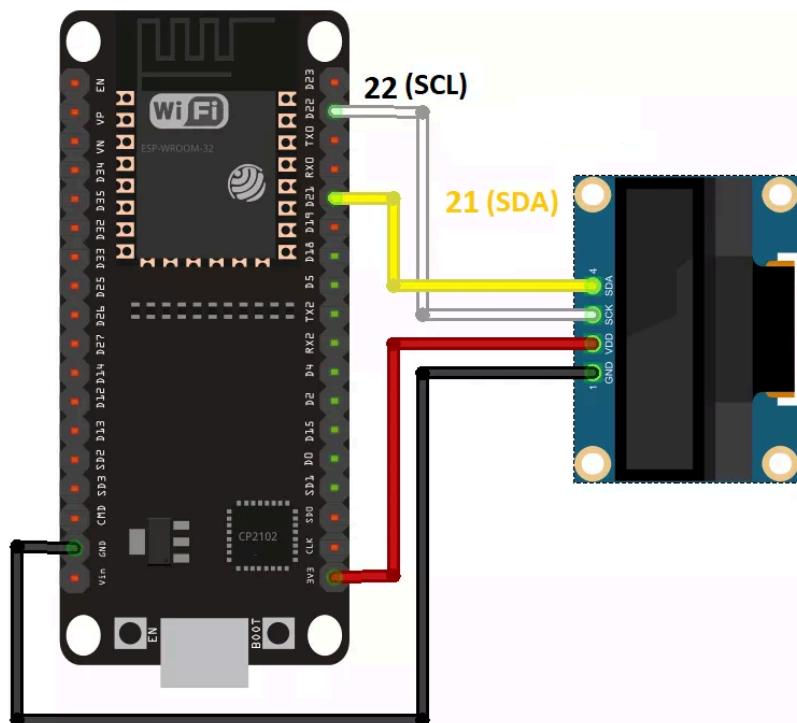
קישורים:

<https://randomnerdtutorials.com/micropython-oled-display-esp32-esp8266/>

חיבור הצג לבקר

OLED	ESP32
Vin	3.3V
GND	GND
SCL	GPIO 22
SDA	GPIO 21

شرطוט חשמלי של החיבור:



```

import time
import framebuf

# register definitions
SET_CONTRAST      = const(0x81)
SET_ENTIRE_ON     = const(0xa4)
SET_NORM_INV      = const(0xa6)
SET_DISP          = const(0xae)
SET_MEM_ADDR      = const(0x20)
SET_COL_ADDR      = const(0x21)
SET_PAGE_ADDR     = const(0x22)
SET_DISP_START_LINE = const(0x40)
SET_SEG_REMAP      = const(0xa0)
SET_MUX_RATIO      = const(0xa8)
SET_COM_OUT_DIR    = const(0xc0)
SET_DISP_OFFSET    = const(0xd3)
SET_COM_PIN_CFG    = const(0xda)
SET_DISP_CLK_DIV   = const(0xd5)
SET_PRECHARGE      = const(0xd9)
SET_VCOM_DESEL     = const(0xdb)
SET_CHARGE_PUMP    = const(0x8d)

class SSD1306:
    def __init__(self, width, height, external_vcc):
        self.width = width
        self.height = height
        self.external_vcc = external_vcc
        self.pages = self.height // 8
        # Note the subclass must initialize self.framebuf to a framebuffer.
        # This is necessary because the underlying data buffer is different
        # between I2C and SPI implementations (I2C needs an extra byte).
        self.poweron()
        self.init_display()

    def init_display(self):
        for cmd in (
            SET_DISP | 0x00, # off
            # address setting
            SET_MEM_ADDR, 0x00, # horizontal
            # resolution and layout
            SET_DISP_START_LINE | 0x00,
            SET_SEG_REMAP | 0x01, # column addr 127 mapped to SEG0
            SET_MUX_RATIO, self.height - 1,
            SET_COM_OUT_DIR | 0x08, # scan from COM[N] to COM0
            SET_DISP_OFFSET, 0x00,
            SET_COM_PIN_CFG, 0x02 if self.height == 32 else 0x12,
            # timing and driving scheme
            SET_DISP_CLK_DIV, 0x80,
            SET_PRECHARGE, 0x22 if self.external_vcc else 0xf1,
            SET_VCOM_DESEL, 0x30, # 0.83*Vcc
            # display
            SET_CONTRAST, 0xff, # maximum
            SET_ENTIRE_ON, # output follows RAM contents
            SET_NORM_INV, # not inverted
            # charge pump

```

```

        SET_CHARGE_PUMP, 0x10 if self.external_vcc else 0x14,
        SET_DISP | 0x01): # on
        self.write_cmd(cmd)
    self.fill(0)
    self.show()

def poweroff(self):
    self.write_cmd(SET_DISP | 0x00)

def contrast(self, contrast):
    self.write_cmd(SET_CONTRAST)
    self.write_cmd(contrast)

def invert(self, invert):
    self.write_cmd(SET_NORM_INV | (invert & 1))

def show(self):
    x0 = 0
    x1 = self.width - 1
    if self.width == 64:
        # displays with width of 64 pixels are shifted by 32
        x0 += 32
        x1 += 32
    self.write_cmd(SET_COL_ADDR)
    self.write_cmd(x0)
    self.write_cmd(x1)
    self.write_cmd(SET_PAGE_ADDR)
    self.write_cmd(0)
    self.write_cmd(self.pages - 1)
    self.write_framebuf()

def fill(self, col):
    self.framebuf.fill(col)

def pixel(self, x, y, col):
    self.framebuf.pixel(x, y, col)

def scroll(self, dx, dy):
    self.framebuf.scroll(dx, dy)

def text(self, string, x, y, col=1):
    self.framebuf.text(string, x, y, col)

class SSD1306_I2C(SSD1306):
    def __init__(self, width, height, i2c, addr=0x3c, external_vcc=False):
        self.i2c = i2c
        self.addr = addr
        self.temp = bytearray(2)
        # Add an extra byte to the data buffer to hold an I2C data/command byte
        # to use hardware-compatible I2C transactions. A memoryview of the
        # buffer is used to mask this byte from the framebuffer operations
        # (without a major memory hit as memoryview doesn't copy to a separate
        # buffer).
        self.buffer = bytearray(((height // 8) * width) + 1)
        self.buffer[0] = 0x40 # Set first byte of data buffer to Co=0, D/C=1
        self.framebuf = framebuffer.FrameBuffer1(memoryview(self.buffer)[1:], width,
height)
        super().__init__(width, height, external_vcc)

```

```

def write_cmd(self, cmd):
    self.temp[0] = 0x80 # Co=1, D/C#=0
    self.temp[1] = cmd
    self.i2c.writeto(self.addr, self.temp)

def write_framebuf(self):
    # Blast out the frame buffer using a single I2C transaction to support
    # hardware I2C interfaces.
    self.i2c.writeto(self.addr, self.buffer)

def poweron(self):
    pass

class SSD1306_SPI(SSD1306):
    def __init__(self, width, height, spi, dc, res, cs, external_vcc=False):
        self.rate = 10 * 1024 * 1024
        dc.init(dc.OUT, value=0)
        res.init(res.OUT, value=0)
        cs.init(cs.OUT, value=1)
        self.spi = spi
        self.dc = dc
        self.res = res
        self.cs = cs
        self.buffer = bytearray((height // 8) * width)
        self.framebuf = framebuffer.FrameBuffer1(self.buffer, width, height)
        super().__init__(width, height, external_vcc)

    def write_cmd(self, cmd):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs.high()
        self.dc.low()
        self.cs.low()
        self.spi.write(bytearray([cmd]))
        self.cs.high()

    def write_framebuf(self):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs.high()
        self.dc.high()
        self.cs.low()
        self.spi.write(self.buffer)
        self.cs.high()

    def poweron(self):
        self.res.high()
        time.sleep_ms(1)
        self.res.low()
        time.sleep_ms(10)
        self.res.high()

```

תוכן הקובץ `yd.run`:

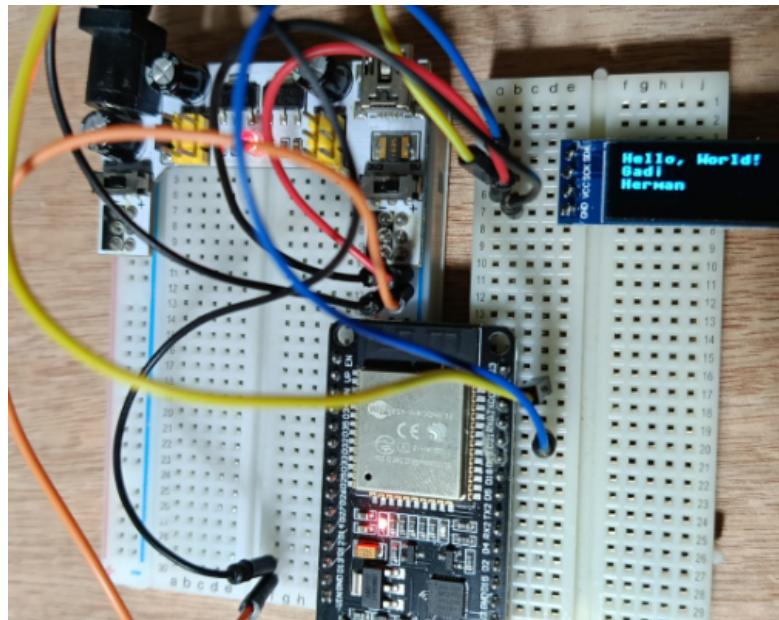
```
from machine import Pin, I2C
import ssd1306
from time import sleep

# using default address 0x3C
i2c = I2C(scl=Pin(22), sda=Pin(21))
oled_width = 128
oled_height = 32
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)

oled.text('Hello, World!', 0, 0)
oled.text('Gadi', 0, 10)
oled.text('Herman', 0, 20)

oled.show()
```

להלן הפלט על הצג:



## משימה 16 - שירותי ענן מבוססי MQTT

קישורים:

<https://github.com/miketeachman/micropython-adafruit-mqtt-esp8266>

<https://io.adafruit.com/api/docs/mqtt.html#adafruit-io-mqtt-api>

<https://www.emqx.com/en/blog/micro-python-mqtt-tutorial-based-on-raspberry-pi>

<https://www.hackster.io/mark-yu/air-quality-system-with-beebotte-and-ifttt-2922c7>

MQTT - Message Queuing Telemetry Transport הוא פרוטוקול תקשורת קל משקל המשמש בעיקר לתקשורת בין מכשירים באינטרנט (TCP). להלן תיאור העקרונות שלו:

1. MQTT פועל על בסיס מודל "פרסום-הרשם" (publish-subscribe). במודל זה, מכשירים יכולים לשלוח ("פרסם") הודעות לנושאים (Topics), ומיכשרים אחרים יכולים "להירשם" (subscribe) לנושאים (Topics) שמשמעותם אותם כדי לקבל את ההודעות הללו.

2. רכיבים עיקריים:

לקוחות: אלו הם המכשירים או התוכנות שמספרמים הודעות או נרשמיים לקבלתן.

ברוקר (מתווך): זהו השירות המרכזי שמנהל את כל התקשורת בין הלקוחות.

3. נושאים (Topics):

כל הודעה ב-MQTT משיכת לנושא מסוים.

נושאים (Topics) מאורגנים בצורה היררכית, כמו מבנה תיקיות.

לדוגמה: "בית/סלון/טפרטורה" יכול להיות נושא למדידת טפרטורה בסלון.

4. יתרונות:

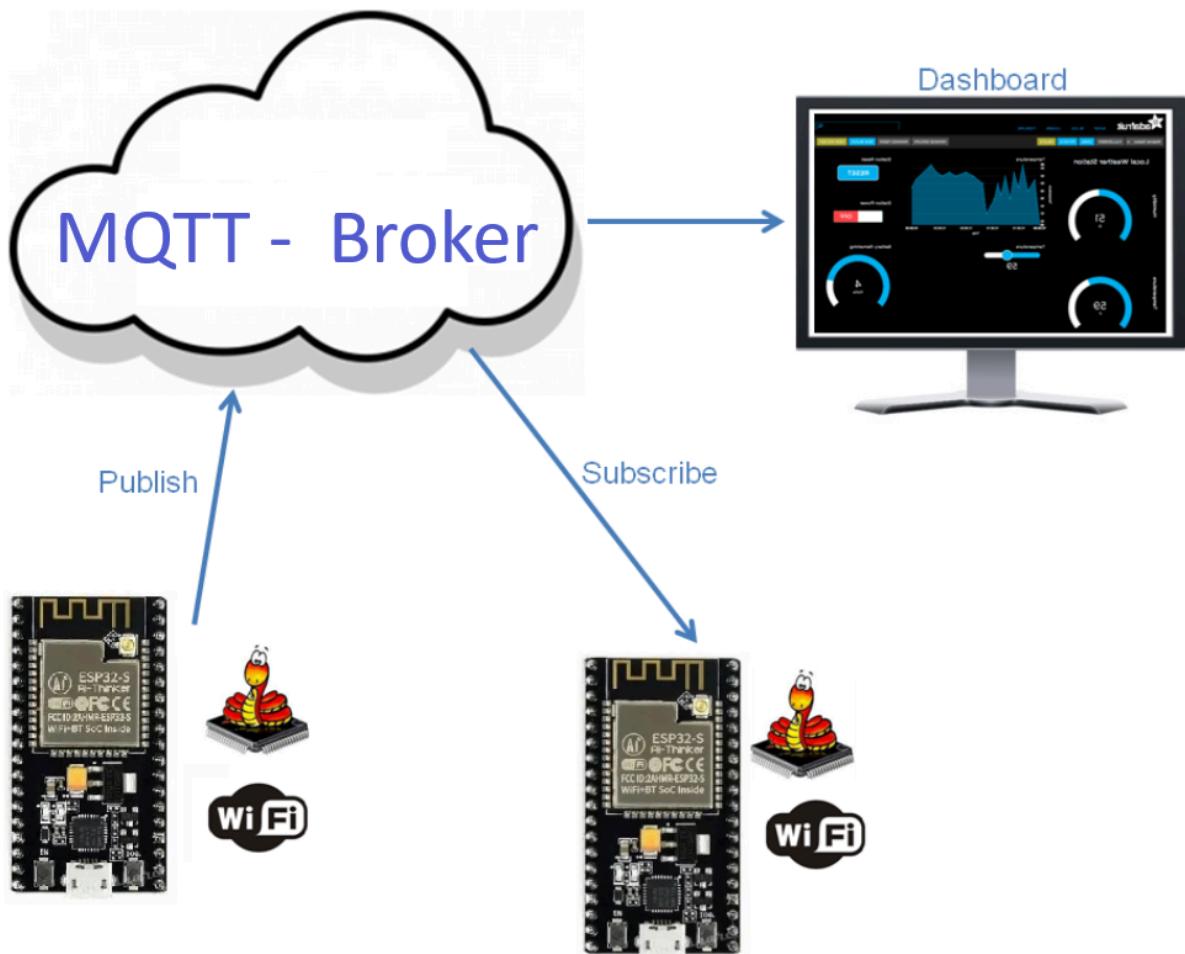
הפרוטוקול צריך מעט משאבי רשות וחומרה, מה שמתאים מאד למיכשי IoT קטנים.

מתאים למיכשרים המופעלים על סוללה.

מאפשר תקשורת כמעט בזמן אמת.

כולל מנגנונים להבטחת העברת הודעות גם ברשות לא יציבות.

5. שימושים נפוצים: מערכות בית חכם, ניטור מכונות בתעשייה, מעקב אחר צי רכבים, איסוף נתונים מחישנים מרוחקים.



**יצירת חשבון** באתר [beebotte.com](https://beebotte.com)

באתר beebotte הוא שירות ענן המיועד לפROYקטים של אינטרנט הדברים (IoT). להלן סקירה כללית על השירות שמציע האתר:

- מאפשרת לחבר מכשיר IoT לענן וניהל אותו מרוחק.
- מאפשר ליצור ממשקי משתמש מותאמים אישית לשיליטה וניהול של מכשיר IoT. כולל אלמנטים כמו כפתורים, מד מתח וגרפים.
- מאפשר לאחסן וניהל נתונים מחישנים או מכשירים.
- מספק ברוקר MQTT לתקשורת ייעילה בין מכשירים.
- מאפשר להציג פעולות אוטומטיות בהתאם על תנאים מסוימים. לדוגמה, שליחת התראה כאשר טמפרטורה עולה מעל סף מסוים.
- מציע תוכנית חינמית עם מגבלות מסוימות, ותוכניות בתשלום עם יכולות נרחבות יותר.

נתחבר לאתר הבא ונפתח בו חשבון משתמש:

<https://beebotte.com/>

לאחר התחברות מוצלחת נקבל את המסר הבא:

The screenshot shows the 'My Channels' section of the beebootte.com interface. On the left is a sidebar with links: Channels, Dashboards, Beerules (beta), Console, Account Settings, Account Usage, and Support. The main area displays four channels:

- esp32**: Created by gadiherman, Private, Created: August 24th 2024.
- Multipass1**: Created by gadiherman, Private, Created: August 23rd 2024.
- temp1**: Created by gadiherman, Private, Created: August 23rd 2024.
- Multipass**: Created by gadiherman, Private, Created: August 23rd 2024.

### שימוש באתר beebootte.com במטרה לשלוט על רכיב חומרה

בחלק זה של הפעולות נדגים כיצד יוצריםلوح בקירה הכלול לחץ שמדליק ומכבה LED בחומרה.

নিচৰ তচিলা ছাই নিচৰ তচিলা ছাই

The screenshot shows the 'Create a new channel' form. The sidebar on the left is identical to the previous one. The main form has the following fields:

- Name:** els32 (highlighted with a red circle)
- Description:** Channel Description (empty)
- Public:**  Public (unchecked)
- Configured Resources:**
  - Resource Name:** led (highlighted with a red circle)
  - Type:** boolean (highlighted with a red circle)
  - SoS:**  SoS (highlighted with a red circle)
- Buttons:** Cancel (gray) and Create channel (green, highlighted with a red circle)

בשלב הבא ניצורلوح בקירה חדש בשם LedControl

Dashboards

Create and manage your dashboards.

TITLE	DESCRIPTION	CREATED ON	SCOPE	VIEWS
My Dashboard		August 21st 2024	Public	64

Showing 1 to 1 of 1 entries

Search:

Previous 1 Next

נכנו לעיצוב הממשק על ידי לחיצה של Add Widget

New Dashboard

Description

Public

+ Add Widget

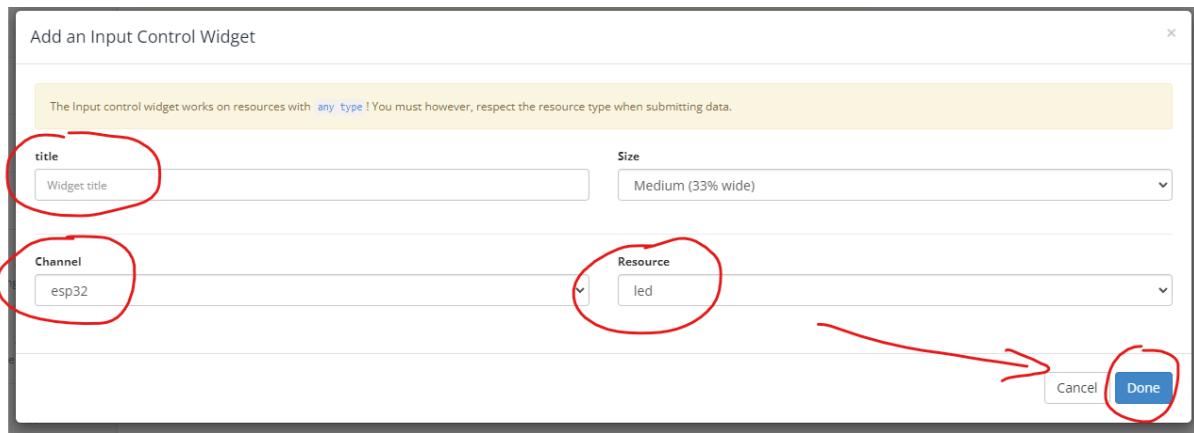
Add Widget Save Changes

נוסף בו לחץן על ידי לחיצה על Create New Block

+ Add Widget

- Public
- Clock
- Text Area
- Basic Value
- Gauge meter
- Timeline Chart
- Multi-line Chart
- Table Chart
- Heat Map
- Google Maps Beta
- Tracker Beta
- On/Off Beta

בשלב הבא נקשר בין הלחצן שייצרנו עם ה- Channel שמו led/esp32 שיצרנו מקודם:



נקבל את הlion הבא:

הדבר האחרון שהוא נדרש כדי לחבר בין האתר שמספק לנו שירות ענן לבין הילד שמחובר לבקר ESP32 הוא קוד הגישה הייחודי ל- Channel. נקבל אותו על ידי ה- Channel Token.

נפתח את סביבת התוכנות כדי לכתוב את הקוד הבא:

הקובץ boot.py

```
# This file is executed on every boot (including wake-boot from deepsleep)
import network

def connect():
    ssid = "XXXX"
    password = "XXXX"
    station = network.WLAN(network.STA_IF)
    if station.isconnected() == True:
        print("Already connected")
        return
```

```

station.active(True)
station.connect(ssid, password)
while station.isconnected() == False:
    pass
print("Connection successful")
print(station.ifconfig())
connect()

```

להלן הגרסת ה-CI פשוטה שנייתן לכתוב ב.tk (גרסת למודית).

הקובץ main.py

```

from umqtt.simple import MQTTClient
import ujson
import utime

def callback_func(topic, msg):
    print((topic, msg))

client = MQTTClient("test_mqtt_client_id", 'mqtt.beebotte.com',
user='token:token_jraXXXXXXXXXXXXku', password='', keepalive=30)
client.connect()
client.set_callback(callback_func)
client.subscribe('esp32/led')

while True:
    print("Checking msg...")
    client.check_msg()
    utime.sleep(1)

```

לאחר צריבת הרכיב וחיבורו לאינטרנט על ידי תקשורת WiFi בכל פעם שנלחץ על הלוחץ שבאטור נקלט ב.tk את ההודעה הבא:

The screenshot shows a terminal window with the title 'Shell'. The command `>>> %Run -c \$EDITOR\_CONTENT` is entered, followed by the previously shown Python code for connecting to a MQTT broker and handling LED messages. The output shows the connection being established to IP '10.0.0.10' and receiving periodic messages from topic 'esp32/led' indicating the state of the LED.

```

Shell <
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Already connected
('10.0.0.10', '255.255.255.0', '10.0.0.138', '10.0.0.138')
Checking msg...
(b'esp32/led', b'{"data":false,"ts":1724509348276,"ispublic":false}')
Checking msg...

