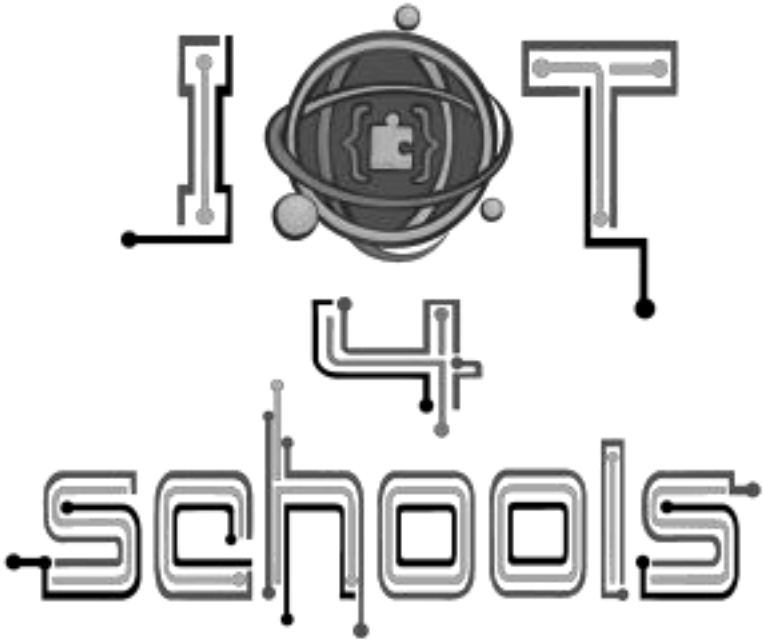
Τεχνικός οδηγός Raspberry Pi Pico και microPython



Συγγραφείς: [1]Angelika Tefelska, [1]Dariusz Tefelski
Συνεισφέροντες: [2]Chrissa Papasarantou, [2]Rene Alimisi

IoT4schools Κοινοπραξία:
[1]Warsaw University of Technology,
[2]EDUMOTIVA,

Τίτλος έργου: “Φέρνοντας το Διαδίκτυο των Πραγμάτων στην εκπαίδευση ως εργαλείο για την αντιμετώπιση των προκλήσεων του 21ου αιώνα”, κωδικός έργου: 2023-1-PL01-KA220-SCH-000154043, Erasmus+ KA220-SCH.

Η στήριξη της Ευρωπαϊκής Επιτροπής για την παραγωγή αυτής της δημοσίευσης δεν συνιστά έγκριση του περιεχομένου, το οποίο αντικατοπτρίζει αποκλειστικά τις απόψεις των συγγραφέων, και η Επιτροπή δεν φέρει ευθύνη για οποιαδήποτε χρήση μπορεί να γίνει των πληροφοριών που περιέχονται σε αυτήν.

Το πρότυπο LaTeX ελήφθη από: https://www.latextemplates.com/template/legrand-orange-book Οι εικόνες κεφαλαίου δημιουργήθηκαν με DALL-E, OpenAI.

Co-funded by the European Union

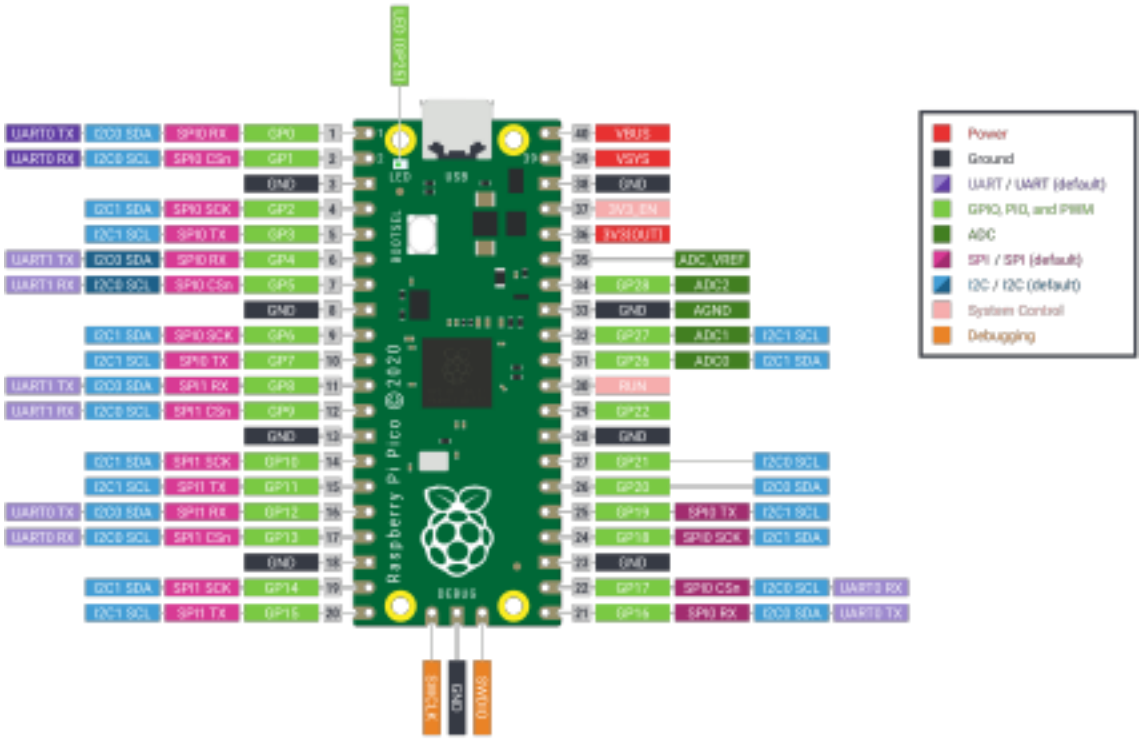
Περιεχόμενα

1. Introduction

Αυτός ο οδηγός είναι μια εισαγωγή στην πλακέτα Raspberry Pi Pico W, η οποία μπορεί να χρησιμοποιηθεί σε έργα Internet of Things (IoT). Η ανάπτυξη της ψηφιοποίησης δείχνει ότι χρησιμοποιούμε όλο και περισσότερο έξυπνες λύσεις που βελτιώνουν την ποιότητα της ζωής μας και μας επιτρέπουν να εξοικονομούμε πόρους όπως η ηλεκτρική ενέργεια ή το νερό. Η εξοικείωση με το θέμα του IoT είναι εξαιρετικά σημαντική στον σύγχρονο κόσμο. Μία από τις πλακέτες που μας επιτρέπει να εισέλθουμε στον κόσμο της τεχνολογίας IoT είναι το Raspberry Pi Pico W, το οποίο είναι συμπαγές, εύκολο στη χρήση και φθηνό. Ως εκ τούτου, αυτός ο οδηγός θα επικεντρωθεί στην παρουσίαση της πλακέτας Raspberry Pi Pico W και της γλώσσας microPython που χρησιμοποιείται για τον προγραμματισμό του. Ο οδηγός απευθύνεται σε μαθητές, φοιτητές, δασκάλους, εκπαιδευτικούς και χομπίστες που δεν έχουν ασχοληθεί στο παρελθόν με την πλακέτα Raspberry Pi Pico W. Σας ενθαρρύνουμε να μάθετε μέσω της εξάσκησης, δηλαδή εκτελώντας δείγματα έργων παράλληλα με έναν οδηγό, που θα σας επιτρέψει να αποκτήσετε, γρήγορα, ευχέρεια στη χρήση της πλακέτας. Απολαύστε τη μάθηση και διασκεδάστε!

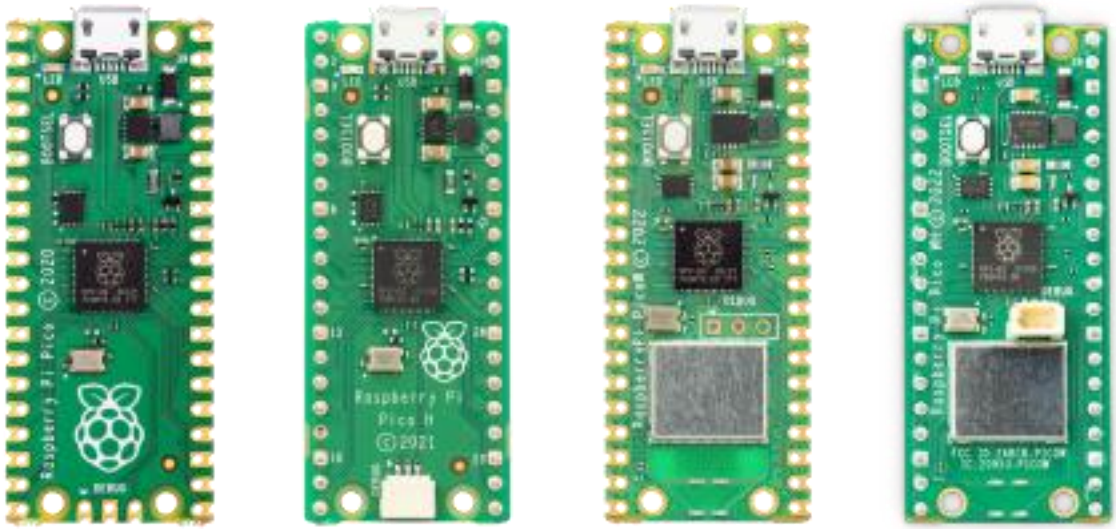
1.1 Raspberry Pi Pico board

Το 2021 κυκλοφόρησαν οι πλακέτες της σειράς Raspberry Pi Pico, οι οποίες ήταν ένα πρωτοποριακό προϊόν που προσφέρθηκε από το Ίδρυμα Raspberry Pi. Οι προηγούμενες εκδόσεις των πλακετών Raspberry Pi ήταν System-on-Chip (SoC), οι οποίες ήταν μικροσκοπικοί υπολογιστές. Από την άλλη πλευρά, η σειρά Raspberry Pi Pico είναι ένας εντελώς διαφορετικός τύπος πλακετών - βασίζονται σε μικροελεγκτές και είναι μια εξαιρετική εναλλακτική λύση για τις πλακέτες Arduino. Η καρδιά της πλακέτας Raspberry Pi Pico είναι το ιδιόκτητο σύστημα RP2040 - ένας μικροελεγκτής διπλού πυρήνα ARM Cortex M0+ που λειτουργεί στα 133 MHz, εξοπλισμένος με 264 KB SRAM και 2 MB μνήμης Flash. Ο στόχος ήταν να παρασχεθεί ένας αποδοτικός μικροελεγκτής με σημαντική υπολογιστική ισχύ που θα ανταποκρίνεται στις προσδοκίες των χομπίστων από όλο τον κόσμο. Εκτός από το εξαιρετικό σύστημα RP2040, αξίζει να σημειωθεί ότι η πλακέτα ήταν εξοπλισμένη με 26 ακίδες εισόδου/εξόδου γενικής χρήσης (GPIO), συμπεριλαμβανομένων 16 καναλιών PWM και διεπαφές επικοινωνίας όπως I2C, SPI, UART. Εάν δεν είστε εξοικειωμένοι με τον αναφερόμενο εξοπλισμό πλακέτας, δεν έχει σημασία, θα συζητήσουμε όλα τα στοιχεία αργότερα στον οδηγό. Η διάταξη των αναφερθέντων στοιχείων στον πίνακα Raspberry Pi Pico φαίνεται στο Σχήμα 1.1. 6 Κεφάλαιο 1. Εισαγωγή



Σχήμα 1.1: Η διάταξη ακροδεκτών του Raspberry Pi Pico. Πηγή εικόνας: https://www.raspberrypi.com.

Η σειρά Raspberry Pi Pico αποτελείται από 4 μικρές πλακέτες (όχι πολύ μεγαλύτερες από ένα pendrive) που φαίνεται στην Εικόνα 1.2.



Σχήμα 1.2: Οι 4 εκδόσεις πλακετών Raspberry Pi Pico. Πηγή εικόνας: https://www.raspberrypi.com.

Η πρώτη πλακέτα στα αριστερά είναι το *Raspberry Pi Pico*, η επόμενη πλακέτα είναι το *Raspberry Pi Pico H*, το οποίο διαφέρει μόνο στο ότι οι ακίδες είναι ήδη συγκολλημένες και η ειδική υποδοχή για σκοπούς εντοπισμού σφαλμάτων είναι παρούσα. Εάν παραγγείλετε την τυπική έκδοση του Raspberry Pi Pico, θα πρέπει να κολλήσετε μόνοι σας τις ακίδες. Αν δεν έχετε ποτέ συγκολλήσει, μην ανησυχείτε. Στο Διαδίκτυο θα βρείτε πολλά βίντεο με παραδείγματα για το πώς να κολλήσετε τους

ακροδέκτες, π. χ. https://www.youtube.com/watch?v=zbhOyCA_4lg. Αν, τελικά, θα προτιμούσατε να μην κολλήσετε ακίδες, μια καλύτερη επιλογή θα ήταν να επιλέξετε πλακέτες με το γράμμα *H*. Ωστόσο, να έχετε υπόψη σας, ότι δεν θα μπορείτε να κολλήσετε ακίδες για σκοπούς αποσφαλμάτωσης σε αυτή την πλακέτα και θα πρέπει να χρησιμοποιήσετε ειδικό σύνδεσμο (καλώδιο και σύνδεσμοι που διατίθενται με τη συσκευή Raspberry Pi Debug probe). Η τρίτη πλακέτα από αριστερά είναι το *Raspberry Pi Pico W*, το οποίο διαθέτει επιπλέον ενσωματωμένη μονάδα WIFI και Bluetooth, η οποία είναι εξαιρετικά σημαντική σε Internet of Things (έργα). Σε αυτόν τον οδηγό, θα επικεντρωθούμε κυρίως στην έκδοση Raspberry Pi Pico W, IoTη οποία είναι απαραίτητη για έργα . Η τέταρτη πλακέτα είναι το IoT*Raspberry Pi Pico WH*, το οποίο διαφέρει από το Raspberry Pi Pico W μόνο στους ακροδέκτες που είναι ήδη κολλημένοι (και στην παρουσία του συνδέσμου debug probe). Επομένως, αν δεν σκοπεύετε να κολλήσετε ακίδες, σας προτείνουμε την έκδοση Raspberry Pi Pico WH για έργα . IoT

Όλες οι πλακέτες Raspberry Pi Pico είναι εξοπλισμένες με μια υποδοχή microUSB η οποία χρησιμοποιείται για τον προγραμματισμό και την τροφοδοσία της πλακέτας. Η πλακέτα μπορεί να προγραμματιστεί σε C/C++ ή mi croPython. Σε αυτόν τον οδηγό, θα επικεντρωθούμε στη microPython ως την ευκολότερη επιλογή για αρχάριους. Για όσους επιθυμούν να μάθουν τον προγραμματισμό του Raspberry Pi Pico σε C/C++, συνιστούμε τον οδηγό για το θέμα αυτό που είναι διαθέσιμος εδώ: https://datasheets.raspberrypi.com/pico/getting-started with-pico.pdf.

Υπάρχουν επίσης διαθέσιμες στην αγορά πλακέτες τρίτων κατασκευαστών, οι οποίες συνήθως περιέχουν πρόσθετα στοιχεία και υποδοχές, π.χ. που επιτρέπουν την εύκολη σύνδεση της μπαταρίας, ή υποδοχή USB-C αντί για microUSB.

Ειδήσεις 1.1.

Πρόσφατα κυκλοφόρησε μια νέα έκδοση του Raspberry Pi Pico 2W, η οποία διαφέρει από τον προκάτοχό της κυρίως στα εξής: ο μικροελεγκτής (ο RP2040 που βασίζεται στον ARM Cortex-M0+ Dual-Core 133 MHz έχει αντικατασταθεί από τον RP2350 που βασίζεται στον ARM Cortex-M33 Dual-Core 150 MHz), η ποσότητα της μνήμης SRAM και Flash αυξήθηκε (2x περισσότερο) και ο αριθμός των καναλιών PWM αυξήθηκε από 16 σε 24. Ως μπόνους εμφανίστηκαν νέοι πυρήνες αρχιτεκτονικής RISC-V για πιο προχωρημένους χρήστες ως εναλλακτική επιλογή: είτε πυρήνες ARM είτε πυρήνες RISC-V, αν και οι μεγαλύτερες δυνατότητες παραμένουν στους πυρήνες ARM.

Ωστόσο, η νέα έκδοση του μικροελεγκτή RP2350 έχει ένα πρόβλημα με τις pull-down αντιστάσεις (βλ. κεφάλαιο για τα ψηφιακά σήματα). Παρά το προαναφερθέν πρόβλημα, το οποίο μπορεί να αμβλυνθεί, η νέα έκδοση είναι μια ενδιαφέρουσα επιλογή αν σκοπεύετε να δημιουργήσετε πιο εκτεταμένα προγράμματα και χρειάζεστε περισσότερη μνήμη. Αυτός ο οδηγός χρησιμοποιεί το Raspberry Pi Pico W, το οποίο είναι αρκετό για τα περισσότερα έργα IoT.

1.2 Εγκατάσταση

Πριν ξεκινήσουμε τον προγραμματισμό του Raspberry Pi Pico, πρέπει να εγκαταστήσετε:

- Thonny editor (https://projects.raspberrypi.org/en/projects/getting -started-with-the-pico/2)
- Raspberry Pi Pico firmware (https://projects.raspberrypi.org/en/projec ts/getting-started-with-the-pico/3).

2. Βασικά ηλεκτρονικά εξαρτήματα

Πριν ξεκινήσουμε την περιπέτειά μας με τον προγραμματισμό του Raspberry Pi Pico W, ας γνωρίσουμε τα βασικά ηλεκτρονικά εξαρτήματα που θα χρησιμοποιήσουμε:

- Αντίσταση - είναι ένα στοιχείο που χρησιμοποιείται για τη μείωση του ρεύματος που το διαρρέει. Διαχέει ενέργεια ως θερμική ενέργεια. Η αντίσταση είναι γραμμικό στοιχείο. Όσο μεγαλύτερη είναι η αντίσταση που χρησιμοποιούμε, τόσο λιγότερο ρεύμα θα διαρρέει το κύκλωμα, σύμφωνα με το νόμο του Ohm:

$$I = \frac{V}{R} \quad (2.1)$$

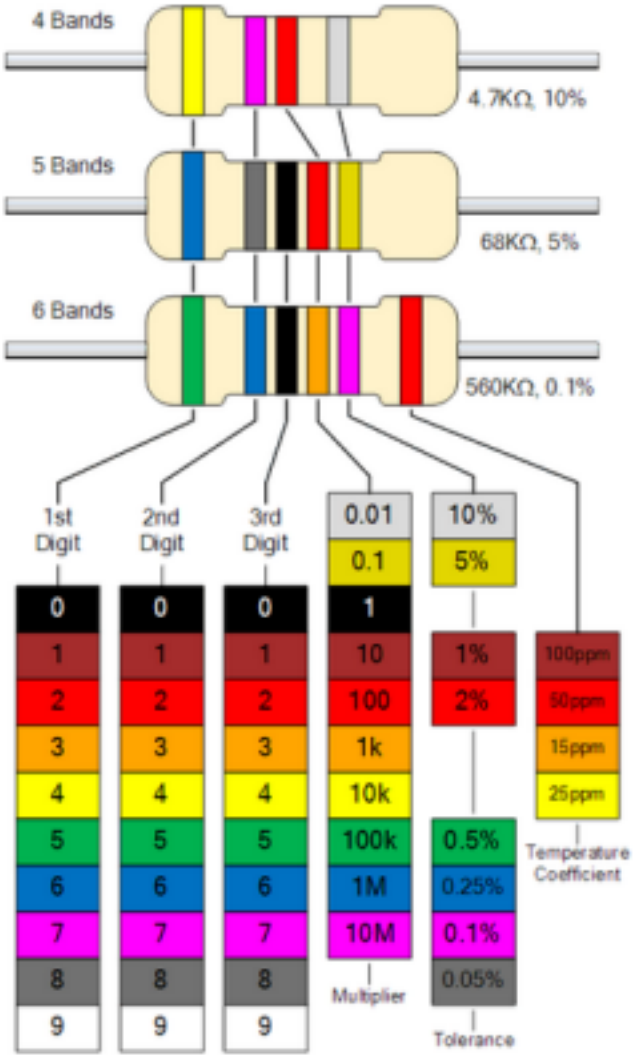
όπου: *I* - ένταση ρεύματος, *R* - αντίσταση.

Η τιμή της αντίστασης ενός αντιστάτη μπορεί να διαβαστεί από τον χρωματικό κώδικα που βρίσκεται πάνω στον αντιστάτη, όπως φαίνεται στο παράδειγμα του Σχ. 2.1. Για κάθε αντιστάτη, θα διαβάσουμε την τιμή της αντίστασης από αριστερά προς τα δεξιά. Για παράδειγμα, στο Σχ. 2.1, ο επάνω αντιστάτης έχει τα ακόλουθα χρώματα ζώνης: κίτρινο (τιμή 4), ροζ (τιμή 7), κόκκινο (x100) και ασημί (ανοχή=10%). Έτσι, η τιμή του είναι *47x100Ω* = 4700Ω = 4,*7kΩ* και η ανοχή του είναι 10%. Η ανοχή δείχνει πόσο μπορεί να διαφέρει η πραγματική αντίσταση από την ονομαστική τιμή (π.χ. για έναν με 10% αντιστάτη , η πραγματική του αντίσταση θα είναι μεταξύ 44*,7kΩ*,*23kΩ* (0,9 - *4,7kΩ*) και 5,17kΩ (1,1 - 4*,7kΩ*)).

