מיקרו פייתון – MicroPython

1. מבוא

מיקרופייתון – MicroPython הוא יישום זעיר של שפת התוכנה Python 3 הכוללת חלק מתוך הספרייה הסטנדרטית של פייתון . על חלק זה בוצעה אופטימיזציה שמאפשרת להריץ תוכניות על מיקרו בקרים מסוימים ומערכות משובצות מחשב ונותנת בקרה על סוגים שונים של פרויקטים אלקטרוניים.

בדומה לפייתון , גם מיקרופייתון היא שפה שעובדת עם מפרש - Interpreter ולא עם קומפיילר כמו שפת C ושפות אחרות. קומפיילר עובר על התוכנית שרשמנו , הופך אותה לאסמבלי ולאחר מכן לשפת מכונה (אפסים ואחדים). לקובץ המכונה יש סיומת exe אם מריצים את התוכנית במחשב או קובץ עם סיומת hex אם יש לטעון אותה לזיכרון התוכנית של מיקרו בקר. במידה ויש טעות בתוכנית הקומפיילר לא יוצר קובץ אסמבלי וכמובן לא קובץ מכונה ולא ניתן להריץ את התוכנית. במפרש Interpreter קיימת תוכנית נוספת המריצה את התוכנית שרשמנו. היא לוקחת פקודה אחר פקודה. היא מתרגמת את הפקודה ומריצה אותה. כאשר יש פקודה לא חוקית התוכנית נעצרת. עבודה עם מפרש איטית בהרבה מעבודה עם קומפיילר ודורשת גם זיכרון גדול יותר. עם זאת היא קומפקטית מספיק כדי להתאים ולעבוד עם רק 256k בתים של נפח קוד ו 16k בתים של זיכרון RAM. כמובן שעם מיקרו בקרים עם נפח זיכרון גדול יותר נוכל להריץ פרויקטים גדולים יותר. לוחות המעבדים של ראספברי פיי הם בנפחי זיכרון גדולים בהרבה וגם ראספברי פי פיקו שהוא מיקרו בקר שיצא לשוק בינואר 2021 הוא בעל נפח זיכרון מתאים לעבודה עם מיקרו פייתון.

ל MicroPython ספריות רבות שמאפשרות עבודה נוחה עם חומרה ומודולים/כרטיסי אלקטרוניקה רבים.

המיקרו פייתון תומך בכרטיסים/לוחות מיקרו בקרים רבים. רשימה של מיקרו בקרים נמצאת בקישור : MicroPython - Python for microcontrollers . נזכיר חלק מהם כאן :

- Pyboard
- Raspberry pi pico
- WiPy
- Teensy
- Micro : bit
- Arduino zero
- ESP32
- ESP8266
- Adafruit

המחלקות והמתודות שבהמשך מתאימות לכרטיס Pyboard וייתכן שלא יתאימו לכרטיסים אחרים.

2. מודולים, מחלקות ומתודות במיקרו פייתון

בפרק זה נתאר מודולים (מחלקות פונקציות ואובייקטים) הנמצאים במיקרופייתון. **רשמנו רק חלק מהמחלקות והפונקציות ולא את כולן. הן מתאימות ל Pyboard . חלקן מתאים גם לעבודה עם Raspberry Pi Pico**

קיימות מספר קטגוריות של מודולים והן :

- מודולים המיישמים קבוצת משנה של פונקציונליות סטנדרטית של פייתון ואינם מיועדים לשינוי על ידי המשתמש .
- מודולים המיישמים קבוצת משנה של פונקציונליות סטנדרטית של פייתון עם אפשרות להרחבה על ידי המשתמש עם קוד פייתון.
- מודולים המיישמים הרחבות של מיקרופייתון לספריות הסטנדרטיות של פייתון.
- מודולים ספציפיים ליציאת מיקרופייתון מסוימת ולכן אינם ניידים ליציאה אחרת.

בחלק מהפורטים ניתן לדעת את השימוש בעזרת ספריות זמינות על ידי הכנסת המשפט הבא ב REPL :

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

מעבר לספריות המובנות שנתאר , ניתן למצוא מודולים רבים נוספים מהספרייה הסטנדרטית של פייתון , כמו גם הרחבות נוספות של מיקרופייתון ב microPython-lib .

3. ספריות סטנדרטיות של פייתון וספריות של מיקרופייתון

את הספריות הסטנדרטיות של פייתון שהותאמו למיקרופייתון נתאר מיד. הן נותנות את פונקציונליות הליבה של המודול והן מיועדות להיות התחליף לספרייה הסטנדרטית של פייתון.

מודולים מסוימים שבהמשך משתמשים בשם פייתון סטנדרטי, אך עם קידומת "u", למשל ujson במקום json. זה מראה שהמודול הוא ספריה של מיקרו פייתון.

אם נרשום import json אז תחילה מחפשים את הקובץ json.py (או את הספרייה json) . אם הוא לא נמצא אז חוזרים לטעינת ujson .

3.1 פונקציות מובנות ותעופות – Built in functions and exeptions

- **cmath** – פונקציות מתמטיות למספרים קומפלקסים (מספרים מרוכבים complex).
- **gc** – שליטה על אוסף האשפה (**g**arbage **c**ollector).
- **math** – פונקציות מתמטיות.
- **uarray** – מערכים (arrays) של נתונים מספריים.
- **uasyncio** – תזמון קלט/פלט אסינכרוני (asynchronous i/o schedular) .
- **ubinascii** – המרות בין בינארי לאסקי (binary ascii).
- **ucollections** – טיפוסים של אוספים (קולקציות) וקונטיינרים.
- **uerrno** - קודים של שגיאת מערכת (error).
- **uhashlib** – אלגוריתמים של hashing (גיבוב).

- **uheapq** - אלגוריתם של תור עדיפות (heap queue).
- **uio** – זרימה של קלט/פלט.
- **ujson** – הצפנה ופענוח JSON .
- **uos** – שרותי "מערכת הפעלה" בסיסיים (**o**prrating **s**ystem).
- **ure** – ביטויים רגילים פשוטים (**re**gular) .
- **uselect** – המתנה לארועים בסט של זרימות.
- **usocket** – מודול סוקט (socket).
- **ussl** – מודול SSL/TLS (Secure Sockets Layer/Transport Layer Security) אבטחה באינטרנט.
- **ustruct** – טיפוסי נתונים מצופפים ומרווחים.
- **usys** - פונקציות מערכת ספציפיות.
- **utime** – פונקציות המתייחסות לזמן .
- **uzlib** – פתיחה של ספריות מכווצות של zlib (zlib היא ספרית נתונים עם קבצים מכווצים).
- **_thread** – תמיכה בהרצה של מספר תוכניות ביחד.