```

כל מה שנשאר לעשות זה להוסיף לפועלה callback\_func את הקוד הבא כדי להפעיל בפועל את ה- LED בהתאם לפקודה שקיבל. נדגים זאת:

```
from machine import Pin
from umqtt.simple import MQTTClient
import ujson
import utime

LED = Pin(2, Pin.OUT)

def callback_func(topic, msg):
    print((topic, msg))
    json_data= ujson.loads(msg)
    dt= json_data["data"]
    if str(dt) == 'True':
        LED.value(1)
    if str(dt) == 'False':
        LED.value(0)

client = MQTTClient("test_mqtt_client_id", 'mqtt.beebotte.com',
user='token:token_jraXXXXXXXXXUku', password='', keepalive=30)
client.connect()
client.set_callback(callback_func)
client.subscribe('esp32/led')

while True:
    print("Checking msg...")
    client.check_msg()
    utime.sleep(1)
```

להלן גרסה יציבה יותר לתקשורת. גרסה המיועדת לשילוב בפרויקטים:

```
from machine import Pin, Timer
from umqtt.simple import MQTTClient
import ujson
import sys
import os

LED = Pin(2, Pin.OUT)
PING_PERIOD = 120

CHANNEL_TOKEN = 'token_jraXXXXXXXXXUku'
CHANNEL_NAME = 'esp32'
RESOURCE_NAME = 'led'
MQTT_SERVER = 'mqtt.beebotte.com'
MQTT_USER = 'token:' + CHANNEL_TOKEN
MQTT_TOPIC = CHANNEL_NAME + '/' + RESOURCE_NAME

def handleTimerInt(timer):
    client.ping()
    print('ping')
```

```

def callback_func(topic, msg):
    print("topic:",topic," msg:", msg)
    json_data= ujson.loads(msg)
    dt= json_data["data"]
    print("*** " + str(dt) + " ***")
    if dt:
        LED.value(1)
    else:
        LED.value(0)

# create a random MQTT clientID
random_num = int.from_bytes(os.urandom(3), 'little')
mqtt_client_id = bytes('client_'+str(random_num), 'utf-8')

client = MQTTClient(mqtt_client_id, MQTT_SERVER, user=MQTT_USER,
password='', keepalive=PING_PERIOD*2 )

myTimer = Timer(0)

try:
    client.connect()
    myTimer.init(period=PING_PERIOD*1000, mode=Timer.PERIODIC,
callback=handleTimerInt)
except Exception as e:
    print('could not connect to MQTT server {}'.format(type(e).__name__, e))
    sys.exit()

client.set_callback(callback_func)
client.subscribe(MQTT_TOPIC)

while True:
    try:
        client.wait_msg()
    except KeyboardInterrupt:
        print('Ctrl-C pressed...exiting')
        client.disconnect()
        sys.exit()

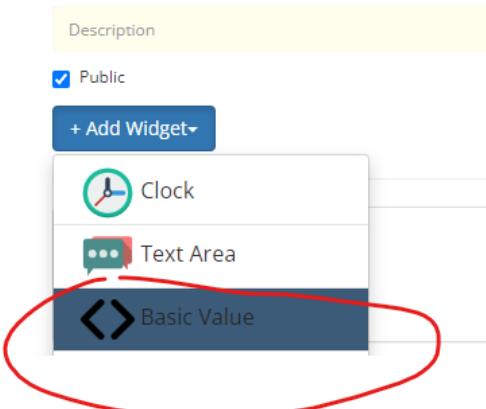
```

### שימוש באתר [beebotte.com](http://beebotte.com) במטרה לקלוט מידע מחיישן המחבר לבקר

בחלק זה של הפעולות נדגים כיצד יוצריםلوح בקירה הכלול משק גրפי המציג טמפרטורה הנקלטת מחיישן המחבר לבקר ESP32.

ניצור תחילה Channel נוסף בשם חדש כדוגמת `esp32_new/Sensor`.  
אפשרות נוספת היא לעדכן את ה- Channel שיצרנו עבור ה- LED ולהוסיף לו RESOURCE חדש. כמוzeigt באירוע:

בשלב הבא נווסף לאותו לוח בקרה שיצרנו כבר עבור ה- LED את הרכיב הבא:



נקבל לוח בקרה הכלל את הרכיבים הבאים:

בשלב הבא נכתוב את הקוד הבא:

```
from machine import Timer
from umqtt.simple import MQTTClient
import utime
import os
import sys

PUBLISH_PERIOD    = 10
msgNumber = 12.24
timeSave = 0

CHANNEL_TOKEN = 'token_jraXXXXXXXXXku'
CHANNEL_NAME   = 'esp32'
RESOURCE_NAME  = 'sensor'
MQTT_SERVER = 'mqtt.beebotte.com'
MQTT_USER = 'token:' + CHANNEL_TOKEN
MQTT_TOPIC = CHANNEL_NAME + '/' + RESOURCE_NAME

def handleTimerInt(timer):
    global msgNumber
    global timeSave
    msg = b'{"data": ' + str(msgNumber) + b', "write": true}'
    client.publish(MQTT_TOPIC,msg, qos=0)
    print("Publish:",msg)
    msgNumber += 1.12
    timeSave = 0

# create a random MQTT clientID
random_num = int.from_bytes(os.urandom(3), 'little')
mqtt_client_id = bytes('client_'+str(random_num), 'utf-8')

client = MQTTClient(mqtt_client_id, MQTT_SERVER, user=MQTT_USER,
password='', keepalive=PUBLISH_PERIOD*2)

myTimer = Timer(0)

try:
    client.connect()
    myTimer.init(period=PUBLISH_PERIOD*1000, mode=Timer.PERIODIC,
callback=handleTimerInt)
except Exception as e:
    print('could not connect to MQTT server {}'.format(type(e).__name__, e))
    sys.exit()

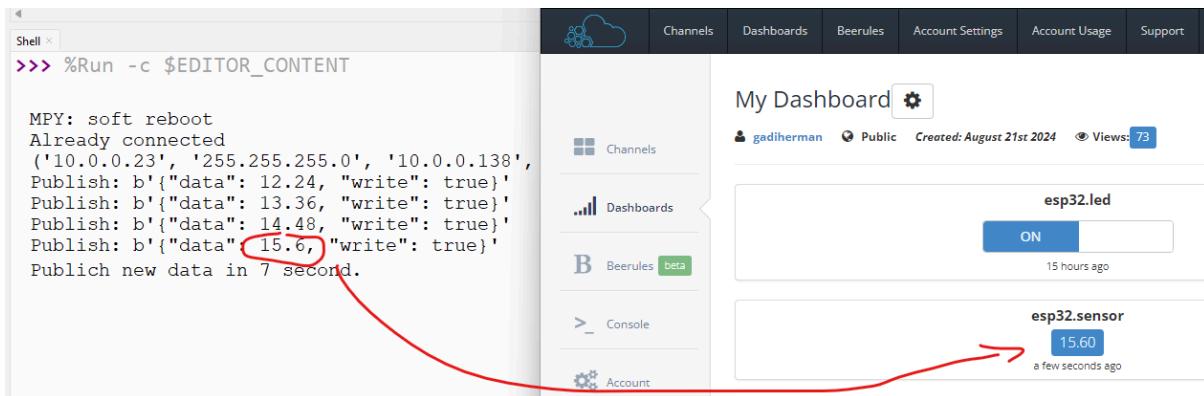
while True:
    try:
        print('Publish new data in', (PUBLISH_PERIOD-timeSave), "second.",
", end="\r")
        utime.sleep(1)
        timeSave+=1
    except KeyboardInterrupt:
        print('Ctrl-C pressed... exiting')
```

```

client.disconnect()
sys.exit()

```

נקבל את הפלט הבא:



### רישום בקר ESP32 לקלטת מידע מבקר אחר

כפי שראינו ב-2 הדוגמאות הקודומות בקר ESP32 מסוגל להירשם לשירות קבלת נתונים. לדוגמה לקבל מידע מלחצן הממוקם במכשיר המשמש שבאטר. כמו כן ראיינו שהקר ESP32 מסוגל גם לספק מידע עבור אחרים שנרשמו למידע. לדוגמה ראיינו קוד המספק למכשיר המשמש שבאטר מידע על טמפרטורה. ניתן לרשום בקר אחד שיספק את המידע ובקר אחר יקבל את המידע (כמובן דרך שירות הענן).

הקוד שמספק את המידע כבר הודגם בדוגמה השנייה. להלן דוגמה לתוכנה בברcker נוספת הנרשם לקבל מידע מהברcker הראשון. במצב זה ניקבל מצב שבקר אחד מספק לענן מידע על הדלקה וכיבוי הנורית הלד ובקר שני מקבל את המידע על הנוריות מהענן ומדליך LED. להלן הקוד:

```

from machine import Timer
from umqtt.simple import MQTTClient
import utime
import os
import sys

PUBLISH_PERIOD    = 10
Led = True
timeSave = 0

CHANNEL_TOKEN = 'token_jraxXXXXXXXXXUku'
CHANNEL_NAME   = 'esp32'
RESOURCE_NAME  = 'led'
MQTT_SERVER = 'mqtt.beebotte.com'
MQTT_USER = 'token:' + CHANNEL_TOKEN
MQTT_TOPIC = CHANNEL_NAME + '/' + RESOURCE_NAME

def handleTimerInt(timer):
    global Led
    global timeSave
    msg1 = b'{"data":true,"ispublic":false}'
    msg2 = b'{"data":false,"ispublic":false}'
    if Led:
        client.publish(MQTT_TOPIC,msg1, qos=0)
        print("Publish:",msg1)
    else:
        client.publish(MQTT_TOPIC,msg2, qos=0)
        print("Publish:",msg2)
    timeSave = utime.time()

client = MQTTClient('esp32', MQTT_SERVER, user=MQTT_USER, password=None)
client.set_callback(handleTimerInt)
client.connect()
client.subscribe(CHANNEL_NAME + '/' + RESOURCE_NAME)

# Set up a timer to publish every PUBLISH_PERIOD seconds
timer = Timer()
timer.init(freq=1/PUBLISH_PERIOD, mode=Timer.PERIODIC, callback=handleTimerInt)

# Main loop
while True:
    utime.sleep(1)

```

```

else:
    client.publish(MQTT_TOPIC, msg2, qos=0)
    print("Publish:", msg2)
Led = not Led
timeSave = 0

# create a random MQTT clientID
random_num = int.from_bytes(os.urandom(3), 'little')
mqtt_client_id = bytes('client'+str(random_num), 'utf-8')

client = MQTTClient(mqtt_client_id, MQTT_SERVER, user=MQTT_USER,
password='', keepalive=PUBLISH_PERIOD*2)

myTimer = Timer(0)

try:
    client.connect()
    myTimer.init(period=PUBLISH_PERIOD*1000, mode=Timer.PERIODIC,
callback=handleTimerInt)
except Exception as e:
    print('could not connect to MQTT server {}'.format(type(e).__name__, e))
    sys.exit()

while True:
    try:
        print('Publish new data in', (PUBLISH_PERIOD-timeSave), "second.",
", end="\r")
        utime.sleep(1)
        timeSave+=1
    except KeyboardInterrupt:
        print('Ctrl-C pressed... exiting')
        client.disconnect()
        sys.exit()

```

נקבל את הפלט הבא:

```

18 def handleTimerIn():
19     global Led
20     global timeSa
21     msg1 = b'{"da'
22     msg2 = b'{"da'
23     if Led:
24         client.pu
25         print("Pu
26     else:
27         client.pu
28         print("Pu
29     Led = not Led
30     timeSave = 0
31

Shell
18
19 Publish: b'{"data":false,"ispublic":false}'
20 Publish: b'{"data":false,"ispublic":false}'
21 Publish: b'{"data":true,"ispublic":false}'
22 Publish: b'{"data":false,"ispublic":false}'
23 Publish: b'{"data":true,"ispublic":false}'
24 Publish: b'{"data":false,"ispublic":false}'
25 Publish: b'{"data":true,"ispublic":false}'
26 Publish: b'{"data":true,"ispublic":false}'
27 Publish: b'{"data":true,"ispublic":false}'
28 Publish: b'{"data":true,"ispublic":false}'
29 Publish: b'{"data":true,"ispublic":false}'
30 Publish: b'{"data":true,"ispublic":false}'
31 Publish: b'{"data":true,"ispublic":false}'

My Dashboard
gadiherman Public Created: August 21st 2024 Views: 77

Channels
Dashboards
Beerules beta
Console
Account Settings

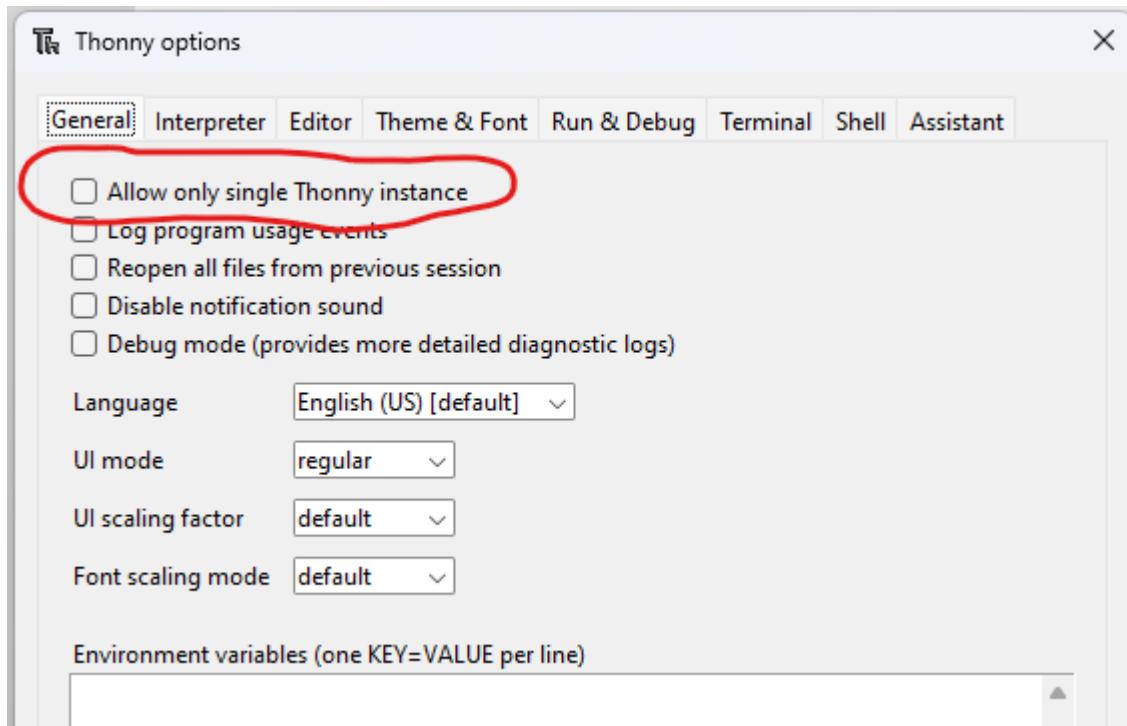
esp32.led
ON
a few seconds ago

esp32.sensor
12.24
4 hours ago

```

## טיפ!

כדי לפתח מספר חלונות של סביבת הפיתוח Thonny שבעל חלון נעובד מול בקר ESP32 אחר. יש לcliffe להגדרות של סביבת העבודה "Tools => General" וaz לבטל את הסימן כפ' שמופיע באIOR הבא:



### כתיבת קוד לבקר ESP32 ועובד ב-duplex (מפרסם ומתקבל מידע בו בזמןית)

כפי שראינו ב-3 הדוגמאות הקודמות בקר ESP32 מסוגל להירשם לשירות קבלת נתונים. לדוגמה לקבל מידע מלחצן הממוקם במכשיר המשמש שבאטר. כמו כן ראיינו שבקר ESP32 מסוגל גם לספק מידע עבור אחרים שנרשמו למידע. לדוגמה ראיינו קוד המספק למכשיר המשמש שבאטר מידע על טמפרטורה.

בדוגמה שלהן נראה כיצד נכתב קוד המפרסם מידע על טמפרטורה ובו זמנית נרשם לשירות הגורם להפעלת טורית ה-LED. להלן הקוד:

```
from machine import Pin, Timer
from umqtt.simple import MQTTClient
import ujson
import sys
import os
from time import sleep

LED = Pin(2, Pin.OUT)
PING_PERIOD = 120

PUBLISH_PERIOD = 10
msgNumber = 12.24

CHANNEL_TOKEN = 'token_jXXXXXXXXXXXXXXXXXXXX'
CHANNEL_NAME = 'esp32'
```

```

MQTT_SERVER = 'mqtt.beebotte.com'
MQTT_USER = 'token:' + CHANNEL_TOKEN

RESOURCE_NAME_PUBLISH = 'sensor'
RESOURCE_NAME_SUBSCRIBE = 'led'
MQTT_TOPIC_PUBLISH = CHANNEL_NAME + '/' + RESOURCE_NAME_PUBLISH
MQTT_TOPIC_SUBSCRIBE = CHANNEL_NAME + '/' + RESOURCE_NAME_SUBSCRIBE

def handleTimer0Int(timer):
    client.ping()
    print('ping')

def handleTimer1Int(timer):
    global msgNumber
    global timeSave
    msg = b'{"data": ' + str(msgNumber) + b', "write": true}'
    client.publish(MQTT_TOPIC_PUBLISH, msg, qos=0)
    print("Publish:", msg)
    msgNumber += 1.12
    timeSave = 0

def callback_func(topic, msg):
    print("topic:", topic, " msg:", msg)
    json_data = ujson.loads(msg)
    dt = json_data["data"]
    print("*** " + str(dt) + " ***")
    if dt:
        LED.value(1)
    else:
        LED.value(0)

# create a random MQTT clientID
random_num = int.from_bytes(os.urandom(3), 'little')
mqtt_client_id = bytes('client' + str(random_num), 'utf-8')

client = MQTTClient(mqtt_client_id, MQTT_SERVER, user=MQTT_USER,
password='', keepalive=PING_PERIOD*2)

timer0 = Timer(0) #subscribe
timer1 = Timer(1) #publish

try:
    client.connect()
    timer0.init(period=PING_PERIOD*1000, mode=Timer.PERIODIC,
callback=handleTimer0Int)
    timer1.init(period=PUBLISH_PERIOD*1000, mode=Timer.PERIODIC,
callback=handleTimer1Int)
except Exception as e:
    print('could not connect to MQTT server {}'.format(type(e).__name__, e))
    sys.exit()

client.set_callback(callback_func)
client.subscribe(MQTT_TOPIC_SUBSCRIBE)

```

```

while True:
    try:
        client.wait_msg()
    except KeyboardInterrupt:
        print('Ctrl-C pressed... exiting')
        client.disconnect()
        sys.exit()

```

נקבל את הפלט הבא:

The screenshot shows a split-screen interface. On the left is a code editor with Python code for an MQTT client. On the right is a Beebotte dashboard showing two channels: 'esp32.led' and 'esp32.sensor'. The 'esp32.led' channel has a value of 'ON' with a timestamp 'a few seconds ago'. The 'esp32.sensor' channel has a value of '14.48' with a timestamp 'a few seconds ago'. Red arrows point from the 'ON' state of the LED and the sensor value to the corresponding sections in the terminal window below, which show the publish and subscribe messages respectively.

```

msg = b'{"data": ' + str(msgNumber) + b'}', "write": true}
client.publish(MQTT_TOPIC_PUBLISH, msg, qos=0)
print("Publish:", msg)
msgNumber += 1.12
timeSave = 0

def callback_func(topic, msg):
    print("topic:", topic, " msg:", msg)
    json_data = ujson.loads(msg)
    dt = json_data["data"]
    print("**** " + str(dt) + " ****")
    if dt:
        LED.value(1)
    else:
        LED.value(0)

# create a random MQTT clientID
random_num = int.from_bytes(os.urandom(3), 'little')

Shell >
7273,"ispublic":true}'
*** True ***
Publish: b'{"data": 12.24, "write": true}'
topic: b'esp32/led'  msg: b'{"data":false,"ispublic":true}'
*** False ***
topic: b'esp32/led'  msg: b'{"data":true,"ispublic":true}'
*** True ***
Publish: b'{"data": 13.36, "write": true}'
Publish: b'{"data": 14.48, "write": true}'

```

ניתן לראות כיצד אותו קוד מאפשר גם להירשם subscribe לשירות esp32.led ובאותו הזמן הקוד מספק כל 10 שניות פרסום publish לשירות esp32.sensor.

שימוש לב שני השירותים יושבים על אותו הערוץ CHANNEL. CHANNEL. קרן:

## esp32

This screenshot shows the configuration page for the 'esp32' service. At the top, it displays the owner 'gadiherman', the visibility 'Public', and the creation date 'Created: August 24th 2024'. Below this is a 'Channel Token' input field containing 'tc'. The main area is titled 'Configured resources' and lists two resources: 'led' and 'sensor'. The 'led' resource is set to 'true' and was updated '5 minutes ago'. The 'sensor' resource has a value of '46.96' and was updated 'a few seconds ago'. A note at the bottom states: 'Note: Using the same channel for both services'.

Resource	Value	Last Update
led	true	5 minutes ago
sensor	46.96	a few seconds ago

Note: Using the same channel for both services

## יצירת חשבון באתר [io.adafruit.com](http://io.adafruit.com)

IO Adafruit הוא שירות ענן המועד לפROYיקטים של אינטרנט של הדברים (IoT). להלן סקירה כללית על השירות שמציע האתר:

- מאפשרת לחבר מכשיר IoT לענן ולנהל אותם מרוחוק.
- תומך ב\_\_). מגוון רחב של התקני חומרה, במיוחד מוצריים של Adafruit.
- מאפשר ליצור ממשק משתמש מותאם אישית לשימושו וניהול של מכשיר IoT. כולל אלמנטים כמו כפתורים, מדים דיגיטליים, גרפים ועוד.
- מאפשר לאחסן ולנהל נתונים מיישנים או מכשירים.
- ניתן להציג את הנתונים בצורה גרפית או ליצא אותם.
- מספק ברוקר MQTT לתקשורת יבילה בין מכשירים.
- מאפשר להציג פעולות אוטומטיות בהתאם מסוימים. לדוגמה, שליחת התראה כאשר טמפרטורה עולה מעל סף מסוים.
- מציע תוכנית חינמית עם מגבלות מסוימות, ותוכניות בתשלום עם יכולות נרחבות יותר.