- Δίοδος LED - είναι ένα στοιχείο ημιαγωγού που μετατρέπει το ρεύμα σε φως. Πιο συγκεκριμένα, τα ηλεκτρόνια που περνούν από ένα υψηλότερο ενεργειακό επίπεδο σε ένα χαμηλότερο σε έναν ημιαγωγό εκπέμπουν ένα φωτόνιο και ανάλογα με την ενέργεια του φωτονίου, λαμβάνουμε διαφορετικά χρώματα LED. Οι λυχνίες LED είναι πολωμένα στοιχεία, πράγμα που σημαίνει ότι το ρεύμα θα ρέει μέσα από αυτά μόνο προς μία κατεύθυνση. Στο σχήμα 2.2 φαίνεται ένα LED με καθοδικό και ανοδικό καλώδιο.

  Η τυπική δίοδος LED χρειάζεται τάση κοντά στα 1,7 V (V_{LED}) και ρεύμα μεταξύ 1 έως 15 *mA*. Η τάση στις ακίδες του Raspberry Pi Pico είναι 3,3 V ($V_{RP)}$, οπότε πρέπει να χρησιμοποιήσετε μια αντίσταση για να περιορίσετε τη ροή ρεύματος:

$$R = V(RP) - V(LED \qquad I = (107; 1600)\Omega \quad (2.2)$$

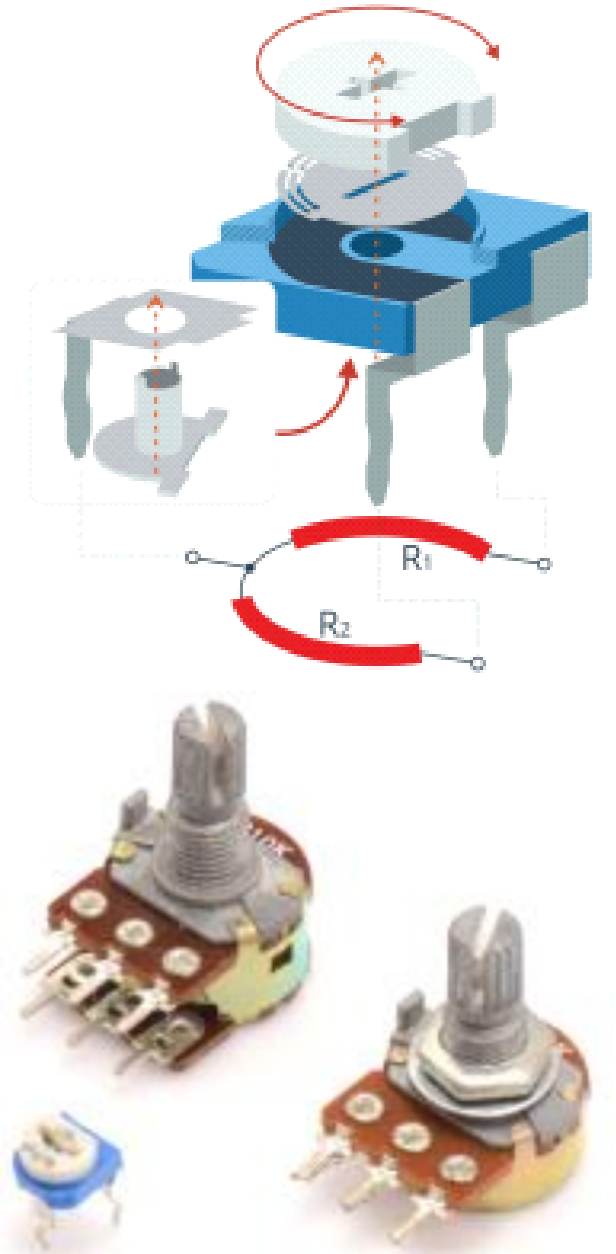


Σχήμα 2.1: Ο χρωματικός κώδικας των αντιστάσεων. Πηγή: https://electronicspost.com/resi stor-color-code/

Σχήμα 2.2: Ένα LED με σημειωμένες την άνοδο και την κάθοδο (αριστερή εικόνα) και ένα σύμβολο διόδου (δεξιά εικόνα). Πηγή: https://www.build-electronic-circuits.com/what-is-an-l ed/ .

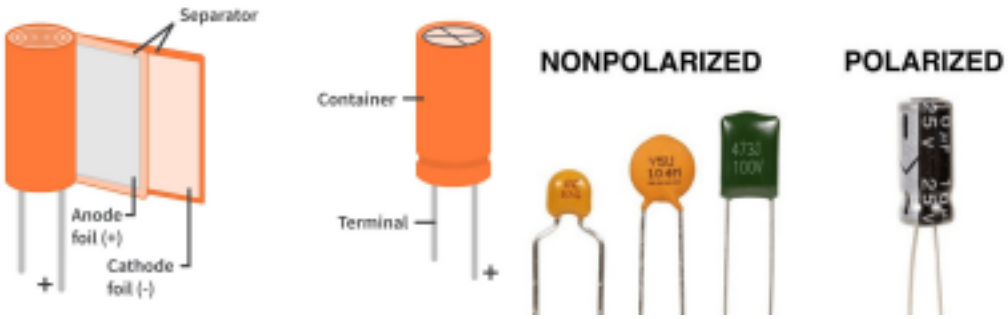
- Ποτενσιόμετρο - είναι ένας μεταβλητός αντιστάτης που σας επιτρέπει να ρυθμίσετε την αντίσταση μεταξύ δύο ποδιών και έτσι να διαιρέσετε την τάση. Αποτελείται από τρία σκέλη, όπου τα δύο σκέλη συνδέονται με τη διαδρομή αντίστασης (βλ. Σχ. 2.3) και το τρίτο σκέλος με το ρυθμιστή. Ρυθμίζοντας τη

θέση του ρυθμιστή, είτε με ένα κουμπί είτε με ένα κατσαβίδι, ανάλογα με τον τύπο του ποτενσιόμετρου, διαιρούμε τη διαδρομή αντίστασης σε δύο αντιστάσεις συνδεδεμένες σε σειρά. Με αυτόν τον τρόπο, αν συνδέσουμε το πρώτο πόδι στα 3,3V και το τρίτο πόδι στα 0V, θα έχουμε μια τάση μεταξύ 0V και 3,3V στο μεσαίο πόδι, ανάλογα με τη θέση του ρυθμιστή.



Σχήμα 2.3: Αρχή λειτουργίας ενός ποτενσιόμετρου. Πηγή: https://forbot.pl/blog/leksykon/potencjometr .

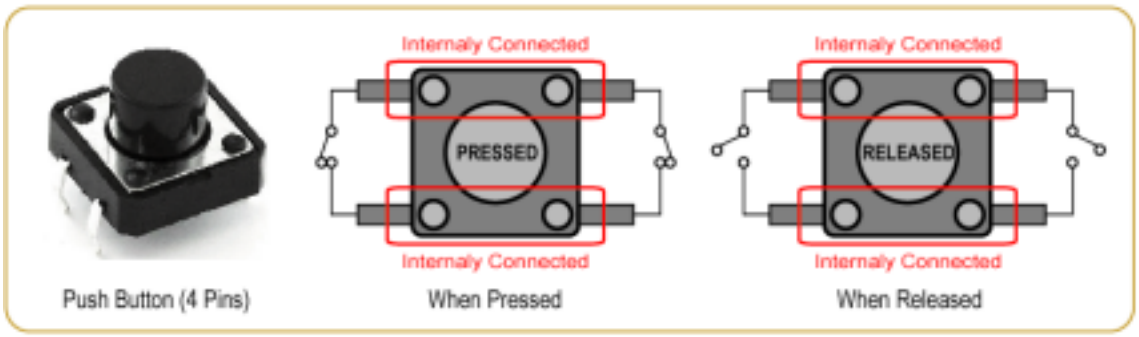
- Πυκνωτής - είναι ένα στοιχείο που συσσωρεύει ηλεκτρικό φορτίο. Αποτελείται από δύο πλάκες και ένα διηλεκτρικό μεταξύ των πλακών (βλέπε αριστερά Σχ. 2.4). Οι πυκνωτές χρησιμοποιούνται, μεταξύ άλλων στοιχείων, π.χ. για το φιλτράρισμα ενός σήματος (ο πυκνωτής δεν περνά τη συνεχή τάση και περνά τους κυματισμούς, συνεπώς μπορεί να εξομαλύνει την τάση, αν συνδεθεί στη γη) ή για τη δημιουργία κυκλωμάτων συντονισμού (εξαγωγή σήματος με συγκεκριμένη συχνότητα).



Σχήμα 2.4: Αριστερή εικόνα: εσωτερική δομή ενός πυκνωτή. Πηγή της εικόνας: https://www.circuitbread.com. Δεξιά εικόνα: παραδείγματα πολωμένων και μη πολωμένων πυκνωτών. Πηγή εικόνας: https://www.ariat-tech.pl

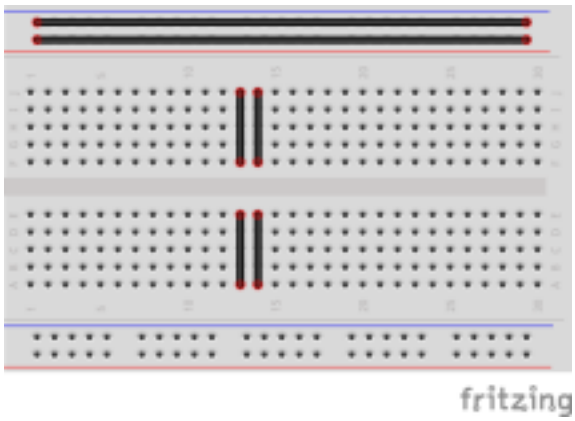
Οι πυκνωτές μπορούν να χωριστούν σε: πυκνωτές που απαιτούν κατάλληλη πόλωση (π.χ. ηλεκτρολυτικοί πυκνωτές) και σε πυκνωτές που δεν απαιτούν (π.χ. κεραμικοί πυκνωτές ή πυκνωτές φύλλων). Αυτοί που χρειάζονται σωστή πόλωση πρέπει να μόνο συνδέονται προς μία καθορισμένη κατεύθυνση (βλ. σημάνσεις στο περίβλημα του πυκνωτή). Οι ηλεκτρολυτικοί πυκνωτές χαρακτηρίζονται από υψηλή χωρητικότητα αλλά δεν είναι αποδοτικοί με σήματα υψηλής συχνότητας λόγω του συντελεστή διάχυσης. Οι κεραμικοί πυκνωτές δεν στεγνώνουν αλλά δεν είναι αποτελεσματικοί με τη διήθηση των χαμηλών συχνοτήτων, επειδή έχουν μικρότερη χωρητικότητα.

- Κουμπί - είναι ένα στοιχείο που κλείνει το κύκλωμα όταν πατηθεί το κουμπί. Όταν το κουμπί δεν είναι πατημένο, το κύκλωμα είναι ανοικτό. Η κατασκευή του κουμπιού φαίνεται στο σχήμα 2.5.



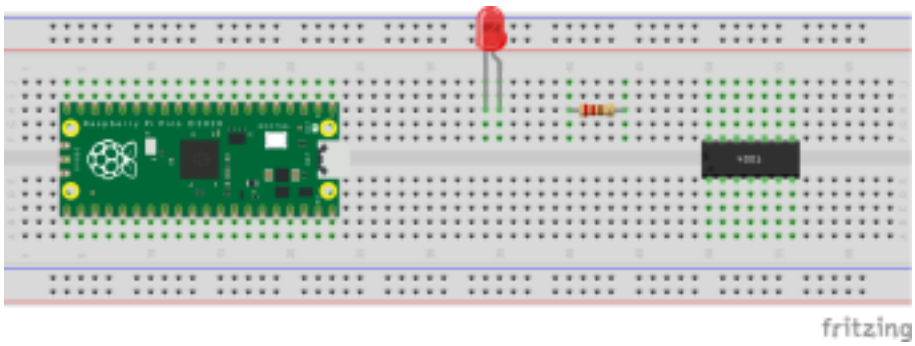
Σχήμα 2.5: Αρχή λειτουργίας του μπουτόν. Πηγή: https://gmostofabd.github.io /8051-Push-Button/.

- Breadboard (πλακέτα) - είναι ένας τύπος πλακέτας που χρησιμοποιείται για τη σύνδεση ηλεκτρονικών εξαρτημάτων, δηλαδή για την κατασκευή ηλεκτρονικών κυκλωμάτων. Η πλακέτα αποτελείται από πολλές οπές που συνδέονται κάθετα, όπως φαίνεται στο σχήμα 2.6 (μαύρη γραμμή).



Σχήμα 2.6: Οι συνδεδεμένες οπές σημειώνονται με μαύρες γραμμές στην πλακέτα.

Έτσι, οι οπές συνδέονται σε στήλες αλλά όχι σε σειρές. Η πλακέτα αποτελείται από δύο μέρη που χωρίζονται από ένα μεγάλο κενό. Τα δύο αυτά μέρη δεν συνδέονται μεταξύ τους. Στις άκρες της πλακέτας υπάρχει ένα τμήμα για τη σύνδεση της τροφοδοσίας και της γείωσης. Συνήθως επισημαίνεται με δύο γραμμές: κόκκινη και μπλε. Στην περίπτωση αυτή, οι οπές συνδέονται οριζόντια και όχι κάθετα. Δηλαδή, κατά μήκος της κόκκινης γραμμής συνδέονται όλες οι οπές και συνήθως εδώ συνδέουμε την τροφοδοσία ρεύματος, δηλαδή 3,3V στην περίπτωση του Raspberry Pi Pico. Ομοίως, όλες οι οπές κατά μήκος της μπλε γραμμής είναι συνδεδεμένες και συνήθως συνδέστε τη γείωση εκεί. Το Σχ. 2.7 δείχνει τη σωστή τοποθέτηση των ηλεκτρονικών του δείγματος εξαρτημάτων , ώστε να μην βραχυκυκλώνονται οι ακροδέκτες τους.



Σχήμα 2.7: Παράδειγμα τοποθέτησης των στοιχείων σε μια πλακέτα έτσι ώστε τα πόδια να μην βραχυκυκλώνονται.

- Αισθητήρες (Sensors)- είναι στοιχεία που επιτρέπουν τη μέτρηση διαφόρων φυσικών μεγεθών, π.χ. θερμοκρασία, πίεση, υγρασία, απόσταση, κίνηση, καρδιακός ρυθμός κ.λπ. Οι αισθητήρες επιστρέφουν τις μετρούμενες τιμές ως: αναλογικά σήματα ή ψηφιακά σήματα που συνήθως μεταδίδονται με τη χρήση διεπαφών επικοινωνίας δεδομένων (δίαυλος).
- Ενεργοποιητές (Actuators) - είναι στοιχεία που εκτελούν κίνηση, π.χ. σερβομηχανισμοί συνεχούς ρεύματος, κινητήρες .
- Ασπίδες (Shields)- είναι πλακέτες που επεκτείνουν τη λειτουργικότητα της βασικής πλακέτας. Υπάρχουν πολλοί τύποι ασπίδων. Στην αρχή, μια πολύ χρήσιμη ασπίδα είναι η GPIO Expander For Raspberry Pi Pico (π.χ. από το Waveshare *Pico στο καπέλο* - βλ. Σχ. 2.8), η οποία σας επιτρέπει να συνδέσετε εξαρτήματα στο Raspberry Pi Pico χωρίς να χρειάζεται να τοποθετήσετε το Raspberry Pi Pico στο breadboard. Αυτή η λύση μειώνει τον κίνδυνο να καταστραφεί η πλακέτα Raspberry Pi Pico από τη λανθασμένη τοποθέτησή της στο breadboard.

Σχήμα 2.8: Waveshare Pico to hat expander. Πηγή: https://www.waveshare.com/pi co-to-hat.htm

3. Εισαγωγή στη γλώσσα Micropython

Στο κεφάλαιο αυτό θα παρουσιαστούν βασικά, επιλεγμένα στοιχεία της γλώσσας , απαραίτητα για τη δημιουργία απλών έργων βασισμένων στο Raspberry Pi Pico. Αυτά είναι τα εξής:

- Μεταβλητή - είναι ένα στοιχείο προγραμματισμού που σας επιτρέπει να αποθηκεύσετε μια δεδομένη τιμή κάτω από ένα επιλεγμένο όνομα, π.χ. όταν δημιουργούμε ένα μαθηματικό πρόγραμμα για την καταμέτρηση των εμβαδών διαφόρων σχημάτων, θα χρησιμοποιούσαμε συχνά τον αριθμό *π* ίσο με, ας υποθέσουμε: 3.14. Αντί να τον εισάγετε χειροκίνητα σε κάθε εξίσωση, μπορείτε να δημιουργήσετε μια μεταβλητή: π=3,14, και τότε ένα πρόγραμμα-παράδειγμα για να μετράτε, π.χ., το εμβαδόν ενός κύκλου θα έμοιαζε ως εξής:

```
pi = 3.14
r = 10
area_circle = pi *r *r
```

1
2
3

Δείτε ότι εδώ έχουμε δημιουργήσει τρεις αριθμητικές μεταβλητές. Η μία αποθηκεύει την τιμή του αριθμού pi (pi), η δεύτερη αποθηκεύει την τιμή της ακτίνας του κύκλου (r) και η τρίτη αποθηκεύει το αποτέλεσμα, δηλαδή το εμβαδόν του κύκλου (area_circle). Θυμηθείτε ότι στην Python και τη Micropython, οι

μεταβλητές δημιουργούνται εισάγοντας πρώτα το όνομα της μεταβλητής της επιλογής σας και μετά το σύμβολο = την τιμή της. Υπάρχουν πολλοί τύποι μεταβλητών. Στο παραπάνω παράδειγμα, παρουσιάσαμε μεταβλητές τύπου *double* (αριθμοί κινητής υποδιαστολής, π.χ. 3,42, -5,68 κ.λπ.). Επιπλέον, οι βασικοί τύποι μεταβλητών περιλαμβάνουν:

- *ακέραιος αριθμός* (π.χ. 0, -4, 12 κ.λπ.),
- *boolean* (λογικές μεταβλητές με δύο πιθανές τιμές: *True* ή *False*),
- *string* (συμβολοσειρές χαρακτήρων, π.χ. "Hello", "Raspberry Pi Pico"). Στο Σχ. 3.1 παρουσιάζεται ένα παράδειγμα προγράμματος στο οποίο πολλές μεταβλητές διαφορετικών τύπων δημιουργήθηκαν . Για την εμφάνισή τους στην οθόνη μπορεί να χρησιμοποιηθεί η συνάρτηση *print()*. Ό,τι βάλουμε ως όρισμα της συνάρτησης print θα εμφανιστεί στο τερματικό. Παρατηρήστε ότι για να ξεκινήσετε το πρόγραμμα, πρέπει να πατήσετε το κουμπί *Εκτέλεση* (πράσινο κουμπί με βέλος). Αν είναι γκριζαρισμένο, ελέγξτε αν η πλακέτα Raspberry Pi Pico είναι συνδεδεμένη στον υπολογιστή μέσω USB και στη συνέχεια επιλέξτε ξανά: *Micropython (Raspberry Pi Pico)...* στην κάτω δεξιά γωνία, όπως φαίνεται από το βέλος στην εικόνα. Αν αποθηκεύσετε το πρόγραμμα στο Raspberry Pi Pico με το όνομα "main.py", θα ξεκινήσει αυτόματα όταν η πλακέτα Raspberry Pi Pico θα τροφοδοτηθεί με ρεύμα, ανεξάρτητα από το αν είναι συνδεδεμένη στον υπολογιστή ή τροφοδοτείται από π.χ. Powerbank.

Σχήμα 3.1: Το πρόγραμμα του παραδείγματος με διαφορετικού τύπου μεταβλητές.

• Λίστες (Lists) - σας επιτρέπουν να δημιουργήσετε ένα σύνολο τιμών. Ένα παράδειγμα λίστας μοιάζει με values=[1, 35, 6, 14] που σημαίνει ότι αποτελείται από 4 στοιχεία που χωρίζονται με κόμμα. Τα στοιχεία αριθμούνται με δείκτες παρόμοια με τα σπίτια κατά μήκος του δρόμου, με τη διαφορά ότι τα αριθμούμε από το μηδέν και όχι από το ένα. Έτσι, για να εξάγετε την τιμή 6 από τη λίστα, πρέπει να καλέσετε την εντολή: values[2] επειδή το 0 στοιχείο είναι η τιμή 1, το 1ο στοιχείο είναι η τιμή 35, το 2ο στοιχείο είναι η τιμή 6 και το 3ο στοιχείο είναι η τιμή 14. Οι δείκτες των στοιχείων δίνονται σε αγκύλες δίπλα στο όνομα της λίστας.