3.2 ספריות ספציפיות של MicroPython

פונקציונליות ספציפית ליישום MicroPython זמינה בספריות הבאות :

- **btree** – בסיס נתונים פשוט של BTree (Binary Tree- BTree - מבנה נתונים בינארי בצורת עץ) .
- **framebuf** – טיפול בחוצץ מסגרת (חלק מזיכרון ה־ RAM במחשב, המוקצה לאחסון של מידע גרפי על תמונה).
- **machine** – פונקציות הקשורות לחומרה.
- **micropython** - גישה ושליטה בנתונים הפנימיים של MicroPython .
- **network** – תצורת הרשת.
- **ubluetooth** – בלוטות רמה נמוכה .
- **ucryptolib** – צופן הצפנה - cryptographic ciphers .
- **uctypes** - גישה לנתונים בינאריים באופן מובנה.

4. ספריות ספציפיות לפורט

במקרים מסוימים לספריות שנראה בהמשך עבור פורט/לוח יש פונקציות או מחלקות דומות לאלו שבספריית **machine** .
במקום כזה הערך בספריה הספציפית של הפורט יהיה עם חומרה ייחודית ללוח המתאים.

כדי לכתוב פונקציות ומחלקות נשתמש במודול machine .

ניתן לקבל את המחלקות והפונקציות שבמודול machine על ידי רישום help (machine) בשורת הפקודה (ב REPL):

```
>>> help (machine)
object <module 'umachine'> is of type module
```

```
__name__ -- umachine
unique_id -- <function>
soft_reset -- <function>
reset -- <function>
reset_cause -- <function>
bootloader -- <function>
freq -- <function>
idle -- <function>
lightsleep -- <function>
deepsleep -- <function>
disable_irq -- <function>
enable_irq -- <function>
time_pulse_us -- <function>
mem8 -- <8-bit memory>
mem16 -- <16-bit memory>
mem32 -- <32-bit memory>
ADC -- <class 'ADC'>
I2C -- <class 'I2C'>
SoftI2C -- <class 'SoftI2C'>
Pin -- <class 'Pin'>
PWM -- <class 'PWM'>
RTC -- <class 'RTC'>
Signal -- <class 'Signal'>
SPI -- <class 'SPI'>
SoftSPI -- <class 'SoftSPI'>
Timer -- <class 'Timer'>
UART -- <class 'UART'>
WDT -- <class 'WDT'>
PWRON_RESET -- 1
WDT_RESET -- 3
```

כדי לגשת לחומרה של כרטיס מסוים יש להשתמש בספרייה המתאימה. לדוגמה : יש להשתמש ב **pyb** אם משתמשים בלוח של python (Pyboard) .

5. ספריות ספציפיות לכרטיס pyboard

הספריות הבאות ספציפיות ל pyboard :

- **pyb** – פונקציות הקשורות ללוח
  - פונקציות הקשורות לזמן.
  - פונקציות הקשורות ל reset .
  - פונקציות הקשורות לפסיקה.
  - פונקציות השייכות לתצרוכת הספק.
  - פונקציות שונות.
  - מחלקות.
- **lcd160cr** - בקרה על תצוגת LCD160CR
  - המחלקה **class LCDR160CR**
  - בנאים – קונסטרקטורים – **Constructors**
  - מתודות סטטיות – **Static methods**
  - חברים במופע – **Instance members**
  - פקודות התקנה - **setup**
  - מתודות גישה לפיקסל – **Pixel access methods** .
  - ציור טקסט – **Drawing text** .
  - ציור צורות פרימיטיביות – **drawing primitive shapes** .
  - מתודות נגיעה במסך – **Touch screen methods** .
  - פקודות מתקדמות – **Advanced commands** .
  - קבועים - **constants** .

6. ספריות ספציפיות ל WiPy

הספריות והמחלקות הבאות ספציפיות ל WiPy :

- **Wipi specific features** – **Wipy** (תכונות ספציפיות ל WiPy) .
  - **Functions** – פונקציות .
- **Class ADCWiPy – analog to digital conversion** – המרה מאנאלוגי לדיגיטאלי.
  - Consructors – בנאים .
  - **Methods** מתודות (פונקציות)
- **class ADCChannel – read analog values from internal or external sources**

- **class TimerWiPy – control hardware timers** מחלקה השולטת על הטיימרים.
  - **Constructors** בנאים .
  - **Methods** מתודות.
- **class TimerChannel – setup a channel for a timer** מחלקה המשייכת כניסה מסוימת לטיימר.
  - **Methods** מתודות.
  - **Constants** קבועים.

7. ספריות ספציפיות ל ESP8266 ו ESP32

הספריות הבאות ספציפיות ל ESP8266 ו ESP32 של חברת Espressif Systems :

- **esp – functions related to the ESP8266 ans ESP32** פונקציות השייכות ל 2 המיקרובקרים הרשומים.
  - **functions** פונקציות.
- **esp32 – functionality specific to ESP32** פונקציונליות ספציפית ל ESP32 .
  - **functions** פונקציות.
  - **Flash partitions** חלוקת זיכרון ה FLASH לדפים (מחיצות) .
  - **RMT** אפשור שליחה וקבלה של פולסים (0 עד 12.5 ננו שניות).
  - **Ultra-Low-Power co-processor** מעבד משותף בהספק אולטרה נמוך .
  - **Constants** – קבועים .
  - **Non-Volatile Storage** - אחסון בזיכרון לא נדיף.

8. ספריות ספציפיות ל RP2040

הספריות הבאות ספציפיות ל RP2040 שהוא המיקרו בקר בראספברי פיי פיקו :

- **rp2 – functionality specific to RP2040** פונקציונליות ספציפית ל RP2040 .
- **PIO related functions** פונקציות המתייחסות לקלט פלט מתוכנת ומאפשרות ליצור ממשקי חומרה נוספים.
- **Classes** מחלקות.

9. המודול machine

המודול **machine** מכיל פונקציות ספציפיות הקשורות לחומרה של לוח/כרטיס מסוים. רוב הפונקציות במודול מאפשרות שליטה וגישה ישירה ובלתי מוגבלת על בלוקים של חומרה במערכות כמו CPU , טיימרים, פסים buses- וכו' . בשימוש לא נכון עלולים לקרות תקלות ובמקרה חמור יותר גם נזק לחומרה.

הערה על הערך המוחזר מפונקציות ומתודות של **machine** : לכל החזרה יש להתייחס כביצוע בהקשר של פסיקה. זה נכון עבור רכיבים פיזיים עם מזהי ID>=0 ורכיבים וירטואליים עם מזהה שלילי כמו -1 .

בהמשך הפרק נזכיר פונקציות ומתודות ולפעמים נקרא למתודה פונקציה ולהפך.

מה ההבדל בין פונקציה ומתודה ?

פונקציה היא קבוצת פקודות עצמאית הכוללת לוגיקה מסוימת וקוראים לה באופן עצמאי והיא מוגדרת מחוץ למחלקה.