נתחבר לאתר הבא ונפתח בו חשבון משתמש:

<https://io.adafruit.com/>

לאחר התחברות מוצלחת נקבל את המסר הבא:

The screenshot shows the Adafruit IO website interface. At the top, there's a navigation bar with links to Shop, Learn, Blog, Forums, LIVE!, AdeBox, and the IO logo. On the right, it says "Hi, Gadi Herman | Account" and shows a shopping cart icon with "0". Below the navigation is a dark header bar with the Adafruit logo, "Devices", "Feeds", "Dashboards", "Actions", and "Power-Ups". To the right of the header are "New Device" and "Help" buttons. The main content area has a green banner that reads: "You are currently using a Adafruit IO Basic plan. For just \$10/month, upgrade to AIO+ to unlock unlimited devices, groups, feeds, dashboards, and more! Learn about the other features and benefits of upgrading your account here." Below the banner, there are two main sections: "Account Status" and "Live Errors". "Account Status" displays metrics for Devices (0 of 2), Groups (1 of 5), Feeds (2 of 10), Dashboards (1 of 5), and Data Rate (0 of 30). "Live Errors" shows "No errors since page load". At the bottom, there's a section for "My Dashboards".

### שימוש באתר [io.adafruit.com](http://io.adafruit.com) המטרה לשימוש על רכיב חומרה

בחלק זה של הפעולות נדגים כיצד יוצריםلوح בקרה הכלול לחץ שמליך ומכבה LED בחומרה.

יצור תחילה Feed חדש בשם led

The screenshot shows the Adafruit IO interface. At the top, there are navigation links: Shop, Learn, Blog, Forums, LIVE!, AdaBox, and IO. On the right, it says "Hi, Gadi Herman | Account" and shows a cart icon with "0". Below the header, the main menu includes Devices, Feeds (which is highlighted), Dashboards, Actions, and Power-Ups. A "New Device" button is also present. The URL in the address bar is "gadiHerman / Feeds". Below the menu, there are two buttons: "+ New Feed" (circled in red) and "+ New Group". A search bar is on the right. The main content area is titled "Default" and lists "Feed Name", "Key", "Last value", and "Recorded". A note at the bottom says "Loaded in 0.24 seconds".

This screenshot shows a modal dialog box titled "Create a new Feed". It has fields for "Name" (containing "led") and "Description" (empty). Below the "Name" field, a note says "Maximum length: 128 characters. Used: 3". At the bottom are "Cancel" and "Create" buttons, with "Create" circled in red. In the background, the Adafruit IO Feeds page is visible, showing the "gadiHerman / Feeds" path and the "+ New Feed" button.

בשלב הבא ניצור לוח בקרה חדש בשם LedControl

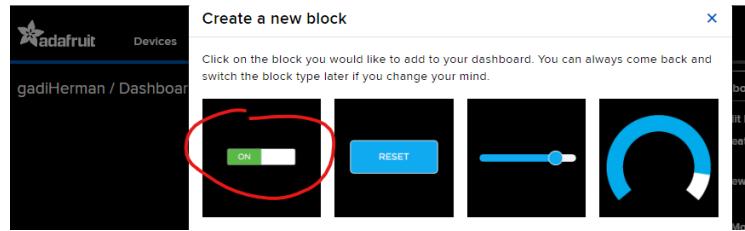
The screenshot shows the Adafruit IO Dashboards page. The top navigation bar includes "Shop", "Learn", "Blog", "Forums", "LIVE!", "AdaBox", and "IO". The main menu items are Devices, Feeds, Dashboards (which is highlighted), Actions, and Power-Ups. A "New Device" button is on the right. The URL in the address bar is "gadiHerman / Dashboards". Below the menu, there is a "+ New Dashboard" button (circled in red) and a search bar. The main content area is titled "Dashboards" and lists a single entry: "Name" (with "LedControl" circled in red) and "Key" (with "ledcontrol").

נכנו לעיצוב הממשק על ידי לחיצה של שמו:

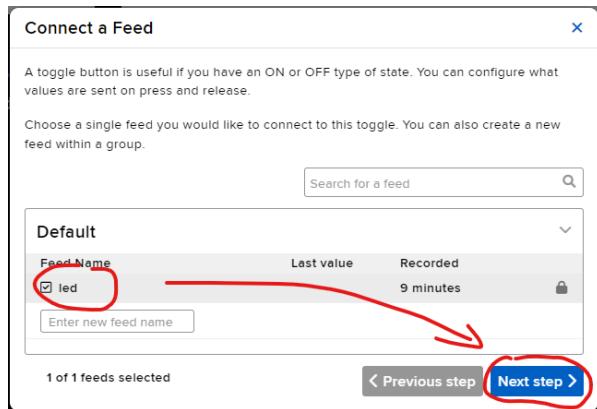
This screenshot shows the "LedControl" dashboard settings. It includes "Dashboard Settings" with "Edit Layout" (circled in red) and "+ Create New Block" (also circled in red). Other options like "View Fullscreen", "Dark Mode", and "Block Borders" are also shown.

נוסף בו לחץ על ידי לחיצה על :Create New Block

The screenshot shows the "gadiHerman / Dashboards / LedControl" page. The main content area displays a dark dashboard template with a "Dashboard Settings" sidebar on the right. The sidebar includes "Edit Layout" (circled in red), "+ Create New Block" (circled in red), "View Fullscreen", "Dark Mode" (switched on), and "Block Borders" (switched on).



בשלב הבא נקשר בין הלחצן שיצרנו עם ה- Feeds ששמו led שיצרנו מקודם:



נגידר בו את המאפיינים הבאים:

Block Title (optional)

✓

Block Preview

Button On Text

✓

Limit of 6 characters for the toggle text. Use the block title to be more descriptive.

Button On Value (uses On Text if blank)

Button Off Text

✓

Limit of 6 characters for the toggle text. Use the block title to be more descriptive.

Button Off Value (uses Off Text if blank)

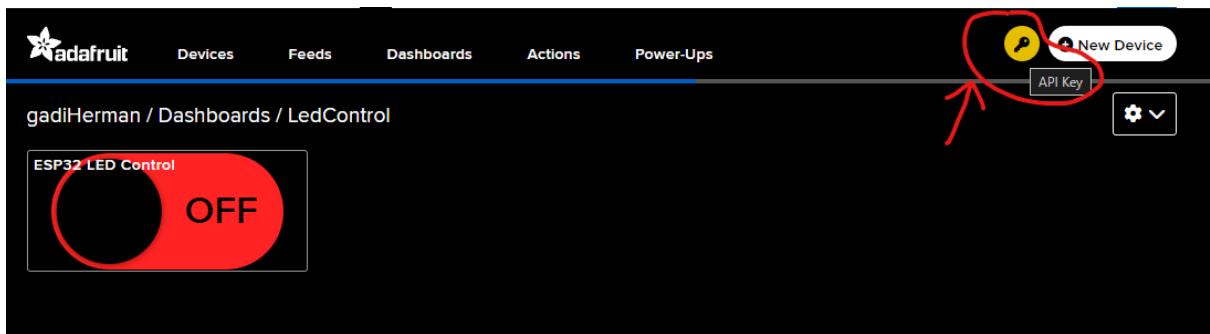
Test Value

Published Value

0 bytes

Create block!
< Previous step

הדבר האחרון שהוא נדרש כדי לחבר בין האתר שמספק לנו שירותים ענן לבין הליד שמחובר לבקר ESP32 הוא קוד הגישה הייחודי לכל משתמש. נקבל אותו על ידי לחיצה על המפתח:



נפתח את סביבת התוכנות כדי לכתוב את הקוד הבא:

הקובץ boot.py

```
# This file is executed on every boot (including wake-boot from deepsleep)
import network

def connect():
    ssid = "XXXX"
    password = "XXXX"

    station = network.WLAN(network.STA_IF)

    if station.isconnected() == True:
        print("Already connected")
        return

    station.active(True)
    station.connect(ssid, password)

    while station.isconnected() == False:
        pass

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

connect()
```

הקובץ main.py

```
import time
from umqtt.robust import MQTTClient
import os
import sys

# the following function is the callback which is
# called when subscribed data is received
def cb(topic, msg):
    print('Received Data: Topic = {}, Msg = {}'.format(topic, msg))

# create a random MQTT clientID
random_num = int.from_bytes(os.urandom(3), 'little')
```

```

mqtt_client_id = bytes('client_'+str(random_num), 'utf-8')

# connect to Adafruit IO MQTT broker using unsecure TCP (port 1883)
ADAFRUIT_IO_URL = b'io.adafruit.com'
ADAFRUIT_USERNAME = b'gadiHerman'
ADAFRUIT_IO_KEY = b'aio_XXX_p31'
ADAFRUIT_IO_FEEDNAME = b'led'

client = MQTTClient(client_id= mqtt_client_id,
                     server=ADAFRUIT_IO_URL,
                     user=ADAFRUIT_USERNAME,
                     password=ADAFRUIT_IO_KEY,
                     ssl=False)

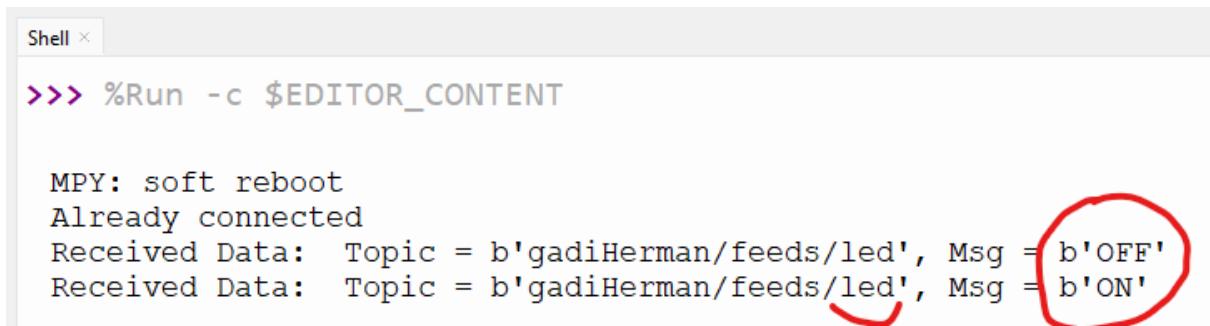
try:
    client.connect()
except Exception as e:
    print('could not connect to MQTT server {}'.format(type(e).__name__, e))
    sys.exit()

mqtt_feedname = bytes('{:s}/feeds/{:s}'.format(ADAFRUIT_USERNAME,
ADAFRUIT_IO_FEEDNAME), 'utf-8')
client.set_callback(cb)
client.subscribe(mqtt_feedname)

# wait until data has been Published to the Adafruit IO feed
while True:
    try:
        client.wait_msg()
    except KeyboardInterrupt:
        print('Ctrl-C pressed... exiting')
        client.disconnect()
        sys.exit()

```

לאחר צריבת הרכיב וחיבורו לאינטרנט על ידי תקשורת WiFi בכל פעם שנלחץ על הלוחץ שבאטר נקלט בברך את ההודעה הבא:



```

Shell >>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Already connected
Received Data: Topic = b'gadiHerman/feeds/led', Msg = b'OFF'
Received Data: Topic = b'gadiHerman/feeds/led', Msg = b'ON'

```

כל מה שנשאר לעשות זה להוסיף לפונקציית `cb` את הקוד הבא כדי להפעיל בפועל את ה- LED בהתאם לפיקודה שקיבל. נדגים זאת:

```

import time
from umqtt.robust import MQTTClient

```

```

import os
import sys
from machine import Pin
Led = Pin(2, mode=Pin.OUT, value=0) # 0V on output

# the following function is the callback which is
# called when subscribed data is received
def cb(topic, msg):
    print('Received Data: Topic = {}, Msg = {}'.format(topic, msg))
    if msg==b'OFF':
        Led.off()
    if msg==b'ON':
        Led.on()

# create a random MQTT clientID
random_num = int.from_bytes(os.urandom(3), 'little')
mqtt_client_id = bytes('client_'+str(random_num), 'utf-8')

# connect to Adafruit IO MQTT broker using unsecure TCP (port 1883)
ADAFRUIT_IO_URL = b'io.adafruit.com'
ADAFRUIT_USERNAME = b'gadiHerman'
ADAFRUIT_IO_KEY = b'aio_____p3l'
ADAFRUIT_IO_FEEDNAME = b'led'

client = MQTTClient(client_id=mqtt_client_id,
                     server=ADAFRUIT_IO_URL,
                     user=ADAFRUIT_USERNAME,
                     password=ADAFRUIT_IO_KEY,
                     ssl=False)

try:
    client.connect()
except Exception as e:
    print('could not connect to MQTT server {}{}'.format(type(e).__name__, e))
    sys.exit()

mqtt_feedname = bytes('{:s}/feeds/{:s}'.format(ADAFRUIT_USERNAME,
ADAFRUIT_IO_FEEDNAME), 'utf-8')
client.set_callback(cb)
client.subscribe(mqtt_feedname)

# following two lines is an Adafruit-specific implementation of the Publish
# "retain" feature
# which allows a Subscription to immediately receive the last Published
# value for a feed,
mqtt_feedname_get = bytes('{:s}/get'.format(mqtt_feedname), 'utf-8')
client.publish(mqtt_feedname_get, '\0')

# wait until data has been Published to the Adafruit IO feed
while True:
    try:
        client.wait_msg()
    except KeyboardInterrupt:

```

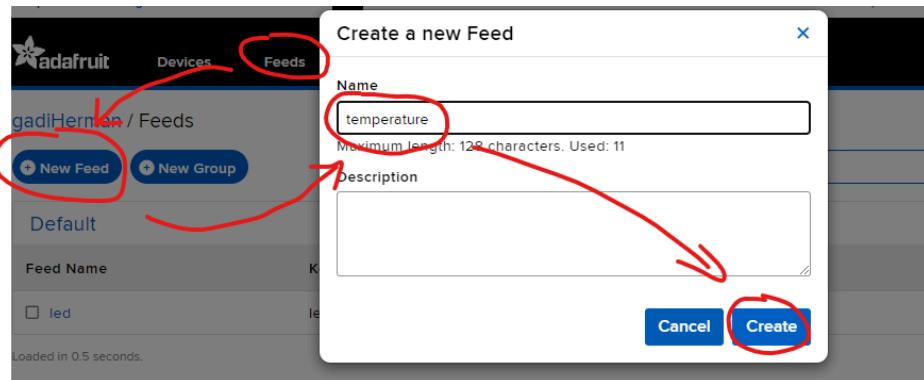
```

print('Ctrl-C pressed... exiting')
client.disconnect()
sys.exit()

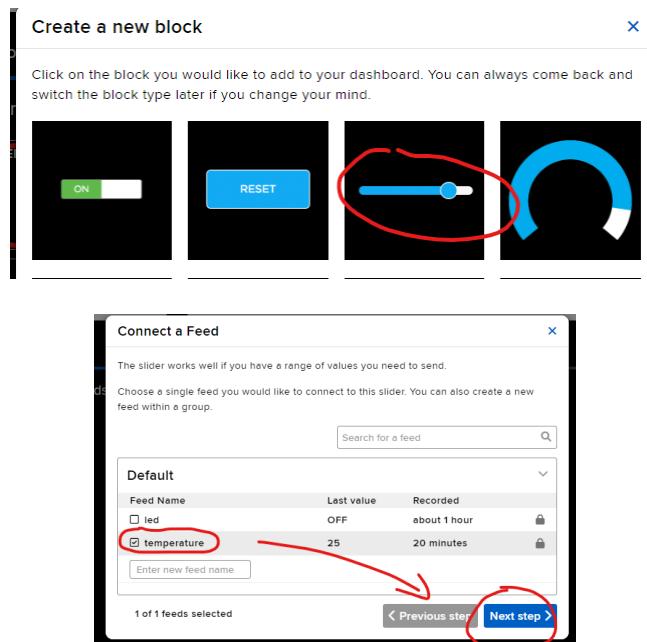
```

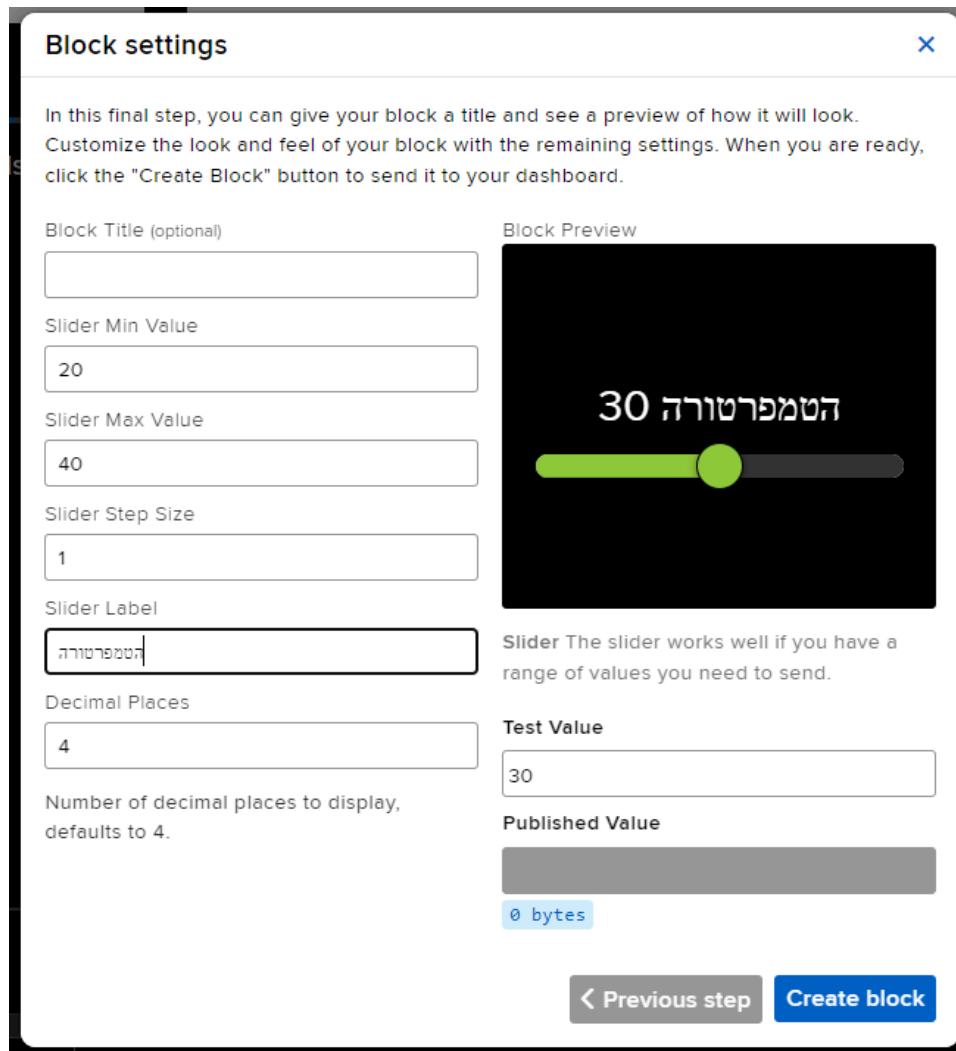
שימוש באתר [adafruit.com](https://adafruit.com) המטרת לקלוט מידע מחיישן המחבר לבקר בחלק זה של הפעולות נדגים כיצד יוצריםلوح בקירה הכלול ממשק גרפי המציג טמפרטורה הנקלטת מחיישן המחבר לבקר ESP32.