• Συναρτήσεις (Functions) - είναι μια υπορουτίνα ή ένα ξεχωριστό τμήμα ενός προγράμματος που εκτελεί μια συγκεκριμένη λειτουργία. Μια συνάρτηση μπορεί να δέχεται δεδομένα εισόδου που ονομάζονται ορίσματα συνάρτησης και μπορεί

να επιστρέφει ένα αποτέλεσμα. Για παράδειγμα, μπορείτε να γράψετε μια συνάρτηση που υπολογίζει το εμβαδόν ενός κύκλου. Για να δηλώσετε μια συνάρτηση, χρησιμοποιήστε τη λέξη def και στη συνέχεια βάλτε το όνομα της συνάρτησης, π.χ. circle_area. Μετά το όνομα της συνάρτησης, τοποθετούμε αγκύλες και μέσα δίνουμε τα ορίσματα της συνάρτησης. Στην περίπτωση του υπολογισμού του εμβαδού ενός κύκλου, πρέπει ο χρήστης να δώσει την ακτίνα του κύκλου και αυτή θα είναι το όρισμα της συνάρτησης. Μετά τις αγκύλες, τοποθετούμε μια άνω και κάτω τελεία. Στο εσωτερικό της συνάρτησης, τοποθετούμε τον κώδικα που θα εκτελεστεί όταν κληθεί η συνάρτηση, δηλαδή τον υπολογισμό του εμβαδού ενός κύκλου χρησιμοποιώντας τον τύπο: A= πr2. Για να επιστρέψει η συνάρτηση ένα αποτέλεσμα, χρησιμοποιούμε τη λέξη return και μετά από αυτήν την τιμή που θα επιστρέψει η συνάρτηση. Έτσι, ο κώδικας θα έμοιαζε ως εξής:

```
def circle_area ( radius ):
    return 3.14* radius **2
```

1
2

Παρατηρήστε ότι η λέξη return τοποθετείται 4 διαστήματα μετά τη λέξη def, η οποία βρίσκεται στην παραπάνω γραμμή. Στο Micropython, η εσοχή υποδεικνύει ποιες γραμμές βρίσκονται μέσα σε μια λειτουργία/βρόχο/δομή και ποιες έξω από αυτήν. Είναι πολύ σημαντικό να βεβαιωθείτε ότι η εσοχή είναι σωστή, διαφορετικά το πρόγραμμα θα εκτελέσει τις γραμμές εκτός συνάρτησης/βρόχου/δομής.

Το επόμενο θέμα είναι η πράξη του πολλαπλασιασμού. Στο Micropython, οι εκθετικές πράξεις εκτελούνται με την τοποθέτηση δύο αστεριών, δηλαδή *το* x^y θα γραφτεί ως $x * *y$.

Για να καλέσετε μια συγκεκριμένη συνάρτηση σε ένα πρόγραμμα, πρέπει απλώς να δώσετε το όνομά της και να βάλετε τα ορίσματα σε αγκύλες, όπως φαίνεται στο Σχήμα 3.2. Σημειώστε ότι παρουσιάζονται δύο εναλλακτικές δυνατότητες κλήσης της συνάρτησης print. Στην περίπτωση του υπολογισμού του εμβαδού του πρώτου κύκλου, το σχόλιο και η τιμή εμφανίζονται ξεχωριστά. Έτσι, αυτές οι δύο πληροφορίες βρίσκονται σε ξεχωριστές γραμμές μετά την κλήση του προγράμματος. Μπορείτε επίσης να εμφανίσετε πολλά κομμάτια πληροφοριών στη συνάρτηση print ταυτόχρονα, χωρίζοντάς τα με κόμματα. Τότε οι πληροφορίες που εμφανίζονται θα βρίσκονται σε μία γραμμή.

Σχήμα 3.2: Το πρόγραμμα του παραδείγματος με τον ορισμό της συνάρτησης.

• δομή if…else - επιτρέπει την εκτέλεση διαφορετικών τμημάτων κώδικα ανάλογα με τη συνθήκη που ικανοποιείται. Στην αρχή γράφουμε τη λέξη *if* και μετά από αυτήν, τη συνθήκη που πρέπει να ικανοποιηθεί. Στο τέλος, τοποθετούμε έναν χαρακτήρα άνω και κάτω τελείας. Όλες οι επόμενες γραμμές που τοποθετούμε μέσα στην *if* (μετά την εσοχή) θα εκτελεστούν μόνο αν η συνθήκη ικανοποιείται . Στη συνέχεια, μπορούμε, αλλά δεν χρειάζεται, να προσθέσουμε ένα μπλοκ και μετά από αυτό ένα χαρακτήρα άνω και κάτω τελεία. Όλες οι επόμενες γραμμές μέσα στο *else* μπλοκ θα εκτελεστούν μόνο αν η *else* δεν έχει ικανοποιηθεί . Ένα παράδειγμα προγράμματος στο Σχ. 3.3.

  Το πρόγραμμα του παραδείγματος χρησιμοποιεί δύο νέα στοιχεία. Το πρώτο είναι η πράξη διαίρεσης με υπόλοιπο (%), η οποία επιστρέφει το υπόλοιπο της διαίρεσης με έναν δεδομένο αριθμό, π.χ. το 12%2 θα επιστρέψει 0 επειδή δεν υπάρχει υπόλοιπο από τη διαίρεση του 12 με το 2, αλλά το 13%2 θα επιστρέψει 1. Ένα άλλο νέο πράγμα είναι η πράξη εξίσωσης. Αν θέλουμε να ελέγξουμε αν ένας αριθμός ισούται με έναν άλλο, χρησιμοποιούμε το διπλό σύμβολο ισότητας (==). Οι υπόλοιπες πράξεις εξίσωσης μοιάζουν ως εξής:

  - a>b (έλεγχος αν το a είναι μεγαλύτερο από το b)
  - a>=b (έλεγχος αν το a είναι μεγαλύτερο ή ίσο του b)
  - a<b (έλεγχος αν το a είναι μικρότερο από το b)
  - a<=b (έλεγχος αν το a είναι μικρότερο ή ίσο του b).

Σχήμα 3.3: Ένα παράδειγμα κώδικα που δείχνει τη χρήση της δομής *if...else*.

Στο παραπάνω πρόγραμμα, μπορείτε να εμφανίσετε ένα μήνυμα σχετικά με το αν ο αριθμός είναι άρτιος ή περιττός με έναν καλύτερο τρόπο. Η συνάρτηση print σας επιτρέπει να συνδυάσετε διάφορες συμβολοσειρές προσθέτοντάς τες με το σύμβολο "+". Για να εμφανίσετε τη μεταβλητή *a*, η οποία είναι ακέραιος τύπος, πρέπει πρώτα να τη μετατρέψετε σε συμβολοσειρά χρησιμοποιώντας τη συνάρτηση . *str().* Ένα παράδειγμα προγράμματος παρουσιάζεται στην Εικ. 3.4

Σχήμα 3.4: Παράδειγμα προγράμματος που δείχνει τη συνένωση συμβολοσειρών για την εμφάνισή τους σε ένα μήνυμα χρησιμοποιώντας τη συνάρτηση print.

- Βρόχος For - είναι ένας βρόχος που σας επιτρέπει να επαναλάβετε ένα συγκεκριμένο κομμάτι κώδικα αρκετές φορές. Ένα παράδειγμα προγράμματος με βρόχο for παρουσιάζεται στην Εικ. 3.5.
  Στην αρχή τοποθετείται η λέξη for, στη συνέχεια η επαναληπτική μεταβλητή (στην προκειμένη περίπτωση ονομάζεται *i*), στη συνέχεια η λέξη *in* και στη συνέχεια το εύρος. Σε αυτή την περίπτωση, το εύρος ήταν αριθμητικό από το 0 έως το 5, πράγμα που σημαίνει ότι η επαναληπτική μεταβλητή *i* θα αυξάνει την τιμή της κατά 1 κάθε φορά που εκτελείται ο βρόχος for, ξεκινώντας από το μηδέν, μέχρι μια τιμή κατά ένα μικρότερη από το ανώτερο εύρος, δηλαδή μέχρι και το 4. Στην περίπτωση αυτή, ο βρόχος for θα εκτελεστεί 5 φορές και σε κάθε επανάληψη η τιμή της επαναληπτικής *i* θα προστίθεται στη μεταβλητή *s*.

Σχήμα 3.5: Παράδειγμα προγράμματος με βρόχο for.

- Βρόχος While - εκτελεί ένα συγκεκριμένο κομμάτι κώδικα όσο πληρούται η συνθήκη. Ένα παράδειγμα προγράμματος που δείχνει τη λειτουργία του βρόχου while φαίνεται στο Σχήμα 3.6. Στην αρχή τοποθετείται η λέξη while, στη συνέχεια η συνθήκη που πρέπει να ικανοποιηθεί για να εκτελεστεί και στο τέλος ο χαρακτήρας άνω και κάτω τελεία. Σε αυτή την περίπτωση, ο βρόχος θα εκτελεστεί μέχρι η μεταβλητή *s* να είναι μικρότερη από 10. Στη μέση του βρόχου, η τιμή της μεταβλητής *s* εμφανίζεται και στη συνέχεια η τιμή της αυξάνεται κατά 2. Η λειτουργία αυτή μπορεί να γραφτεί με διάφορους τρόπους. Ένας από αυτούς είναι: *s*= s+2 ή, εν συντομία, *s*+ = 2, το οποίο θα προσθέσει επίσης την τιμή 2 στην τρέχουσα τιμή της μεταβλητής *s.*

Σχήμα 3.6: Παράδειγμα προγράμματος με βρόχο while.

Το πρόγραμμα του παραδείγματος χρησιμοποιεί το σύμβολο #. Οτιδήποτε τοποθετείται στην ίδια γραμμή μετά το σύμβολο # είναι σχόλιο που δεν εκτελείται από το πρόγραμμα.

Στην περίπτωση του προγραμματισμού μικροελεγκτών, χρησιμοποιείται συνήθως ένας άπειρος βρόχος, ο οποίος επαναλαμβάνει ένα δεδομένο τμήμα κώδικα συνεχώς μέχρι η παροχή ρεύματος στην πλακέτα να διακοπεί. Τις περισσότερες φορές, ένας άπειρος βρόχος εκτελείται με τη χρήση ενός βρόχου *while.*

Εισαγωγή στη γλώσσα Micropython καθορίζοντας μια λογική τιμή *True* ως συνθήκη:

```
while True :
        print ("It is infinite loop")
```

1
2

Τα εισαγόμενα, επιλεγμένα στοιχεία της γλώσσας είναι, σύμφωνα με τους συγγραφείς της Micropython, τα πιο χρήσιμα για την έναρξη της εργασίας με την πλακέτα Raspberry Pi Pico W. Αν χρειάζεστε περισσότερες πληροφορίες σχετικά με τη γλώσσα Micropython, είναι καλύτερο να εξοικειωθείτε με το εγχειρίδιο για τη γλώσσα αυτή.

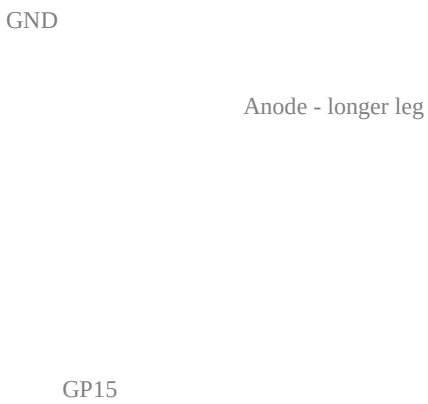
4. Τα ψηφιακά σήματα

Τα ψηφιακά σήματα είναι σήματα που έχουν μία από τις δύο δυνατές τιμές 0 ή 1, η οποία στην περίπτωση του μικροελεγκτή Raspberry Pi Pico αντιστοιχεί σε τιμές τάσης 0 V ή 3,3 V. Μπορούν να χρησιμοποιηθούν για τον έλεγχο στοιχείων όπως οι λυχνίες LED (ενεργοποίηση ή απενεργοποίηση) ή για τη λήψη πληροφοριών από στοιχεία όπως ένα κουμπί (πατημένο ή μη πατημένο) ή ένας αισθητήρας κίνησης (ανίχνευση κίνησης ή μη ανίχνευση κίνησης). Η ανάγνωση ή η παραγωγή ψηφιακών σημάτων γίνεται με τη χρήση ακροδεκτών. Οι ακροδέκτες GPIO είναι ακροδέκτες που μπορούν να χρησιμοποιηθούν τόσο ως όσο *είσοδοι/έξοδοι γενικού σκοπού (GPIO)*, από τους οποίους μπορούμε να διαβάσουμε πληροφορίες, π.χ. για το πάτημα ενός κουμπιού, και ως ακροδέκτες εξόδου, στους οποίους μπορούμε να παράγουμε ψηφιακά σήματα, π.χ. για να ανάψουμε ένα LED. Στο Σχ. 4.1 φαίνονται οι 40 ακίδες από την πλακέτα Raspberry Pi Pico, οι οποίες αριθμούνται από πάνω αριστερά. Οι ακροδέκτες που χρησιμεύουν ως GPIO συντομογραφούνται με *GPx* (πράσινα ορθογώνια), όπου *x* είναι ο αύξων αριθμός του ακροδέκτη GPIO, ξεκινώντας από το 0 έως το 28.

Σχήμα 4.1: Η διάταξη ακροδεκτών του Raspberry Pi Pico. Πηγή της εικόνας: https://www.raspberrypi.com.

4.1 Παράδειγμα 1: Έργο LED που αναβοσβήνει

Ας ξεκινήσουμε την περιπέτεια του Raspberry Pi Pico με ένα έργο που αναβοσβήνει LED. Αρχικά, συνδέστε το LED στον επιλεγμένο ακροδέκτη GPIO, όπως φαίνεται στην Εικ. 4.2.



Σχήμα 4.2: Σύνδεση του ηλεκτρονικού κυκλώματος από το παράδειγμα 1.

Τώρα ας χρησιμοποιήσουμε το Thonny για να γράψουμε ένα πρόγραμμα σε Micropython που θα ενεργοποιεί τη δίοδο και θα την απενεργοποιεί εναλλάξ κάθε 1s. Για να το κάνετε αυτό, ακολουθήστε τα εξής βήματα:

1. Ας ξεκινήσουμε με την ενσωμάτωση της βιβλιοθήκης στη Micropython:

```
import machine
```

Η βιβλιοθήκη *machine* περιέχει όλες τις απαραίτητες οδηγίες για την επικοινωνία με το Raspberry Pi Pico. Η εντολή *import συμ*περιλαμβάνει την καθορισμένη βιβλιοθήκη στο έργο.

2. Στη συνέχεια, πρέπει να ρυθμίσετε την ακίδα GP15 ώστε να λειτουργεί ως ακίδα εξόδου, επειδή ελέγχουμε το LED στέλνοντας μια τιμή 1 ή 0 στην ακίδα GP15. Η συνάρτηση *Pin από τη βιβλιοθήκη machine* χρησιμοποιείται για το σκοπό αυτό. Για να καλέσουμε συναρτήσεις από βιβλιοθήκη στη γλώσσα Micropython, πρώτα δίνουμε το όνομα της βιβλιοθήκης και μετά την τελεία το όνομα της συνάρτησης, δηλαδή *machine.Pin()*. Η συνάρτηση Pin δέχεται δύο ορίσματα. Το πρώτο είναι ο αριθμός του ακροδέκτη GPIO. Το δεύτερο είναι ο προσδιορισμός του αν το pin GPIO θα λειτουργεί ως είσοδος (*machine.Pin.IN*) ή ως έξοδος (*machine.Pin.OUT)*. Επομένως, σε αυτή την περίπτωση η κλήση της συνάρτησης θα έμοιαζε ως εξής: *machine.Pin(15, machine.Pin.OUT)*. Το αντικείμενο που επέστρεψε η συνάρτηση *Pin* ανατέθηκε στη μεταβλητή *led* που δημιουργήθηκε.

```
# Including machine library
import machine

# Configuring GPIO pin 15 as output
led = machine . Pin (15 , machine . Pin . OUT )
```

4.1 Παράδειγμα 1: έργο LED που αναβοσβήνει

3. Στο επόμενο βήμα τοποθετούμε έναν άπειρο βρόχο, ο οποίος θα περιέχει το κύριο μέρος του προγράμματος:

```
# Including machine library
import machine

# Configuring GPIO pin 15 as output
led = machine . Pin (15 , machine . Pin . OUT )

while True :
```

4. Στο επόμενο βήμα ανάβουμε το LED στέλνοντας την τιμή 1 στον ακροδέκτη GP15. Για τον ορισμό της τιμής στο pin χρησιμοποιείται η συνάρτηση *value()*, η οποία δέχεται ως όρισμα την τιμή 0 ή 1.

```
# Including machine library
import machine

# Configuring GPIO pin 15 as output
led = machine . Pin (15 , machine . Pin . OUT )

while True :
        led . value (1)
```

5. Τώρα το πρόγραμμα θα πρέπει να περιμένει 1s ώστε να δούμε το αποτέλεσμα του φωτισμού της διόδου. Για το σκοπό αυτό θα χρησιμοποιήσουμε τη βιβλιοθήκη *utime*, η οποία περιέχει συναρτήσεις για καθυστερήσεις. Πρώτα, πρέπει να προσθέσουμε τη βιβλιοθήκη:

```
# Including libraries
import machine
import utime

# Configuring GPIO pin 15 as output
led = machine . Pin (15 , machine . Pin . OUT )

while True :
        led . value (1)
```

6. Η συνάρτηση *sleep* από τη βιβλιοθήκη *utime* σας επιτρέπει να καθυστερήσετε το πρόγραμμα για έναν επιλεγμένο αριθμό δευτερολέπτων. Ας ορίσουμε μια καθυστέρηση 1s μετά το άναμμα της διόδου:

```
# Including libraries
import machine
import utime

# Configuring GPIO pin 15 as output
led = machine . Pin (15 , machine . Pin . OUT )

while True :
        led . value (1)
        utime . sleep (1)
```

7. Το τελευταίο βήμα είναι να απενεργοποιήσετε το LED στέλνοντας την τιμή 0 στον ακροδέκτη GP15 και να περιμένετε 1s για να δείτε το αποτέλεσμα.

```
# Including libraries
import machine
import utime

# Configuring GPIO pin 15 as output
led = machine . Pin (15 , machine . Pin . OUT )

while True :
        led . value (1)
        utime . sleep (1)
        led . value (0)
        utime . sleep (1)
```

Εκτελέστε τον παραπάνω κώδικα στον επεξεργαστή Thonny και δείτε το αποτέλεσμα. Εάν η δίοδος αναβοσβήνει κάθε 1s, έχετε κάνει τα πάντα σωστά. Αν όχι, ελέγξτε πρώτα τη σύνδεση της διόδου LED στην πλακέτα και στη συνέχεια ελέγξτε αν δεν κάνατε κάποιο λάθος στο πρόγραμμα όταν ξαναγράψατε τον

κώδικα.

Συμβουλή 4.1.1

Στο παραπάνω παράδειγμα, θέτουμε την τιμή του ακροδέκτη GP15 σε 1 ή 0 για να ενεργοποιήσουμε ή να απενεργοποιήσουμε τη δίοδο. Για το σκοπό αυτό χρησιμοποιήθηκε η συνάρτηση value(). Ωστόσο, το πρόγραμμα μπορεί να γραφτεί ακόμα πιο απλά. Η συνάρτηση toggle() αλλάζει την τιμή σε ένα συγκεκριμένο pin στο αντίθετο, δηλαδή αν η τιμή είχε οριστεί σε 1, αλλάζει σε 0, και αν η τιμή είχε οριστεί σε 0, αλλάζει σε 1. Έτσι, στο παραπάνω παράδειγμα, το εσωτερικό του άπειρου βρόχου θα άλλαζε σε:

```
while True :
        led . toggle ()
        utime . sleep (1)
```

1
2
3

Συμβουλή 4.1.2

Αν αποθηκεύσετε το πρόγραμμα στο Raspberry Pi Pico (όχι στον υπολογιστή) με το όνομα "main.py", θα ξεκινήσει αυτόματα όταν η πλακέτα Raspberry Pi Pico τροφοδοτηθεί με ρεύμα, ανεξάρτητα από το αν είναι συνδεδεμένη με τον υπολογιστή ή τροφοδοτείται από π.χ. Powerbank.

4.2 Παράδειγμα 2: Ενεργοποίηση/απενεργοποίηση LED με κουμπί πίεσης

Σε αυτό το παράδειγμα, η λυχνία LED θα ανάψει ή θα σβήσει όταν πατηθεί το κουμπί. Για να το κάνετε αυτό, πρώτα συνδέστε το κύκλωμα σύμφωνα με το Σχ. 4.3.

Το κουμπί έχει συνδεθεί στην ακίδα GP14 και η λυχνία LED στην GP15. Τώρα ανοίξτε το Thonny και ας γράψουμε κάποιο κώδικα Micropython που θα ελέγχει το LED ανάλογα με το πάτημα του κουμπιού. Για να το κάνετε αυτό, ακολουθήστε τα παρακάτω βήματα:

GP14

Σχήμα 4.3: Σύνδεση του ηλεκτρονικού κυκλώματος από το παράδειγμα 2.