מתודה גם היא קבוצה של פקודות עצמאית שקוראים לה בהתייחסות לאובייקט מסוים והיא מוגדרת בתוך מחלקה.

9.1 הצגת הפונקציות והמחלקות במודול machine, מהירות המעבד ואת ה id של הכרטיס

ניתן לרשום בשורת הפקודה (REPL) את המילים :

```
help (machine)
```

ואז נקבל את המחלקות והפונקציות של המודול machine . אם נרשום

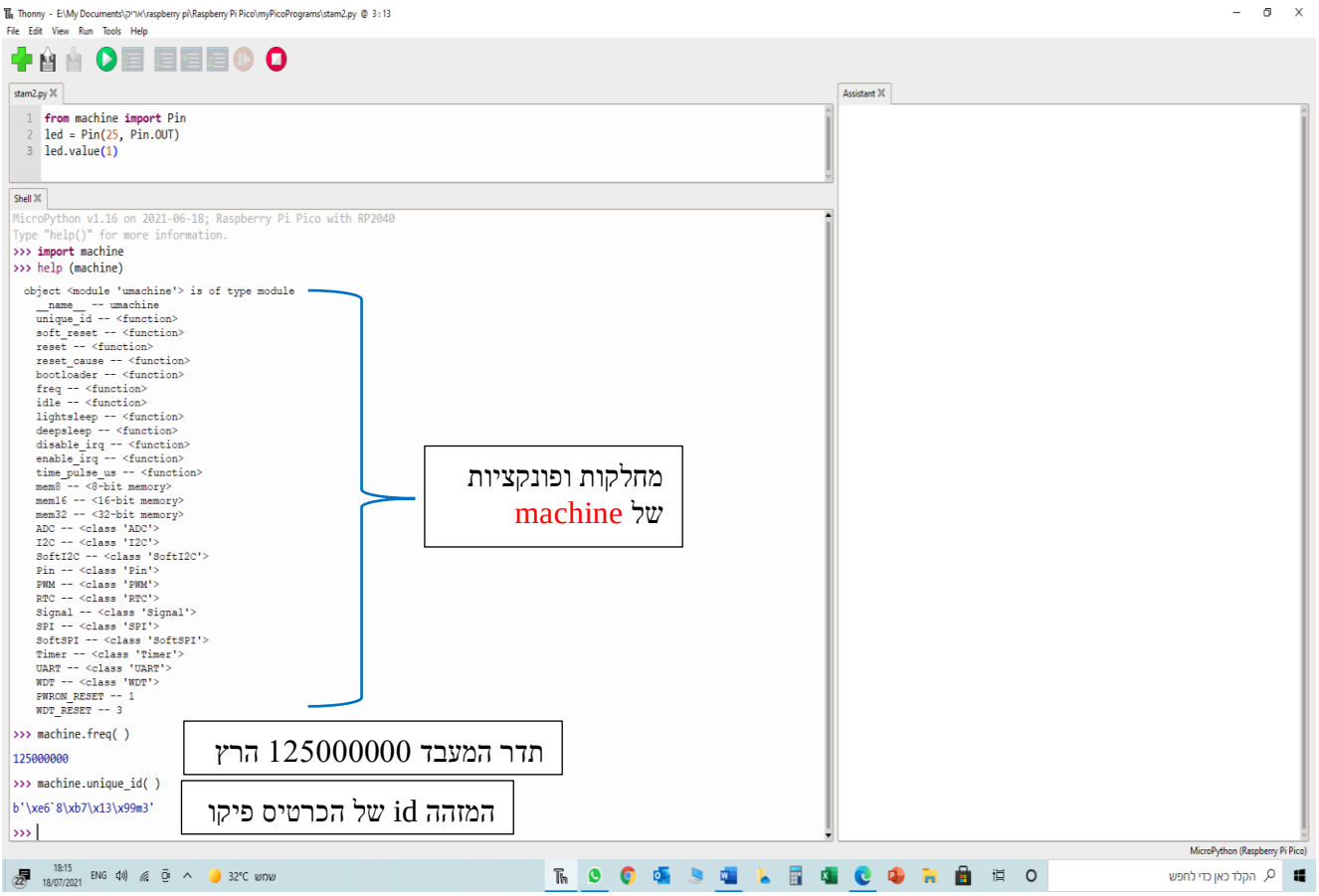
```
machine.freq( )
```

נקבל את תדר העבודה של המעבד. אם נרשום את הפקודה :

```
machine.unique_id( )
```

נקבל מחרוזת בתים למזהה הייחודי של הכרטיס.

הדבר נראה באיור הבא בעזרת כתיבת הפקודות ב Shell של תוכנת Thonny :



איור 1 : הצגת הפונקציות והמחלקות במודול machine, מהירות המעבד ואת ה id של הכרטיס

9.2 פונקציות המתייחסות ל Reset

machine.reset ( )

איפוס הרכיב בצורה הדומה ללחיצה על לחצן RESET חיצוני.

machine.soft_reset ( )

ביצוע איפוס "רך" של המפרש (interpreter) , ביטול כל האובייקטים של פייתון ואיפוס מחדש של הערימה (heap) של פייתון . מנסים לשמור על המתודה שבאמצעותה המשתמש מחובר אל ה REPL במיקרו פייתון (למשל טורי , USB , WiFi).

machine.reset_cause ( )

הערך המוחזר הוא סיבת ה reset . הערכים האפשריים הם (אין צורך לדעת את המספר עצמו ומספיק לרשום) :

machine.PWRON_RESET - איפוס בהפעלת החשמל או על ידי לחצן reset.

machine.HARD_RESET - איפוס על ידי לחצן reset.

machine.PWRON_RESET - איפוס על ידי טיימר של כלב השמירה – Watch Dog Timer.

machine.DEEPSLEEP_RESET - איפוס שנגרם על ידי Deepsleep.

9.3 פונקציות המתייחסות לפסיקות

machine.disable_irq( )

חסימת בקשות פסיקה. הפונקציה מחזירה ערך שיש להעביר אותו לפונקציה machine.enable_irq( ) כדי לשחזר את מצב הפסיקות המקורי לפני שחסמנו אותן.

machine.enable_irq(state)

אפשר מחדש בקשות פסיקה. הפרמטר state הוא הערך שהוחזר מהקריאה האחרונה לפונקציה machine.disable_irq( ) .