יצור תחילה Feeds נוספים בשם temperature

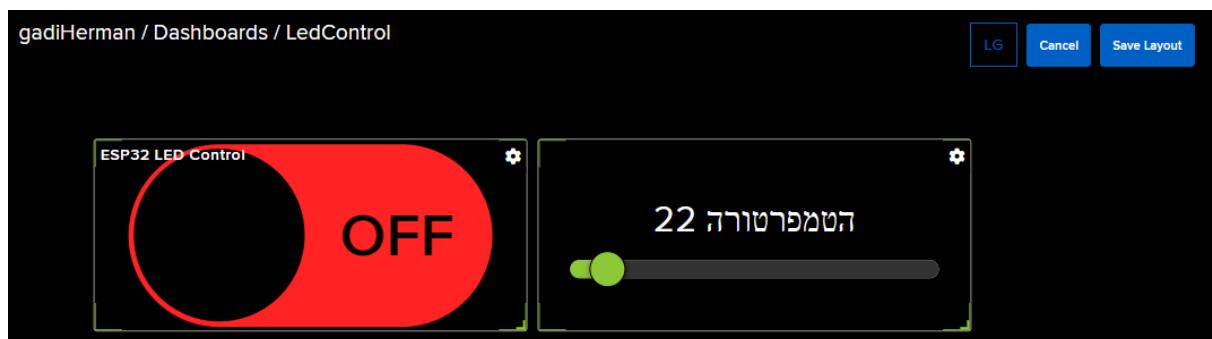


בשלב הבא נוסיף לאותוلوح בקירה שיצרנו כבר עבור ה- LED את הרכיב הבא:





נקבל לוח בקרה הכלול את הרכיבים הבאים:



בשלב הבא נכתוב את הקוד הבא:

```
import time
from umqtt.robust import MQTTClient
import os
import sys

# create a random MQTT clientID
```

```

random_num = int.from_bytes(os.urandom(3), 'little')
mqtt_client_id = bytes('client_'+str(random_num), 'utf-8')

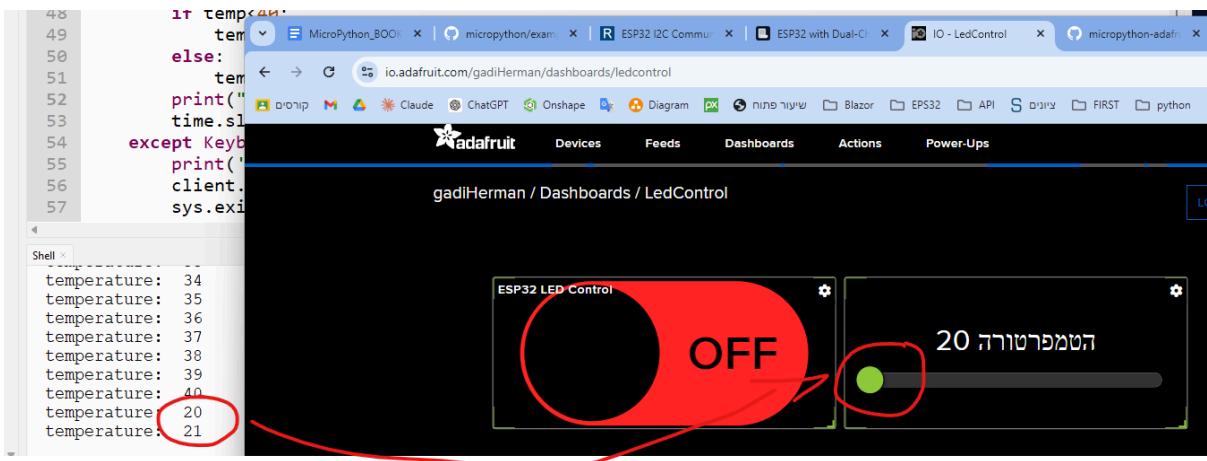
# connect to Adafruit IO MQTT broker using unsecure TCP (port 1883)
#
# To use a secure connection (encrypted) with TLS:
#   set MQTTClient initializer parameter to "ssl=True"
#   Caveat: a secure connection uses about 9k bytes of the heap
#           (about 1/4 of the micropython heap on the ESP8266 platform)
ADAFRUIT_IO_URL = b'io.adafruit.com'
ADAFRUIT_USERNAME = b'gadiHerman'
ADAFRUIT_IO_KEY = b'aio_____p31'
ADAFRUIT_IO_FEEDNAME = b'temperature'

client = MQTTClient(client_id=mqtt_client_id,
                     server=ADAFRUIT_IO_URL,
                     user=ADAFRUIT_USERNAME,
                     password=ADAFRUIT_IO_KEY,
                     ssl=False)
try:
    client.connect()
except Exception as e:
    print('could not connect to MQTT server {}'.format(type(e).__name__, e))
    sys.exit()

# publish free heap statistics to Adafruit IO using MQTT
#
# format of feed name:
#   "ADAFRUIT_USERNAME/feeds/ADAFRUIT_IO_FEEDNAME"
mqtt_feedname = bytes('{:s}/feeds/{:s}'.format(ADAFRUIT_USERNAME,
ADAFRUIT_IO_FEEDNAME), 'utf-8')
PUBLISH_PERIOD_IN_SEC = 10
temp=20
while True:
    try:
        client.publish(mqtt_feedname,
                       bytes(str(temp), 'utf-8'),
                       qos=0)
        if temp<40:
            temp+=1
        else:
            temp=20
        print("temperature: ", temp)
        time.sleep(PUBLISH_PERIOD_IN_SEC)
    except KeyboardInterrupt:
        print('Ctrl-C pressed... exiting')
        client.disconnect()
        sys.exit()

```

נקבל את הפלט הבא:



### רישום בקר לקלט מידע מברך אחר

כפי שראינו ב-2 הדוגמאות הקודומות בקר ESP32 מסוגל להירשם לשירות קבלת נתונים. לדוגמה לקבל מידע מלחצן הממוקם במכשיר המשמש שבatter. כמו כן ראיינו שבקר ESP32 מסוגל גם לספק מידע עבור אחרים שנרשמו למידע. לדוגמה ראיינו קוד המספק למכשיר המשמש שבatter מידע על טמפרטורה.

ניתן כמובן לרשום בקר אחד שיספק את המידע ובקר אחר יקבל את המידע (כמובן דרך שירות הענן).

הקוד שמספק את המידע כבר הודגם בדוגמה השנייה. להלן דוגמה לתוכנה בברך נוסף הנרשם לקבל מידע מהברך הראשון. במצב זה נקלט מצב שבברך אחד מספק לענן מידע עם טמפרטורה ובברך שני מקבל את המידע על הטמפרטורה מהענן ומדליק LED או מערכת קירור אחרת במידה והטמפרטורה גבוהה מ- 30 מעלות. להלן :

```

import time
from umqtt.robust import MQTTClient
import os
import sys
from machine import Pin
Led = Pin(2, mode=Pin.OUT, value=0)

# the following function is the callback which is
# called when subscribed data is received
def cb(topic, msg):
    print('Received Data: Topic = {}, Msg = {}'.format(topic, msg))
    if int(msg) < 30:
        Led.off()
    else:
        Led.on()

# create a random MQTT clientID
random_num = int.from_bytes(os.urandom(3), 'little')
mqtt_client_id = bytes('client_'+str(random_num), 'utf-8')

# connect to Adafruit IO MQTT broker using unsecure TCP (port 1883)
ADAFRUIT_IO_URL = b'io.adafruit.com'
ADAFRUIT_USERNAME = b'gadiHerman'
ADAFRUIT_IO_KEY = b'aio_UKmN85DbcrlQpriPBAmRTgbm0p31'
ADAFRUIT_IO_FEEDNAME = b'temperature'

```

```

client = MQTTClient(client_id= mqtt_client_id,
                     server=ADAFRUIT_IO_URL,
                     user=ADAFRUIT_USERNAME,
                     password=ADAFRUIT_IO_KEY,
                     ssl=False)

try:
    client.connect()
except Exception as e:
    print('could not connect to MQTT server {}{}'.format(type(e).__name__, e))
    sys.exit()

mqtt_feedname = bytes('{:s}/feeds/{:s}'.format(ADAFRUIT_USERNAME, ADAFRUIT_IO_FEEDNAME), 'utf-8')
client.set_callback(cb)
client.subscribe(mqtt_feedname)

# wait until data has been Published to the Adafruit IO feed
while True:
    try:
        client.wait_msg()
    except KeyboardInterrupt:
        print('Ctrl-C pressed... exiting')
        client.disconnect()
        sys.exit()

```

**בקר ESP32 שגם מספק מידע וגם מקבל מידע בו בזמןית**

להלן דוגמת קוד המשלבת פונקציית publish וsubscribe באותו הזמן:

```

import time
from umqttr.robust import MQTTClient
import os
import sys
from machine import Pin
Led = Pin(2, mode=Pin.OUT, value=0) # 0V on output

# the following function is the callback which is
# called when subscribed data is received
def cb(topic, msg):
    print('Subscribe: Received Data: Topic = {}, Msg = {}{}'.format(topic, msg))
    if msg==b'OFF':
        Led.off()
    if msg==b'ON':
        Led.on()

# create a random MQTT clientID
random_num = int.from_bytes(os.urandom(3), 'little')
mqtt_client_id = bytes('client_'+str(random_num), 'utf-8')

```

```

ADAFRUIT_IO_URL = b'io.adafruit.com'
ADAFRUIT_USERNAME = b'gadiHerman'
ADAFRUIT_IO_KEY = b'aio_XXX_0p3l'
ADAFRUIT_IO_SUB_FEEDNAME = b'led'
ADAFRUIT_IO_PUB_FEEDNAME = b'temperature'

client = MQTTClient(client_id= mqtt_client_id,
                     server=ADAFRUIT_IO_URL,
                     user=ADAFRUIT_USERNAME,
                     password=ADAFRUIT_IO_KEY,
                     ssl=False)

try:
    client.connect()
except Exception as e:
    print('could not connect to MQTT server {}{}'.format(type(e).__name__, e))
    sys.exit()

# format of feed name: "ADAFRUIT_USERNAME/feeds/ADAFRUIT_IO_FEEDNAME"
mqtt_pub_feedname = bytes('{:s}/feeds/{:s}'.format(ADAFRUIT_USERNAME,
ADAFRUIT_IO_PUB_FEEDNAME), 'utf-8')
mqtt_sub_feedname = bytes('{:s}/feeds/{:s}'.format(ADAFRUIT_USERNAME,
ADAFRUIT_IO_SUB_FEEDNAME), 'utf-8')
client.set_callback(cb)
client.subscribe(mqtt_sub_feedname)
PUBLISH_PERIOD_IN_SEC = 10
SUBSCRIBE_CHECK_PERIOD_IN_SEC = 0.5
accum_time = 0
temp=20
while True:
    try:
        # Publish
        if accum_time >= PUBLISH_PERIOD_IN_SEC:
            print('Publish: temperature = {}'.format(temp))
            client.publish(mqtt_pub_feedname,
                           bytes(str(temp), 'utf-8'),
                           qos=0)
        accum_time = 0
        if temp<40:
            temp+=1
        else:
            temp=20
        # Subscribe. Non-blocking check for a new message.
        client.check_msg()

        time.sleep(SUBSCRIBE_CHECK_PERIOD_IN_SEC)
        accum_time += SUBSCRIBE_CHECK_PERIOD_IN_SEC
    except KeyboardInterrupt:
        print('Ctrl-C pressed... exiting')
        client.disconnect()
        sys.exit()

```

נקבל את הפלט הבא:

The screenshot shows the Adafruit IO dashboard for a project named "gadiHerman / Dashboards / LedControl". The interface includes a red "ESP32 LED Control" button labeled "OFF", a slider labeled "הטמפרטורה 33", and a shell terminal window. The shell window displays MQTT message logs:

```
Shell x
Publish: temperature = 25
Publish: temperature = 26
Publish: temperature = 27
Publish: temperature = 28
Publish: temperature = 29
Subscribe: Received Data: Topic = b'gadiHerman/feeds/led', Msg = b'OFF'
Subscribe: Received Data: Topic = b'gadiHerman/feeds/led', Msg = b'ON'
Publish: temperature = 30
Subscribe: Received Data: Topic = b'gadiHerman/feeds/led', Msg = b'OFF'
Publish: temperature = 31
Publish: temperature = 32
Publish: temperature = 33
```

## משימה 17 - הפעלת רכיב השמעת קבצי MP3 מושם על YX5300

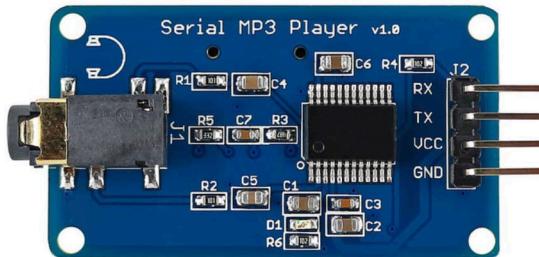
קישור:

[https://github.com/GadiHerman/ESP32\\_YX5300\\_Serial\\_MP3\\_Player](https://github.com/GadiHerman/ESP32_YX5300_Serial_MP3_Player)

נגן MP3 מושם על רכיב YX5300 תומך בהשמעת קבצי בפורמט MP3 ו-WAV בתדר דגימה של 8kHz עד 48kHz. קבצי השמע מאוחסנים בכרטיסי מיקרו SD שמתחבר לשקע כרטיס TF בגב הלוח. הנגן כולל בקר פנימי השולט על השמעת MP3 על ידי שליחת פקודות טוריות למודול דרך יציאת UART.

להלן מאפייני הרכיב:

Item	Min	Typical	Max	Unit
Power Supply(VCC)	3.2	5	5.2	VDC
Current ( @VCC=5V )	/	/	200	mA
Logic interface	3.3V / 5V TTL			/
Supported Card Type	Micro SD card(<=2G); Mirco SDHC card(<=32G)			/
File system format	Fat16 / Fat32			/
Uart baud rate	9600			bps



הרכיב כולל מספר הדקים:

1. ממוקם הבקרה הכלול את ההדים GND, VCC, TX, RX לאדמה של ה-ESP32 הדק VCC מחובר ל-5V. הדק TX (שידור) מחובר להדק RX של בקר ESP32 (קיבלה) מחובר להדק TX של בקר ESP32 .
2. שקע כרטיס TF בצד האחורית של ה-PCB לחיבור כרטיס מיקרו SD הכלול קבצי MP3/WAV.
3. מחוון השמעה (LED יירוק) מהבהבת בזמן ההשמעה. דלוק קבוע בכל זמן אחר.
4. שקע פלט קול לאוזניות או מגבר חיצוני.

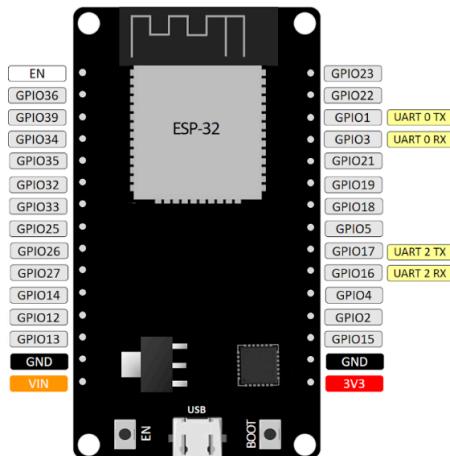
לבקר ESP32 יש 3 ממוקמי תקשורת UART (שרק אחד זמין לשימוש) על פי המפרט הבא:

UART0: (GPIO 1 and GPIO3) משמש לתקשורת מול המחשב -

UART1: (GPIO 9 and GPIO10) – connected to the ESP32 SPI flash memory, so you can't use them.

זמן לשימוש - UART2: (GPIO 17 and GPIO 16)

להלן מיקום הדקי ה- UART בעקר:



כלומר UART0 לא זמין וUART2 זמין דרך הדקים tx=17 tx=16 rx=16rx נחבר את התקשרות בין הבקר לבין רכיב ההשמעה.

יש לארಗן את קבצי השמע בזיכרון ה- SD על פי הפורמט הבא:

```
+-- 01
|   + 001-file_description.mp3
|   + 002-file_description.mp3
|
+- 02
|   + 003-file_description.mp3
|   + 004-file_description.mp3
|   + 005-file_description.mp3
|
+- 03
|   + 006-file_description.mp3
|   + 007-file_description.mp3
```

להלן מימוש מחלקה המטפלת בתקשרות על הרכיב. יש ליצור קובץ הכלול התוכן הבא:

```
from machine import UART
from time import sleep

class MD_YX5300:
    def __init__(self, UART_NUMBER=2):
        self.command=bytearray()
        self.command.append(0x7e)
        self.command.append(0xFF)
        self.command.append(0x06)
        self.command.append(0x00)
        self.command.append(0x00)
        self.command.append(0x00)
        self.command.append(0xEF)
```

```

        self.uart = UART(UART_NUMBER, 9600)
        # set volume to mid point (0=min 30=max)
        self.volume_level=15
        self.set_volume(self.volume_level)
        sleep(0.5)

    def play_next(self):
        self.command[3]=0x01
        self.uart.write(self.command)

    def play_previous(self):
        self.command[3]=0x02
        self.uart.write(self.command)

    def play_track(self,track_id):
        self.command[3]=0x03
        self.command[6]=track_id
        self.uart.write(self.command)

    def play(self):
        self.command[3]=0x03
        self.command[6]=1
        self.uart.write(self.command)

    def volume_up(self,step_count=1):
        if self.volume_level<=(30-step_count):
            self.volume_level=self.volume_level+step_count
        else:
            self.volume_level=30
        self.set_volume(self.volume_level)

    def volume_down(self,step_count=1):
        if self.volume_level>=step_count:
            self.volume_level=self.volume_level-step_count
        else:
            self.volume_level=0
        self.set_volume(self.volume_level)

    def set_volume(self,level):
        self.command[3]=0x06
        self.command[6]=level
        self.uart.write(self.command)

    def sleep_module(self):
        self.command[3]=0x0A
        self.uart.write(self.command)

    def wakeup_module(self):
        self.command[3]=0x0B
        self.uart.write(self.command)

    def reset_module(self):
        self.command[3]=0x0C
        self.uart.write(self.command)

```

```

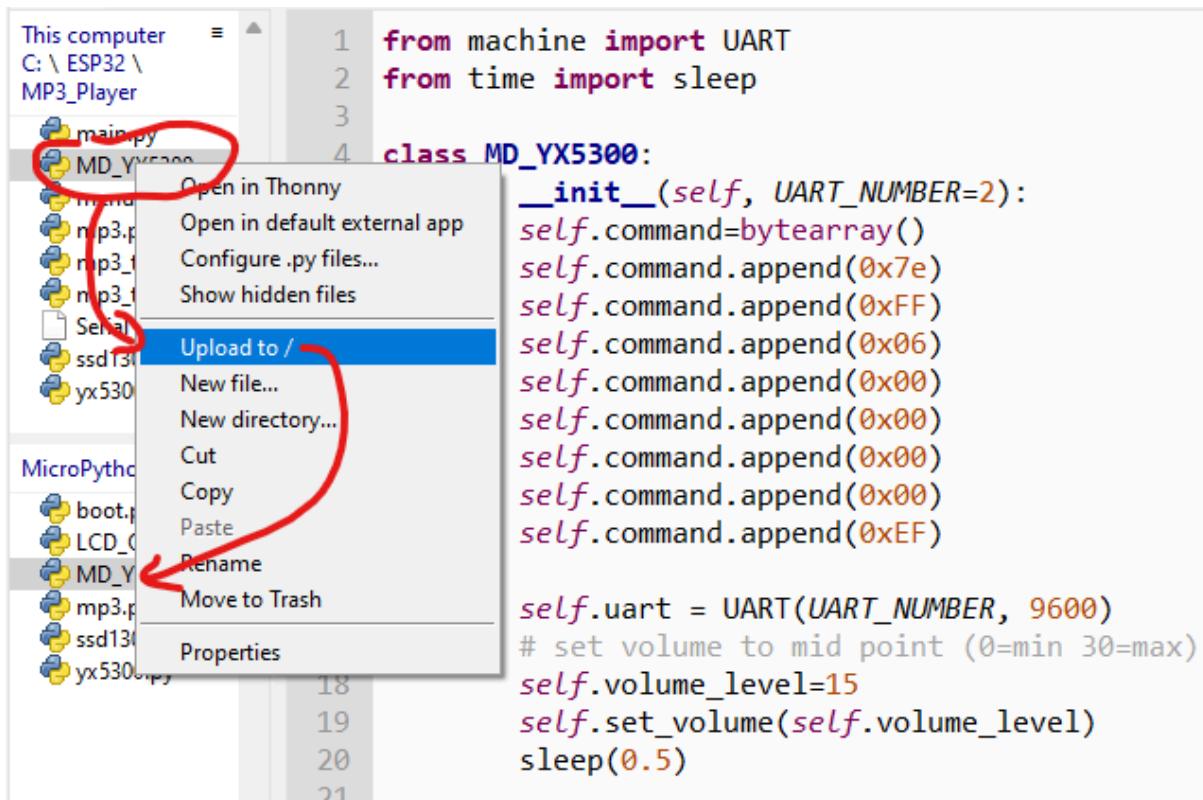
def pause(self):
    self.command[3]=0x0E
    self.uart.write(self.command)

def resume(self):
    self.command[3]=0x0D
    self.uart.write(self.command)

def stop(self):
    self.command[3]=0x16
    self.uart.write(self.command)

```

כדי לבדוק את הרכיב יש להעתיק את קובץ המחלקה לבקר על ידי ביצוע השלבים הבאים:



להלן קוד בדיקה בסיסי לרכיב:

```

import MD_YX5300

mp3 = MD_YX5300.MD_YX5300()
mp3.set_volume(30)
mp3.play_track(15)

```

להלן דוגמת קוד למימוש נגן הכלול קלט ופלט דרך חלון ה- Shell

```
import MD_YX5300
```

```

def display_menu(menu_items):
    print("==== MD_YX5300 Test Menu ====")
    for i, item in enumerate(menu_items, 1):
        print(f"{i}. {item}")
    print("=====")

def get_user_choice(menu_items):
    while True:
        try:
            choice = int(input("Enter the number of your choice (0 to exit): "))
            if choice == 0:
                return None
            if 1 <= choice <= len(menu_items):
                return menu_items[choice - 1]
            else:
                print("Invalid choice. Please try again.")
        except ValueError:
            print("Invalid input. Please enter a number.")

def main():
    menu_items = [
        "Play Next Track",
        "Play Previous Track",
        "Play Track by ID",
        "Play First Track",
        "Volume Up",
        "Volume Down",
        "Set Volume",
        "Sleep Module",
        "Wakeup Module",
        "Reset Module",
        "Pause",
        "Resume",
        "Stop"
    ]
    player = MD_YX5300.MD_YX5300(UART_NUMBER=2)

    while True:
        display_menu(menu_items)
        choice = get_user_choice(menu_items)

        if choice is None:
            print("Exiting the test menu. Goodbye!")
            break

        if choice == "Play Next Track":
            player.play_next()
        elif choice == "Play Previous Track":
            player.play_previous()
        elif choice == "Play Track by ID":
            track_id = int(input("Enter the track ID to play: "))

```

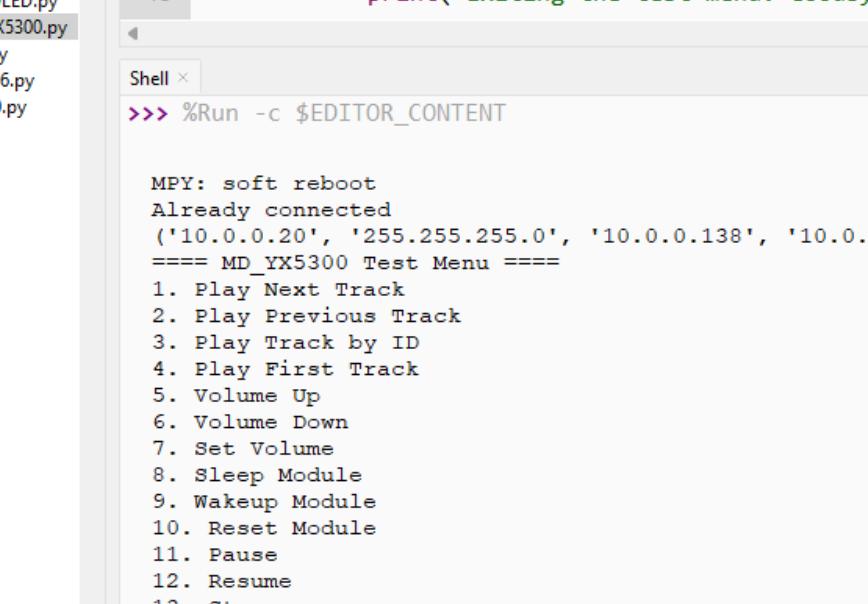
```

        player.play_track(track_id)
    elif choice == "Play First Track":
        player.play()
    elif choice == "Volume Up":
        step_count = int(input("Enter the number of steps to increase
volume: "))
        player.volume_up(step_count)
    elif choice == "Volume Down":
        step_count = int(input("Enter the number of steps to decrease
volume: "))
        player.volume_down(step_count)
    elif choice == "Set Volume":
        volume_level = int(input("Enter the volume level (0-30): "))
        player.set_volume(volume_level)
    elif choice == "Sleep Module":
        player.sleep_module()
    elif choice == "Wakeup Module":
        player.wakeup_module()
    elif choice == "Reset Module":
        player.reset_module()
    elif choice == "Pause":
        player.pause()
    elif choice == "Resume":
        player.resume()
    elif choice == "Stop":
        player.stop()

if __name__ == "__main__":
    main()

```

נקבל את הפלט הבא:



The screenshot shows a MicroPython development environment. On the left, a file tree lists several Python files: boot.py, LCD\_OLED.py, MD\_YX5300.py, mp3.py, ssd1306.py, and yx5300.py. The main window displays a code editor with the following content:

```
44
45
46
if choice is None:
    print("Exiting the test menu. Goodbye!")
```

Below the code editor is a terminal window titled "Shell". It contains the command:

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

The terminal output shows the execution of the code, resulting in the following test menu:

```
MPY: soft reboot
Already connected
('10.0.0.20', '255.255.255.0', '10.0.0.138', '10.0.0.138')
===== MD_YX5300 Test Menu =====
1. Play Next Track
2. Play Previous Track
3. Play Track by ID
4. Play First Track
5. Volume Up
6. Volume Down
7. Set Volume
8. Sleep Module
9. Wakeup Module
10. Reset Module
11. Pause
12. Resume
13. Stop
=====
Enter the number of your choice (0 to exit):
```

## משימה 18 - הפעלת צג גרפי 2x16 I2C LCD

במשימה זה תלמד כיצד להשתמש בתצוגת LCD 2x16 עם בקר ESP32 כדי להציג טקסט ומספרים כתווים על המסך. הצג בגודל 16x2 תווים יכול להציג 16 תווים בשורה על פני שתי שורות של תווים. LCD-הברker מסוג 00804478hd, כמו כן יש לו גם מודול I2C שמחובר אליו ובכך מקל על החיבור בין הצג לברker ESP32.