1. Πρώτα, πρέπει να προσθέσετε τη βιβλιοθήκη machine και να ρυθμίσετε το pin GP15, στο οποίο το είναι συνδεδεμένο LED, ως έξοδο, όπως έγινε στο προηγούμενο παράδειγμα.

```
import machine

led = machine . Pin (15 , machine . Pin . OUT )
```

2. Στη συνέχεια θα πρέπει να διαμορφώσετε τον ακροδέκτη GP14, στον οποίο είναι συνδεδεμένο το κουμπί, ως στοιχείο εισόδου (επιλογή μηχανής.Pin.IN) γιατί θα διαβάσουμε την τιμή από το κουμπί (είτε έχει πατηθεί). Το κουμπί θα πρέπει επίσης να συνδεθεί με αντιστάσεις που ονομάζονται pull-up ή pull-down, οι οποίες είναι ήδη τοποθετημένες μέσα στην πλακέτα Raspberry Pi Pico και συνδέονται με όλες τις ακίδες GPIO. Αυτές οι αντιστάσεις είναι απαραίτητες για να αποφευχθεί η αιωρούμενη είσοδος, δηλαδή μια κατάσταση στην οποία θα διαβάζονταν πληροφορίες από τον ακροδέκτη GPIO ότι το κουμπί είναι πατημένο όταν δεν το έχουμε πατήσει καθόλου. Αυτό μπορεί να θεωρηθεί ως δυσλειτουργία και σοβαρό σχεδιαστικό λάθος. Οι συρόμενες αντιστάσεις συνδέουν το κουμπί με τη γείωση, πράγμα που σημαίνει ότι εάν δεν πατηθεί το κουμπί, θα διαβαστεί η τιμή 0. Οι αντιστάσεις έλξης συνδέουν το κουμπί στα 3,3 V, πράγμα που σημαίνει ότι εάν δεν πατηθεί το κουμπί, θα διαβαστεί η τιμή 1. Σε αυτή την περίπτωση, θα χρησιμοποιήσουμε pull-down αντιστάσεις. Για το σκοπό αυτό, το machine.Pin.PULL_DOWN θα πρέπει να παρέχεται ως το τρίτο όρισμα της συνάρτησης Pin και το απέναντι καλώδιο του κουμπιού πρέπει να συνδεθεί στα +3,3 V. Εάν θέλουμε να χρησιμοποιήσουμε αντιστάσεις έλξης, θα πρέπει να περάσουμε το machine.Pin.PULL_UP ως τρίτο όρισμα και το αντίθετο καλώδιο κουμπιού πρέπει να συνδεθεί στη γείωση (GND), η οποία έχει δυναμικό 0 V.

```
import machine

led = machine . Pin (15 , machine . Pin . OUT )
but = machine . Pin (14 , machine . Pin . IN , machine . Pin . PULL_DOWN )
```

3. Στο επόμενο βήμα, ας απενεργοποιήσουμε πρώτα το LED ορίζοντας την τιμή 0:

```
import machine

led = machine . Pin (15 , machine . Pin . OUT )
but = machine . Pin (14 , machine . Pin . IN , machine . Pin . PULL_DOWN )

led . value (0)
```

4. Τώρα ας δημιουργήσουμε τον κύριο βρόχο του προγράμματος και σε αυτόν ας ελέγξουμε την κατάσταση του κουμπιού. Εάν η τιμή που επιστρέφεται από το κουμπί είναι ίση με 1, σημαίνει ότι το κουμπί έχει πατηθεί:

```
import machine

led = machine . Pin (15 , machine . Pin . OUT )
but = machine . Pin (14 , machine . Pin . IN , machine . Pin . PULL_DOWN )

led . value (0)

while True :
        if but . value () ==1:
```

5. Στη συνέχεια, ας αλλάξουμε την κατάσταση LED στο αντίθετο όταν πατηθεί το κουμπί:

```
import machine

led = machine . Pin (15 , machine . Pin . OUT )
but = machine . Pin (14 , machine . Pin . IN , machine . Pin . PULL_DOWN )

led . value (0)

while True :
        if but . value () ==1:
                led . toggle ()
```

6. Το τελευταίο βήμα είναι να προσθέσετε μια καθυστέρηση 1 δευτερολέπτου. Αυτή η καθυστέρηση μπορεί να είναι μικρότερη αλλά είναι απαραίτητη γιατί όταν

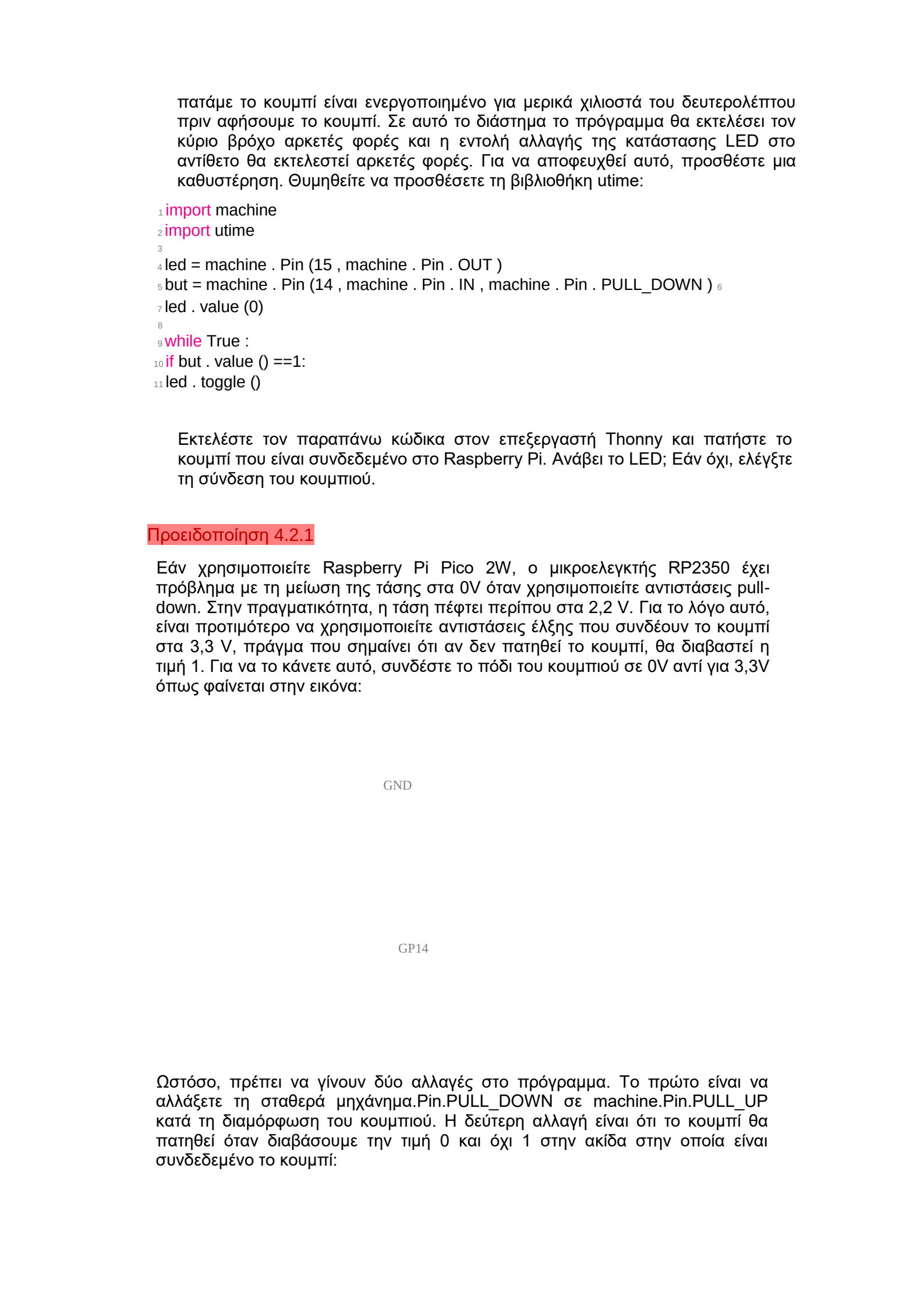
πατάμε το κουμπί είναι ενεργοποιημένο για μερικά χιλιοστά του δευτερολέπτου πριν αφήσουμε το κουμπί. Σε αυτό το διάστημα το πρόγραμμα θα εκτελέσει τον κύριο βρόχο αρκετές φορές και η εντολή αλλαγής της κατάστασης LED στο αντίθετο θα εκτελεστεί αρκετές φορές. Για να αποφευχθεί αυτό, προσθέστε μια καθυστέρηση. Θυμηθείτε να προσθέσετε τη βιβλιοθήκη utime:

```
import machine
import utime

led = machine . Pin (15 , machine . Pin . OUT )
but = machine . Pin (14 , machine . Pin . IN , machine . Pin . PULL_DOWN )
led . value (0)

while True :
if but . value () ==1:
led . toggle ()
```

Εκτελέστε τον παραπάνω κώδικα στον επεξεργαστή Thonny και πατήστε το κουμπί που είναι συνδεδεμένο στο Raspberry Pi. Ανάβει το LED; Εάν όχι, ελέγξτε τη σύνδεση του κουμπιού.

Προειδοποίηση 4.2.1

Εάν χρησιμοποιείτε Raspberry Pi Pico 2W, ο μικροελεγκτής RP2350 έχει πρόβλημα με τη μείωση της τάσης στα 0V όταν χρησιμοποιείτε αντιστάσεις pull-down. Στην πραγματικότητα, η τάση πέφτει περίπου στα 2,2 V. Για το λόγο αυτό, είναι προτιμότερο να χρησιμοποιείτε αντιστάσεις έλξης που συνδέουν το κουμπί στα 3,3 V, πράγμα που σημαίνει ότι αν δεν πατηθεί το κουμπί, θα διαβαστεί η τιμή 1. Για να το κάνετε αυτό, συνδέστε το πόδι του κουμπιού σε 0V αντί για 3,3V όπως φαίνεται στην εικόνα:



Ωστόσο, πρέπει να γίνουν δύο αλλαγές στο πρόγραμμα. Το πρώτο είναι να αλλάξετε τη σταθερά μηχάνημα.Pin.PULL_DOWN σε machine.Pin.PULL_UP κατά τη διαμόρφωση του κουμπιού. Η δεύτερη αλλαγή είναι ότι το κουμπί θα πατηθεί όταν διαβάσουμε την τιμή 0 και όχι 1 στην ακίδα στην οποία είναι συνδεδεμένο το κουμπί:

```
import machine
import utime

led = machine . Pin (15 , machine . Pin . OUT )
but = machine . Pin (14 , machine . Pin . IN , machine . Pin . PULL_UP ) led

. value (0)

while True :
        if but . value () ==0:
                led . toggle ()
        utime . sleep (1)
```

4.3 Παράδειγμα 3: Φως που ανάβει με αισθητήρα κίνησης

Σε αυτό το παράδειγμα, θα δημιουργήσουμε αυτόματο διακόπτη φωτός που θα ανάβει το φως όταν ανιχνευτεί η κίνηση από τον αισθητήρα κίνησης. Για το σκοπό αυτό, θα χρησιμοποιήσουμε μια δίοδο LED και έναν αισθητήρα κίνησης HC-SR501 PIR που φαίνεται στην Εικ. 4.4 Σύμφωνα με το εγχειρίδιο του αισθητήρα PIR HC-SR501, αυτός ο αισθητήρας πρέπει να συνδεθεί σε τροφοδοτικό από 5V έως 20V. Στην πλακέτα Raspberry Pi Pico W, παρέχεται τροφοδοτικό 5V στον ακροδέκτη VBUS. Όταν χρησιμοποιείτε αισθητήρες που τροφοδοτούνται από τάση υψηλότερη από 3,3 V, βεβαιωθείτε ότι το επιστρεφόμενο σήμα έχει τάση όχι μεγαλύτερη από 3,3 V, διαφορετικά μπορεί να καταστρέψουμε την πλακέτα Raspberry Pi Pico. Πού μπορώ να βρω τέτοιες πληροφορίες; Ο ευκολότερος τρόπος είναι να βρείτε ένα φύλλο δεδομένων για έναν δεδομένο αισθητήρα στο Διαδίκτυο και να διαβάσετε τι τάση έχει το υψηλό σήμα που παράγεται από τον αισθητήρα. Σε αυτήν την περίπτωση, ο κατασκευαστής δηλώνει ότι ο αισθητήρας PIR HC-SR501 επιστρέφει σήματα με τιμή 3,3V ή 0V.

Σχήμα 4.4: PIR Motion Sensor HC-SR501. Πηγή: https://www.unoelectro.com.ar.

Συνδέστε λοιπόν τον αισθητήρα κίνησης PIR και τη δίοδο LED στην πλακέτα Raspberry Pi Pico W όπως φαίνεται στην Εικ. 4.5.

Σχήμα 4.5: Σύνδεση του ηλεκτρονικού κυκλώματος από το παράδειγμα3.

Τώρα ας γράψουμε ένα πρόγραμμα στον επεξεργαστή Thonny που θα ανάψει το LED για 5 δευτερόλεπτα όταν ανιχνευτεί κίνηση. Για να το κάνετε αυτό, ακολουθήστε τα εξής βήματα:

1. Πρώτα πρέπει να εισαγάγετε τις βιβλιοθήκες *machine* και *utime*:

```
import machine
import utime
```

```
1
2
```

2. Στη συνέχεια, πρέπει να διαμορφώσετε την ακίδα GP15 στην οποία είναι συνδεδεμένη η λυχνία LED ως έξοδο και να απενεργοποιήσετε τη λυχνία LED:

```
import machine
import utime

LED = machine . Pin (15 , machine . Pin . OUT )
LED . value (0)
```

```
1
2
3
4
```

3. Ο αισθητήρας κίνησης PIR είναι ένα στοιχείο εισόδου που επιστρέφει 0 ή 1 ανάλογα με το αν έχει ανιχνεύσει κίνηση. Επομένως, η ακίδα GP22, στην οποία είναι συνδεδεμένος ο αισθητήρας κίνησης PIR, θα πρέπει να διαμορφωθεί ως είσοδος:

```
import machine
import utime

LED = machine . Pin (15 , machine . Pin . OUT )
LED . value (0)

PIR = machine . Pin (22 , machine . Pin . IN )
```

4. Τώρα στον κύριο βρόχο πρέπει να διαβάσετε την τιμή που επιστρέφεται από τον αισθητήρα κίνησης και να την εμφανίσετε στο τερματικό:

```
import machine
import utime

LED = machine . Pin (15 , machine . Pin . OUT )
LED . value (0)

PIR = machine . Pin (22 , machine . Pin . IN )

while True :
        motion = PIR . value ()
        print ( motion )
```

5. Τώρα εκτελέστε το πρόγραμμα και ελέγξτε πώς συμπεριφέρεται ο αισθητήρας κίνησης. Το Σχ. 4.4 δείχνει δύο ποτενσιόμετρα που επισημαίνονται ως "Χρονισμός εξόδου" και "Ευαισθησία". Μπορείτε να τα χρησιμοποιήσετε για να ρυθμίσετε τον αισθητήρα κίνησης. Σας συνιστώ να ρυθμίσετε το ποτενσιόμετρο "Output timing" στη χαμηλότερη δυνατή τιμή και να προσαρμόσετε την ευαισθησία όπως επιθυμείτε. Μην ανησυχείτε εάν ο αισθητήρας επιστρέψει την τιμή 1 πολλές φορές μετά την ανίχνευση κίνησης. Αυτό είναι φυσιολογικό επειδή

η μικρότερη διάρκεια σήματος είναι μεγαλύτερη από τον χρόνο εκτέλεσης του κύριου βρόχου προγράμματος. Αφού ρυθμίσετε τον αισθητήρα κίνησης, συνεχίστε να γράφετε το πρόγραμμα.

6. Στο επόμενο βήμα, το LED θα πρέπει να ανάψει όταν ανιχνευτεί κίνηση για 5 δευτερόλεπτα, διαφορετικά το LED θα πρέπει να είναι σβηστό:

```
import machine
import utime

LED = machine . Pin (15 , machine . Pin . OUT )
LED . value (0)

PIR = machine . Pin (22 , machine . Pin . IN )

while True :
        motion = PIR . value ()
        print ( motion )
        if( motion ==1) :
                LED . value (1)
                utime . sleep (5)
        else :
                LED . value (0)
```

Το πρόγραμμα είναι έτοιμο. Δοκιμάστε τη λειτουργία του.

4.4 PWM

Ορισμένα εξαρτήματα, όπως τα LED RGB ή οι σερβομηχανισμοί ελέγχονται χρησιμοποιώντας τη διαμόρφωση πλάτους παλμού (PWM). Αυτή η τεχνική επιτρέπει τη δημιουργία ορθογώνιων σημάτων με καθορισμένο κύκλο λειτουργίας από 0 έως 100%. Για παράδειγμα, αν ο κύκλος λειτουργίας έχει ρυθμιστεί στο 20%, τότε για το 20% της διάρκειας παλμού έχουμε υψηλό σήμα και για το 80% της διάρκειας παλμού έχουμε χαμηλό σήμα. Παραδείγματα κυματομορφών σήματος που παράγονται χρησιμοποιώντας την τεχνική PWM φαίνονται στο Σχ. 4.6.

Εικόνα 4.6: Παράδειγμα σημάτων PWM με κύκλους λειτουργίας 10% (επάνω διάγραμμα), 50% (μεσαίο διάγραμμα) και 90% (κάτω διάγραμμα). Πηγή: https://www.robotyka.net.pl/pwm-modulacja-sz erokosci-impulsu/.

4.4.0.1 Παράδειγμα 4: RGB LED

Μια δίοδος RGB αποτελείται από τρεις διόδους στα χρώματα κόκκινο, πράσινο και μπλε. Για να δημιουργήσετε ένα επιλεγμένο χρώμα, ελέγχετε το γέμισμα κάθε χρώματος με ένα PWM. Για παράδειγμα, το χρώμα της ελιάς είναι ένα μείγμα κόκκινου και πράσινου. Στον κώδικα RGB, για να το δημιουργήσετε, πρέπει να ορίσετε 50% κόκκινο γέμισμα και 50% πράσινο γέμισμα. Τα σήματα PWM αυτό το καθιστούν δυνατό, επομένως χρησιμοποιούνται για τον έλεγχο των διόδων RGB. Το RGB LED έχει 4 ακίδες όπως φαίνεται στην Εικ. 4.7.

Σχήμα 4.7: Τύποι διόδων RGB LED. Πηγή: https://www.build-electronic-cir cuits.com/rgb-led/ .

Το μεγαλύτερο προβάδισμα είναι το κοινό προβάδισμα. Εάν χρησιμοποιούμε μια κοινή καθοδική δίοδο RGB, το μεγαλύτερο καλώδιο θα πρέπει να συνδεθεί στο GND. Και ο φωτισμός ενός συγκεκριμένου χρώματος γίνεται με τη ρύθμιση της υψηλής κατάστασης σε ένα κόκκινο, πράσινο ή μπλε καλώδιο. Εάν η δίοδος RGB έχει κοινή άνοδο, τότε το μεγαλύτερο καλώδιο θα πρέπει να συνδεθεί στα 3,3 V και ο φωτισμός ενός συγκεκριμένου χρώματος γίνεται με τη ρύθμιση της χαμηλής κατάστασης σε ένα δεδομένο καλώδιο. Σε αυτό το παράδειγμα, θα χρησιμοποιήσουμε μια κοινή καθοδική δίοδο RGB, η οποία θα πρέπει να συνδεθεί όπως φαίνεται στο Σχ. 4.8. Τα υπόλοιπα 3 πόδια είναι οι αγωγοί για τα μεμονωμένα χρώματα: κόκκινο, πράσινο και μπλε. Μην

ξεχάσετε να προσθέσετε αντιστάσεις περιορισμού ρεύματος σε κάθε καρφίτσα ελέγχου χρώματος (3 x 330 Ω θα πρέπει να είναι η βέλτιστη).

Σχήμα 4.8: Παράδειγμα σύνδεσης κοινής καθόδου RGB LED. Χρησιμοποιήσαμε αντιστάσεις 3 x 330 Ω για τον περιορισμό ρεύματος κάθε LED.

Αφού συνδέσετε τη δίοδο RGB, ανοίξτε το Thonny και ξεκινήστε να γράφετε τον κώδικα που θα εμφανίζει τις ακολουθίες χρωμάτων: κόκκινο, πράσινο, μπλε, ροζ, λευκό και κίτρινο. Για να το κάνετε αυτό, πρέπει:

1. Αρχικά, προσθέστε τις βιβλιοθήκες *machine και utime*:

```
import machine
import utime
```

```
1
2
```

2. Στη συνέχεια, πρέπει να ορίσετε ποιες ακίδες θα χρησιμοποιηθούν για τη δημιουργία σημάτων PWM. Αυτό γίνεται χρησιμοποιώντας τη συνάρτηση PWM από τη βιβλιοθήκη *machine*, η οποία παίρνει ως όρισμα ένα αντικείμενο Pin. Όπως αναφέρθηκε, το RGB LED αποτελείται από τρεις διόδους (κόκκινο, πράσινο και μπλε), επομένως πρέπει να διαμορφώσετε τρεις ακίδες ως PWM.

```
import machine
import utime

red = machine . PWM ( machine . Pin (11) )
green = machine . PWM ( machine . Pin (12) )
blue = machine . PWM ( machine . Pin (13) )
```

```
1
2
3
4
5
6
```

3. Στη συνέχεια, πρέπει να ρυθμίσετε τη συχνότητα σήματος PWM. Για αυτό χρησιμεύει η συνάρτηση *freq*:

```
import machine
import utime

red = machine . PWM ( machine . Pin (11) )
green = machine . PWM ( machine . Pin (12) )
blue = machine . PWM ( machine . Pin (13) )

red . freq (1000)
green . freq (1000)
blue . freq (1000)
```

3. Στο Raspberry Pi Pico W, η ανάλυση των σημάτων PWM είναι 16 bit. Αυτό σημαίνει ότι η πλήρωση σήματος PWM έχει οριστεί από 0 έως 65535, όπου το 65535 είναι 100% γέμισμα. Ωστόσο, τα χρώματα δίνονται στον κώδικα RGB, όπου οι τιμές είναι από 0 έως 255. Ως εκ τούτου, είναι βολικό να δημιουργηθούν συναρτήσεις για τη ρύθμιση της πλήρωσης σε μια δεδομένη δίοδο, οι οποίες θα μετατρέψουν την τιμή που δίνεται στο εύρος (0; 255) στο εύρος (0; 65535), έτσι ώστε ο χρήστης να μην χρειάζεται να υπολογίσει ξανά αυτές τις τιμές χειροκίνητα. Για να μετατρέψετε τις τιμές που δίνονται στο εύρος (0; 255) στο εύρος (0; 65535), οι τιμές που δίνονται θα πρέπει να πολλαπλασιαστούν επί 65535/255=257:

```
import machine
import utime

red = machine . PWM ( machine . Pin (11) )
green = machine . PWM ( machine . Pin (12) )
blue = machine . PWM ( machine . Pin (13) )

red . freq (1000)
green . freq (1000)
blue . freq (1000)


def set_color (r , g , b ):
red . duty_u16 (r *257)
green . duty_u16 ( g *257)
blue . duty_u16 ( b *257)
```

5. Τώρα πρέπει να δημιουργήσουμε τον κύριο βρόχο while, στον οποίο θα οριστούν τα χρώματα κόκκινο, πράσινο και μπλε και θα υπάρχει καθυστέρηση 1 δευτερολέπτων μεταξύ τους:

```
import machine
import utime

red = machine . PWM ( machine . Pin (11) )
green = machine . PWM ( machine . Pin (12) )
blue = machine . PWM ( machine . Pin (13) )

red . freq (1000)
green . freq (1000)
blue . freq (1000)

def set_color (r , g , b ):
        red . duty_u16 (r *257)
        green . duty_u16 ( g *257)
        blue . duty_u16 ( b *257)

while True :
        set_color (255 ,0 ,0) # red
        utime . sleep (1)

        set_color (0 ,255 ,0) # green
        utime . sleep (1)

        set_color (0 ,0 ,255) # blue
        utime . sleep (1)
```

6. Εκτελέστε το πρόγραμμα και δείτε εάν αυτά τα τρία χρώματα εμφανίζονται στη δίοδο RGB με τη σειρά. Εάν όχι, ελέγξτε τη σύνδεση και τον κώδικα. Εάν εμφανίζονται, προσθέστε τα υπόλοιπα χρώματα για εμφάνιση, δηλαδή: ροζ, λευκό και κίτρινο.

```
import machine
import utime
```

```
3
4 red = machine . PWM ( machine . Pin (11) )
5 green = machine . PWM ( machine . Pin (12) )
6 blue = machine . PWM ( machine . Pin (13) )
7
8 red . freq (1000)
9 green . freq (1000)
10 blue . freq (1000)
11
12 def set_color (r , g , b ):
13 red . duty_u16 (r *257)
14 green . duty_u16 ( g *257)

15 blue . duty_u16 ( b *257)
16
17 while True :
18 set_color (255 ,0 ,0) # red
19 utime . sleep (1)
20 set_color (0 ,255 ,0) # green
21 utime . sleep (1)
22 set_color (0 ,0 ,255) # blue
23 utime . sleep (1)
24 set_color (255 ,20 ,147) # pink
25 utime . sleep (1)
26 set_color (255 ,255 ,255) # white
27 utime . sleep (1)
28 set_color (255 , 255 , 0) # yellow
29 utime . sleep (1)
```

Δοκιμάστε το ολοκληρωμένο πρόγραμμα.