9.4 פונקציות המתייחסות להספק

machine.freq( )

מחזירה את תדר המעבד ביחידות של הרץ.

machine.idle( )

העברת המעבד למצב סרק. כדי להקטין את תצרוכת ההספק לפרקי זמן קצרים או ארוכים. המעבד עצמו מפסיק לעבוד אבל הרכיבים הפריפריאליים בתוך המיקרו בקר ממשיכים לעבוד. המעבד חוזר לעבוד ברגע שיש פסיקה כלשהי .

machine.sleep( )

העברת המעבד למצב שינה. במצב זה המעבד והמעגלים הפריפריאליים בתוך המיקרו בקר לא עובדים חוץ מה WLAN . הביצוע מתחדש מהנקודה שבה הפעלנו את הפונקציה. כדי לעורר מחדש את המיקרו יש לקבוע את התצורה מראש של מקורות ההתעוררות .

machine.deepsleep( )

עצירת המעבד והרכיבים ההיקפיים שבמיקרו בקר כולל ממשקי רשת (אם קיימים) . הביצוע מתחדש מהתוכנית הראשית כמו ב reset . ניתן לבדוק את סיבת ה reset כדי לדעת האם הגענו ממצב machine.DEEPSLEEP שנסביר בסעיפים הבאים. כדי שההתעוררות תתרחש בפועל יש לקבוע את התצורה של מקורות ההתעוררות כמו שינוי pin או פסק זמן של שעון RTC .

machine.wake_reason( )

מקבלים את סיבת ההתעוררות. ישנן מספר קבועים אפשריים והם :

machine.IDLE - חזרה ממצב idle .

machine.SLEEP – חזרה ממצב שינה sleep .

machine.DEEPSLEEP – חזרה ממצב שינה עמוקה .

9.5 פונקציות שונות

machine.rng( )

הפונקציה מחזירה מספר אקראי (רנדומאלי) של 24 ביטים שנוצר על ידי התוכנה.

machine.unique_id( )

הפונקציה מחזירה מחרוזת בתים עם מזהה ייחודי של הלוח/ או ה SoC (System On Chip – מערכת ברכיב – ג'וק שבו יש את המרכיבים של מחשב) . הוא משתנה מלוח/SOC אחד למשנהו . בחלק מבפורטים של מיקרו פייתון המזהה תואם לכתובת MAC ברשת.

machine.time_pulse_us(pin, pulse_level, timeout_us=1000000)

הפונקציה מודדת זמן בהדק מסוים ומחזירה את משך הזמן במיקרו שניות . השימוש בפונקציה יעיל למדידה של רוחב דופק של פולס. הארגומנט ***pin*** הוא מספר ההדק הרצוי. ***pulse_level*** הוא הרמה הלוגית של הפולס שרוצים למדוד (0 או 1) . **timeout_us=1000000** הוא מספר במיקרו שניות שאומר מהו זמן המדידה המקסימאלי שמאפשרים . אם הערך בהדק הוא הערך שקבענו ב ***pulse_level*** אז המדידה מתחילה מייד. אם ערך המתח בהדק שונה מזה שקבענו ב ***pulse_level*** הפונקציה ממתינה עד שהערך בהדק יהיה ברמה הלוגית שקבענו ומכאן מתחילה המדידה עד שהערך בהדק משנה את מצבו.

הפונקציה מחזירה את הערך -2 אם עבר פסק הזמן שרשמנו ב **timeout_us** שקבענו ולא הגיעה הרמה הרצויה .

הפונקציה מחזירה את הערך -1 אם עבר פסק הזמן של המדידה עצמה ברמת המתח שקבענו. הזמן של פסק הזמן זהה בשני המקרים ונתון במיקרו שניות בארגומנט ***timeout_us*** .

10. מחלקות במיקרו פייתון

המחלקות במיקרו פייתון הן המחלקות הבאות :

- class Pin – control I/O pins בקרה על הדקי קלט פלט
- class UART – duplex serial communication bus UART תקשורת טורית דו כיוונית
- class SPI – a Serial Peripheral Interface bus protocol (master side) SPI פרוטוקול
- class I2C – a two-wire serial protocol I2C פרוטוקול
- class RTC – real time clock שעון זמן אמת
- class Timer – control hardware timers בקרה על הטיימרים הטוריים
- class TimerChannel — setup a channel for a timer קביעת ערוץ לטיימר
- class WDT – watchdog timer טיימר כלב השמירה
- class ADC – analog to digital conversion המרה אנלוגית לדיגיטאלית
- class ADCChannel — read analog values from internal or external sources קריאת ערכים אנלוגיים ממקורות פנימיים או חיצוניים
- class SD – secure digital memory card SD כרטיס דיגיטאלי

נעבור על המחלקות וכיצד להשתמש בהן.

11. המחלקה Pin

ניתן להוסיף במיקרו פייתון מודולים של חומרה שיעזרו לנו לעבוד עם הפיקו . לדוגמה במודול **machine** יש אובייקט Pin שבעזרתו ניתן לשלוט על הדקי הרכיב , לקבוע האם ההדק יהיה קלט או פלט , לכתוב להדק רמות לוגיות של 0 או 1 או לקרא מההדק את הרמה הלוגית שיש בו. נעבוד עם הלד בכרטיס הפיקו המתחברת להדק GPIO 25 . אם נשים בהדק זה '1' הלד נדלקת ואם שמים בה '0' הלד כבויה. לפני שנרשום קוד כלשהו נסביר מספר מושגים במיקרו פייתון.

האובייקט Pin משמש לשלוט על הדקי הקלט/פלט (הנקראים GPIO). אובייקטים של Pin משויכים להדק פיזי של המיקרו בקר ובעזרתו ניתן להוציא מתחים של 0 ו 1 להדק או לקרא מתח כניסה הנמצא בהדק. למחלקה (class) הנקראת Pin יש מתודות לקבוע את מצב ההדק (קלט או פלט וכו') ואת המתח שנוציא להדק (אם קבענו אותו כפלט) או נקרא ממנו (אם קבענו אותו לפלט) . עבור ההדקים האנלוגיים של הפיקו יש מחלקה בשם ADC .

כדי לעבוד עם האובייקט Pin יש לייבא - import - את המחלקה machine . לשם כך נרשום :

```
from machine import Pin
```

11.1 בנאים – constructors

Class**machine.pin**(id, mode=-1 , pull=-1, *, value, drive, alt)

האובייקט Pin נבנה בעזרת הארגומנטים הבאים :

id - מזהה - המציין באופן חד משמעי הדק קלט/פלט מסוים. האפשרויות עבור המזהה הן מספר שלם - int , מחרוזת string או tuple (עם פורט ומספר הדק [port, pin]) . אם נוסף ארגומנט בקונסטרקטור הוא משמש לאתחול ההדק.

mode – מציין את אופן העבודה של ההדק שיכול להיות אחד מהבאים :