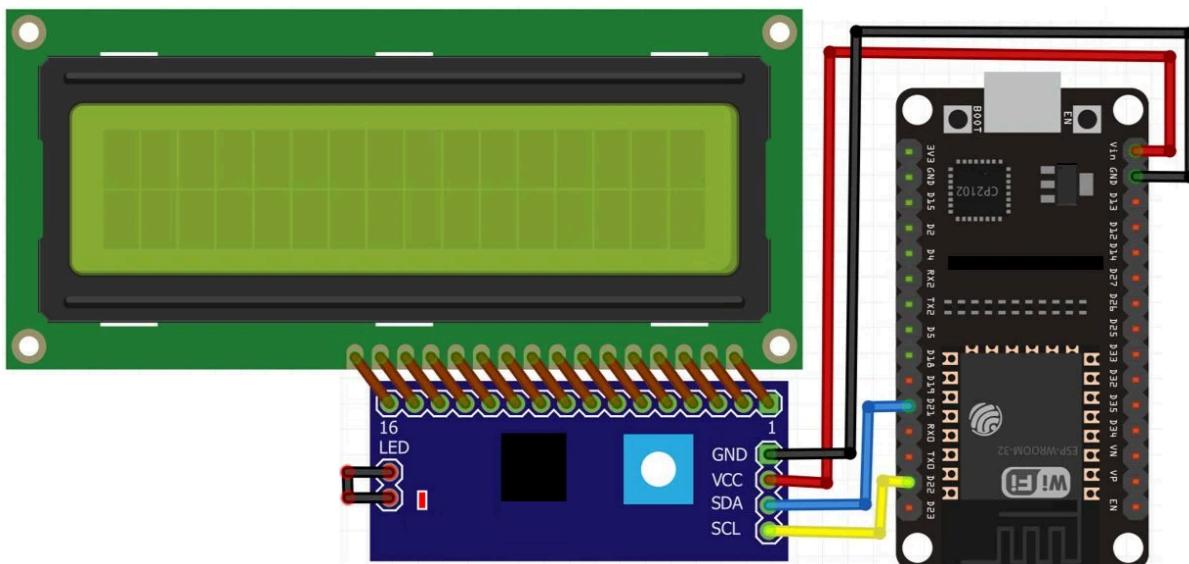
קישורים:

<https://microcontrollerslab.com/i2c-lcd-esp32-esp8266-micropython-tutorial/>

חיבור הצג לברker

OLED	ESP32
Vin	3.3V
GND	GND
SCL	GPIO 22
SDA	GPIO 21

شرطוט חשמלי של החיבור:



בשלב הראשון נעזר בקוד הבא כדי לאתר את כתובות הרכיב:

```
from machine import I2C, Pin  
import machine
```

```

i2c = I2C(scl=Pin(22), sda=Pin(21))

devices = i2c.scan()

if len(devices) == 0:
    print("No i2c device !")
else:
    print('i2c devices found:',len(devices))
for device in devices:
    print("At address: ",hex(device))

```

נקבל את הפלט הבא:

```

>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Already connected
('10.0.0.20', '255.255.255.0', '10.0.0.138', '10.0.0.138')
Warning: I2C(-1,...) is deprecated, use SoftI2C(...) instead
i2c devices found: 2
At address: 0x27
At address: 0x3c
>>>

```

ניתן לראות שיש לנו 2 רכיבים המתחברים על אותו ערוץ I2C האחד בכתובת 27 והשני 3CH.

תוכן הקובץ עטוף :

```

from utime import sleep_ms
from machine import I2C

class I2C_LCD():
    def __init__(self, i2c, addr = 0x27):
        self.i2c=i2c
        self.buf = bytearray(1)
        self.BK = 0x08
        self.RS = 0x00
        self.E = 0x04
        self.ADDR = addr
        self.setcmd(0x33)
        sleep_ms(5)
        self.send(0x30)
        sleep_ms(5)
        self.send(0x20)
        sleep_ms(5)
        for i in [0x28, 0x0C, 0x06, 0x01]:
            self.setcmd(i)
        self.px, self.py = 0, 0

```

```

        self.pb = bytearray(16)

    def setReg(self, dat):
        self.buf[0] = dat
        self.i2c.writeto(self.ADDR, self.buf)
        sleep_ms(1)

    def send(self, dat):
        d=(dat&0xF0)|self.BK|self.RS
        self.setReg(d)
        self.setReg(d|0x04)
        self.setReg(d)

    def setcmd(self, cmd):
        self.RS=0
        self.send(cmd)
        self.send(cmd<<4)

    def setdat(self, dat):
        self.RS=1
        self.send(dat)
        self.send(dat<<4)

    def autoaddr(self):
        for i in range(32, 63):
            try:
                if self.i2c.readfrom(i, 1):
                    return i
            except:
                pass
        raise Exception('I2C address detect error!')

    def write_cgram(self, buf, reg=0):
        n = len(buf)
        self.setcmd(0x40 + (reg%8)*8)
        for i in range(n):
            self.setdat(buf[i])

    def clear(self):
        self.setcmd(1)

    def backlight(self, on):
        if on:
            self.BK=0x08
        else:
            self.BK=0
        self.setcmd(0)

    def on(self):
        self.setcmd(0x0C)

    def off(self):
        self.setcmd(0x08)

```

```

def shl(self):
    self.setcmd(0x18)

def shr(self):
    self.setcmd(0x1C)

def char(self, ch, x=-1, y=0):
    if x>=0:
        a=0x80
        if y>0:
            a=0xC0
        self.setcmd(a+x)
    self.setdat(ch)

def puts(self, s, x=0, y=0):
    if type(s) is not str:
        s = str(s)
    if len(s)>0:
        self.char(ord(s[0]),x,y)
        for i in range(1, len(s)):
            self.char(ord(s[i]))

def newline(self):
    self.px = 0
    if self.py < 1:
        self.py += 1
    else:
        for i in range(16):
            self.char(self.pb[i], i)
            self.char(32, i, 1)
            self.pb[i] = 32
def print(self, s):
    if type(s) is not str:
        s = str(s)
    for i in range(len(s)):
        d = ord(s[i])
        if d == ord('\n'):
            self.newline()
        else:
            self.char(d, self.px, self.py)
            if self.py:
                self.pb[self.px] = d
            self.px += 1
            if self.px > 15:
                self.newline()

```

תוכן הקובץ HelloWorld.py

```
from machine import I2C, Pin
from i2c_lcd import I2C_LCD
from time import sleep_ms

i2c = I2C(scl=Pin(22), sda=Pin(21))
LCD = I2C_LCD(i2c,0x27)

LCD.print("ABCDEFGHIJKLM NOPQRSTUVWXYZ")
sleep_ms(4000)
LCD.clear()
LCD.puts("Hello Word (-: ")
n = 0
while 1:
    LCD.puts(n, 0, 1)
    n += 1
    sleep_ms(1000)
```

להלן הפלט על הציג:



## משימה 19 - הפעלת צג LCD גרפי צבעוני 240\*320 פיקסלים מבוסס על ILI9341

במשימה זו נமמש קוד להפעלת צג גרפי צבעוני בגודל 240\*320 פיקסלים מבוסס על ILI9341. הצג מחובר לבקר דרך משך SPI. לצגים גרפיים יש 2 מאפיינים הקובעים את הביצועים שלהם:

זמן רישום: Rendering time: הזמן שלוקח לעבד מסגרת (frame) אחת עבור הצג. זמן זה מושפע מקצב העבודה של הבקר, הבקר של עבד בקצב של 200MHz אמור להספיק לביוזו העבודה.

זמן שטוחה: Flushing time: הזמן שלוקח להעביר מסגרת אחת למרחק. זמן זה מושפע מקצב התקשורת בין הצג למרחק בפעולות זו התקשרות בין השניים מתבצע דרך משך טורי SPI. קצב התקשרות המקסימלי שמסוגל בкар ESP32 להעביר דרך משך SPI הוא 40MHz. עבור צג בגודל 240\*320 פיקסלים שבו כל פיקסל בגודל של 16bit - אנו זקוקים לפחות 1228000 סיביות כדי להעביר מסגרת אחת למרחק.

$$240 \times 320 = 1228000 \text{ bit}$$

כלומר בקצב העבודה של 40MHz אנו יכולים להעביר למשך מסגרת ב- 31msec.

$$\frac{1}{40M} = 31msec$$

כמובן לנו שזמן זה יש להוסיף את הזמן שלוקח למעבד להכין את התמונה.

קישורים:

<https://github.com/adafruit/micropython-adafruit-rgb-display/releases>

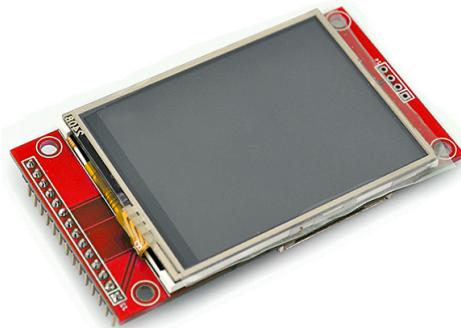
<https://cdn-learn.adafruit.com/downloads/pdf/micropython-displays-drawing-shapes.pdf?timestamp=1570796654>

<https://github.com/adafruit/micropython-adafruit-rgb-display>

[https://www.youtube.com/watch?v=9wsJzVeN\\_m8](https://www.youtube.com/watch?v=9wsJzVeN_m8)

<https://github.com/adafruit/micropython-adafruit-gfx>

<https://www.digikey.co.il/en/maker/projects/3407b01f06f647b48983cd0cd8c809>



תקשורת SPI בкар ESP32

There are two SPI drivers. One is implemented in software (bit-banging) and works on all pins, and is accessed via the machine.SPI class.

There are two hardware SPI channels that allow faster transmission rates (up to 80Mhz). These may be used on any IO pins that support the required direction and are otherwise unused (see Pins and

GPIO) but if they are not configured to their default pins then they need to pass through an extra layer of GPIO multiplexing, which can impact their reliability at high speeds. Hardware SPI channels are limited to 40MHz when used on pins other than the default ones listed below.

	HSPI (id=1)	VSPI (id=2)
sck	14	18
mosi	13	23
miso	12	19

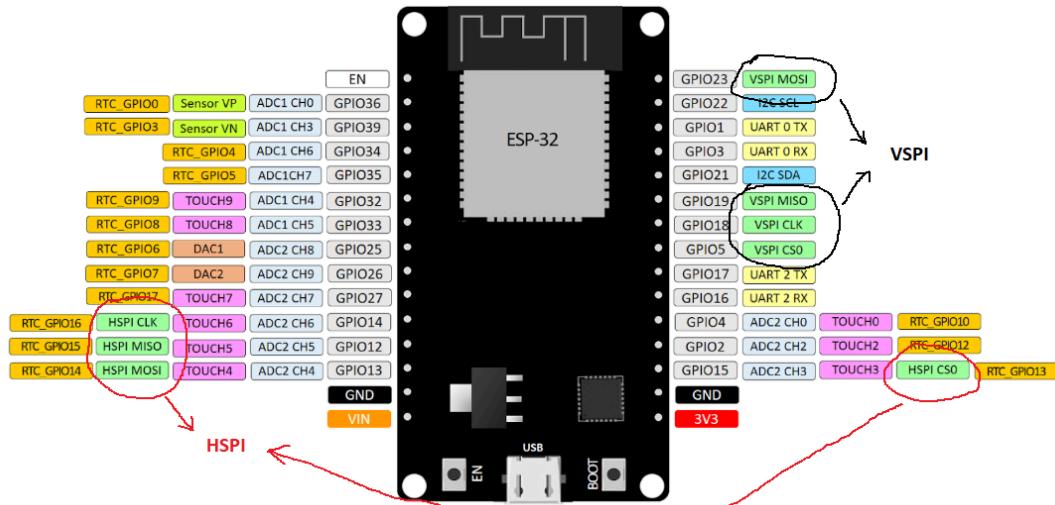
Hardware SPI has the same methods as Software SPI above:

```
from machine import Pin, SPI

hspi = SPI(1, 10000000, sck=Pin(14), mosi=Pin(13), miso=Pin(12))

vspi = SPI(2, baudrate=80000000, polarity=0, phase=0, bits=8, firstbit=0,
           sck=Pin(18), mosi=Pin(23), miso=Pin(19))
```

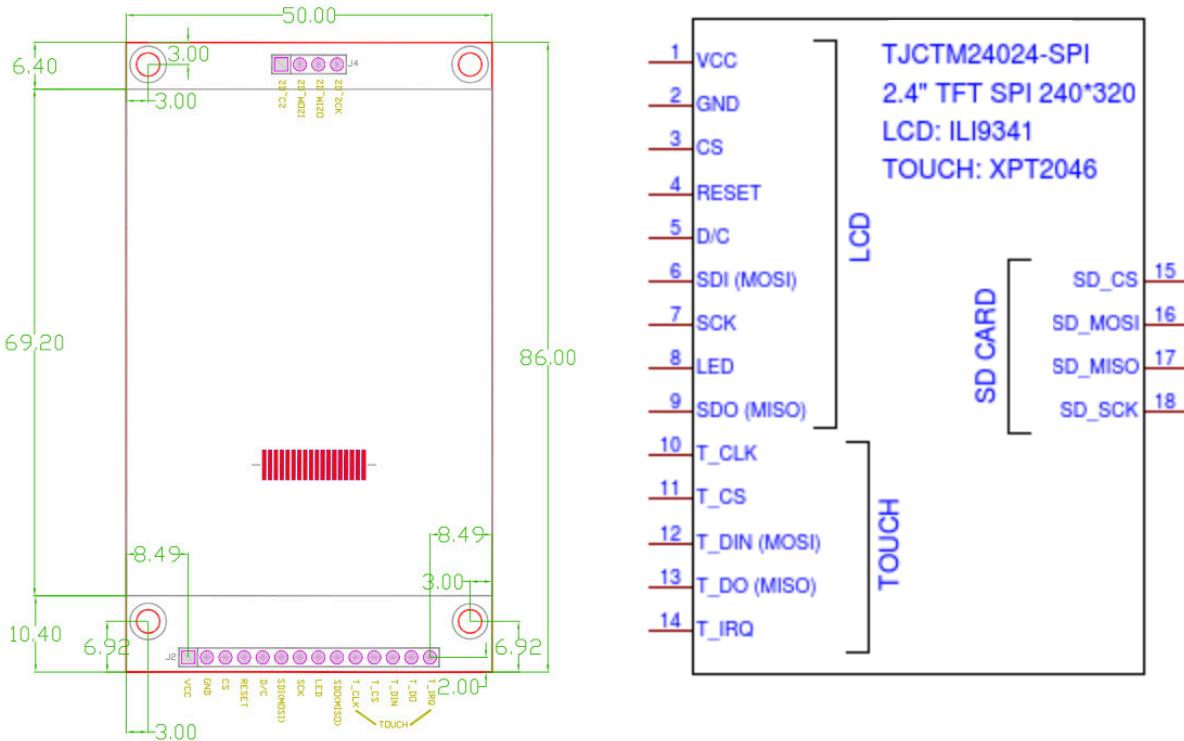
מיפוי רגילים:



מערכת החיבור בין הציג לבקра:

ESP32	ILI9341 LCD color display
Vcc (3.3v)	RST
D5	CS
D4	D/C
D23 (MOSI)	DIN

D18 (SCK)	CLK
3.3V	VCC
Vcc (3.3v)	LED
GND	GND



ILI9341 SPI TFT	EasyESP-1 or NodeMCU pins
VCC	3.3V
GND	GND
CS	D2
RESET	RST
DC	D1
SDI/MOSI	D7
SCK	D5
LED	3.3V
SDO/MISO	D6 (optional)

RGB565

Especially (cheap) screens used with embedded devices do not provide 24 bit color-depth. Moreover, storing and/or transmitting 3 bytes per pixel is consuming quite some memory and creates latency. RGB565 requires only 16 (5+6+5) bits/2 bytes and is commonly used with embedded screens. It provides 5 bits for Red and Blue and 6 bits for Green. Providing 5 bits for 2 colors and 6 bits for another seems asymmetric but storing and transmitting something which cannot nicely be packed in bytes would be complicated. Note that since we have less bits (information) available, we can represent less colors. While RGB888 provides  $2^{24}=16\ 777\ 216$  colors, RGB565 only provides  $2^{16}=65\ 536$  colors.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
r7	r6	r5	r4	r3	g7	g6	g5	g4	g3	g2	b7	b6	b5	b4	b3

שאלה:

```
help('modules')
```

להלן הפלט:

## משימה 20 - קרייה וכתיבה של תג RFID תוך שימוש ב- RC522

קישורים:

<https://github.com/rdagger/micropython-ili9341>

RC522 הוא קורא/כותב RFID נפוץ מאוד המיועד לקרייה וכתיבה של תג RFID וקרטיסים חכמים הפעלים בתדר של 13.56MHz. הוא מבוסס על השבב MFRC522 של חברת NXP.

מאפיינים עיקריים:

- מתח הפעלה: 3.7V
- ממשך תקשורת: SPI
- טווח קרייה: עד 5 ס"מ ( תלוי בתג )
- תומיכה בתקנים: ISO 14443A/MIFARE
- צירכיט זרם נמוכה: C-Amb13 בזמן פעולה
- מידות קטנות: BD"כ 60x40x40 מ"מ
- מחיר נמוך יחסית

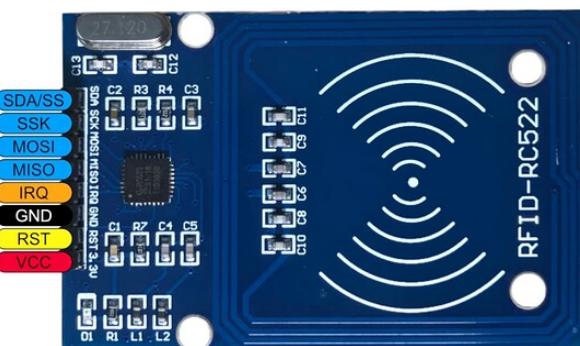
שימושים עיקריים:

1. בקרת כניסה:
  - מערכות געילה אלקטרוניות
  - זיהוי עובדים
  - בקרת גישה למתקנים
2. מערכות תשלום:
  - קרטיסי תחבורה ציבורית
  - מערכות תשלום במזנונים
  - קרטיסי חבר במועדונים

יתרונות:

- קל לשימוש ותוכנות
- תומיכה נרחבת ו眾多 זמינות
- אמינות גבוהה
- מחיר משתלם

חיבור בסיסי: הרכיב מתחבר למקroit-בקר דרך ממשך SPI. להלן הדקן הרכיב:



#### dagshim leshimush:

1. יש להקפיד על חיבור למתח 3.3V בלבד
2. מומלץ להשתמש בספירות מוכנות
3. יש לשמר על מרחוק מינימלי בין קוראים במדיה ומשתמשים ביוטר אחד
4. רצוי להוסיף קבל סינון על קו המתח

#### תקשורת SPI בבקר ESP32

תקשורת SPI - Serial Peripheral Interface היא פרוטוקול תקשורת טורי סינכרוני המשמש לתקשורת בין מיקרו-בקרים והתקנים היקפיים. ESP32 תומך עד 4 ערוצי SPI שונים (SPI0-SPI3).