Συμβουλή 4.4.1

Τι πρέπει να αλλάξει στο παραπάνω παράδειγμα εάν έχουμε μια κοινή λυχνία LED ανόδου RGB; Δύο πράγματα πρέπει να αλλάξουν:

1. Η σύνδεση του RGB LED. Το μαύρο καλώδιο από το Σχ. 4.8 θα πρέπει να συνδεθεί σε 3.3V και όχι σε GND. 2. Τα μεμονωμένα χρώματα στο LED κοινής ανόδου RGB ανάβουν σε χαμηλή κατάσταση και όχι σε υψηλή κατάσταση. Έτσι, για να ορίσετε 100% γέμισμα, θα πρέπει να ορίσετε την τιμή 0 και όχι 65535 σε μια δεδομένη ακίδα. Ως εκ τούτου, θα πρέπει να τροποποιήσετε τη συνάρτηση set_color() ως εξής:

```
def set_color (r , g , b ):
        red . duty_u16 (65535 - r *257)
        green . duty_u16 (65535 - g *257)
        blue . duty_u16 (65535 - b *257)
```

```
1
2
3
4
```

Προειδοποίηση 4.4.1

Όταν χρησιμοποιείτε περισσότερες ακίδες που έχουν διαμορφωθεί για τη δημιουργία σημάτων PWM, να είστε προσεκτικοί. Στο Raspberry Pi Pico W

έχουμε 8 κανάλια PWM, το καθένα με δύο εξόδους A και B (βλ. Εικ. 4.9). Κάθε έξοδος με τον ίδιο αριθμό, π.χ. Τα A[5] και B[5] δεν είναι εντελώς ανεξάρτητα. Μπορούμε να χρησιμοποιήσουμε αυτές τις δύο εξόδους σε ένα πρόγραμμα και να ορίσουμε διαφορετικούς κύκλους λειτουργίας PWM, αλλά η συχνότητα αυτών των σημάτων θα είναι η ίδια.

Σχήμα 4.9: Τα PWM κανάλια. Πηγή: https://www.codrey.com/raspberry-pi/raspberry-pi-pico-pwm-primer/.

5. Τα αναλογικά σήματα

Εκτός από τα ψηφιακά σήματα που έχουν δύο πιθανές καταστάσεις 0 ή 1 (0 V ή 3,3 V αντίστοιχα στο πρότυπο TTL χαμηλής τάσης), τα αναλογικά σήματα μεταφέρουν πληροφορίες όσον αφορά την τάση ή το ρεύμα. Αυτά παρέχουν συνεχείς τιμές σε ένα δεδομένο εύρος, π.χ. ένας αισθητήρας που μετράει τη θερμοκρασία θα επιστρέψει

τάση ανάλογη της θερμοκρασίας (π.χ. 235 mV θα αντιστοιχεί στη θερμοκρασία των 23,5oC. Μπορούμε να πούμε ότι το κέρδος του αισθητήρα ισούται με 10 mV/oC). Για την ανάγνωση της τάσης των αναλογικών σημάτων, είναι απαραίτητος ένας μετατροπέας αναλογικού σε ψηφιακό (ADC). Μια τέτοια συσκευή είναι ήδη ενσωματωμένη στον μικροελεγκτή Raspberry Pi Pico. Ο ενσωματωμένος μετατροπέας είναι 12-bit. Εάν θέλουμε πιο ακριβείς μετρήσεις, μπορούμε να συνδέσουμε έναν εξωτερικό μετατροπέα στο Raspberry Pi Pico (περισσότερα για αυτό αργότερα στον οδηγό).

5.1 Παράδειγμα 5: Μέτρηση θερμοκρασίας

Η μέτρηση της θερμοκρασίας μπορεί να γίνει με δύο τρόπους: χρησιμοποιώντας τον ενσωματωμένο αισθητήρα θερμοκρασίας (biased bipolar diode) ή έναν εξωτερικό αισθητήρα θερμοκρασίας, π.χ. LM35. Σε αυτό το παράδειγμα θα εφαρμοστούν και οι δύο μέθοδοι και θα συγκριθούν τα αποτελέσματα. Ας ξεκινήσουμε με τον ενσωματωμένο αισθητήρα θερμοκρασίας. Ανοίξτε το πρόγραμμα επεξεργασίας Thonny και εκτελέστε τα ακόλουθα βήματα:

1. Πρώτα πρέπει να προσθέσετε τις βιβλιοθήκες machine και utime:

```
import machine
import utime
```

2. Στη συνέχεια, πρέπει να διαμορφώσετε τον ακροδέκτη ώστε να λειτουργεί ως ADC, για το σκοπό αυτό χρησιμοποιείται η συνάρτηση ADC (αριθμός ακίδας). Ο ενσωματωμένος αισθητήρας θερμοκρασίας είναι συνδεδεμένος στο τέταρτο κανάλι ADC:

```
import machine
import utime

built_in_temp = machine . ADC (4)
```

3. Στη συνέχεια, στον κύριο βρόχο while, διαβάστε την τιμή από το ADC χρησιμοποιώντας τη συνάρτηση read_u16(). Αν και το ADC είναι 12 bit, η συνάρτηση read_u16 μετατρέπει αμέσως την τιμή από 12 bit σε 16 bit:

```
import machine
import utime

built_in_temp = machine . ADC (4)

while True :
        built_in_temp . read_u16 ()
```

4. Για να λάβετε την τιμή τάσης, μετατρέψτε τις τιμές που επιστρέφονται από τη συνάρτηση read_u16 στην περιοχή (0 έως 65535) στην τάση (0 έως 3.3)V:

```
import machine
import utime

built_in_temp = machine . ADC (4)

while True :
        voltage1 = built_in_temp . read_u16 () *3.3/65535
```

5. Στη συνέχεια, πρέπει να μετατρέψετε την τάση σε θερμοκρασία σύμφωνα με τον τύπο στην τεκμηρίωση RP2040. Το αποτέλεσμα θα εμφανίζεται στο τερματικό κάθε 1 δευτερόλεπτο:

```
import machine
import utime

built_in_temp = machine . ADC (4)

while True :
        voltage1 = built_in_temp . read_u16 () *3.3/65535 temp1 = 27
        - ( voltage1 -0.706) /0.001721
        print ( temp1 )
        utime . sleep (1)
```

Δοκιμάστε το πρόγραμμα.

Η μέτρηση με χρήση του ενσωματωμένου αισθητήρα θερμοκρασίας επιβαρύνεται με δύο βασικά προβλήματα. Το πρώτο είναι η υψηλή ανακρίβεια της μέτρησης της θερμοκρασίας που προκύπτει από την ευαισθησία των τιμών ανάγνωσης ανάλογα με την τάση αναφοράς. Μια αλλαγή στην τάση αναφοράς κατά 1% προκαλεί ήδη σφάλμα ένδειξης θερμοκρασίας περίπου 4oC. Ένα άλλο πρόβλημα είναι το γεγονός ότι ο ενσωματωμένος αισθητήρας θερμοκρασίας μετράει πραγματικά τη θερμοκρασία του RP2040 και όχι του περιβάλλοντος. Επομένως, κατά τη διάρκεια εντατικής λειτουργίας του RP2040, η θερμοκρασία ανάγνωσης μπορεί να είναι πολύ υψηλότερη από τη θερμοκρασία περιβάλλοντος. Εάν θέλετε πιο ακριβή μέτρηση θερμοκρασίας, είναι προτιμότερο να χρησιμοποιήσετε έναν εξωτερικό αισθητήρα θερμοκρασίας. Τώρα ας επεκτείνουμε το πρόγραμμα προσθέτοντας ένδειξη θερμοκρασίας από τον αναλογικό αισθητήρα LM35. Για να το κάνετε αυτό, συνδέστε τον αισθητήρα LM35 στο Raspberry Pi Pico W όπως φαίνεται στην Εικ. 5.1. Οι ακίδες στις οποίες οδηγούνται τα κανάλια ADC είναι: GP26, GP27 και GP28 (δείτε την Εικ. 1.1 - σκούρα πράσινα σημάδια).

Σχήμα 5.1: Παράδειγμα σύνδεσης αισθητήρα LM35.

6Τώρα ας διαμορφώσουμε τον ακροδέκτη GP26 ώστε να λειτουργεί ως ADC:

```
import machine
import utime

built_in_temp = machine . ADC (4)
external_temp = machine.ADC(26)

while True :
        voltage1 = built_in_temp . read_u16 () *3.3/65535 temp1 = 27
        - ( voltage1 -0.706) /0.001721
        print ( temp1 )
        utime . sleep (1)
```

7. Στο επόμενο βήμα, διαβάστε την τάση που επιστρέφει ο αισθητήρας LM35 και πολλαπλασιάστε το αποτέλεσμα επί 100 για να λάβετε τη θερμοκρασία σε βαθμούς Κελσίου σύμφωνα με την τεκμηρίωση του αισθητήρα LM35 (το κέρδος του αισθητήρα είναι 10 mV/oC, επομένως χρειαζόμαστε τάση σε mV, άρα πολλαπλασιάζουμε με 1000 και διαιρούμε με το κέρδος του αισθητήρα: 10, για να έχουμε τη θερμοκρασία σε oC. Επομένως, πολλαπλασιάστε την τιμή μετά την μετατροπή ADC επί 100):

```
import machine
import utime

built_in_temp = machine . ADC (4)
external_temp = machine . ADC (26)

while True :
voltage1 = built_in_temp . read_u16 () *3.3/65535 temp1 = 27 - ( voltage1 -
0.706) /0.001721
voltage2 = external_temp . read_u16 () *3.3/65535 temp2 = voltage2 *100


utime . sleep (1)
```

8. Το τελευταίο βήμα είναι να εμφανιστούν και τα δύο αποτελέσματα στο τερματικό σε μία γραμμή:

```
import machine
import utime

built_in_temp = machine . ADC (4)
external_temp = machine . ADC (26)

while True :
        voltage1 = built_in_temp . read_u16 () *3.3/65535 temp1 = 27
        - ( voltage1 -0.706) /0.001721

        voltage2 = external_temp . read_u16 () *3.3/65535 temp2 =
        voltage2 *100
        print (" Temp1 ="+ str ( temp1 ) +" Temp2 ="+ str ( temp2 ) )
        utime . sleep (1)
```

Δοκιμάστε το πρόγραμμα και συγκρίνετε τα αποτελέσματα.

Η χρήση εξωτερικών ADC για τη λήψη πιο ακριβών μετρήσεων τάσης θα συζητηθεί αργότερα σε αυτόν τον οδηγό.

6. Διακοπές (Interrupts)

Οι διακοπές μας δίνουν έναν μηχανισμό που επιτρέπει να σταματήσουμε την εκτέλεση ενός προγράμματος για το χρόνο μιας ειδικής διαδικασίας και μετά να επιστρέψουμε στη στιγμή που το πρόγραμμα σταμάτησε και να το συνεχίσουμε (βλ. Εικ. 6.1). Οι διακοπές μπορούν να προκληθούν από εξωτερικά ή εσωτερικά σήματα.

Σχήμα 6.1: Πώς λειτουργούν οι διακοπές. Πηγή: https://makergram.com/blog/what-is-i nterrupts/

Για την καλύτερη κατανόηση της ιδέας, οι διακοπές εκτελούνται αυτόματα σε πολλές στιγμές της ζωής, π.χ. όταν διαβάζουμε ένα βιβλίο και χτυπάει το τηλέφωνο, σταματάμε να διαβάζουμε το βιβλίο για να απαντήσουμε στο τηλέφωνο και αφού τελειώσει η συζήτηση, επιστρέφουμε στην ανάγνωση του βιβλίου. Ακριβώς η ίδια ιδέα εμφανίζεται και στους μικροελεγκτές. Οι εξωτερικές διακοπές μπορούν να ενεργοποιηθούν από τα ακόλουθα σήματα στην επιλεγμένη ακίδα GPIO:

- Pin.IRQ_RISING - όταν το σήμα αλλάζει από χαμηλό σε υψηλό.
- Pin.IRQ_FALLING - όταν το σήμα αλλάζει από υψηλό σε χαμηλό.

6.1 Παράδειγμα 6: Ηχητικά σήματα στα φανάρια

Για να κατανοήσουμε καλύτερα την ιδέα της διακοπής, θα φτιάξουμε ένα φανάρι πεζών με ηχητικό σήμα για τυφλούς ή άτομα με προβλήματα όρασης όταν πατηθεί ένα κουμπί. Για το σκοπό αυτό, θα χρησιμοποιήσουμε 3 λυχνίες LED για να σηματοδοτήσουμε το φανάρι, έναν βομβητή με γεννήτρια ήχου και ένα κουμπί που θα ενεργοποιεί το ηχητικό σήμα για τουλάχιστον μία πλήρη ακολουθία φώτων.

Αρχικά, συνδέστε το σύστημα σύμφωνα με το Σχήμα 6.2.

GP17

GP16

GP10; GP11; GP12

Σχήμα 6.2: Σύνδεση του ηλεκτρονικού κυκλώματος από το παράδειγμα 6.

Στη συνέχεια, ανοίξτε το πρόγραμμα επεξεργασίας Thonny και ακολουθήστε τα εξής βήματα:

1. Αρχικά, προσθέστε τις απαραίτητες βιβλιοθήκες: machine και utime και διαμορφώστε τις ακίδες στις οποίες συνδέονται τα LED ως έξοδοι:

```
import machine
import utime

led_red = machine . Pin (10 , machine . Pin . OUT )
led_yellow = machine . Pin (11 , machine . Pin . OUT )
led_green = machine . Pin (12 , machine . Pin . OUT )
```

1
2
3
4
5
6

2. Στη συνέχεια, διαμορφώστε τον ακροδέκτη GP16 στον οποίο είναι συνδεδεμένος ο βομβητής ως ακροδέκτης για τη δημιουργία σήματος PWM και ρυθμίστε τη συχνότητα σήματος PWM στα 1000 Hz.

```
import machine
import utime

led_red = machine . Pin (10 , machine . Pin . OUT )
led_yellow = machine . Pin (11 , machine . Pin . OUT )

led_green = machine . Pin (12 , machine . Pin . OUT )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)
```

4. Στον κύριο βρόχο του προγράμματος, τα LED θα πρέπει να ανάβουν σύμφωνα με τη σειρά:
   - κόκκινη λυχνία LED αναμμένη, άλλα LED σβηστά
   - κόκκινες και κίτρινες λυχνίες LED αναμμένες, πράσινες σβηστές
   - πράσινο LED αναμμένο και άλλες λυχνίες LED σβηστές

```
import machine
import utime

led_red = machine . Pin (10 , machine . Pin . OUT )
led_yellow = machine . Pin (11 , machine . Pin . OUT )
led_green = machine . Pin (12 , machine . Pin . OUT )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

while True :
        led_red . value (1)
        led_yellow . value (0)
        led_green . value (0)

        led_red . value (1)
        led_yellow . value (1)
        led_green . value (0)

        led_red . value (0)
        led_yellow . value (0)
        led_green . value (1)
```

4. Στη συνέχεια, ας δημιουργήσουμε μια λογική μεταβλητή sound_active, η οποία θα έχει μία από τις δύο πιθανές τιμές: True ή False. Όταν έχει την τιμή True, θα εκκινήσουμε το βομβητή και θα δημιουργήσουμε ένα ηχητικό σήμα που θα ενημερώνει εάν τα φώτα είναι κόκκινα ή πράσινα. Επιπλέον, ας δημιουργήσουμε μια συνάρτηση make_sound(duration), στην οποία θα δημιουργήσουμε ένα ηχητικό σήμα. Για το σκοπό αυτό, θα χρησιμοποιήσουμε τη συνάρτηση

duty_u16(), η οποία θα ορίσει το καθήκον του σήματος PWM. Αυτή η συνάρτηση δέχεται τιμές από 0 έως 65535. Η αλλαγή λειτουργίας θα επηρεάσει την ένταση του ήχου που παράγεται από το βομβητή. Τα ηχητικά σήματα σε διαφορετικές χώρες μπορεί να διαφέρουν, αλλά κατά κανόνα θεωρείται ότι κατά τη διάρκεια ενός κόκκινου ή κίτρινου φωτός, το ηχητικό σήμα παράγεται σε μεγαλύτερα διαστήματα από ό,τι κατά τη διάρκεια ενός πράσινου φωτός. Για να ελέγξουμε τα διαστήματα μεταξύ των παραγόμενων ηχητικών σημάτων, θα δημιουργήσουμε μια μεταβλητή *duration* που δίνεται ως όρισμα στη συνάρτηση:

```
import machine
import utime

led_red = machine . Pin (10 , machine . Pin . OUT )
led_yellow = machine . Pin (11 , machine . Pin . OUT )
led_green = machine . Pin (12 , machine . Pin . OUT )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

sound_active=False

def make_sound(duration):
if sound_active :
buzzer . duty_u16 (16383) % turn on buzzer utime . sleep (1)
buzzer . duty_u16 (0) % turn off buzzer
utime . sleep ( duration )
else :
buzzer . duty_u16 (0)

# Because when the sound signal is turned on , #the total delay is 1 +
duration . So here we #add 1 to maintain the same delay .

utime . sleep ( duration +1)

while True :
led_red . value (1)
...
```

5. Τώρα ας προσθέσουμε την παραγωγή ήχου μετά από κάθε φανάρι. Η συσκευή θα πρέπει να παράγει ένα προειδοποιητικό ηχητικό σήμα 5 φορές. Για το σκοπό αυτό δημιουργείται ένας βρόχος for, ο οποίος θα εκτελεστεί 5 φορές και θα καλεί τη συνάρτηση make_sound 5 φορές. Στην περίπτωση του κόκκινου ή του κίτρινου, το διάστημα μεταξύ των ηχητικών σημάτων πρέπει να είναι 2 δευτερόλεπτα και στην περίπτωση του πράσινου 1 δευτερόλεπτο. Ως εκ τούτου, αυτές οι παράμετροι δόθηκαν στη συνάρτηση make_sound:

```
import machine
import utime

led_red = machine . Pin (10 , machine . Pin . OUT )
led_yellow = machine . Pin (11 , machine . Pin . OUT )
led_green = machine . Pin (12 , machine . Pin . OUT )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

sound_active = False

def make_sound ( duration ):
```

```
if sound_active :
buzzer . duty_u16 (16383) % turn on buzzer utime . sleep (1)
buzzer . duty_u16 (0) % turn off buzzer
utime . sleep ( duration )
else :
buzzer . duty_u16 (0)
utime . sleep ( duration +1)

while True :
led_red . value (1)
led_yellow . value (0)
led_green . value (0)
for i in range (0 ,5) :
make_sound (2)

led_red . value (1)
led_yellow . value (1)
led_green . value (0)
for i in range (0 ,5) :
make_sound (2)

led_red . value (0)
led_yellow . value (0)
led_green . value (1)
for i in range (0 ,5) :
make_sound (1)
```