- **Pin.In** - ההדק מוגדר ככקלט .
- **Pin.out** - ההדק מוגדר כיציאה.
- Pin.OPEN_DRAIN - ההדק מוגדר כטרנזיסטור FET עם מרזב פתוח – ללא נגד במרזב - (בדומה ל Open Collector בטרנזיסטור רגיל) . מצב זה עובד כך : אם מוציאים להדק 0 אז בהדק יש רמת מתח נמוכה, קרובה ל 0 וולט כי ה FET בתוך הרכיב נמצא במצב ON . אם מוציאים 1 להדק הוא נמצא במצב של עכבה גבוהה (הנקרא גם מצב שלישי - High Z) .
- Pin.ALT - ההדק מוגדר לבצע פעולה חילופית שהיא ספציפית להדק. כל המתודות האחרות אינן עובדות על הדק שהוגדר בצורה כזו חוץ מ Pin.init() . ניסיון להפעיל את אחת המתודות על הדק זה תוביל לתוצאה לא מוגדרת.
- Pin.ALT_OPEN_DRAIN – דומה ל Pin.ALT אבל ההדק הוגדר כ Open Drain (מרזב פתוח) .

pull – מראה האם להדק מחובר נגד משיכה חלש - weak pull resistor - מעלה או מטה . נגד משיכה חלש הוא נגד של כ 100 קילו אוהם שמחובר בין ההדק לאדמה או בין ההדק למתח הספק. אם הנגד מחובר לאדמה זה נקרא – PULL DOWN . אם הנגד מחובר בין ההדק למתח ה V_{DD} של הג'וק זה נקרא pull up . באופן מעשי לא מחברים נגד אלא במקום הנגד יש טרנזיסטור FET העובד קרוב לאזור הקטעון שלו וההתנגדות RDS שלו היא בסדר גודל של מאה קילו אוהם ויותר. האפשרויות הן :

- **None** - אין נגד משיכה למעלה או למטה .
- **Pin.PULL_UP** – מראה שלהדק מחובר נגד משיכה למעלה בתוך הרכיב .
- **Pin.PULL_DOWN** – מראה שלהדק מחובר נגד משיכה למטה בתוך הרכיב.

value – תקף רק עבור אופני העבודה Pin.OUT ועבור Pin.OPEN_DRAIN ומציין מה הערך ההתחלתי שיהיה בהדק. אם לא רושמים ערך אז מצב ההדק יישאר ללא שינוי.

drive – מציין את הספק היציאה של ההדק והוא יכול להיות אחד מהמצבים הבאים :
Pin.LOW_POWER , PIN.MED_POWER ו **Pin.HIGH_POWER** . הכוונה לגודל זרם היציאה שתלוי ביכולת של ההדק המסוים.

alt – מציין שלהדק יכול להיות תפקיד נוסף והערכים שהוא יכול לקבל תלויים בפורט . ארגומנט זה תקף במצבים **Pin.ALT** או **Pin.ALT_OPEN_DRAIN** . מצב זה תומך כאשר להדק יש כמה תפקידים כמו למשל הדק שמשמש לתקשורת טורית I2C ורוצים להשתמש בו גם לתפקיד נוסף כמו הדלקת לד.

נרשום תוכנית לדוגמה ונסביר את הפקודות :

```
from machine import Pin   # Pin מייבאים את האובייקט machine מהמחלקה

p0 = Pin(0, Pin.OUT)      # create an output pin on pin #0  - 0 יצירה של יציאה בהדק

p0.value(0)               # set the value low –  0 מוציאים להדק

p0.value(1)               # set the value high –  1 מוציאים להדק

# create an input pin on pin #2, with a pull up resistor–קביעת הדק 2 ככניסה עם נגד משיכה למעלה
p2 = Pin(2, Pin.IN, Pin.PULL_UP)

print(p2.value())   # read and print the pin value –  2 קריאה והדפסה של הערך בהדק

p0.mode(p0.IN)      # reconfigure pin #0 in input mode – שינוי תפקיד הדק 0 ממצב פלט למצב קלט
```

11.2 מתודות Methods

Pin.init(mode=-1, pull=-1, *, value, drive, alt)

אתחול מחדש של הדק בעזרת הפרמטרים הנתונים (מוסברים בפסקה הקודמת על הבנאים). רק הארגומנטים שצוינו מוגדרים. שאר מצבי ההדק לא משתנים. אין החזרה של ערך.

Pin.value([x])

המתודה מאפשרת לקבוע ולקבל את הרמה הלוגית של הדק (תלוי אם שמים בארגומנט **x** 0 או 1). אם לא רשום הארגומנט אז המתודה מקבלת את הרמה הלוגית של ההדק (תלוי במתח הנמוך או הגבוה שיש בהדק). אם רשמנו במקום x את הערך 0 או 1 ההדק מקבל את הרמה שרשמנו. ההתנהגות של המתודה תלויה במצב העבודה של ההדק :

Pin.IN - המתודה מחזירה את הערך המעשי שנמצא כרגע בהדק.

Pin.OUT – ההתנהגות והערך החוזר מהמתודה לא מוגדרים.

Pin.OPEN_DRAIN - אם ההדק במצב '0' אז ההתנהגות והערך החוזר מהמתודה לא מוגדרים. אם ההדק במצב '1' המתודה מחזירה את המצב המעשי הנוכחי של ההדק.

אם רושמים ערך בארגומנט **x** המתודה מגדירה את רמת הלוגיקה שתהיה בהדק (0 או 1) . הארגומנט **x** יכול להיות כל דבר שניתן להמרה לבוליאני. אם תוצאת ההמרה היא True אז בהדק יהיה '1' , אחרת יהיה בהדק '0' . ההתנהגות של המתודה תלויה במצב העבודה שקבענו להדק :

Pin.IN – הערך שרשמנו נשמר בחוצץ היציאה של ההדק. מצב ההדק לא משתנה אלא נשאר במצב של עכבה גבוהה. הערך שיש בחוצץ יעבור להדק רק כאשר נשנה את מצה ההדק ל **Pin.OUT** או **Pin.OPEN_DRAIN** .

Pin.OUT - הערך שרשמנו מועבר אל ההדק מיידית.

Pin.OPEN_DRAIN – אם הערך הוא '0' ההדק נקבע למצב נמוך. אם הוא נקבע ל '1' אז ההדק במצב עכבה גבוהה.

אם שמים ערך בארגומנט x לא מוחזר ערך מהמתודה.

Pin._call_([x])

ניתן לקרא לאובייקטים של הדק. מתודת ה call נותנת קיצור דרך מהיר לקבוע ולקבל את הערך של הדק. המתודה הזו שוות ערך למתודה Pin.value([x]) .

Pin.on - שמים בהדק '1' .

Pin.off - שמים בהדק '0' .

Pin.mode([mode]) - מקבלת או קובעת את מצב העבודה של ההדק. מצב הארגומנט mode מוסבר בפסקה הקודמת על הבנאים.

Pin.pull([pull]) - מקבלת או קובעת את מצב נגד המשיכה למטה או למעלה או ללא נגד משיכה. מצב הארגומנט pull מוסבר בפסקה הקודמת על הבנאים.