#### קווי התקשורת העיקריים:

- SCLK - אות שעון
- MOSI - Master Out Slave In
- MISO - Master In Slave Out
- CS/SS - בחרית ההתקן

**הדקם ב-ESP32:** ESP32 מציע מספר אפשרויות לחיבור SPI:

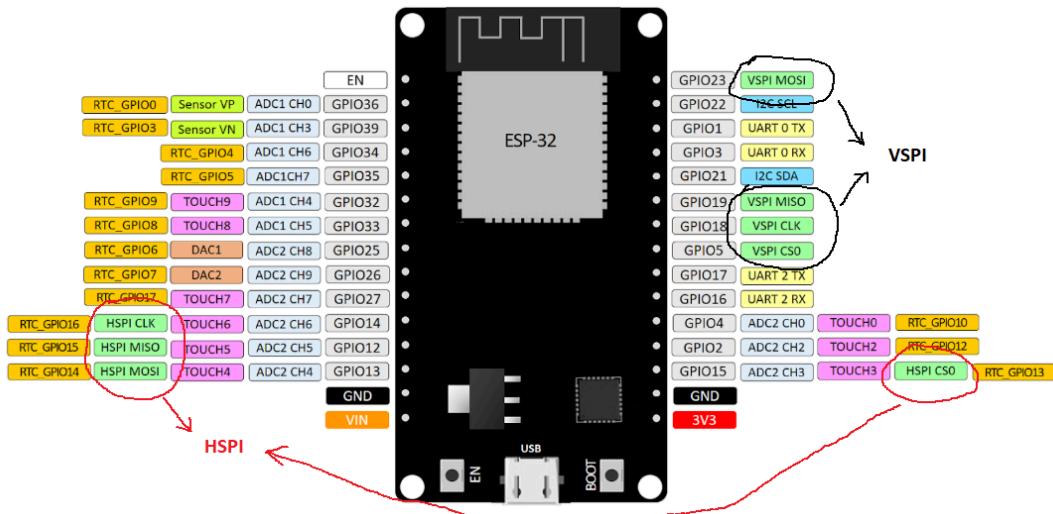
##### .1 VSPI - SPI3 - ברירת המחדל:

- SCLK: GPIO18
- MOSI: GPIO23
- MISO: GPIO19
- CS: GPIO5

##### .2 HSPI - SPI2

- SCLK: GPIO14
- MOSI: GPIO13
- MISO: GPIO12
- CS: GPIO15

#### מייפוי רגליים:



מערכת החיבור בין RFID-RC522 לברker :ESP32

ESP32	RFID-RC522
Vcc (3.3v)	VCC +3.3V
RST	22
GND	GND
MISO	19
MOSI	23
SCK	18
SS/SDA	5

#### שימושים נפוצים:

- תקשורת עם צגים
- קריאה/כתיבה לכרטיסי SD
- תקשורת עם חיישנים
- תקשורת עם מודול RF
- העברת נתונים להתקנים היקפיים

להלן דוגמת קוד לשימוש בתקשרות SPI:

```
from machine import Pin, SPI
import time

class SPIExample:
    def __init__(self):
        # הגדלת SPI פינוי
        self.sck = Pin(18, Pin.OUT)      # שעון #
        self.mosi = Pin(23, Pin.OUT)     # מוצא נתונים #
        self.miso = Pin(19, Pin.IN)      # כניסה נתונים #
```

```

        self.cs = Pin(5, Pin.OUT)          בחרית רכיב #

# אתחול אובייקט SPI
self.spi = SPI(2, baudrate=1000000,
               polarity=0, phase=0,
               sck=self.sck,
               mosi=self.mosi,
               miso=self.miso)

def write_data(self, data):
    """ SPI פונקציה לשילוח נתונים דרך
    """
    self.cs.value(0) # הפעלה CS (active low)
    self.spi.write(bytes(data))
    self.cs.value(1) # ביטול CS

def read_data(self, bytes_count):
    """ SPI פונקציה לקריאת נתונים דרך
    """
    self.cs.value(0)
    data = self.spi.read(bytes_count)
    self.cs.value(1)
    return data

def transfer_data(self, data):
    """ פונקציה לשילוח וקבלת בו-זמן נתונים
    """
    self.cs.value(0)
    received = self.spi.write_readinto(bytes(data), bytes(len(data)))
    self.cs.value(1)
    return received

# דוגמת שימוש
def main():
    spi_example = SPIExample()

    # דוגמה לשילוח נתונים #
    data_to_send = [0x55, 0xAA, 0x0F]
    print("שלח נתונים:", data_to_send)
    spi_example.write_data(data_to_send)

    # דוגמה לקריאת נתונים #
    received_data = spi_example.read_data(3)
    print("התבלו נתונים:", list(received_data))

    # דוגמה להעברה דו-כיוונית #
    transfer_data = [0x12, 0x34, 0x56]
    received = spi_example.transfer_data(transfer_data)
    print("נתונים שהעבירו:", transfer_data)
    print("נתונים שהתקבלו:", list(received))

```

```

if __name__ == "__main__":
    main()

```

הסברים נוספים על הקוד:

**1. אתחול SPI:**

- בחירת עroz
- הגדרת קצב השעון (baudrate)
- קביעת הקוטביות והפaza
- הגדרת הפינים

**2. פעולות בסיסיות:**

- write - שליחת נתונים
- read - קריית נתונים
- העברת DO-씨ונייה - write\_readinto

**3. טיפים לשימוש:**

- תמיד לשחרר את-CS בסיום התקשרות
- להתאים את מהירות השעון לתקן המחבר
- לוודא חיבור נכון של הפינים

להלן מימוש המחלקה המתפל בקורא הcredטיסים, יש ליצור קובץ בשם : mfrc522.py ולשמר אותו הקובץ בזיכרון בקר ה-ESP32.

```

from machine import Pin, SPI
import utime
#https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf
class MFRC522:

    OK = const(0)
    NOTAGERR = const(1)
    ERR = const(2)
    REQIDL = const(0x26)
    REQALL = const(0x52)
    AUTHENT1A = const(0x60)
    AUTHENT1B = const(0x61)

    def __init__(self, spi, cs=None, rst=None):
        self.spi = spi
        self.cs = cs
        self.rst = rst
        if self.rst is None:
            self.rst = DummyPin()
        if self.cs is None:
            self.cs = DummyPin()
        self.cs.init(self.cs.OUT, value=1)
        self.rst.init(self.rst.OUT, value=1)
        self.initSensor()

    def initSensor(self):
        self.reset()
        self._wreg(0x2A, 0x8D)
        self._wreg(0x2B, 0x3E)
        self._wreg(0x2D, 30)
        self._wreg(0x2C, 0)

```

```

        self._wreg(0x15, 0x40)
        self._wreg(0x11, 0x3D)
        self.antenna_on()

    def reset(self):
        self._wreg(0x01, 0x0F)

    def antenna_on(self, on=True):
        """
        Turns on the antenna of an MFRC522 RFID module by setting the TxControlReg
        register.

        If the antenna is already on, this method does nothing.
        """
        if on and ~(self._rreg(0x14) & 0x03):
            self._sflags(0x14, 0x03)
        else:
            self._cflags(0x14, 0x03)

    def request(self, mode):
        self._wreg(0x0D, 0x07)
        stat, recv, bits = self._tocard(0x0C, [mode])
        if (stat != self.OK) | (bits != 0x10):
            stat = self.ERR
        return stat, bits

    def anticoll(self):
        ser_chk = 0
        ser = [0x93, 0x20]
        self._wreg(0x0D, 0x00)
        stat, recv, bits = self._tocard(0x0C, ser)
        if stat == self.OK:
            if len(recv) == 5:
                for i in range(4):
                    ser_chk = ser_chk ^ recv[i]
                if ser_chk != recv[4]:
                    stat = self.ERR
            else:
                stat = self.ERR
        return stat, recv

    def select_tag(self, uid):
        """
        Selects a tag or card for communication.

        Args:
            uid (list): The unique identifier of the tag or card.

        Returns:
            int: The status of the command execution (1 or 0).
        """
        buf = [0x93, 0x70] + uid[:5]
        buf += self._crc(buf)
        stat, recv, bits = self._tocard(0x0C, buf)
        return self.OK if (stat == self.OK) and (bits == 0x18) else self.ERR

    def authentication(self, mode, addr, sect, uid):
        return self._tocard(0x0E, [mode, addr] + sect + uid[:4])[0]

    def stop_crypto1(self):
        self._cflags(0x08, 0x08)

```

```

def read(self, addr):
    data = [0x30, addr]
    data += self._crc(data)
    stat, recv, _ = self._tocard(0x0C, data)
    return recv if stat == self.OK else None

def write(self, addr, data):
    buf = [0xA0, addr]
    buf += self._crc(buf)
    stat, recv, bits = self._tocard(0x0C, buf)
    if not (stat == self.OK) or not (bits == 4) or not ((recv[0] & 0x0F) ==
0x0A):
        stat = self.ERR
    else:
        buf = data + self._crc(data)
        stat, recv, bits = self._tocard(0x0C, buf)
        if not (stat == self.OK) or not (bits == 4) or not ((recv[0] & 0x0F) ==
0x0A):
            stat = self.ERR
    return stat

def _wreg(self, reg, val):
    self.cs.value(0)
    self.spi.write(b'\xc' % int(0xff & ((reg << 1) & 0x7e)))
    self.spi.write(b'\c' % int(0xff & val))
    self.cs.value(1)

def _rreg(self, reg):
    self.cs.value(0)
    self.spi.write(b'\c' % int(0xff & (((reg << 1) & 0x7e) | 0x80)))
    val = self.spi.read(1)
    self.cs.value(1)
    return val[0]

def _sflags(self, reg, mask):
    self._wreg(reg, self._rreg(reg) | mask)

def _cflags(self, reg, mask):
    self._wreg(reg, self._rreg(reg) & (~mask))

def _tocard(self, cmd, send):
    recv = []
    bits = irq_en = wait_irq = n = 0
    stat = self.ERR
    if cmd == 0x0E:
        irq_en = 0x12
        wait_irq = 0x10
    elif cmd == 0x0C:
        irq_en = 0x77
        wait_irq = 0x30
    self._wreg(0x02, irq_en | 0x80)
    self._cflags(0x04, 0x80)
    self._sflags(0x0A, 0x80)
    self._wreg(0x01, 0x00)
    for c in send:
        self._wreg(0x09, c)
    self._wreg(0x01, cmd)

```

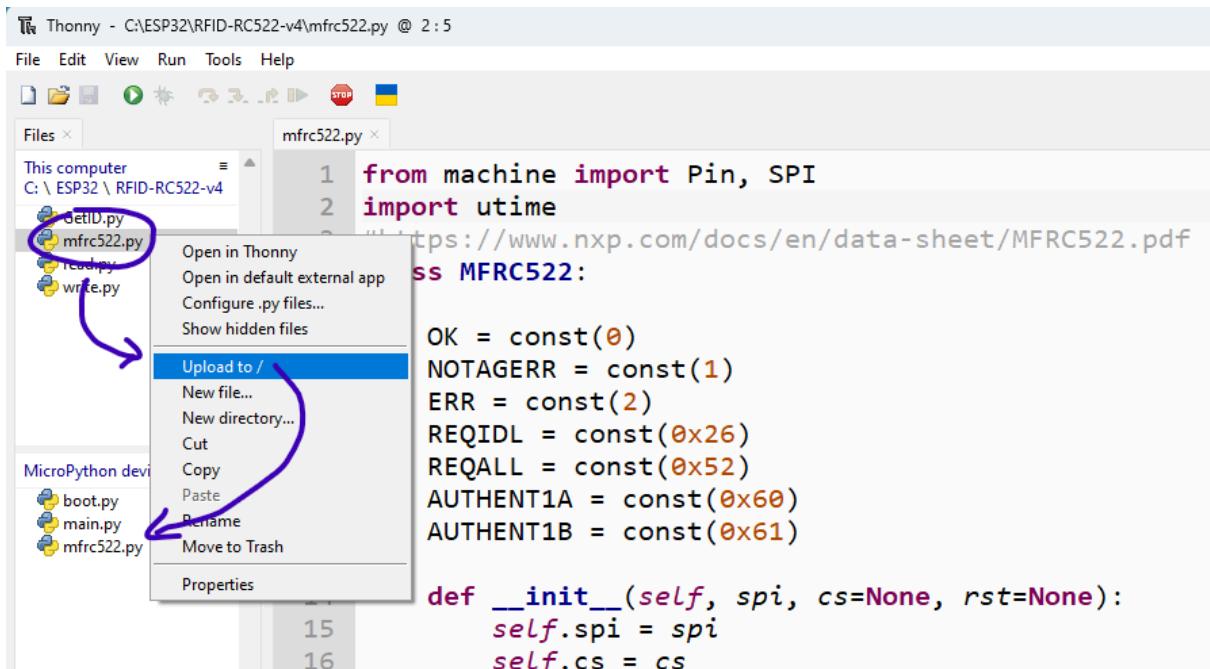
```

if cmd == 0x0C:
    self._sflags(0x0D, 0x80)
i = 2000
while True:
    n = self._rreg(0x04)
    i -= 1
    if ~((i != 0) and ~(n & 0x01) and ~(n & wait_irq)):
        break
self._cflags(0x0D, 0x80)
if i:
    if (self._rreg(0x06) & 0x1B) == 0x00:
        stat = self.OK
        if n & irq_en & 0x01:
            stat = self.NOTAGERR
    elif cmd == 0x0C:
        n = self._rreg(0x0A)
        lbits = self._rreg(0x0C) & 0x07
        if lbits != 0:
            bits = (n - 1) * 8 + lbits
        else:
            bits = n * 8
        if n == 0:
            n = 1
        elif n > 16:
            n = 16
        for _ in range(n):
            recv.append(self._rreg(0x09))
    else:
        stat = self.ERR
return stat, recv, bits

def _crc(self, data):
    self._cflags(0x05, 0x04)
    self._sflags(0x0A, 0x80)
    for c in data:
        self._wreg(0x09, c)
    self._wreg(0x01, 0x03)
    i = 0xFF
    while True:
        n = self._rreg(0x05)
        i -= 1
        if not ((i != 0) and not (n & 0x04)):
            break
    return [self._rreg(0x22), self._rreg(0x21)]

```

ניתן לראות כיצד מעבירים את הקובץ לבקר:



דוגמה 1: קריית קוד הבקר ובניית מנגנון אבטחה בסיסי

להלן הקוד:

```

from mfrc522 import MFRC522
from machine import Pin, SPI, reset

spi = SPI(2,baudrate=100000, polarity=0, phase=0, sck=Pin(18, Pin.OUT),
mosi=Pin(23, Pin.OUT), miso=Pin(19, Pin.IN))
rdr = MFRC522(spi, cs=Pin(5, Pin.OUT), rst=Pin(22, Pin.OUT))
print("Place the card close to the sensor")
print('Presse Ctrl-C to exit')

ApprovedCardList = [
    [58, 58, 199, 36],
    [249, 65, 207, 153]
]

try:
    while True:
        stat, tag_type = rdr.request(rdr.REQIDL)
        if stat == rdr.OK:
            stat, raw_uid = rdr.anticoll()
            if stat == rdr.OK:
                cardId=[raw_uid[0], raw_uid[1], raw_uid[2], raw_uid[3]]
                print("Card detected. id:",cardId)

                for lst in ApprovedCardList:
                    if lst == cardId:
                        print("OK")

except KeyboardInterrupt:
    print('Ctrl-C pressed...exiting')
    reset()

```

## דוגמה 2: כתיבת בлок נתונים לזכרון של כרטיס ה-RFID

להלן הקוד:

```
from mfrc522 import MFRC522
from machine import Pin, SPI, reset

spi = SPI(2,baudrate=100000, polarity=0, phase=0, sck=Pin(18, Pin.OUT),
mosi=Pin(23, Pin.OUT), miso=Pin(19, Pin.IN))
rdr = MFRC522(spi, cs=Pin(5, Pin.OUT), rst=Pin(22, Pin.OUT))
print("Place the card close to the sensor")
print('Presse Ctrl-C to exit')

try:
    while True:
        stat, tag_type = rdr.request(rdr.REQIDL)
        if stat == rdr.OK:
            (stat, idCard) = rdr.anticoll()
            if stat == rdr.OK:
                print("Card detected")
                cardId=[idCard[0], idCard[1], idCard[2], idCard[3]]
                print(" - id :", cardId)
                if rdr.select_tag(idCard) == rdr.OK:
                    key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
                    DataToSend =
[ord('G'),ord('a'),ord('d'),ord('i'),0,1,2,3,4,5,6,7,8,9,10,255]
                    if rdr.authentication(rdr.AUTHENT1A, 10, key, idCard)
== rdr.OK:
                        stat = rdr.write(10, DataToSend)
                        rdr.stop_crypto1()
                        if stat == rdr.OK:
                            print("Data written to card")
                        else:
                            print("Failed to write data to card")
                    else:
                        print("Authentication error")
                else:
                    print("Failed to select tag")
except KeyboardInterrupt:
    print('Ctrl-C pressed...exiting')
    reset()
```

## דוגמה 3: קריית בлок נתונים לזכרון של כרטיס ה-RFID

להלן הקוד:

```
from mfrc522 import MFRC522
from machine import Pin, SPI, reset

spi = SPI(2,baudrate=100000, polarity=0, phase=0, sck=Pin(18, Pin.OUT),
mosi=Pin(23, Pin.OUT), miso=Pin(19, Pin.IN))
rdr = MFRC522(spi, cs=Pin(5, Pin.OUT), rst=Pin(22, Pin.OUT))
print("Place the card close to the sensor")
```

```

print('Presse Ctrl-C to exit')

try:
    while True:
        stat, tag_type = rdr.request(rdr.REQIDL)
        if stat == rdr.OK:
            stat, idCard = rdr.anticoll()
            if stat == rdr.OK:
                print("Card detected")
                cardId=[idCard[0], idCard[1], idCard[2], idCard[3]]
                print(" - id :", cardId)
            if rdr.select_tag(idCard) == rdr.OK:
                key = [255, 255, 255, 255, 255, 255]
                if rdr.authentication(rdr.AUTHENT1A, 10, key, idCard) ==
rdr.OK:
                    print("Read data from card: %s" % rdr.read(10))
                    rdr.stop_crypto1()
                else:
                    print("Authentication error")
            else:
                print("Failed to select tag")
except KeyboardInterrupt:
    print('Ctrl-C pressed...exiting')
    reset()

```

## משימה 21 - עדכון RTC פנימי בברouter תוך שימוש ב- API מבוסס JSON

קישורים:

<https://www.hackster.io/alankrantas/very-simple-micropython-esp8266-esp-12-web-clock-3c5c6f>

<https://gist.github.com/drrk/4a17c4394f93d0f9123560af056f6f30>

<https://randomnerdtutorials.com/esp32-ntp-client-date-time-arduino-ide/>

<https://docs.micropython.org/en/latest/esp32/quickref.html#real-time-clock-rtc>

<https://techtutorialsx.com/2017/06/11/esp32-esp8266-micropython-http-get-requests/>

<https://docs.micropython.org/en/latest/library/ujson.html>

[https://www.w3schools.com/python/python\\_tuples.asp](https://www.w3schools.com/python/python_tuples.asp)

<http://docs.micropython.org/en/v1.9.3/pyboard/library/pyb.RTC.html#pyb.RTC.datetime>

במשימה זו נפתח קוד תוכנה המתחבר לאינטרנט כדי לקבל עדכון זמן דרך הכתובת הבאה:

<http://worldtimeapi.org/api/timezone/Asia/Jerusalem>

קישור זה מחזיר לנו מחרוזת טקסט בפורמט JSON הכול את השעה המדוייקת באופן הבא:



```
{"week_number":40,"utc_offset":"+03:00","utc_datetime":"2019-10-01T16:10:25.763035+00:00","unixtime":1569946225,"timezone":"Asia/Jerusalem","raw_offset":7200,"dst_until":"2019-10-26T23:00:00+00:00","dst_offset":3600,"dst_from":"2019-03-29T00:00:00+00:00","dst":true,"day_of_year":274,"day_of_week":2,"datetime":"2019-10-01T19:10:25.763035+03:00","client_ip":"77.138.54.225","abbreviation":"IDT"}
```

נבחן את מבנה הנתונים שהשתקבל

```
{  
    "week_number": 40,  
    "utc_offset": "+03:00",  
    "utc_datetime": "2019-10-01T16:10:25.763035+00:00",  
    "unixtime": 1569946225,  
    "timezone": "Asia/Jerusalem",  
    "raw_offset": 7200,  
    "dst_until": "2019-10-26T23:00:00+00:00",  
    "dst_offset": 3600,  
    "dst_from": "2019-03-29T00:00:00+00:00",  
    "dst": true,  
    "day_of_year": 274,  
    "day_of_week": 2,
```

```

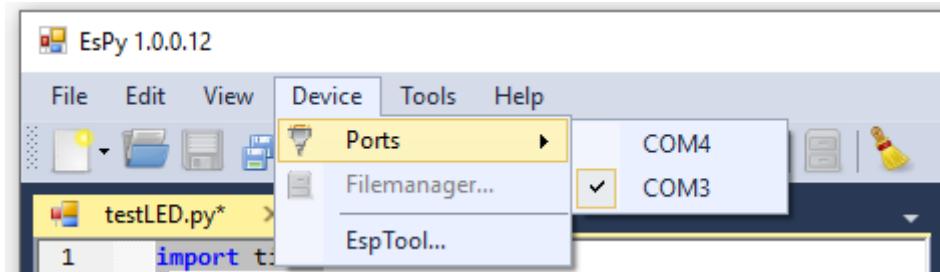
    "datetime": "2019-10-01T19:10:25.763035+03:00",
    "client_ip": "77.138.54.225", "abbreviation": "IDT"
}

```

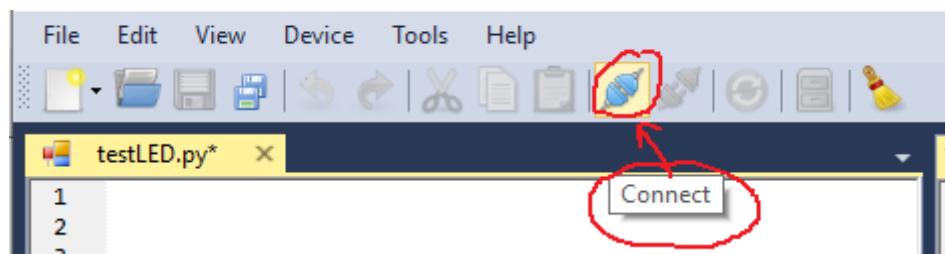
בתרגום זה נחלץ את השעה מຕוך datetime ונעדכן בעזרתו את השעון הפנימי של הבקר.