6. Τώρα ας διαμορφώσουμε την ακίδα GP17, στην οποία είναι συνδεδεμένο το κουμπί, ως είσοδο. Επιπλέον, ας διαμορφώσουμε τη διακοπή. Για να το κάνετε αυτό, χρησιμοποιήστε τη συνάρτηση: *irq(trigger, handler)*. Το πρώτο όρισμα καθορίζει σε ποιον τύπο σήματος είναι ευαίσθητη η διακοπή, π.χ. μια ακμή που πέφτει ή ανέρχεται. Σε περίπτωση ανερχόμενης ακμής, η διακοπή θα ενεργοποιηθεί όταν πατηθεί το κουμπί και σε περίπτωση πτώσης, όταν απελευθερωθεί το κουμπί. Στο παράδειγμά μας, θα χρησιμοποιήσουμε μια ανερχόμενη άκρη. Το δεύτερο όρισμα είναι το όνομα της συνάρτησης που δημιουργήσαμε, η οποία θα εκτελεστεί όταν συμβεί μια διακοπή. Θα δημιουργήσουμε μια συνάρτηση που ονομάζεται sound_for_blind στο επόμενο βήμα:

```
import machine
import utime

led_red = machine . Pin (10 , machine . Pin . OUT )
led_yellow = machine . Pin (11 , machine . Pin . OUT )
led_green = machine . Pin (12 , machine . Pin . OUT )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

sound_active = False

def make_sound ( duration ):
if sound_active :
buzzer . duty_u16 (16383) % turn on buzzer utime . sleep (1)
buzzer . duty_u16 (0) % turn off buzzer
utime . sleep ( duration )
else :
buzzer . duty_u16 (0)
```

```
21 utime . sleep ( duration +1)
22
23 button = machine.Pin(17, machine.Pin.IN, machine.Pin.PULL_DOWN)
24 button.irq(trigger=machine.Pin.IRQ_RISING, handler=sound_for_blind)
25
26 while True :
27 led_red . value (1)
28 ...
```

7. Τώρα ας δημιουργήσουμε μια συνάρτηση που θα καλείται όταν συμβεί μια διακοπή. Μέσα σε αυτή τη συνάρτηση θα αλλάξουμε την τιμή της μεταβλητής sound_active σε True. Δεδομένου ότι αυτή είναι μια μεταβλητή που ορίζεται εκτός της συνάρτησης, θα πρέπει να καλέσουμε την εντολή global variable_name που ενημερώνει ότι αυτή η μεταβλητή δημιουργείται εκτός της συνάρτησης:

```
import machine
import utime

led_red = machine . Pin (10 , machine . Pin . OUT )
led_yellow = machine . Pin (11 , machine . Pin . OUT )
led_green = machine . Pin (12 , machine . Pin . OUT )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

sound_active = False

def make_sound ( duration ):
        if sound_active :
                buzzer . duty_u16 (16383) % turn on buzzer
                utime . sleep (1)
                buzzer . duty_u16 (0) % turn off buzzer
                utime . sleep ( duration )
        else :
                buzzer . duty_u16 (0)
                utime . sleep ( duration +1)

 def sound_for_blind(pin):
        global sound_active
        sound_active = True
        print (" Sound active : "+ str ( sound_active ))

button = machine . Pin (17 , machine . Pin .IN , machine . Pin . ,→
     PULL_DOWN )
button . irq ( trigger = machine . Pin . IRQ_RISING , handler = ,→
     sound_for_blind )

while True :
        led_red . value (1)
        ...
```

```
1
2
```

Τώρα, όταν ο χρήστης πατήσει το κουμπί, θα κληθεί η συνάρτηση sound_for_blind, ανεξάρτητα από το ποια γραμμή κώδικα στον βρόχο while θα εκτελέσει το πρόγραμμα.

8. Το τελευταίο βήμα είναι να απενεργοποιήσετε το ηχητικό σήμα μετά από τουλάχιστον μία πλήρη σειρά εναλλαγής φωτός. Για να γίνει αυτό, ας δημιουργήσουμε μια μεταβλητή counts που θα μετράει πόσες φορές έχει εκτελεστεί η πλήρης ακολουθία μετά τη διακοπή. Στη συνέχεια, στο κύριο πρόγραμμα, θα ελέγξουμε εάν οι μετρήσεις είναι μεγαλύτερες ή ίσες με 1, εάν ναι, απενεργοποιούμε το ηχητικό σήμα θέτοντας τη μεταβλητή sound_active σε False. Εάν το ηχητικό σήμα είναι ενεργοποιημένο, τότε με κάθε επανάληψη του βρόχου while θα αυξήσουμε την τιμή της μεταβλητής counts κατά 1:

```
import machine
import utime

led_red = machine . Pin (10 , machine . Pin . OUT )
led_yellow = machine . Pin (11 , machine . Pin . OUT )
led_green = machine . Pin (12 , machine . Pin . OUT )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

sound_active = False

def make_sound ( duration ):
if sound_active :
buzzer . duty_u16 (16383)
utime . sleep (1)
buzzer . duty_u16 (0)
utime . sleep ( duration )
```

```
else :
buzzer . duty_u16 (0)
utime . sleep ( duration +1)

def sound_for_blind ( pin ):
global sound_active
sound_active = True
print (" Sound active : "+ str ( sound_active ))

button = machine . Pin (17 , machine . Pin .IN , machine . Pin . ,→
        PULL_DOWN )
button . irq ( trigger = machine . Pin . IRQ_RISING , handler = ,→
        sound_for_blind )

counts=0

while True :
#How many times, the full sequence has been executed
if counts >=1:
sound_active = False
counts =0
if sound_active :
counts =+1

led_red . value (1)
led_yellow . value (0)
led_green . value (0)
for i in range (0 ,5) :
make_sound (2)

led_red . value (1)
led_yellow . value (1)
led_green . value (0)
for i in range (0 ,5) :
make_sound (2)

led_red . value (0)
led_yellow . value (0)
led_green . value (1)
for i in range (0 ,5) :
make_sound (1)
```

Το πρόγραμμα είναι τώρα έτοιμο, μπορείτε να το δοκιμάσετε.

7. Αισθητήρες (Sensors)

Οι αισθητήρες χρησιμοποιούνται για τη μέτρηση διαφόρων φυσικών μεγεθών, όπως

θερμοκρασία, πίεση, ένταση ακτινοβολίας UV, επιτάχυνση, ανίχνευση ήχου, συγκέντρωση αερίου κ.λπ. Οι αισθητήρες μπορούν να επιστρέψουν αναλογικές τιμές (π.χ. αισθητήρας θερμοκρασίας που συζητείται στο κεφάλαιο 5) ή ψηφιακές τιμές. Οι ψηφιακές τιμές μπορεί να είναι είτε 0-1 (π.χ. αισθητήρας κίνησης, αισθητήρας ανίχνευσης ήχου) είτε μπορεί να είναι τιμές από ένα δεδομένο εύρος και, στη συνέχεια, οι σειριακές διεπαφές επικοινωνίας (δίαυλοι) όπως I2C, SPI, UART ή 1−wire χρησιμοποιούνται για τη μεταφορά δεδομένων. Η ψηφιακή μετάδοση έχει ένα πλεονέκτημα έναντι της αναλογικής μετάδοσης, καθώς είναι πολύ λιγότερο ευαίσθητη σε εξωτερικές παρεμβολές και θόρυβο.

Η σειριακή διεπαφή επικοινωνίας αποτελείται από μια ομάδα γραμμών, οι οποίες χρησιμοποιούνται για την αποστολή δεδομένων μεταξύ συνδεδεμένων συσκευών. Οι σειριακές διεπαφές μπορούν να χωριστούν σε δύο ομάδες: σύγχρονες και ασύγχρονες. Η πρώτη ομάδα χρησιμοποιεί πρόσθετη γραμμή σειριακού ρολογιού, η οποία συγχρονίζει όλες τις συσκευές που είναι συνδεδεμένες στη διεπαφή επικοινωνίας. Οι πιο δημοφιλείς σύγχρονοι δίαυλοι είναι SPI (Serial Peripheral Interface) και I2C (Inter-Integrated Circuit), το οποίο επίσης επισημαίνεται ως I2C, IIC ή TW I (Two Wire Interface-Διασύνδεση δύο καλωδίων). Οι πιο δημοφιλείς διεπαφές ασύγχρονης σειριακής επικοινωνίας είναι το UART (Universal asynchronous receiver-transmitter) και το 1-wire. Σε αυτό το κεφάλαιο, θα συζητήσουμε έναν αισθητήρα ανά τύπο ψηφιακής επικοινωνίας. Άλλοι αισθητήρες χρησιμοποιούνται με τους περισσότερους τρόπους παρόμοια με αυτούς που συζητούνται παρακάτω.

7.1 Παράδειγμα 7: αισθητήρας στάθμευσης (parking sensor)

Οι αισθητήρες στάθμευσης εκπέμπουν ήχο όταν η απόσταση μεταξύ αυτοκινήτου και εμποδίου είναι μικρή, π.χ. λιγότερο από 1 μέτρο όταν παρκάρετε στην όπισθεν. Η συχνότητα του εκπεμπόμενου ήχου αυξάνεται όσο πιο κοντά είναι η απόσταση από το εμπόδιο. Για να δημιουργήσουμε έναν αισθητήρα στάθμευσης, θα χρησιμοποιήσουμε τον αισθητήρα απόστασης υπερήχων HC-SR04 και έναν βομβητή. Αρχικά, συνδέστε το σύστημα σύμφωνα με το Σχ. 7.1.

Τώρα μεταβείτε στο πρόγραμμα επεξεργασίας Thonny και ξεκινήστε να δημιουργείτε ένα πρόγραμμα που θα διαβάζει πρώτα την απόσταση από ένα εμπόδιο χρησιμοποιώντας έναν αισθητήρα απόστασης υπερήχων και στη συνέχεια θα παράγει έναν προειδοποιητικό ήχο εάν το εμπόδιο είναι πιο κοντά από 1 m. Για να το κάνετε αυτό, ακολουθήστε αυτά τα βήματα:

1. Αρχικά, ας προσθέσουμε τις απαραίτητες βιβλιοθήκες, δηλαδή *machine* και *utime*, και ας διαμορφώσουμε τις ακίδες: GP16 (buzzer) ως PWM, GP18 (trigger) ως έξοδο και GP19 (echo) ως είσοδο. Επιπλέον, ας ορίσουμε τη συχνότητα παραγωγής σήματος PWM στα 1000 Hz.

VBUS

GP16

Σχήμα 7.1Συνδέοντας το ηλεκτρονικό κύκλωμα του παραδείγματος 7.

```
import machine
import utime

trigger = machine . Pin (18 , machine . Pin . OUT )
echo = machine . Pin (19 , machine . Pin . IN )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)
```

```
1
2
3
4
5
6
7
8
```

2. Στη συνέχεια, στον κύριο βρόχο, ας προσθέσουμε ένα τμήμα κώδικα για να μετρήσουμε την απόσταση από τον αισθητήρα απόστασης υπερήχων. Σύμφωνα με την τεκμηρίωση για τον αισθητήρα HC-SR04 (βλ. Εικ. 7.2), ρυθμίστε πρώτα το χαμηλό σήμα στο trigger για μικρό χρονικό διάστημα, π.χ. 2μs. Στη συνέχεια, ρυθμίστε το υψηλό σήμα για 10 μs. Για να δημιουργήσετε καθυστερήσεις σε μικροδευτερόλεπτα, χρησιμοποιήστε τη συνάρτηση: utime.sleep_us(). Στο επόμενο βήμα, ρυθμίστε το χαμηλό σήμα στο trigger.

Σχήμα 7.2: Αρχή μέτρησης απόστασης στον αισθητήρα απόστασης υπερήχων HC-SR04. Πηγή: https://www.electronicoscaldas.com/datasheet/HC-SR04.pdf.

```
import machine
import utime

trigger = machine . Pin (18 , machine . Pin . OUT )
echo = machine . Pin (19 , machine . Pin . IN )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

while True :
        trigger . value (0)
        utime . sleep_us (2)
        trigger . value (1)
        utime . sleep_us (10)
        trigger . value (0)
```

3. Τώρα πρέπει να μετρήσουμε πόσο κράτησε το υψηλό σήμα στον ακροδέκτη ηχούς, επειδή η διάρκεια του σήματος στον ακροδέκτη ηχούς σχετίζεται με την απόσταση. Για να το κάνουμε αυτό, θα χρησιμοποιήσουμε τη συνάρτηση utime.ticks_us(), η οποία μετρά πόσος χρόνος έχει περάσει σε μs από την έναρξη του προγράμματος. Αρχικά, θα δημιουργήσουμε έναν βρόχο while που θα εκτελείται όσο το σήμα είναι χαμηλό. Μέσα, θα τοποθετήσουμε τη συνάρτηση tick_us(). Με αυτόν τον τρόπο, θα λάβουμε πληροφορίες για το πότε το σήμα ήταν χαμηλό για τελευταία φορά.

```
import machine
import utime

trigger = machine . Pin (18 , machine . Pin . OUT )
echo = machine . Pin (19 , machine . Pin . IN )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

while True :
        trigger . value (0)
        utime . sleep_us (2)
        trigger . value (1)
        utime . sleep_us (10)
        trigger . value (0)

        while echo . value () ==0:
                signal_off = utime . ticks_us ()
```

4. Ομοίως, θα μετρήσουμε πότε το σήμα ήταν τελευταία φορά ψηλά στην ακίδα ηχούς (echo pin):

```
import machine
import utime

trigger = machine . Pin (18 , machine . Pin . OUT ) echo =
machine . Pin (19 , machine . Pin . IN )

buzzer = machine . PWM ( machine . Pin (16) )

buzzer . freq (1000)

while True :
trigger . value (0)
utime . sleep_us (2)
trigger . value (1)
utime . sleep_us (10)
trigger . value (0)

```

```
while echo . value () ==0:
signal_off = utime . ticks_us ()

while echo . value () ==1:
signal_on = utime . ticks_us ()
```

5. Η διαφορά μεταξύ του χρόνου της τελευταίας εμφάνισης του υψηλού και του χαμηλού σήματος είναι η διάρκεια του υψηλού παλμού στην ακίδα ηχούς. Πώς μεταφράζουμε τη διάρκεια του παλμού σε απόσταση; Κοιτάξτε το Σχ. 7.3 που δείχνει την ιδέα της μέτρησης της απόστασης με έναν αισθητήρα απόστασης υπερήχων.

Σχήμα 7.3: Αρχή μέτρησης της απόστασης με χρήση αισθητήρα υπερήχων. Πηγή: https://www.researchgate.net/figure/A-block-diagram-of-Ultrasonic-se nsor-working-principles_fig5_344385811.

Στην αρχή εκπέμπεται ένα ηχητικό κύμα, το οποίο ανακλάται από το αντικείμενο και επιστρέφει στον αισθητήρα. Επομένως, στον μετρούμενο χρόνο t, το κύμα διανύει τη διπλάσια απόσταση μεταξύ του αισθητήρα και του αντικειμένου και κινείται με ταχύτητα περίπου 340 m/s (η ταχύτητα του ήχου στον αέρα). Επομένως, μπορούμε να γράψουμε την εξίσωση για την ταχύτητα:

$$v = \frac{2d}{t} \quad (7.1) \qquad d = v \cdot \frac{t}{2} = 0.034 \frac{cm}{\mu s} \cdot \frac{t}{2} \approx \frac{t}{58} \quad (7.2)$$

Ως εκ τούτου, η λαμβανόμενη διάρκεια παλμού θα πρέπει να διαιρεθεί με το 58 για να ληφθεί η απόσταση σε εκατοστά:

```
import machine
import utime

trigger = machine . Pin (18 , machine . Pin . OUT )
echo = machine . Pin (19 , machine . Pin . IN )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

while True :
        trigger . value (0)
        utime . sleep_us (2)
        trigger . value (1)
        utime . sleep_us (10)
        trigger . value (0)

        while echo . value () ==0:
                signal_off = utime . ticks_us ()

        while echo . value () ==1:
                signal_on = utime . ticks_us ()

        diff = signal_on - signal_off
        distance = diff /58.0
        print (" Distance ="+ str ( distance ))
```

6. Το τελευταίο βήμα είναι να δημιουργήσετε έναν προειδοποιητικό ήχο στον βομβητή. Για να γίνει αυτό, ελέγχουμε αν η απόσταση είναι μικρότερη από 100 cm. Αν ναι, παράγουμε ένα σήμα PWM με τον επιλεγμένο κύκλο λειτουργίας στον βομβητή, εκπέμποντας έτσι ένα ηχητικό κύμα. Η εισαγόμενη τιμή θα μεταφραστεί στην ένταση του ήχου που εκπέμπεται. Στη συνέχεια ορίζουμε καθυστέρηση ανάλογη της απόστασης. Σε αυτή την περίπτωση, η απόσταση διαιρέθηκε με το

50, έτσι ώστε ο προειδοποιητικός ήχος να μην είναι πολύ μεγάλος. Η τιμή του 50 επιλέχθηκε εμπειρικά, δεν έχει πολύ φυσικό νόημα. Στη συνέχεια σβήνουμε τον βομβητή θέτοντας το fill στο 0 και περιμένουμε επίσης χρόνο ανάλογο της απόστασης.

```
import machine
import utime

trigger = machine . Pin (18 , machine . Pin . OUT )
echo = machine . Pin (19 , machine . Pin . IN )

buzzer = machine . PWM ( machine . Pin (16) )
buzzer . freq (1000)

while True :
trigger . value (0)
utime . sleep_us (2)
trigger . value (1)
utime . sleep_us (10)
trigger . value (0)


while echo . value () ==0:
signal_off = utime . ticks_us ()

while echo . value () ==1:
signal_on = utime . ticks_us ()

diff = signal_on - signal_off
distance = diff /58.0
print (" Distance ="+ str ( distance ))

if distance <100:
buzzer . duty_u16 (16383)
utime . sleep ( distance /50)
buzzer . duty_u16 (0)
utime . sleep ( distance /50)
```

Τώρα το πρόγραμμα είναι έτοιμο και μπορείτε να το δοκιμάσετε.

7.2 Παράδειγμα 8: GPS

Σε αυτό το παράδειγμα, θα επικεντρωθούμε στον ασύγχρονο δίαυλο επικοινωνίας UART, ο οποίος θα χρησιμοποιηθεί για τη λήψη δεδομένων από τη μονάδα GPS Waveshare Neo-6m/7m. Το Raspberry Pi Pico έχει 2 θύρες UART: UART0 και UART1 (βλ. Εικ. 1.1). Θα συνδέσουμε τη μονάδα GPS στο UART1 (ακίδες GP4 και GP5). Απλώς θυμηθείτε ότι οι γραμμές προς Rx (Receiver) και Tx (Transmitter) πρέπει να συνδέονται σταυρωτά, δηλαδή η γραμμή Rx από τη μονάδα GPS στο Tx του Raspberry Pi Pico (GP4) και η γραμμή Tx από το GPS στο Rx του Raspberry Pi Pico (GP5). Συνδέουμε το τροφοδοτικό της μονάδας GPS στα 3,3V. Συνδέστε λοιπόν το σύστημα όπως φαίνεται στην Εικ. 7.4.

Σχήμα 7.4: Σύνδεση του ηλεκτρονικού κυκλώματος από το παράδειγμα 8.

Τώρα ανοίξτε το πρόγραμμα επεξεργασίας Thonny και ας προσπαθήσουμε να γράψουμε κώδικα που θα λαμβάνει πλαίσια δεδομένων από τη μονάδα GPS:

1. Αρχικά, πρέπει να προσθέσετε τις βιβλιοθήκες *machine* και *utime* και να διαμορφώσετε το UART. Για το σκοπό αυτό, χρησιμοποιήστε τη συνάρτηση machine.UART(), η οποία λαμβάνει ως πρώτο όρισμα τον αριθμό θύρας UART, δηλαδή την τιμή 0 ή 1. Συνδέσαμε τη μονάδα GPS στο UART1, οπότε εισάγουμε 1. Στη συνέχεια, πρέπει να δώσετε τον ρυθμό μετάδοσης και τις ακίδες στις οποίες είναι συνδεδεμένη η μονάδα GPS. Εάν κοιτάξετε το Σχ. 1.1, το UART1 είναι διαθέσιμο σε δύο σετ ακίδων: GP4 και GP5 ή GP8 και GP9. Επομένως, πρέπει να υποδείξετε ποιες ακίδες χρησιμοποιούνται για τη σύνδεση της μονάδας GPS:

```
import machine
import utime

uart = machine . UART (1 , baudrate =9600 , tx = machine . Pin (4) , ,→ rx =
    machine . Pin (5) )
```

```
1
2
3
4
```

2. Στη συνέχεια στον κύριο βρόχο ελέγχουμε αν υπάρχουν διαθέσιμα δεδομένα στο buffer UART με τη συνάρτηση any() και αν υπάρχει, διαβάζουμε τη γραμμή με τη συνάρτηση readline(). Εμφανίζουμε τη γραμμή ανάγνωσης στο τερματικό:

```
import machine
import utime

uart = machine . UART (1 , baudrate =9600 , tx = machine . Pin (4) , ,→ rx =
    machine . Pin (5) )

while True :
    if uart . any () :
        line = uart . readline ()
        print ( line )
    utime . sleep (1)
```

```
1
2
3
4


5
6
7
8
9
10
```

Όταν εκτελείτε το πρόγραμμα, στην αρχή θα έχετε το αποτέλεσμα όπως φαίνεται

στο Σχήμα 7.5.

Σχήμα 7.5: Παράδειγμα εξόδου από μονάδα GPS χωρίς επιδιόρθωση στους δορυφόρους.

Αυτό το αποτέλεσμα σημαίνει ότι η μονάδα GPS δεν έχει ακόμη διορθωθεί στους δορυφόρους και στέλνει κενούς χαρακτήρες ανάμεσα στα κόμματα. Όταν διορθωθεί, θα εμφανιστούν δεδομένα ανάμεσα στα κόμματα, τα οποία θα μπορούμε να ερμηνεύσουμε (π.χ. όπως φαίνεται στην Εικ. 7.6).