Pin.drive([drive]) - מקבלת או נותנת את רמת הספק היציאה של ההדק. מצב הארגומנט drive מוסבר בפסקה הקודמת על הבנאים. זמינות ב WiPy (תוכנה עבור IOT – Internet Of Things – אינטרנט של דברים).

Pin.irq.(handler=None, trigger=(Pin.IRQ_FALLING | Pin.IRQ_RISING), *, priority=1, wake=None)

קביעה של צורת עבודה עם פסיקה. אם מצב העבודה של ההדק הוא **Pin.IN** (קלט) אז מקור הפסיקה הוא בהדק החיצוני של המיקרו. אם מצב ההדק **Pin.OUT** אז מקור הפסיקה הוא חוצץ היציאה של ההדק. אם מצב העבודה של ההדק הוא **Pin.OPEN_DRAIN** אז מקור הפסיקה הוא חוצץ היציאה עבור מצב של '0' וההדק החיצוני עבור מצב של '1' .

הארגומנטים הם :

handler - פונקציה אופציונלית שנקרא לה כאשר תהיה בקשת פסיקה.

trigger – קובעים את האירוע שיכול ליצור את הפסיקה.

האפשרויות הן :

Pin.IRQ_FALLING – הפסיקה תקרה כאשר יש ירידה (מעבר מ '1' ל '0') .

Pin.IRQ_RISING – הפסיקה תקרה כאשר יש עלייה (מעבר מ '0' ל '1') .

Pin.IRQ_LOW_LEVEL – הפסיקה תקרה כאשר יש רמה נמוכה של '0' .

Pin.IRQ_HIGH_LEVEL – הפסיקה תקרה כאשר יש רמה גבוהה של '1' .

ניתן לקבוע קבלת פסיקה על יותר מאפשרות אחת. לדוגמה : כדי לקבל פסיקה כשיש ירידה או עלייה נרשום בארגומנט :

```
trigger=(Pin.IRQ_FALLING | Pin.IRQ_RISING)
```

priority - מגדירים את רמת העדיפות של הפסיקה. ערך גבוה מראה עדיפות גבוהה . הערכים שניתן לרשום ספציפיים להדק.

wake - בוחרים את מצב צריכת ההספק שבו נקבל פסיקה. המצבים האפשריים הם : **machine.IDLE** - מצב סרק , **machine.SLEEP** - מצב שינה, **machine.DEEPSLEEP** - מצב שינה עמוקה.

12. המודול utime

מודול זה מיישם קבוצת משנה של המודול CPython . בתיעוד המקורי של CPython זה נקרא **time** .
במודול **utime** פונקציות לקבלת הזמן והתאריך הנוכחיים , למדידת מרווחי זמן ולביצוע השהיות.
תקופת הזמן מתחילת מדידת הזמן Epoch : בפורטים עם Unix מתבצעת מדידת זמן מתאריך 01-01-1970 שעה 00:00:00 . בפורטים אחרים של מיקרו בקרים הזמן נמדד מתאריך 01-01-2000 שעה 00:00:00 .
שמירה/תחזוקה של התאריך/זמן בלוח השנה בפועל: פעולה זו דורשת RTC (שעון זמן אמת). במערכות עם מערכת הפעלה בסיסית (כולל RTOS מסוימים) ה RTC יכול להיות משמעותי. קביעה ושמירה על לוח השנה המעשי הוא באחריות מערכת ההפעלה ה OS/RTOS וזה מתבצע מחוץ למיקרו פייתון. משתמשים ב API של מערכת ההפעלה כדי לבקש את התאריך והזמן. במיקרו בקרים תלוי באובייקט **machine.RTC( )** . כדי לכוון את הזמן והתאריך משתמשים בפונקציה **machine.RTC( ).datetime(tuple)** ומתחזקים אותם באמצעים הבאים :

- על ידי סוללת גיבוי (שייתכן שיש בכרטיס שבו נמצא המיקרו בקר).
- שימוש בפרוטוקול זמן של הרשת (מחייב התקנה של הפורט/המשתמש .
- קביעה ידנית של המשתמש בכל הפעלת חשמל (ישנם לוחות שקובעים את זמן ה RTC עבור hard reset ויש כאלה שנדרש כיוון חוזר שלהם).

אם הזמן/תאריך לא נשמרים על ידי מיקרו פייתון אז ייתכן שהפונקציות הבאות שמתייחסות לזמן מוחלט יתנהגו בצורה לא צפויה.

12.1 הפונקציות של utime

utime.gmtime([secs])

utime.localtime([secs])

המרה של הזמן **secs** שמבוטא בשניות מאז תקופת הזמן Epoch (מוסבר למעלה) ל 8 בתים כ tuple המכיל :

year, month, mday, hour, minute, second, weekday, yearday

שנה חודש היום בחודש שעה דקות שניות היום בשבוע היום בשנה

אם לא רושמים את הארגומנט **secs** אז משתמשים בזמן הנוכחי מה RTC .

הפונקציה **gmtime( )** מחזירה tuple של התאריך–זמן ב UTC שהוא זמן אוניברסלי מתואם או באנגלית Coordinated Universal Time הוא תקן הזמן לפיו נמדדים ומתואמים השעונים במדינות השונות. אזורי זמן מסביב לעולם מתוארים ביחס לתקן זה. לדוגמה: אזור הזמן בישראל הוא UTC+2 בחורף, ו UTC+3-בשעון הקיץ.

לדוגמה : (2021, 7, 19, 9, 59, 49, 6, 200)

הפונקציה **localtime( )** מחזירה tuple של התאריך–זמן בזמן מקומי.

לדוגמה : (2021, 7, 19, 10, 2, 6, 6, 200)

הפורמט של הערכים ב tuple הוא :

- year - השנה - כולל את המאה . לדוגמה 2021 .
- month - חודש – מ 1 עד 12 .
- mday – היום בחודש – מ 1 עד 31 .
- hour – שעה – מ 0 עד 23 .
- minute – מ 0 עד 59 .
- second – מ 0 עד 59 .
- weekday – היום בשבוע – מ 0 עד 6 (מיום שני עד ראשון).

- yearday – היום בשנה – מ 1 עד 366 .

utime.mktime( )

הפונקציה מקבלת ארגומנט של 8-tuple שמבטאת את הזמן המקומי. היא מחזירה מספר שלם שמראה כמה שניות עברו מאז 1 בינואר 2000 .

utime.sleep(seconds)

הפונקציה מבצעת שינה (השהייה) בשניות בהתאם למספר ה seconds שרשמנו כארגומנט. ישנם לוחות שיקבלו גם מספר float עבור מספר עם חלק של שבר של שניות. ישנם לוחות שלא יקבלו מספר float . במקרה כזה יש להשתמש בפונקציות sleep_ms( ) ו sleep_us( ) שנסביר עכשיו.

utime.sleep_ms(ms)

השהייה במילי שניות בהתאם לזמן בארגומנט **ms** . צריך להיות מספר חיובי או 0 .

utime.sleep_us(us)

השהייה במיקרו שניות בהתאם לזמן בארגומנט **us** . צריך להיות מספר חיובי או 0 .