### שלב 1: חיבור הבקר למחשב

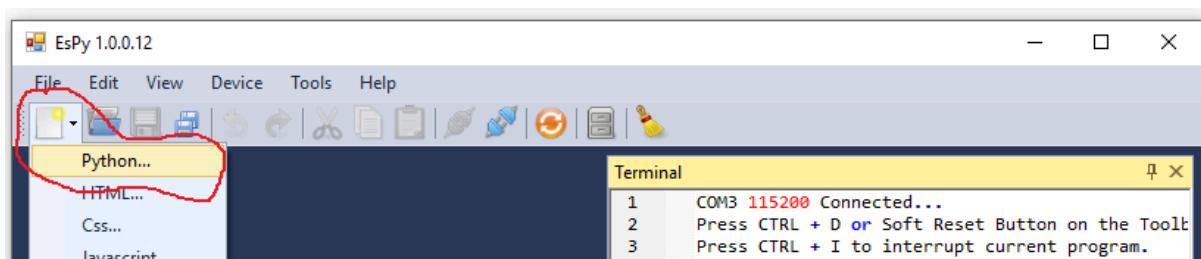
נעיל את קובץ הרצה exe ונתחבר לבקר על ידי לחיצה על Device ו- Port בחר את מספר המפתח (Port) שלו מוחובר הבקר שלנו



ואז נלחץ על Connect



נפתח קובץ Python חדש לכתיבה קוד התוכנית:



### שלב 2: כתיבת קבצי הקוד

להלן קוד התוכנית:

```

import network
import urequests
import ujson

```

```

import utime
from machine import RTC

ssid = "*****"          #wifi name
pw = "*****"            #wifi password
web_query_delay = 200000 #interval time of web JSON query
url = "http://worldtimeapi.org/api/timezone/Asia/Jerusalem"

# wifi connection
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(ssid, pw)
while not wifi.isconnected():
    pass
print("IP Address:" , str(wifi.ifconfig()[0]))

#internal RTC(Real Time Clock)
rtc = RTC()
update_time = utime.ticks_ms() - web_query_delay

while True:
    if not wifi.isconnected():
        machine.reset() # if lose wifi connection reboot ESP32
    if utime.ticks_ms() - update_time >= web_query_delay:
        response = urequests.get(url)
        if response.status_code == 200: # if query success
            #print("Date&Time in JSON format: " + response.text)
            js = ujson.loads(response.text) # you can also use parsed =
response.json()
            s_dt = str(js["datetime"])
            year = int(s_dt[0:4])
            month = int(s_dt[5:7])
            day = int(s_dt[8:10])
            hour = int(s_dt[11:13])
            minute = int(s_dt[14:16])
            second = int(s_dt[17:19])

```

```

subsecond = int(round(int(s_dt[20:26]) / 10000))

# update internal RTC

rtc.datetime((year, month, day, 0, hour, minute, second,
subsecond))

update_time = utime.ticks_ms()

print("RTC updated\n")

dt = rtc.datetime()

s_date = "{:02}/{:02}/{:4}".format(dt[1], dt[2], dt[0])

s_time = "{:02}:{:02}:{:02}".format(dt[4], dt[5], dt[6])

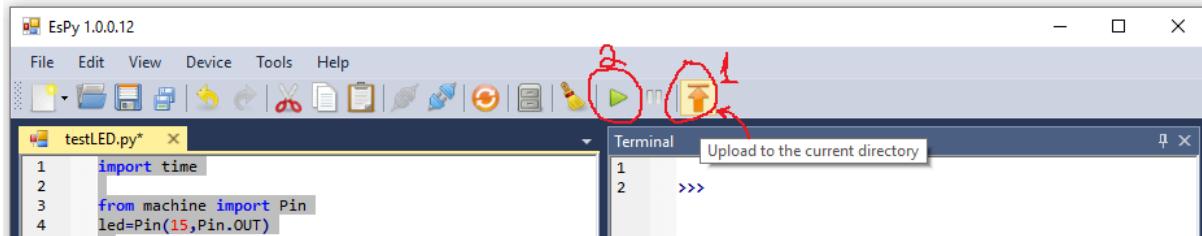
print("Date: " + s_date)

print("Time: " + s_time)

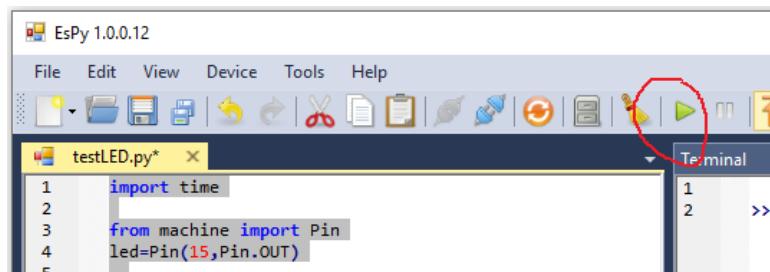
utime.sleep(2)

```

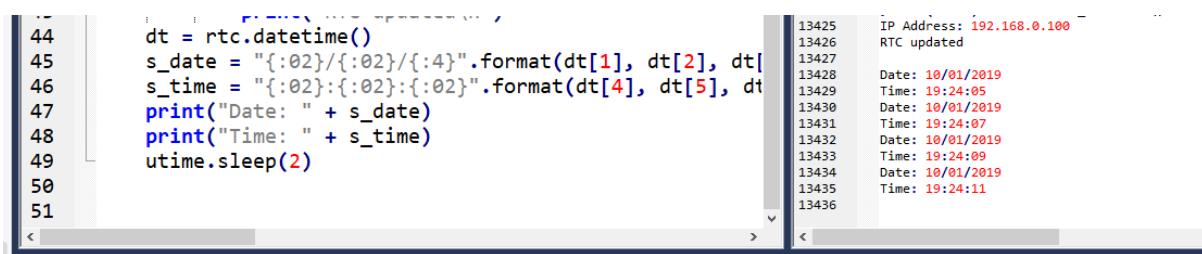
נצרוב את הקובץ שיצרנו על שימוש בלחץ



להרצת התוכנית בברker נעמוד על הקובץ שכתבנו ונלחץ על Run



נקבל פלט הדומה לזה המוצג כאן:



## משימה 22 - תקשורת אלחוטית מבוססת מקמ"ש NRF24L01

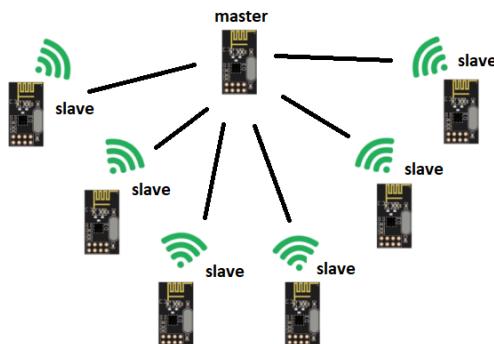
קישורים:

<https://github.com/micropython/micropython/tree/master/drivers/nrf24l01>

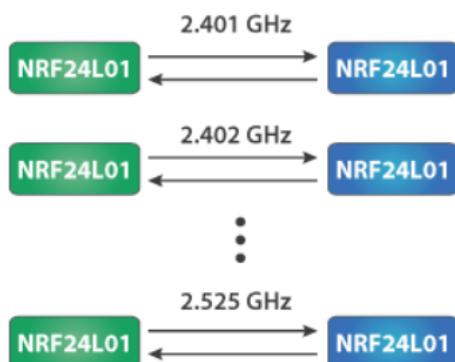
<http://docs.micropython.org/en/v1.9.3/pyboard/library/pyb.SPI.html>

مصدر מקלט NRF24L01 עובד בתדר של 2.4 גיגה הרץ. הרכיב מסוגל להעביר תקשורת ספרטיטית בקצב שבין עד 250Kbps 2Mbps. בטוחה שנע בין סמך מטרים בחלל סגור ועד 100 מטר בחלל פתוח.

מקמ"ש ( مصدر/מקלט ) NRF24L01 יכול להשתמש ב- 125 ערוצי תקשורת שונים. כאשר כל ערוץ יהיה לתקשורת בו זמינות בין יחידה מרכזית אחת ייחד עם עד 6 יחידות משנה נוספות.



כל ערוץ תקשורת מקבל תדר פנוי בתחום שבין 2.401GHz עד 2.525GHz



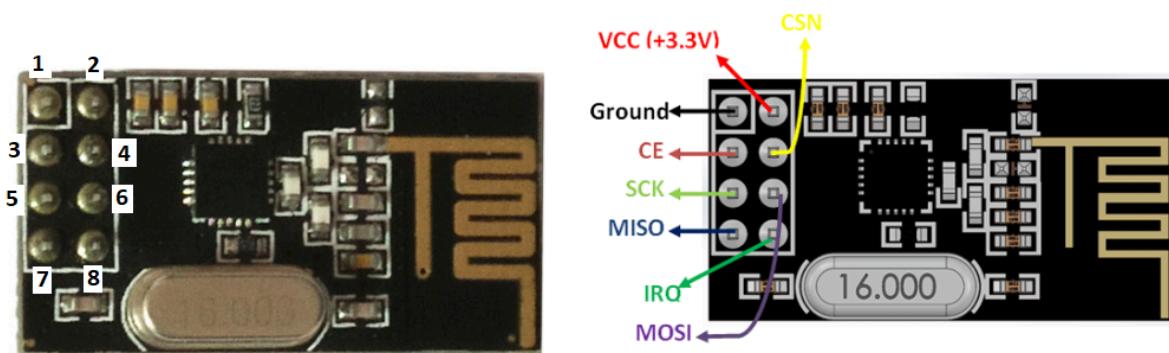
להלן רשימת המאפיינים הטכניים של הרכיב

- 2.4GHz RF transceiver Module
- Operating Voltage: 3.3V
- Nominal current: 50mA
- Range : 50 – 200 feet
- Operating current: 250mA (maximum)
- Communication Protocol: SPI
- Baud Rate: 250 kbps - 2 Mbps.
- Channel Range: 125
- Maximum Pipelines/node : 6
- Low cost wireless solution

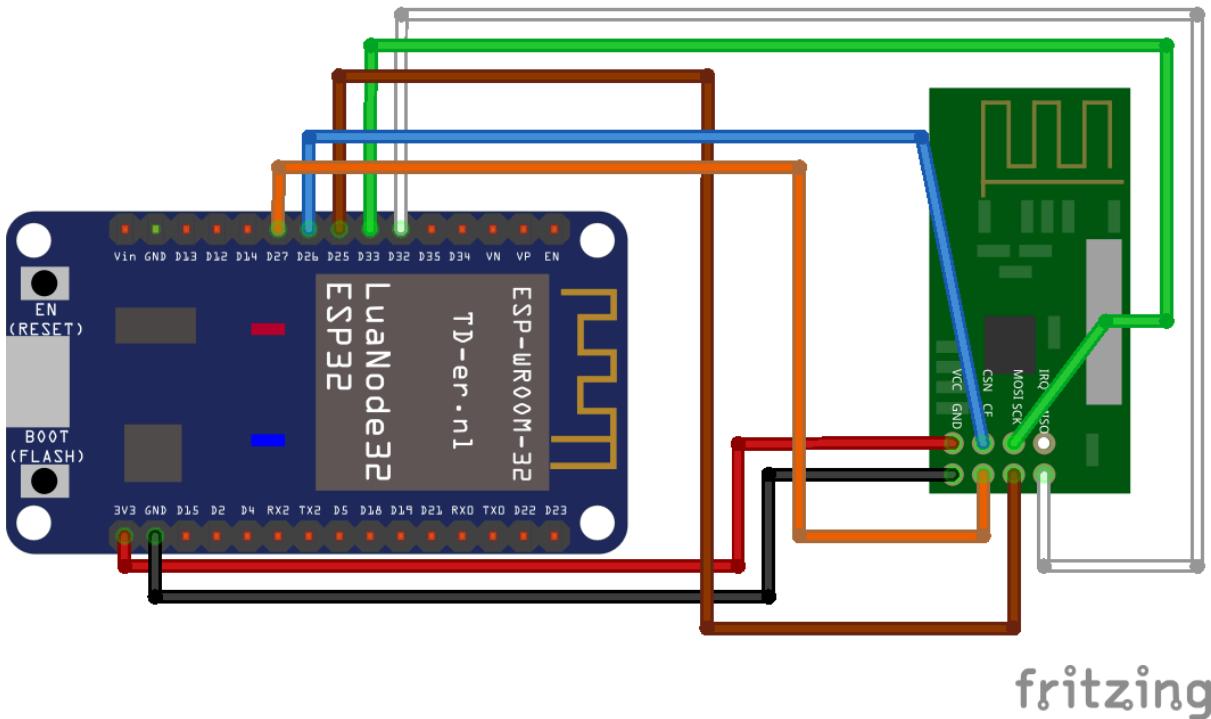
שימוש לבן מקום מ"ש עובד במתוח עבודה של 3.3V ולא 5V. למרות זאת הדק' הבדיקה של הרכיב יכולים להתחבר לבקרים כמו UNO Arduino המספקים מתוח בקרה ברמה של 5V.

מיופי רגליים:

- 1 - Ground - Connected to the Ground of the system
- 2 - Vcc - Powers the module using 3.3V
- 3 - CE - Used to enable SPI communication
- 4 - CSN - This pin has to be kept high always, else it will disable the SPI
- 5 - SCK - Provides the clock pulse using which the SPI communication works
- 6 - MOSI - Connected to MOSI pin of MCU, for the module to receive data from the MCU
- 7 - MISO - Connected to MISO pin of MCU, for the module to send data from the MCU
- 8 - IRQ - It is an active low pin and is used only if interrupt is required



אופן החיבור בין בקר ESP32 לבין מוקם מ"ש NRF24L01:



fritzing

### תקשורת SPI בברker ESP32

There are two SPI drivers. One is implemented in software (bit-banging) and works on all pins, and is accessed via the `machine.SPI` class.

There are two hardware SPI channels that allow faster transmission rates (up to 80Mhz). These may be used on any IO pins that support the required direction and are otherwise unused (see Pins and GPIO) but if they are not configured to their default pins then they need to pass through an extra layer of GPIO multiplexing, which can impact their reliability at high speeds. Hardware SPI channels are limited to 40MHz when used on pins other than the default ones listed below.

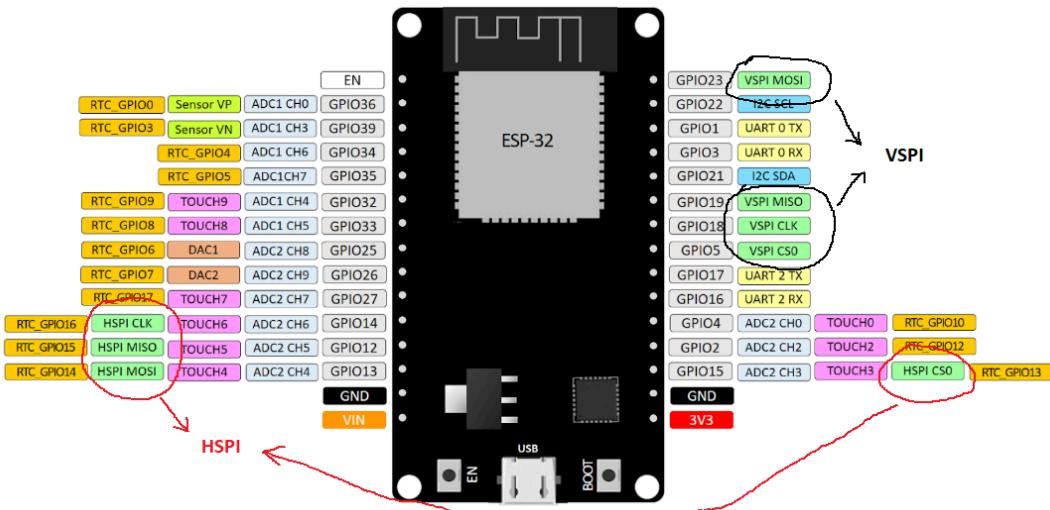
	HSPI (id=1)	VSPI (id=2)
sck	14	18
mosi	13	23
miso	12	19

Hardware SPI has the same methods as Software SPI above:

```
from machine import Pin, SPI

hspi = SPI(1, 10000000, sck=Pin(14), mosi=Pin(13), miso=Pin(12))

vspi = SPI(2, baudrate=80000000, polarity=0, phase=0, bits=8, firstbit=0,
           sck=Pin(18), mosi=Pin(23), miso=Pin(19))
```



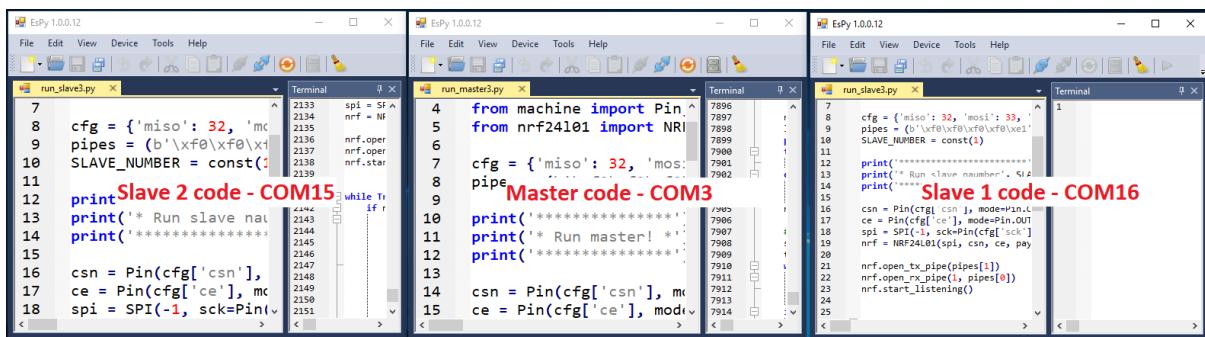
בתרגיל זה נעשה שימוש ב-SPI\_7 כאשר נשנה את הגדרות הדק'י התקשרות שבין הבקר למקם"ש על ידי תוכנה. להלן הוראת האתחול של תקשורת SPI בקוד תוך כדי הגדרת הדק'י העובדה:

```
vsopi = SPI(2, baudrate=2000000, polarity=0, phase=0, bits=8, firstbit=0,
sck=Pin(18), mosi=Pin(23), miso=Pin(19))
```

במשימה זו נפתח קוד תוכנה המיציר תקשורת נתונים מבוססת טקסט בין יחידת בוס (Master) לבין 2 יחידות משנה (Slave).

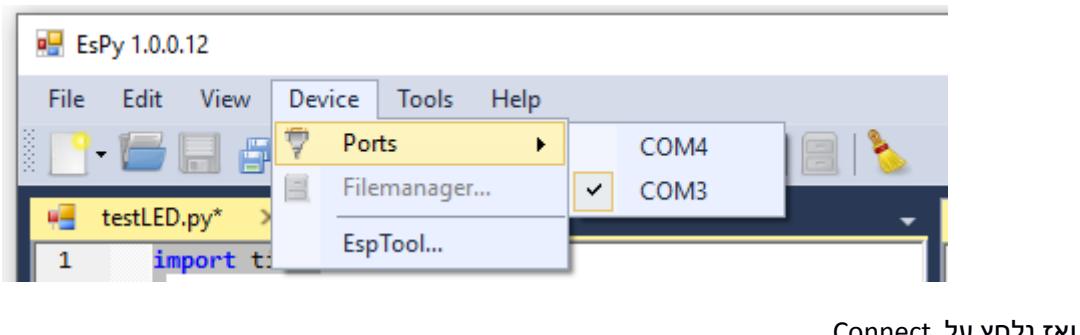
במשימה נעשה שימוש בשלוש בקרים ESP32 המחברות כל אחת למקם"ש NRF24L01 ועובדים בו זמנית. אחד מהם מוגדר בתוכנה כ-Master ו-2 האחרים יוגדרו בתוכנה-C-Slave.

כדי למש את התרגיל נפתח במקביל 3 סביבות פיתוח על גבי המחשב ובכל אחת מהם נתחבר לבקר אחר בהתאם למפתח שאלוי מחוברים כל אחד מהבקרים.

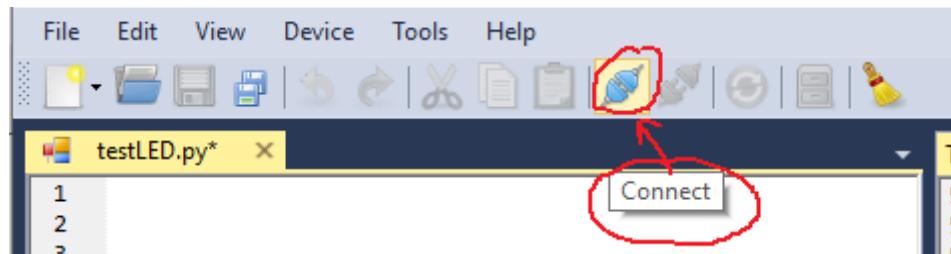


**שלב 1: חיבור שלוש בקרים למחשב אחד**

נפעיל את קובץ ההרצה EsPy.exe שלוש פעמים על אותו המחשב ונתחבר לכל אחד מהבקרים על ידי לחיצה על Port - Device



ואז נלחץ על Connect

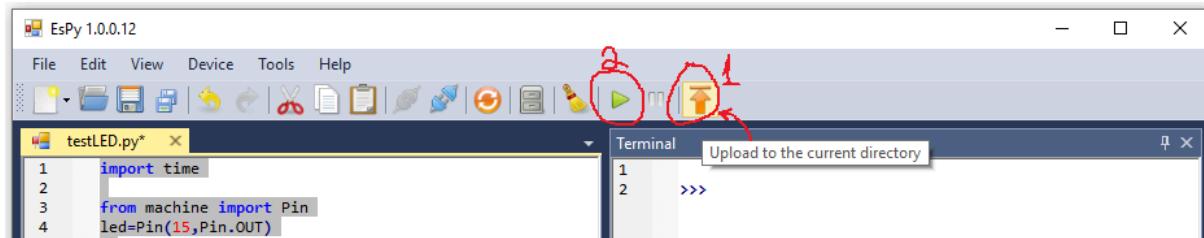


נפתח קובץ Python חדש בכל אחד מהບקרים כדי להעתיק לבקר את קוד המחלקה `nrf24l01.py`.

גרסה עדכנית של המחלקה ניתן להוריד דרך ה קישור הבא:

<https://github.com/micropython/micropython/tree/master/drivers/nrf24l01>

נצרוב את הקובץ שפתחנו על כל אחד משלוש הבקרים על שימוש בלחצן "Upload to the current directory"



## שלב 2: כתיבת קבצי הקוד

כתבו 3 קודים שונים האחד לבקר שיממש את השרת (Master) ו-2 קודים ל-2 תוכנות Save שימתינו לפניות מהשרת.