Σχήμα 7.6: Παράδειγμα εξόδου από το GPS, το οποίο διορθώθηκε στους δορυφόρους. Πηγή: https: //www.waveshare.com/wiki/File:UART-GPS-NEO-6M-User-Manual-2.png.

Πώς να ερμηνεύσετε δεδομένα από το GPS; Τα δεδομένα που λαμβάνονται από το GPS καταγράφονται σύμφωνα με το πρωτόκολλο NMEA (National Marine Electronics Association), στο οποίο κάθε ακολουθία ξεκινά με ένα αναγνωριστικό, π.χ. GPGGA - Global Positioning System Fix Data - Δεδομένα επιδιόρθωσης του παγκόσμιου εντοπισμού θέσης. Για να διαβάσετε λοιπόν τα δεδομένα που σας ενδιαφέρουν, πρέπει να βρείτε το κατάλληλο αναγνωριστικό ακολουθίας, στο οποίο βρίσκονται οι πληροφορίες που αναζητάτε. Σε αυτή την περίπτωση, θέλουμε να διαβάσουμε τη θέση, επομένως μας ενδιαφέρει μόνο το πλαίσιο GPGGA. Τα υπόλοιπα πλαίσια συζητούνται εδώ: https://aprs.gids.nl/nmea/ . Το πλαίσιο GPPGA μοιάζει με το σχήμα 7.7.

Σχήμα 7.7: GPGGA ακολουθία. Πηγή: https://aprs.gids.nl/nmea/

3. Για να εξαγάγετε ένα πλαίσιο GPGGA, μετατρέψτε πρώτα το αναγνωσμένο κείμενο σε μορφή byte σε μια συμβολοσειρά με κωδικοποίηση utf-8. Για το σκοπό αυτό, χρησιμοποιείται η συνάρτηση decode(). Στη συνέχεια, εμφανίζουμε μόνο εκείνες τις γραμμές που ξεκινούν με $GPGGA. Για το σκοπό αυτό, χρησιμοποιούμε τη συνάρτηση startswith(), η οποία επιστρέφει True εάν η συμβολοσειρά ξεκινά με την επιλεγμένη λέξη που δίνεται σε αγκύλες:

```
import machine
import utime

uart = machine . UART (1 , baudrate =9600 , tx = machine . Pin (4) , ,→ rx =
    machine . Pin (5) )

while True :
    if uart . any () :
        line = uart . readline ()
        line = line . decode ('utf -8 ')
        if line . startswith ('$GPGGA ') :
            print ( line )
    utime . sleep (1)
```

4. Στη συνέχεια, πρέπει να χωρίσουμε τη συμβολοσειρά σε τμήματα (fragments).

Κάθε πληροφορία στη συμβολοσειρά χωρίζεται με κόμμα, επομένως θα χρησιμοποιήσουμε τη συνάρτηση split(), η οποία χωρίζει τη συμβολοσειρά σε θραύσματα σύμφωνα με τον χαρακτήρα διαχωρισμού που δίνεται στις αγκύλες.

```
import machine
import utime

uart = machine . UART (1 , baudrate =9600 , tx = machine . Pin (4) , ,→ rx =
    machine . Pin (5) )

while True :
    if uart . any () :
        line = uart . readline ()
        line = line . decode ('utf -8 ')
        if line . startswith ('$GPGGA ') :
            parts = line . split (',')
            latitude = parts [2]
            longitude = parts [4]
            print ( latitude +","+ longitude )
    utime . sleep (1)
```

Τώρα το πρόγραμμα είναι έτοιμο και μπορεί να δοκιμαστεί. Για να ελέγξουμε αν έχουμε διαβάσει τη σωστή τοποθεσία, μπορούμε να χρησιμοποιήσουμε το google maps. Στη συνέχεια, στο πεδίο αναζήτησης θα πρέπει να εισαγάγουμε το γεωγραφικό πλάτος και μήκος με τη μορφή π.χ.: 52 13.30093, 21 00.41831.

7.3 Παράδειγμα 9: Μετεωρολογικός σταθμός (Weather station)

Η σειριακή περιφερειακή διεπαφή αποτελείται από τέσσερις γραμμές:

• SPI SCK - σειριακό ρολόι, που επιτρέπει το συγχρονισμό συσκευών,

• SPI TX (παλαιότερα ονομαζόταν MOSI - Master Output Slave Input) - γραμμή, η οποία μεταδίδει bit από την κύρια συσκευή σε όλες τις υποτελείς συσκευές,

• SPI RX (παλαιότερα ονομαζόταν MISO - Master Input Slave Output) - γραμμή, η οποία μεταδίδει bits από slave συσκευές στην κύρια συσκευή,

• SPI CS (Chip Select) - γραμμή, η οποία ενεργοποιεί την επικοινωνία με επιλεγμένη εξαρτημένη συσκευή. Η εξαρτημένη συσκευή αρχίζει να ακούει και να αποκρίνεται όταν

οριστεί μια χαμηλή τιμή στη γραμμή CS της. Αυτή η γραμμή μερικές φορές ονομάζεται Slave Select και στη συνέχεια επισημαίνεται ως SS

> Για να επισημάνετε ότι η ενεργή κατάσταση είναι χαμηλή στο όνομα γραμμής, τοποθετείται συχνά μια γραμμή πάνω από το κείμενο, όπως SS ή CS.

Η σχηματική σύνδεση μεταξύ συσκευών (block diagram) φαίνεται στο Σχ. 7.8. Η κύρια συσκευή είναι μόνο μία και αυτή η συσκευή λαμβάνει πληροφορίες από slave συσκευές κατά παραγγελία. Η κύρια συσκευή παράγει το σήμα ρολογιού, το οποίο θα συγχρονίσει όλες τις συσκευές. Είναι δυνατή η σύνδεση πολλών βοηθητικών συσκευών (π.χ. αισθητήρων) στην κύρια (π.χ. πλακέτα Raspberry Pi Pico), αλλά όλες οι εξαρτημένες συσκευές πρέπει να έχουν ξεχωριστές γραμμές SS/CS. Αξίζει να σημειωθεί ότι το SPI είναι δίαυλος full-duplex. Δηλαδή, τα δεδομένα μπορούν να μεταδοθούν και να ληφθούν από τη συσκευή ταυτόχρονα.

Σχήμα 7.8: Σχηματική σύνδεση μεταξύ συσκευών που χρησιμοποιούν δίαυλο SPI. Πηγή: https: //forbot.pl/blog/kurs-stm32-f4-10-obsluga-spi-wyswietlacz-oled-id13475.

Η αποκλειστική βιβλιοθήκη για τη χρήση του διαύλου SPI ονομάζεται SPI, αλλά οι περισσότεροι αισθητήρες έχουν έτοιμες βιβλιοθήκες για το MicroPython, τις οποίες μπορείτε να βρείτε στο πρόγραμμα επεξεργασίας Thonny. Αυτό το παράδειγμα θα σας δείξει πώς χρησιμοποιούνται έτοιμες βιβλιοθήκες για αισθητήρες, οι οποίοι στέλνουν δεδομένα χρησιμοποιώντας τη διεπαφή SPI. Για παράδειγμα, θα φτιάξουμε έναν μετεωρολογικό σταθμό με βάση τον αισθητήρα BME280, ο οποίος επιτρέπει τη μέτρηση της θερμοκρασίας, της πίεσης και της υγρασίας. Πρώτα, πρέπει να εγκαταστήσετε το library. Στο Thonny, μπορούμε να αναζητήσουμε έτοιμες βιβλιοθήκες επιλέγοντας από το μενού Εργαλεία → Διαχείριση πακέτων (*Tools → Manage Packages)*. Στη συνέχεια, πρέπει να εισαγάγετε το όνομα του αισθητήρα για τον οποίο αναζητάτε βιβλιοθήκη. Εφόσον το BME280 στέλνει δεδομένα μέσω SPI και I2C, μπορείτε επιπλέον να προσθέσετε το όνομα του διαύλου κατά την αναζήτηση μιας βιβλιοθήκης, π.χ. όπως φαίνεται στο Σχ. 7.9. Στη συνέχεια, επιλέξτε μία από τις έτοιμες βιβλιοθήκες. Σε αυτό το παράδειγμα, θα χρησιμοποιήσουμε τη βιβλιοθήκη bme280-upy. Κάντε κλικ σε αυτό και στη συνέχεια πατήστε το κουμπί Εγκατάσταση (*Install)*. Στη συνέχεια, πρέπει να συνδέσετε τον αισθητήρα στο Raspberry Pi Pico, το οποίο έχει 2 ανεξάρτητους διαύλους SPI: SPI0 και SPI1 (βλ. Εικ. 1.1). Θα χρησιμοποιήσουμε το SPI0 διαθέσιμο στις ακίδες GP16-GP19.

Συνδέστε τον αισθητήρα σύμφωνα με το Σχ. 7.11. Κατά την εγκατάσταση της βιβλιοθήκης bme280-upy (βλ. Σχ. 7.10), παρέχονται βασικές πληροφορίες για τη βιβλιοθήκη μαζί με συνδέσμους προς το αποθετήριο και τη σελίδα PyPI.

Σχήμα 7.9:Αναζητώντας βιβλιοθήκη για τον αισθητήρα BME280.

Σχήμα 7.10: Πληροφορίες σχετικά με τη βιβλιοθήκη *bme280-upy* library.

Σχήμα 7.11: Σύνδεση του ηλεκτρονικού κυκλώματος για το παράδειγμα 9.

Ανοίξτε τη σελίδα PyPI και σημειώστε ότι συνήθως ο δημιουργός της βιβλιοθήκης παρέχει ένα παράδειγμα του τρόπου χρήσης της βιβλιοθήκης μαζί με άλλες απαραίτητες πληροφορίες.

Ας δημιουργήσουμε ένα πρόγραμμα που διαβάζει τη θερμοκρασία, την πίεση και την υγρασία από τον αισθητήρα, με βάση το παράδειγμα από τη σελίδα PyPI της βιβλιοθήκης bme280-upy. Για να το κάνετε αυτό, ακολουθήστε αυτά τα βήματα:

a. Αρχικά προσθέστε τις βιβλιοθήκες machine και bme280:

```
import machine
import bme280
```

b. Στη συνέχεια, πρέπει να διαμορφώσετε τον αισθητήρα προσδιορίζοντας ποιο δίαυλο θα χρησιμοποιήσετε για επικοινωνία και εάν είναι SPI, θα πρέπει επίσης να καθορίσετε τον ακροδέκτη CS που χρησιμοποιήθηκε:

```
import machine
import bme280

bme = bme280 . BME280 ( spiBus =0 , spiCS =17)
```

c. Στη συνέχεια, πρέπει να διαβάσετε τα δεδομένα από τον αισθητήρα, ο οποίος επιστρέφει τη θερμοκρασία σε βαθμούς Κελσίου. Στην περίπτωση της υγρασίας, πρέπει να πολλαπλασιάσετε το αποτέλεσμα επί 100 για να λάβετε ένα αποτέλεσμα σε ποσοστό και η πίεση πρέπει να διαιρεθεί με το 100 για να λάβετε ένα αποτέλεσμα σε hPa:

```
import machine
import bme280

bme = bme280 . BME280 ( spiBus =0 , spiCS =17)

while True :
        temperature , humidity , pressure = bme . readForced ( ,→ filter =
    bme280 . FILTER_2 , tempOversampling = ,→ bme280 . OVSMPL_4
            , humidityOversampling = bme280 ,→ . OVSMPL_4 ,
                pressureOversampling = bme280 .
      ,→ OVSMPL_4 )
    humidity = humidity *100
    pressure = pressure /100
     print ("T="+str ( temperature )+", humidity ="+ str ( ,→ humidity
                )+", pressure ="+ str ( pressure ))
```

Εκτελέστε το πρόγραμμα και δοκιμάστε πώς λειτουργεί.

7.4 Παράδειγμα 10: Μέτρηση ποιότητας αέρα εσωτερικού χώρου

Ο δίαυλος Inter-Integrated Circuit αποτελείται από δύο γραμμές:
• Το SDA (Serial Data Line) που είναι μια γραμμή δεδομένων, η οποία χρησιμοποιείται για την αποστολή δεδομένων μεταξύ κύριας και δευτερεύουσας συσκευής.
• Το SCL (Serial Clock Line). Και οι δύο γραμμές συνδέονται στο VCC με αντιστάσεις έλξης.

Σε αυτό το παράδειγμα, δεν θα εμβαθύνουμε στο πρωτόκολλο μετάδοσης I2C, αλλά θα χρησιμοποιήσουμε μια έτοιμη βιβλιοθήκη αισθητήρων. Σημειώστε στο Σχ. 1.1 ότι έχουμε δύο διαύλους I2C (που επισημαίνονται ως I2C0 και I2C1) που οδηγούνται σε πολλές ακίδες. Για παράδειγμα, θα δημιουργήσουμε ένα πρόγραμμα για την ανάγνωση των παραμέτρων ποιότητας του αέρα σε ένα δωμάτιο όπως: Δείκτης ποιότητας αέρα, συγκέντρωση CO2 (eC02) και συγκέντρωση συνολικών πτητικών οργανικών ενώσεων (TVOC). Για το σκοπό αυτό, θα χρησιμοποιήσουμε τον αισθητήρα ποιότητας αέρα DFROBOT ENS160. Ο αισθητήρας πρέπει να συνδεθεί στο Raspberry Pi σύμφωνα με τον πίνακα αρ. 7.1.

ENS160 sensor	Raspberry Pi Pico W
3V3	3V3
GND	GND
SCL	GP15
SDA	GP14

Πίνακας 7.1: Σύνδεση του αισθητήρα DFROBOT ENS160 στο Raspberry Pi Pico W.

Αυτή τη φορά, δεν θα βρούμε έτοιμη βιβλιοθήκη στον διαχειριστή πακέτων του επεξεργαστή Thonny. Τι να κάνετε σε μια τέτοια περίπτωση; Μπορείτε να αναζητήσετε μια έτοιμη βιβλιοθήκη στο Διαδίκτυο και να την κατεβάσετε, π.χ. από το github. Σε αυτό το παράδειγμα, θα χρησιμοποιήσουμε το αποθετήριο https://github.com/TimHanewich/Air-Quality-IoT/tree/master. Για να χρησιμοποιήσετε την έτοιμη βιβλιοθήκη που είναι κοινόχρηστη στο github, πρέπει

να κάνετε λήψη των πηγών της κάνοντας κλικ στο κουμπί Κώδικας (Code) και επιλέγοντας Λήψη ZIP (Download ZIP) όπως φαίνεται στο Σχ. 7.12.

Σχήμα 7.12: Λήψη της βιβλιοθήκης από το αποθετήριο github.

Στη συνέχεια, πρέπει να αποσυμπιέσετε το αρχείο zip στον φάκελο. Μεταβείτε στο Thonny και συνδεθείτε στον πίνακα Raspberry Pi Pico και, στη συνέχεια, επιλέξτε ΠΡΟΒΟΛΗ → Αρχεία (*VIEW → Files*) στο επάνω μέρος. Στη συνέχεια, στην αριστερή πλευρά θα έχουμε μια προεπισκόπηση των καταλόγων στον υπολογιστή και από κάτω το περιεχόμενο του πίνακα Raspberry Pi Pico. Αντιγράψτε τη βιβλιοθήκη ENS160 (src/ENS160.py) στο Raspberry Pi Pico όπως φαίνεται στην Εικ. 7.13.
Συνήθως, στο αποθετήριο που κατεβάσατε υπάρχει ένα παράδειγμα του τρόπου χρήσης της βιβλιοθήκης. Σε αυτήν την περίπτωση, θα βρούμε επίσης ένα παράδειγμα που ονομάζεται main.py αλλά θα περιέχει πολλές περισσότερες πληροφορίες από αυτές που χρειαζόμαστε (σύνδεση με τον αισθητήρα AHT21 και σύνδεση με WIFI). Ανοίξτε το αρχείο main.py και αφαιρέστε τα περιττά στοιχεία, τότε το πρόγραμμα θα πρέπει να μοιάζει με αυτό:

Σχήμα 7.13: Εγκατάσταση της ληφθείσας βιβλιοθήκης στην πλακέτα Raspberry Pi Pico.

```
import machine
import utime
import ENS160

# set up
print (" Setting up ENS160 ")
i2c = machine . I2C (1 , sda = machine . Pin (14) , scl = machine . Pin (15) )
ens = ENS160 . ENS160 ( i2c )
ens . reset ()
utime . sleep (0.5)
ens . operating_mode = 2
utime . sleep (2.0)

while True :
# take reading from ENS160
print (" Taking ENS160 measurements ... ")
aqi = ens . AQI
eco2 = ens . CO2
tvoc = ens . TVOC
print ("AQI : " + str ( aqi ) + ", ECO2 : " + str ( eco2 ) + ", TVOC : " + str ( tvoc ) )
utime . sleep (1)
```

Εκτελέστε το πρόγραμμα και δοκιμάστε το. Πώς να ερμηνεύσετε τα δεδομένα ανάγνωσης; Απλώς εξοικειωθείτε με τους πίνακες στην τεκμηρίωση του αισθητήρα (βλ. Σχ. 7.14, 7.15, 7.16).

Σχήμα 7.14: AQI Reference.

Σχήμα 7.15: eC02 Concentration Reference.

Σχήμα 7.16: TVOC Reference.

7.5 Παράδειγμα 11: Μέτρηση θερμοκρασίας με χρήση μετατροπέα A/D

Σε αυτό το παράδειγμα θα μάθουμε πώς να διαβάζουμε την τιμή τάσης από έναν μετατροπέα αναλογικού σε ψηφιακό (ADC). Μέχρι τώρα χρησιμοποιούσαμε τον ενσωματωμένο μετατροπέα αναλογικού σε ψηφιακό, ο οποίος είναι 12 bit (ανάλυση περίπου 800 μV). Αν θέλουμε καλύτερη ακρίβεια, μπορούμε να χρησιμοποιήσουμε έναν εξωτερικό μετατροπέα, π.χ. ADS1115 (16 bit - ανάλυση περίπου 76 μV). Σε αυτό το παράδειγμα θα συνδέσουμε τον αισθητήρα θερμοκρασίας LM35 στον μετατροπέα ADS1115 σύμφωνα με το Σχ. 7.17.

Σχήμα 7.17: Σύνδεση του ηλεκτρονικού κυκλώματος του παραδείγματος 11.

Στην περίπτωση του μετατροπέα ADS1115, δεν θα χρησιμοποιήσουμε έτοιμη βιβλιοθήκη που μπορεί να βρεθεί στο Github, αλλά θα γράψουμε τη δική μας βιβλιοθήκη. Γιατί; Τα δύο τελευταία παραδείγματα έδειξαν πώς να χρησιμοποιούμε τους διαύλους SPI και I2C με έτοιμες βιβλιοθήκες και δεν εμβαθύναμε στο πρωτόκολλο μετάδοσης δεδομένων. Τώρα δεν θα χρησιμοποιήσουμε έτοιμες βιβλιοθήκες για να δείξουμε πώς να επικοινωνούμε με έναν μετατροπέα ή έναν άλλο αισθητήρα με γενική βιβλιοθήκη διεπαφής επικοινωνίας (η οποία είναι διαθέσιμη εκτός συσκευασίας στο Micropython για το Raspberry Pi Pico). Αρχικά, θα πρέπει να ξεκινήσετε βρίσκοντας ένα φύλλο δεδομένων για τον μετατροπέα ADS1115 στο Διαδίκτυο (https://www.ti.com/lit/ds/syml ink/ads1115.pdf).

Στη συνέχεια, πρέπει να βρείτε τρία πράγματα στην τεκμηρίωση:

- πρωτόκολλο γραφής
- πρωτόκολλο ανάγνωσης
- καταχώρηση διευθύνσεων μαζί με τις τιμές που πρέπει να τους αποσταλούν για να ορίσετε τις δεδομένες παραμέτρους.