כאשר רוצים לבצע השהייה אבל בזמן ההשהיה להריץ פקודות נעזרים בפונקציות הבאות :

utime.ticks_ms( )

הפונקציה מחזירה ערך ממונה הסופר מילי שניות . הערך במונה הולך וגדל והוא התחיל עם נקודת התייחסות שרירותית (בדרך כלל כשהתחלנו להריץ את התוכנית). המונה מגיע למקסימום שלו ואז מתאפס וממשיך לספור מצב זה נקרא **wrap around** – עוטף סביב . לערך המקסימלי נקרא בשם TICKS_MAX . נקרא לזמן המקסימאלי של המונה TICKS_PERIOD - מחזור טיקים (כמו רעש תקתוק שעון אנאלוגי) והוא שווה ל :

TICKS_PERIOD = TICKS_MAX + 1.

הוספנו 1 כי המונה מתחיל לספור מ 0 . התוצאה כמובן במילי שניות. ערך זה הוא חזקה של 2 והוא שונה מכרטיס מיקרו בקר אחד לאחר . כמובן שהערך לא יכול להיות שלילי . אותו זמן מקסימלי משמש גם לפונקציות ticks_ms( ) , ticks_us( ) ו ticks_cpu( ) שנסביר בהמשך. הפונקציות האלו יחזירו ערך בטווח של 0 עד TICKS_MAX . הערך המוחזר מהפונקציה לא משמעותי עבורנו כי לא מעניין כמה זמן עבר מרגע הפעלת המערכת. אנחנו נשתמש בפונקציות ticks_diff( ) ו ticks_add( ) (שנסביר בהמשך) למדוד הפרשי זמן או חיבור זמנים. השימוש בפונקציות אלו יכול

לתת לנו אפשרות לבצע השהיית זמן ובזמן הזה להריץ את התוכנית שנרצה . להבדיל מהפונקציות של utime.sleep שמבצעות השהייה שבה ממתינים שההשהיה תסתיים ולא ניתן להריץ תוכנית בזמן ההשהיה.

הערה : ביצוע פעולות מתמטיות כמו חיבור או חיסור או פעולות כמו (<, <=, >, >=) על הערך החוזר מהפונקציה יכולים לגרום לתוצאות לא חוקיות. ביצוע פעולות מתמטיות ולאחר מכן להעביר את התוצאות כארגומנטים לפונקציה ticks_diff( ) או ticks_add( ) יובילו גם לתוצאות לא חוקיות.

utime.ticks_us( )

בדיוק כמו **utime.ticks_ms( )** אבל הערך החוזר הוא במיקרו שניות.

utime.ticks_cpu( )

דומה לפונקציות ticks_ms( ) ו ticks_us( ) , אך עם הרזולוציה הגבוהה ביותר האפשרית במערכת. בדרך כלל מדובר בתדר השעון של המעבד המרכזי, ולכן הפונקציה נקראת כך. אבל זה לא חייב להיות שעון CPU, יחידת התזמון המדויקת (רזולוציה) של פונקציה זו אינה **נמצאת במודול utime ,** אך ניתן לקרא את הערך בדפי הנתונים של הכרטיס שעובדים איתו. פונקציה זו מיועדת לבחינות של ביצועים עדינות מאוד או לולאות הדוקות מאוד בזמן אמת. הימנע משימוש בו בקוד נייד. זמינות: לא בכל כרטיס מיקרו יש את הפונקציה הזו.

utime.ticks_add(ticks, delta)

הערך שבין המספר ticks עם חיבור של delta. המספר יכול להיות חיובי או שלילי. נותנים ערך ב ticks והפונקציה מאפשרת לחשב את הטיקים בין הערך delta לפני או אחרי . ticks הוא פרמטר ישיר של הערך החוזר מהפונקציות ticks_ms( ) או ticks_us( ) או ticks_cpu( ) או מקריאה קודמת ל ticks_add( ) . delata הוא מספר שלם או ביטוי מספרי. הפונקציה שימושית לחישוב תאריכי יעד עבור אירועים/משימות שונות.

הערה : יש להשתמש בפונקציה ticks_diff( ) כדי לחשב תאריכי יעד).

דוגמה : נמצא מה ערך ה ticks לפני 100msec .

```
# Find out what ticks value there was 100ms ago
print(ticks_add(time.ticks_ms(), -100))

# Calculate deadline for operation and test for it    חשב תאריך יעד לפעולה ובדוק האם הגיע הזמן
deadline = ticks_add(time.ticks_ms(), 200)
while ticks_diff(deadline, time.ticks_ms()) > 0:
    do_a_little_of_something()

# Find out TICKS_MAX used by this port    בכרטיס TICKS_MAX מצא את
print(ticks_add(0, -1))
```

utime.ticks_diff(ticks1, ticks2)

מדידת הפרש הטיקים בין הערכים המוחזרים מהפונקציות ticks_ms( ) או ticks_us( ) או ticks_cpu( ) . הערך המתקבל יכול להיות חיובי או שלילי.

יש להשתמש בפונקציה הזו למדידת הפרש טיקים . כדי להימנע ממקרה של wrap around (המונה הגיע למקסימום והתחיל לספור מחדש) כדאי לבדוק הפרש טיקים שאין ביניהם הפרשי זמן ארוכים מידי.

הפונקציה שימושית בין השאר לביצוע polling על הדק למשך זמן רצוי . במקרה זה סדר הארועים ידוע ואנחנו עובדים עם מספרים חיוביים בלבד.

לדוגמה : נעשה polling על הדק כלשהו כדי לראות האם הרמה הלוגית של ההדק משתנה ל 1 . זמן הבדיקה לא יעלה על 500 מיקרו שניות:

```
# Wait for GPIO pin to be asserted, but at most 500us
start = time.ticks_us()#start העברת זמן המונה במילי שניות למשתנה
while pin.value() == 0:  # 0 לולאה שנמצאים בה כל עוד ההדק ב
if time.ticks_diff(time.ticks_us(), start) > 500:
 raise TimeoutError  # תעופה
```

הערה : לא להעביר ערכים של time( ) (שנסביר בהמשך) ל ticks_diff( ) .

utime.time( )

הפונקציה מחזירה את מספר השניות השלם מאז הפעלת המערכת (Epoch) בהנחה שה RTC הוגדר ומכוון. אם אין RTC עם גיבוי סוללה אז מקבלים ערך של השניות שעברו מרגע שהופעל הכרטיס או שהוא קיבל reset . הדיוק של הערך החוזר הוא 1 שנייה. אם צריך דיוק גבוה יותר כדאי להשתמש ב time_ns( ) .

utime.time_ns( )

דומה ל time( ) אבל מחזיר את הזמן בננו שניות.