להלן קוד התוכנית המעודכן ל-Master :

```
import sys
import ustruct as struct
import utime
from machine import Pin, SPI
from nrf24l01 import NRF24L01

cfg = { 'miso': 32, 'mosi': 33, 'sck': 25, 'csn': 26, 'ce': 27}
```

```

pipes = (b'\xf0\xf0\xf0\xf0\xe1', b'\xf0\xf0\xf0\xf0\xd2')

print('*****')
print('* Run master! *')
print('*****')

csn = Pin(cfg['csn'], mode=Pin.OUT, value=1)
ce = Pin(cfg['ce'], mode=Pin.OUT, value=0)
spi = SPI(-1, sck=Pin(cfg['sck']), mosi=Pin(cfg['mosi']),
miso=Pin(cfg['miso']))
nrf = NRF24L01(spi, csn, ce, payload_size=16)

nrf.open_tx_pipe(pipes[0])
nrf.open_rx_pipe(1, pipes[1])
nrf.start_listening()

num_successes = 0
num_failures = 0
led_state = 0

while num_successes < 10 and num_failures < 10:

    nrf.stop_listening()      #stop listening and send packet
    lst = [utime.ticks_ms(),17,18,19]
    print('Sending:', lst)
    try:
        nrf.send(struct.pack('iiii', lst[0],lst[1],lst[2],lst[3]))
    except OSError:
        pass

    nrf.start_listening()     #start listening again

    # wait for response
    start_time = utime.ticks_ms()
    timeout = False
    while not nrf.any() and not timeout:

```

```

        if utime.ticks_diff(utime.ticks_ms(), start_time) > 250:
            timeout = True

    if timeout:
        print('Failed, response timed out')
        num_failures += 1

    else:
        rlst = struct.unpack('ii', nrf.recv())
        print('Got OK response from slave', rlst[1] , 'in',
utime.ticks_diff(utime.ticks_ms(), rlst[0]), 'ms')
        num_successes += 1

    utime.sleep_ms(250)

print('Master finished sending; successes=%d, failures=%d' %
(num_successes, num_failures))

```

להלן קוד התוכנית המיעודת לכל אחד מ-2 ה-Slave:

```

import sys
import ustruct as struct
import utime
from machine import Pin, SPI
from nrf24l01 import NRF24L01
from micropython import const

cfg = {'miso': 32, 'mosi': 33, 'sck': 25, 'csn': 26, 'ce': 27}
pipes = (b'\xf0\xf0\xf0\xe1', b'\xf0\xf0\xf0\xd2')
SLAVE_NUMBER = const(1)

print('*****')
print('* Run slave naumber', SLAVE_NUMBER, '*')
print('*****')

csn = Pin(cfg['csn'], mode=Pin.OUT, value=1)
ce = Pin(cfg['ce'], mode=Pin.OUT, value=0)

```

```

spi = SPI(-1, sck=Pin(cfg['sck']), mosi=Pin(cfg['mosi']),
miso=Pin(cfg['miso']))

nrf = NRF24L01(spi, csn, ce, payload_size=16)

nrf.open_tx_pipe(pipes[1])
nrf.open_rx_pipe(1, pipes[0])
nrf.start_listening()

while True:
    if nrf.any():
        while nrf.any():
            buf = nrf.recv()
            lst = struct.unpack('iiii', buf)
            print('received:', lst[0], lst[1], lst[2], lst[3])
            utime.sleep_ms(15)

        # Give master time to get into receive mode.
        utime.sleep_ms(10)
        nrf.stop_listening()
        try:
            nrf.send(struct.pack('ii', lst[0], SLAVE_NUMBER))
        except OSError:
            pass
        print('sent response')
        nrf.start_listening()

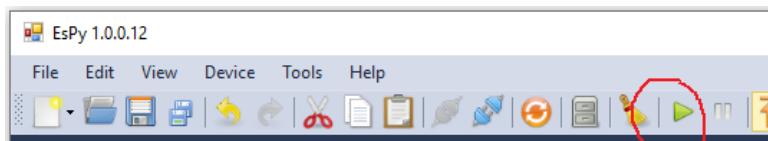
```

הבדל היחיד בין 2 תוכנות ה-Slave הוא מספרו כפי שמוגדר בשורה הבאה:

```
SLAVE_NUMBER = const(1)
```

הגדירו לכל אחד מתוכנות ה-Slave מספר שונה.

נצרוב את כל אחד מהקבצים שיצרנו על שימוש בלחץ "Upload to the current directory" להרצת התוכנית ב.tk נultimo על הקובץ שכתבנו ונלחץ על Run



**חשיבות:** תוכנות ה- Slave חייבות לזרע תחילת לפני הרצת תוכנת ה- Master. זה בגלל שתוכנת ה- Master היא זו שיזומת תקשורת עם ה- Slave. והאחרונה (תוכנת ה- Slave) מחזירה ל- Master הודעה אישור. ללא תוכנת Slave נקבל הודעה timed out Failed.

**נקבל פلت הדומה לזה המוצג כאן:**

The screenshot shows two MicroPython IDE windows. The top window, titled 'run\_master3.py', contains the following code:

```
led_state = 0
while num_successes < 10 and num_failures < 10:
    nrf.stop_listening()      #stop listening and send
    lst =[utime.ticks_ms(),17,18,19]
    print('Sending:', lst)
    try:
        nrf.send(struct.pack('iiii', lst[0],lst[1],1
    except OSError:
        pass

    nrf.start_listening()     #start listening again
# wait for response
```

The bottom window, titled 'run\_slave3.py', contains the following code:

```
import sys
import ustruct as struct
import utime
from machine import Pin, SPI
from nrf24l01 import NRF24L01
from micropython import const

cfg = {'miso': 32, 'mosi': 33, 'sck': 25, 'csn': 26, 'ce': 27}
pipes = (b'\xf0\xf0\xf0\xf0\xe1', b'\xf0\xf0\xf0\xf0\xd2')
SLAVE_NUMBER = const(1)

print('*'*16)
print('* Run slave naumber', SLAVE_NUMBER, '*')
print('*'*16)
```

Both windows have a terminal tab on the right. The master terminal shows the following output:

```
*****  
* Run master! *  
*****  
Sending: [29691506, 17, 18, 19]  
Got OK response from slave 1 in 42 ms  
Sending: [29691800, 17, 18, 19]  
Got OK response from slave 1 in 40 ms  
Sending: [29692092, 17, 18, 19]  
Got OK response from slave 1 in 40 ms  
Sending: [29692305, 17, 18, 19]  
Got OK response from slave 1 in 39 ms  
Sending: [29692676, 17, 18, 19]  
Got OK response from slave 1 in 40 ms  
Sending: [29692968, 17, 18, 19]  
Got OK response from slave 1 in 40 ms  
Sending: [29693260, 17, 18, 19]  
Got OK response from slave 1 in 39 ms  
Sending: [29693551, 17, 18, 19]  
Got OK response from slave 1 in 40 ms  
Sending: [29693843, 17, 18, 19]  
Got OK response from slave 1 in 39 ms  
Sending: [29694134, 17, 18, 19]  
Got OK response from slave 1 in 40 ms  
Master finished sending; successes=10, failures=0
```

The slave terminal shows the following output:

```
*****  
* Run Slave naumber 1 *  
*****  
received: 29691506 17 18 19  
sent response  
received: 29691800 17 18 19  
sent response  
received: 29692092 17 18 19  
sent response  
received: 29692385 17 18 19  
sent response  
received: 29692676 17 18 19  
sent response  
received: 29692968 17 18 19  
sent response  
received: 29693260 17 18 19  
sent response  
received: 29693551 17 18 19  
sent response  
received: 29693843 17 18 19  
sent response  
received: 29694134 17 18 19  
sent response
```

## נספח א' - בדיקת הספריות הזמינות לתוכנות ב- MicroPython תחת בקר ESP32

בלה בלה.

פתרונות:

<https://docs.micropython.org/en/latest/library/index.html>

שאלה:

```
help('modules')
```

להלן הפלט:

```
Plus any modules on the filesystem
>>> help('modules')
    main          framebuffer      socket          upip
    _boot         gc              ssl              upip_utarfile
    _onewire      hashlib         struct          upysh
    _thread       heapq            sys              urandom
    _webrepl      inisetup        time             ure
    apal06        io              ubinascii       urequests
    array         json            ucollections   uselect
    binascii       machine         ucryptolib     usocket
    btree          math            uctypes         ussl
    builtins      micropython    uerrno          ustruct
    cmath          neopixel       uhashlib        utime
    collections   network        uhashlib       utimeq
    dht           ntptime        uheapq          uwebsocket
    ds18x20       onewire        uio              uzlib
    errno          os              ujson           webrepl
    esp            random         umqtt/robust   webrepl_setup
    esp32          re              umqtt/simple  websocket_helper
    flashbdev     select         uos              zlib
Plus any modules on the filesystem
>>> █
```

## **נספח ב' - תכונות אסינכרוני**

בלה בלה.

קישוריהם:

<https://github.com/peterhinch/micropython-async/blob/master/TUTORIAL.md>

התקינה:

Hardware without internet connectivity (copy source)

If micropip.py is not to be used the files should be copied from source. The following instructions describe copying the bare minimum of files to a target device, also the case where uasyncio is to be frozen into a compiled build as bytecode. For the latest release compatible with official firmware files must be copied from the official micropython-lib.

Clone the library to a PC with

```
$ git clone https://github.com/micropython/micropython-lib.git
```

On the target hardware create a uasyncio directory (optionally under a directory lib) and copy the following files to it:

uasyncio/uasyncio/\_\_init\_\_.py

uasyncio.core/uasyncio/core.py

uasyncio.synchro/uasyncio/synchro.py

uasyncio.queues/uasyncio/queues.py

The uasyncio modules may be frozen as bytecode in the usual way, by placing the uasyncio directory and its contents in the port's modules directory and rebuilding.

שאלות:

```
import uasyncio as asyncio
async def bar():
    count = 0
    while True:
        count += 1
        print(count)
        await asyncio.sleep(1)      # Pause 1s

loop = asyncio.get_event_loop()
```

```
loop.create_task(bar())    Schedule ASAP  
loop.run_forever()
```

להלן הפלט:

שאלה:

```
import uasyncio as asyncio  
from uasyncio.synchro import Lock  
  
async def task(i, lock):  
    while 1:  
        await lock.acquire()  
        print("Acquired lock in task", i)  
        await asyncio.sleep(0.5)  
        lock.release()  
  
    async def killer():  
        await asyncio.sleep(10)  
  
loop = asyncio.get_event_loop()  
  
lock = Lock() # The global Lock instance  
  
loop.create_task(task(1, lock))  
loop.create_task(task(2, lock))  
loop.create_task(task(3, lock))  
  
loop.run_until_complete(killer()) # Run for 10s
```

להלן הפלט:

שאלה:

```
help('modules')
```

להלן הפלט:

## נספח ג' - יבוא ספרייה קוד תוך שימוש הכלים PIP יעודיים ל-

בלה בלה.

קישורים:

<https://lemariva.com/blog/2018/03/tutorial-installing-dependencies-on-micropython>

שאלות:

```
ssid_ = 'WIFI NAME'
wp2_pass = 'WIFI PASSWORD'

def do_connect():
    import network
    sta_if = network.WLAN(network.STA_IF)
    if not sta_if.isconnected():
        print('connecting to network....')
        sta_if.active(True)
        sta_if.connect(ssid_, wp2_pass)
        while not sta_if.isconnected():
            pass
    print('network config:', sta_if.ifconfig())

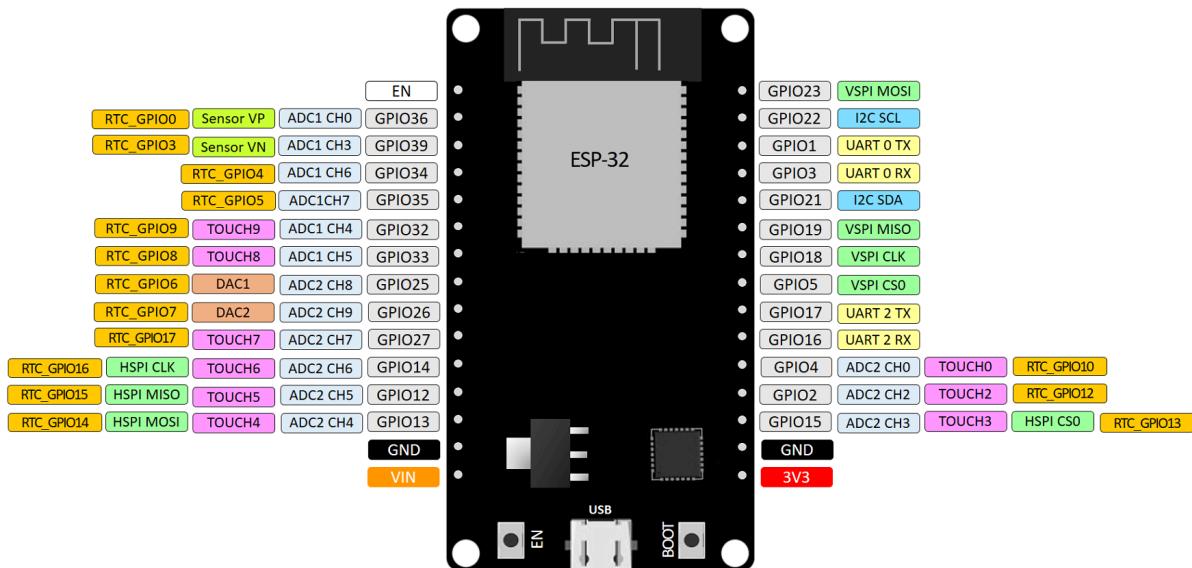
do_connect()

import upip
upip.install('notes-pico')
```

להן הפלט:

```
;32mI (5978) network: CONNECTEDm
;32mI (6568) event: sta ip: 192.168.0.100, mask: 255.255.255.0, gw: 192.168.0.1
;32mI (6568) network: GOT_IPm
network config: ('192.168.0.100', '255.255.255.0', '192.168.0.1', '192.168.0.1')
;32mI (6598) modsocket: Initializingm
Installing to: /lib/
Warning: micropython.org SSL certificate is not validated
Installing notes-pico 0.9.5 from https://files.pythonhosted.org/packages/6d/de/5e877aefba75ed6badc1b7a154efd1f8ea9ce6602a0104d77c11c2e0adea/notes-pico-0.9.5.tar.gz
Installing picoweb 1.7.2 from https://files.pythonhosted.org/packages/a7/cb/1b48103a22f8d67985af5e30ee157c195d2a37b1286466e8744ad8e7e1be/picoweb-1.7.2.tar.gz
Installing utemplate 1.3.1 from https://files.pythonhosted.org/packages/75/11/59ac69a862232afc9ffccadcc95395cfb7b28c17610edc061089d4f03f8/utemplate-1.3.1.tar.gz
Installing pycopy-logging 0.3 from https://files.pythonhosted.org/packages/56/85/47a6790260c85f0ddad460124df9a6ddbaa0b0ac33b0ac89194f6f106276/pycopy-logging-0.3.tar.gz
Installing picopy-pkg_resources 0.2.1 from https://files.pythonhosted.org/packages/05/4a/5481a3225d43195361695645d78f4439527278088c0822fadaf2e93378c/pycopy-pkg_resources-0.2.1.tar.gz
Installing pycopy-btreedb 0.4.1 from https://files.pythonhosted.org/packages/a2/a3/086de7d0374a361359b7e387cef19af40af6f4a79426dc9e83ae6dd1c8/pycopy-btreedb-0.4.1.tar.gz
Installing pycopy-uasyncio 3.2 from https://files.pythonhosted.org/packages/2a/51/d3c436a463adc7752d6009bcbfa9527ad95e3543349240f0e02ed2e5f8/pycopy-uasyncio-3.2.tar.gz
Installing pycopy-uasyncio.core 2.3.2 from https://files.pythonhosted.org/packages/ca/b2/c5bbab0bde7022b6d927a614c026d7bf310d3f8a20e031571fb1a08a433/pycopy-uasyncio.core-2.3.2.tar.gz
>>>
```

## נספח ד' - מיפוי הבדיקה בקר ESP32



\*קיימים בשוק מוצרי ESP32 שבהם הבדיקה הרכיב ממופים באופן שונה.

מיפוי הבדיקה הקלט/פלט של בקר ESP32 :

Notes	Output	Input	GPIO	Notes	Output	Input	GPIO
	OK	OK	16	outputs PWM	OK	pulled up	0
	OK	OK	17	debug output at boot	OK	TX pin	1
	OK	OK	18	on-board LED	OK	OK	2
	OK	OK	19	HIGH at boot	RX pin	OK	3
	OK	OK	21		OK	OK	4
	OK	OK	22	outputs PWM	OK	OK	5
	OK	OK	23	SPI flash	x	x	6
	OK	OK	25	SPI flash	x	x	7
	OK	OK	26	SPI flash	x	x	8
	OK	OK	27	SPI flash	x	x	9
	OK	OK	32	SPI flash	x	x	10
	OK	OK	33	SPI flash	x	x	11
input only		OK	34	boot fail if pulled high	OK	OK	12
input only		OK	35		OK	OK	13
input only		OK	36	outputs PWM	OK	OK	14
input only		OK	39	On board LED -PWM	OK	OK	15

## נספח ה' - עדכון קושחה לבקר ESP32

מקור:

<https://www.youtube.com/watch?v=4kiNU-dNcf0>

נתקין תוכנה המאפשרת לעדכן קושחה מהכתובת הבאה:

<https://www.espressif.com/en/support/download/other-tools>

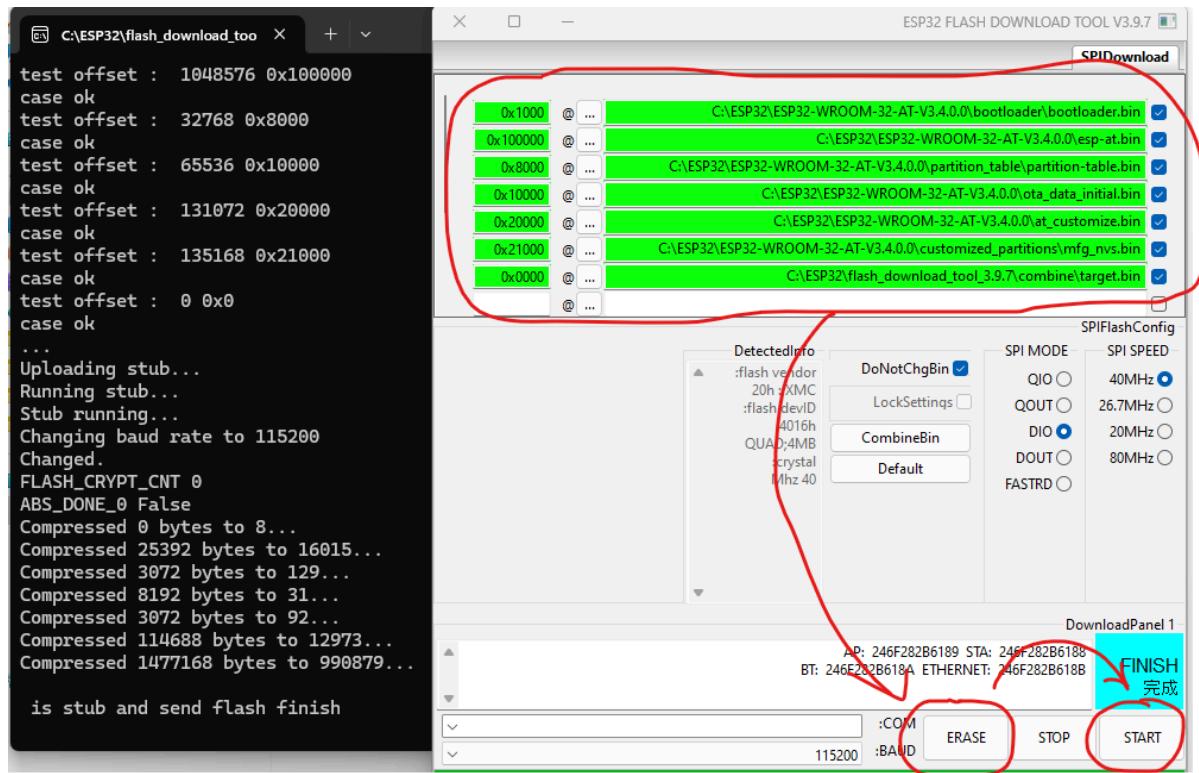
The screenshot shows the Espressif Support website's 'Download' section. In the 'Tools' category, there is a single result for 'Flash Download Tools'. The result table includes columns for Title, Platform, Version, Release Date, and Download. The 'Title' column shows '+ Flash Download Tools', the 'Platform' column shows 'Windows PC', the 'Version' column shows 'V3.9.6', and the 'Release Date' column shows '2024.06.07'. A red circle highlights the 'Title' cell, and another red circle highlights the 'Download' button.

נוריד את הגרסה העדכנית ביותר לקושחה שבאתר הבא:

[https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT\\_Binary\\_Lists/esp\\_at\\_binaries.html](https://docs.espressif.com/projects/esp-at/en/latest/esp32/AT_Binary_Lists/esp_at_binaries.html)

The screenshot shows the Espressif Documentation website's 'Released Firmware' page for the ESP32-WROOM-32 Series. The page lists several firmware files, with the first one circled in red: 'v3.4.0.0 ESP32-WROOM-32-AT-V3.4.0.0.zip (Recommended)'. The page also includes a note about using the latest version of firmware and a 'Note' section with instructions for modifying firmware configurations.

נפתח את התוכנה ונקבע בה את הפרמטרים הבאים:



נחבר את ה-ESP32 למחשב נלחץ על ERASE לאחר קבלת הודעה FINISH נלחץ על START כדי לזרוב את התוכנה העדכנית.

\*קיימים בשוק מוצרי ESP32 שבהם הדק' הרכיב ממופים באופן שונה.

## תנאי השימוש

תנאי השימוש במסמך זה האם לפי הסטנדרט הבא:

### You are free:

- to Share** – to copy, distribute and transmit the material
- to Remix** – to adapt the material

### Under the following conditions:

**Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

**NonCommercial** — You may not use the material for commercial purposes.

**ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.