13. המחלקה Timer (טיימר) – שליטה על טיימרים של חומרה

טיימרים עוסקים בתזמון של זמנים ואירועים. הטיימרים שונים בין כרטיסי/לוחות האלקטרוניקה השונים. המחלקה timer במיקרו פייתון מאפשרת לנו לבצע השהיות , ביצוע מדידות של זמנים ועוד. בגלל השוני בין המיקרו בקרים השונים שיש בכרטיסי האלקטרוניקה השונים ייתכן שהתנהגות בלוח אחד לא תהיה זהה בלוח אחר. **כאן מדובר בכרטיס Pyboard** .

13.1 בנאים – constructors

classmachine.Timer(id, …)

בניית אובייקט חדש של timerשל המזהה id . id של 1- בונה טיימר וירטואלי (אם הוא נתמך על ידי הלוח)

13.2 מתודות – methods

Timer.init(mode, *, width=16)

אתחול הטיימר. לדוגמה :

```
tim.init(Timer,PERIODIC)      # טיימר של 16 ביטים הפועל בצורה מחזורית
tim.init((Timer,ONE_SHOT, width=32))      # טיימר של 32 ביטים שסופר פעם אחת בלבד
```

הארגומנטים :

mode יכול להיות :

- **Timer.ONE_SHOT** - הטיימר רץ פעם אחת בלבד עד שתקופת הזמן שנקבעה לו מסתיימת.
- **Timer.PERIODIC** - הטיימר עובד בצורה מחזורית בתדירות שנקבעה לו.
- **Timer.PWM** - מוציאים אות PWM בהדק.
- **width** – צריך להיות 16 או 32 ביטים. לתדירויות נמוכות מ 5 הרץ (או זמנים גדולים מ 200 מילי שניות) יש להשתמש בטיימרים של 32 ביטים. אופן 32-bit קיים רק באופנים **ONE_SHOT** ו **PERIODIC** .

Timer.deinit( )

מפסיק את האתחול של המונה. עוצר את הטיימר ולא מאפשר את העבודה שלו.

Timer.channel(channel, **, freq, period, polarity=Timer.POSITIVE, duty_cycle=0)

אם העברנו רק את מזהה הערוץ מוחזר אובייקט ערוץ שאותחל קודם לכן (או **None** אם אין ערוץ קודם). אחרת ערוץ הטיימר מאותחל ומוחזר.

אופן העבודה הוא זה שהתצורה שלו נקבעה לאובייקט הטיימר ששימש ליצירת הערוץ.

- **Channel** - אם רוחב (כמות) הביטים של הטיימר היא 16 אז יכול להיות או **TIMER.A** או **TIMER.B** .

  אם מספר הביטים 32 אז חייב להיות **TIMER.A | TIMER.B** .
- **freq** - קובעים את התדר בהרצים .
- **period** - קובעים את הזמן במיקרו שניות.

הערה חשובה : נותנים רק אחד מהם , או freq או period ואף פעם לא את שניהם.

- **polarity** – זה שימושי רק במצב PWM ומגדיר את קוטביות ה duty cycle .
- **duty_cycle** –שימושי רק במצב PWM . זה האחוז של ה duty_cycle באחוזים מ 0.00 ועד 100.00 . היות ו WiPy לא תומך במספרים עם נקודה צפה (float) ה duty cycle צריך להינתן בתחום של 0 – 10000 . 10000 יציין 100.00 , 5050 יציין 50.50 וכך הלאה.

הערה : כשהערוץ באופן PWM אז באופן אוטומטי הוגדר לו מספר הדק ולכן אין צורך להגדיר לו הדק עם המחלקה של Pin .

ההדקים התומכים בעבודה עם PWM הם (בכרטיס ה Pyboard):

- GP24 בטיימר 0 ערוץ A .
- GP25 בטיימר 1 ערוץ A .
- GP9 בטיימר 2 ערוץ B .
- GP10 בטיימר 3 ערוץ A .
- GP11 בטיימר 3 ערוץ B .

Timerchannel.freq([value]**)**

מקבלים או קובעים את התדירות בהרצים של ערוץ הטיימר.

Timerchannel.period([value]**)**

קובעים או מקבלים את זמן המחזור של הטיימר במיקרו שניות.

Timerchannel.duty_cycle([value]**)**

- קובעים או מקבלים את ה duty cycle של אות ה PWM . הערך הוא באחוזים /(100.00 – 0.00). היות ו WiPy לא תומך במספרים עם נקודה צפה (float) ה duty cycle צריך להינתן בתחום של 0 – 10000 . 10000 יציין 100.00 , 5050 יציין 50.50 וכך הלאה.

13.3 קבועים

אופני העבודה עם הטיימר:

Timer.ONE_SHOT

Timer.PERIODIC

14. המחלקה TimerChannel – ערוץ הטיימר - קביעת הערוץ עבור הטיימר

הערוצים של טיימר משמשים ליצירה/לכידה (generate/capture) של אות באמצעות הטיימר.

האובייקטים של TimmerChannel נוצרים בעזרת המתודות של Timer.channel( ) .

14.1 מתודות

Timerchannel.irq**(*, trigger, priority=1, handler=None)**

ההתנהגות של הקריאה תלויה באופן העבודה של ערוץ הטיימר.

אם אופן העבודה **Timer.PERIODIC** אז הקריאה מתבצעת במחזוריות בתדר או זמן המחזור שקבענו.

אם אופן העבודה **Timer.ONE_SHOT** אז הקריאה מתבצעת כאשר הזמן שקבענו הסתיים.

אם אופן העבודה **Timer.PWM** אז הקריאה מבוצעת כאשר מגיעים לערך ה duty cycle .

הפרמטרים המקובלים הם :

priority – רמת הפסיקה. הערך הוא בין 1 ל 7 . ערך גבוה מייצג עדיפות גבוהה.

handler – פונקציה אופציונלית שתופעל כאשר הפסיקה מופעלת.

trigger - חייב להיות **Timer.TIMEOUT** כשאופן הפעולה הוא **Timer.PERIODIC** או **Timer.ONE_SHOT** או **Timer.PWM** כאשר ההפעלה חייבת להיות שווה ל **Timer.MATCH** .

ביבליוגרפיה :

1. Overview — MicroPython 1.16 documentation
2. MicroPython - Python for microcontrollers
3. MicroPython: An Intro to Programming Hardware in Python – Real Python
4. class Timer – control hardware timers — MicroPython 1.9.3 documentation