Αυτό είναι ένα γενικό διάγραμμα του τι πρέπει να κάνετε, και πιο αναλυτικά:

α. Σαρώστε ποιες συσκευές είναι συνδεδεμένες στο δίαυλο I2C για να βρείτε τη διεύθυνση του μετατροπέα A/D. Για να το κάνετε αυτό, διαμορφώστε πρώτα τους διαύλους I2C χρησιμοποιώντας τη συνάρτηση machine.I2C(), η οποία παίρνει τον αριθμό διαύλου I2C ως πρώτο όρισμα (0 για το I2C0 ή 1 για το I2C1 - οι αριθμοί είναι στο Σχ. 1.1. Στην περίπτωσή μας, θα είναι I2C1). Το δεύτερο όρισμα είναι η συχνότητα ρολογιού του διαύλου I2C. Το τρίτο και το τέταρτο όρισμα είναι οι ακίδες που χρησιμοποιήθηκαν για τη σύνδεση SDA και SCL.

```
import machine
import utime

i2c = machine . I2C (1 , freq =400000 , scl = machine . Pin (15) , ,→ sda =
    machine . Pin (14) )
```

```
1
2
3
4
```

b. Στη συνέχεια, χρησιμοποιούμε τη συνάρτηση scan(), η οποία επιστρέφει τις

διευθύνσεις των συσκευών που είναι συνδεδεμένες στο δίαυλο I2C με τη μορφή λίστας. Στη συνέχεια διαβάζουμε τις τιμές από τη λίστα χρησιμοποιώντας έναν βρόχο for. Για να εμφανίσουμε τα δεδομένα σε δεκαεξαδικό κώδικα, όπως συνήθως γράφονται οι διευθύνσεις, χρησιμοποιούμε τη συνάρτηση hex, η οποία μετατρέπει τις δεκαδικές τιμές σε δεκαεξαδικές:

```
import machine
import utime

i2c = machine . I2C (1 , freq =400000 , scl = machine . Pin (15) , ,→ sda =
    machine . Pin (14) )
devices = i2c . scan ()
for device in devices :
        print (hex ( device ))
```

```
1
2
3
4

5
6
7
```

c. Εκτελέστε τον κώδικα και, στη συνέχεια, αποθηκεύστε τη διεύθυνση στη μεταβλητή ADS1115_I2C_ADDRESS:

```
import machine
import utime

i2c = machine . I2C (1 , freq =400000 , scl = machine . Pin (15) , ,→ sda =
    machine . Pin (14) )
devices = i2c . scan ()
for device in devices :
    print (hex ( device ))

ADS1115_I2C_ADDRESS = 0 x48
```

```
1
2
3
4

5
6
7
8
9
```

Συμβουλή 7.5.1

Μπορείτε να ελέγξετε τη ληφθείσα διεύθυνση του μετατροπέα A/D ADS1115 με τον πίνακα 7.2 του φύλλου δεδομένων ADS1115. Η δεκαεξαδική τιμή 0x48 είναι 0b1001000 σε δυαδική μορφή. Μπορείτε

να χρησιμοποιήσετε:

print(bin(device))

στο πρόγραμμά σας.

d. Στη συνέχεια, πρέπει να βρείτε τις διευθύνσεις των καταχωρητών μέσα στο ADC, όπου αποθηκεύονται οι ρυθμίσεις παραμέτρων και οι τιμές που μετρήθηκαν από το ADC. Οι διευθύνσεις βρίσκονται στην τεκμηρίωση του αισθητήρα στον Πίνακα 6, όπως φαίνεται στο Σχ. 7.18. Σημειώστε ότι σε αυτόν τον πίνακα, ο καταχωρητής δείκτη διευθύνσεων περιγράφεται με μεμονωμένα bit. Το ένα byte έχει 8 bit [7:0]. Τα bit από το 7 (παλαιότερο) έως το 2 είναι δεσμευμένα και θα πρέπει πάντα να γράφετε 0 σε αυτά. Τα bits από το 1 έως το 0 είναι σημαντικά και έχουμε 4 διευθύνσεις, όπου χρησιμοποιούμε μόνο τις διευθύνσεις '00' και '01'.

   Περάστε τις διευθύνσεις σε μεταβλητές του προγράμματος.

Σχήμα 7.18: Πίνακας 6 από το φύλλο δεδομένων. Πηγή: https://www.ti.com/lit/ds/symlink/ ads1115.pdf

```
import machine
import utime

i2c = machine . I2C (1 , freq =400000 , scl = machine . Pin (15) , ,→ sda =
      machine . Pin (14) )
devices = i2c . scan ()
for device in devices :
                print (hex ( device ))

ADS1115_I2C_ADDRESS = 0 x48
ADS1115_CONVERSION_REG = 0 x00
ADS1115_CONFIG_REG = 0 x01
```

e. Στη συνέχεια, πρέπει να αναζητήσετε πληροφορίες σχετικά με το ποιες παράμετροι του ADC μπορούν να ρυθμιστούν και ποιες τιμές πρέπει να αποσταλούν στον καταχωρητή διαμόρφωσης για το σκοπό αυτό. Όλα αυτά

τα δεδομένα βρίσκονται στον Πίνακα 8 (βλ. Σχ. 7.19 και 7.20)) και οι χαρακτηρισμοί των γραμμάτων h υποδεικνύουν τιμές γραμμένες σε δεκαεξαδικό κώδικα, π.χ. 1h=0x01, το οποίο πρέπει να οριστεί σε μεμονωμένα bit του καταχωρητή διαμόρφωσης 16-bit. Σε αυτό το παράδειγμα, θα εκτελέσουμε μία μόνο μέτρηση από το κανάλι 0 στην περιοχή ±4,096V. Δεν θα χρησιμοποιήσουμε συγκριτικό. Οι απαραίτητες τιμές έχουν καταγραφεί για τις μεταβλητές:

```
import machine
import utime

i2c = machine . I2C (1 , freq =400000 , scl = machine . Pin (15) , ,→ sda =
    machine . Pin (14) )
devices = i2c . scan ()
for device in devices :
print (hex ( device ))

ADS1115_I2C_ADDRESS = 0 x48
ADS1115_CONVERSION_REG = 0 x00
ADS1115_CONFIG_REG = 0 x01

ADS1115_CONFIG_OS_SINGLE = 0 x8000 # Start a single ,→ conversion
ADS1115_CONFIG_MUX_AIN0 = 0 x4000 # AIN0 Chain ( Single - ,→ ended
    channel )
        ADS1115_CONFIG_GAIN = 0 x0200 # Gain amplifier + -4.096 V

ADS1115_CONFIG_MODE_SINGLE = 0 x0100 # Single - shot mode
ADS1115_CONFIG_DR_128SPS = 0 x0080 # Data rate = 128 ,→ SPS
ADS1115_CONFIG_COMP_DISABLED = 0 x0003 # disable ,→
    comparator
```

Σχήμα 7.19: Περιγραφή πεδίου καταχωρητή διαμόρφωσης - μέρος 1. Πηγή: https://www.ti.com/l it/ds/symlink/ads1115.pdf

Σχήμα 7.20: Περιγραφή πεδίου καταχωρητή διαμόρφωσης - μέρος 2. Πηγή: https://www.ti.com/l it/ds/symlink/ads1115.pdf

f. Στο επόμενο βήμα θα συνδυάσουμε όλα τα σετ bit σε μια τιμή 16 bit για να το στείλουμε στον καταχωρητή διαμόρφωσης:

```
1 import machine
2 import utime
3
4 i2c = machine . I2C (1 , freq =400000 , scl = machine . Pin (15) , ,→ sda =
        machine . Pin (14) )
5 devices = i2c . scan ()
6 for device in devices :
7 print (hex ( device ))
8
9 ADS1115_I2C_ADDRESS = 0 x48
10 ADS1115_CONVERSION_REG = 0 x00
11 ADS1115_CONFIG_REG = 0 x01
12
13 ADS1115_CONFIG_OS_SINGLE = 0 x8000
14 ADS1115_CONFIG_MUX_AIN0 = 0 x4000
15 ADS1115_CONFIG_GAIN = 0 x0200
16 ADS1115_CONFIG_MODE_SINGLE = 0 x0100
17 ADS1115_CONFIG_DR_128SPS = 0 x0080
18 ADS1115_CONFIG_COMP_DISABLED = 0 x0003
19
20 ADS1115_CONFIG = (
21 ADS1115_CONFIG_OS_SINGLE
22 | ADS1115_CONFIG_MUX_AIN0

23 | ADS1115_CONFIG_GAIN
24 | ADS1115_CONFIG_MODE_SINGLE
25 | ADS1115_CONFIG_DR_128SPS
26 | ADS1115_CONFIG_COMP_DISABLED
27 )
```

g. Στο επόμενο βήμα, θα δημιουργήσουμε τη συνάρτηση για την αποστολή της τιμής μας στον καταχωρητή. Για να γίνει αυτό, πρέπει να δούμε πώς μοιάζει το πρωτόκολλο επικοινωνίας. Σύμφωνα με την τεκμηρίωση (βλ. Σχ. 7.21), πρέπει πρώτα να στείλετε τη διεύθυνση της συσκευής (διεύθυνση ADC I2C), μετά τη διεύθυνση μητρώου (π.χ. διαμόρφωση) και μετά τα byte δεδομένων. Ο μετατροπέας ADS1115 είναι συσκευή 16 bit και πρώτα πρέπει να στείλετε τα 8 πρώτα bit (ονομασία: D15-D8) και μετά τα 8 επόμενα bit (ονομασία: D7-D0). Αυτό ονομάζεται Big-endian. Συνήθως στη μνήμη του υπολογιστή μας οι τιμές αποθηκεύονται σε διαμόρφωση Little-endian που είναι αντίθετη από αυτήν. Θα πρέπει να το αντιμετωπίσουμε αυτό. Θα είναι πιο βολικό να δημιουργήσετε μια συνάρτηση (function) για την αποστολή δεδομένων και μέσα έναν πίνακα 4 στοιχείων με τα δεδομένα που πρέπει να σταλούν (διεύθυνση συσκευής, διεύθυνση μητρώου, πρώτα στη σειρά bit, επόμενα στη σειρά bit). Στον πίνακα, θα αποθηκεύσουμε δεδομένα με τη μορφή byte, επομένως θα χρησιμοποιήσουμε τη συνάρτηση bytearray(), η οποία δημιουργεί έναν πίνακα με δεδομένα αποθηκευμένα σε μορφή byte.

Σχήμα 7.21: Διάγραμμα χρονισμού. Πηγή: https://www.ti.com/lit/ds/symlink/ads1115.pdf

```
import machine
...

ADS1115_CONFIG = (
ADS1115_CONFIG_OS_SINGLE
| ADS1115_CONFIG_MUX_AIN0
| ADS1115_CONFIG_GAIN
| ADS1115_CONFIG_MODE_SINGLE
| ADS1115_CONFIG_DR_128SPS
| ADS1115_CONFIG_COMP_DISABLED
)

def write_ads1115_register ( register , value ) :
data = bytearray ([ register , ( value >> 8) & 0 xFF , ,→ value & 0 xFF ])
```

Στο παραπάνω απόσπασμα, όταν δημιουργούμε έναν bytearray, μπορεί να παρατηρήσετε δύο άγνωστες καταχωρήσεις:

- (value >> 8) – αυτή είναι μια πράξη μετατόπισης 8 bits προς τα δεξιά, δηλαδή όταν τα δεδομένα είναι 16 bits, π.χ. για την τιμή 1010 1010 0011 0011, το αποτέλεσμα θα είναι 0000 0000 1010 1010. Έτσι, τα 8 πρώτα bits θα μεταπηδήσουν στη θέση των 8 τελευταίων bits, και αντί για αυτά θα υπάρχουν μόνο μηδενικά.
- value & 0xFF – αυτή είναι μια λογική πράξη AND,το αποτέλεσμα της οποίας θα είναι μια αμετάβλητη τιμή 16-bit. Γιατί μια τέτοια πράξη; Εάν η μεταβλητή *value* περιείχε κάτι περισσότερο από τα 16 bit που μας ενδιαφέρουν, τότε πρέπει να απαλλαγούμε από τις υπόλοιπες πληροφορίες και να κρατήσουμε μόνο τα 16 bits. Στην Python, η οποία είναι μια δυναμική γλώσσα, δεν μπορούμε να είμαστε σίγουροι για τον τύπο της τιμής. Αυτός θα μπορούσε να είναι τιμή 32-bit με καθορισμένη τιμή στα σημαντικά bits και και η πράξη μετατόπισης θα μπορούσε να φέρει τα bits σε ένα πεδίο 16-bit. Επομένως, για να είστε σίγουροι, είναι καλύτερα να χρησιμοποιήσετε την πράξη του λογικού AND.

h. Το επόμενο βήμα είναι η αποστολή δεδομένων από τον πίνακα στο ADC μέσω του διαύλου I2C:

```
import machine
...

ADS1115_CONFIG = (
            ADS1115_CONFIG_OS_SINGLE
            | ADS1115_CONFIG_MUX_AIN0
            | ADS1115_CONFIG_GAIN
            | ADS1115_CONFIG_MODE_SINGLE
            | ADS1115_CONFIG_DR_128SPS
            | ADS1115_CONFIG_COMP_DISABLED
)

def write_ads1115_register ( register , value ) : data = bytearray ([
    register , ( value >> 8) & 0 xFF , ,→ value & 0 xFF ])
    i2c . writeto ( ADS1115_I2C_ADDRESS , data )
```

i. Γνωρίζουμε ήδη πώς να στέλνουμε δεδομένα. Ήρθε η ώρα να δημιουργήσετε μια συνάρτηση για την ανάγνωση δεδομένων. Για να γίνει αυτό, πρέπει να βρούμε πληροφορίες σχετικά με το πρωτόκολλο ανάγνωσης δεδομένων στο φύλλο δεδομένων (βλ. Σχ. 7.22). Η ανάγνωση των δεδομένων γίνεται σε δύο βήματα. Το πρώτο βήμα είναι να στείλουμε τη διεύθυνση ADC I2C και τη διεύθυνση μητρώου (ρυθμίζοντας το Address Pointer Register) από τα οποία θέλουμε να διαβάσουμε δεδομένα (κόκκινα ορθογώνια στο Σχ. 7.22). Το δεύτερο βήμα είναι να διαβάσουμε δύο byte δεδομένων από τη διεύθυνση ADC (σκούρα μπλε ορθογώνια στο Σχ. 7.22). Ας δημιουργήσουμε λοιπόν μια συνάρτηση που θα διαβάζει δεδομένα από τον καταχωρητή:

:

Σχήμα 7.22: Διάγραμμα χρονισμού. Πηγή: https://www.ti.com/lit/ds/symlink/ads1115.pdf

```
import machine
...

ADS1115_CONFIG = (
ADS1115_CONFIG_OS_SINGLE
| ADS1115_CONFIG_MUX_AIN0
| ADS1115_CONFIG_GAIN
| ADS1115_CONFIG_MODE_SINGLE
| ADS1115_CONFIG_DR_128SPS
| ADS1115_CONFIG_COMP_DISABLED
)

def write_ads1115_register ( register , value ) :
data = bytearray ([ register , ( value >> 8) & 0 xFF , ,→ value & 0 xFF ])
i2c . writeto ( ADS1115_I2C_ADDRESS , data )

        def read_ads1115_register ( register , num_bytes ):

i2c . writeto ( ADS1115_I2C_ADDRESS , bytearray ([ ,→ register ]) )
return i2c . readfrom ( ADS1115_I2C_ADDRESS , num_bytes ,→ )
```

j. Έχουμε ήδη συναρτήσεις (functions) για ανάγνωση και εγγραφή δεδομένων μέσω του διαύλου I2C. Τώρα ας δημιουργήσουμε μια συνάρτηση στην οποία θα διαμορφώσουμε τον AD μετατροπέα, θα διαβάσουμε δεδομένα από αυτόν και θα τα μετατρέψουμε σε τάση. Εδώ θα χρησιμοποιήσουμε τη συνάρτηση *from_bytes(),* η οποία μετατρέπει τιμές από τη μορφή μεμονωμένων byte σε δεκαδική τιμή. Σημειώστε ότι πρέπει να προσδιορίσουμε τον τρόπο με τον οποίο είναι διατεταγμένα τα byte στη μνήμη στη συνάρτηση *from_bytes().* Επειδή αυτά διαβάστηκαν ως Big-endian, χρησιμοποιούμε τον προσδιοριστή 'big'.

```
import machine
...

def read_ads1115_register ( register , num_bytes ): i2c . writeto (
        ADS1115_I2C_ADDRESS , bytearray ([ ,→ register ]) )
        return i2c . readfrom ( ADS1115_I2C_ADDRESS , num_bytes ,→ )

def read_adc () :
        # Sending configuration to configuration register
        write_ads1115_register ( ADS1115_CONFIG_REG ,
             ,→ ADS1115_CONFIG )

        # Wait for measurement to finish
        utime . sleep (0.01)

        # Reading conversion value
        result = read_ads1115_register (
             ,→ ADS1115_CONVERSION_REG , 2)
        raw_value = int . from_bytes ( result , 'big ')
        if raw_value >= 0 x8000 : # Correction for negative ,→ numbers
              raw_value -= 0 x10000

          # Convert value to voltage
          voltage = raw_value * (4.096 / 32768) # Range ,→ + -4.096 V/
               max value
          return voltage
```

1
2
3
4
5

6

7
8
9

k. Τώρα ας προσθέσουμε τον κύριο βρόχο στον οποίο θα διαβάσουμε την τιμή της τάσης και θα τη μετατρέψουμε σε θερμοκρασία (ο πολλαπλασιασμός με το 100 προκύπτει από την τεκμηρίωση για τον αισθητήρα θερμοκρασίας LM35):

```
import machine
import utime

i2c = machine . I2C (1 , freq =400000 , scl = machine . Pin (15) , ,→ sda =
        machine . Pin (14) )
devices = i2c . scan ()

for device in devices :
print (hex ( device ))

ADS1115_I2C_ADDRESS = 0 x48
ADS1115_CONVERSION_REG = 0 x00
ADS1115_CONFIG_REG = 0 x01

ADS1115_CONFIG_OS_SINGLE = 0 x8000
ADS1115_CONFIG_MUX_AIN0 = 0 x4000
ADS1115_CONFIG_GAIN = 0 x0200
ADS1115_CONFIG_MODE_SINGLE = 0 x0100
ADS1115_CONFIG_DR_128SPS = 0 x0080
ADS1115_CONFIG_COMP_DISABLED = 0 x0003

ADS1115_CONFIG = (
ADS1115_CONFIG_OS_SINGLE
| ADS1115_CONFIG_MUX_AIN0
| ADS1115_CONFIG_GAIN
| ADS1115_CONFIG_MODE_SINGLE
| ADS1115_CONFIG_DR_128SPS
| ADS1115_CONFIG_COMP_DISABLED
)

def write_ads1115_register ( register , value ) :
data = bytearray ([ register , ( value >> 8) & 0 xFF , ,→ value & 0 xFF ])
i2c . writeto ( ADS1115_I2C_ADDRESS , data )

def read_ads1115_register ( register , num_bytes ): i2c . writeto (
        ADS1115_I2C_ADDRESS , bytearray ([ ,→ register ]) )
```

```
35 return i2c . readfrom ( ADS1115_I2C_ADDRESS , num_bytes ,→ )
36
37 def read_adc () :
38 # Sending configuration to configuration register 39 write_ads1115_register (
                ADS1115_CONFIG_REG , ,→ ADS1115_CONFIG )
40
41 # Wait for measurement to finish
42 utime . sleep (0.01)
43
44 # Reading conversion value
45 result = read_ads1115_register (
                ,→ ADS1115_CONVERSION_REG , 2)
46 raw_value = int . from_bytes ( result , 'big ') 47 if raw_value >= 0 x8000 : #
Correction for negative ,→ numbers
48 raw_value -= 0 x10000
49
```



```
50 # Convert value to voltage
51 voltage = raw_value * (4.096 / 32768) # Range ,→ + -4.096 V/ max value
52 return voltage
53
54 while True :
55 voltage = read_adc ()
56 temp = voltage *100
57 print ("T="+str ( temp ) )
58 utime . sleep (1)
```

Ο κώδικας είναι έτοιμος και μπορούμε να τον δοκιμάσουμε.

7.6 Παράδειγμα 12: Μέτρηση θερμοκρασίας με χρήση 1-wire bus

Η διεπαφή 1-wire αναπτύχθηκε από την εταιρεία Dallas Semiconductor. Αυτός ο δίαυλος αποτελείται από μία μόνο γραμμή. Αυτή η διεπαφή λειτουργεί παρόμοια με το δίαυλο I2C, αλλά τα bit 0 και 1 ορίζονται από τη χρονική διάρκεια του χαμηλού σήματος. Το 1-wire bus λειτουργεί πιο αργά από το I2C. Η μέγιστη ταχύτητα είναι 16 kbit/s.

Σε αυτό το παράδειγμα, θα διαβάσουμε τη θερμοκρασία από τον αισθητήρα θερμοκρασίας DS18B20 μέσω του διαύλου 1-wire και θα εμφανίσουμε τη θερμοκρασία ανάγνωσης σε μια οθόνη LCD με μετατροπέα I2C (μερικές φορές ο μετατροπέας αγοράζεται ξεχωριστά για την οθόνη). Για να το κάνετε αυτό, συνδέστε το σύστημα όπως στο Σχκ. 7.23

.

Σχήμα 7.23: Σύνδεση του ηλεκτρονικού κυκλώματος του παραδείγματος 12.

Τώρα ας γράψουμε ένα πρόγραμμα που θα διαβάζει δεδομένα από τον αισθητήρα DS18B20 και θα τα εμφανίζει στην οθόνη. Για να το κάνετε αυτό, ακολουθήστε τα εξής βήματα:

a. Εγκαταστήστε τη βιβλιοθήκη onwire και το ds18x20 χρησιμοποιώντας τη διαχείριση πακέτων στο πρόγραμμα επεξεργασίας Thonny (δείτε τα Σχ. 7.24 και 7.25).

Σχήμα 7.24: Εγκατάσταση της βιβλιοθήκης onewire.

Σχήμα 7.25: Εγκατάσταση της βιβλιοθήκης DS18x20.

b. Τώρα ας προσθέσουμε τις απαραίτητες βιβλιοθήκες:

```
import machine
import onewire
import ds18x20
import utime
```

c. Στη συνέχεια, πρέπει να καθορίσετε ποιος ακροδέκτης GPIO θα χρησιμεύσει ως 1-wire δίαυλος. Για να το κάνετε αυτό, χρησιμοποιήστε τη συνάρτηση onewire.OneWire(*pin number*). Το επόμενο βήμα είναι να διαμορφώσετε τον αισθητήρα DS18B20, δηλαδή να καλέσετε τη συνάρτηση: ds18x20.DS18X20(), η οποία παίρνει ως όρισμα ένα αντικείμενο κλάσης onewire:

```
import machine
import onewire
import ds18x20
import utime

one_wire_bus = onewire . OneWire ( machine . Pin (13) )
ds18b20_sensor = ds18x20 . DS18X20 ( one_wire_bus )
```

d. Στο επόμενο βήμα, πρέπει να προσδιορίσετε την διεύθυνση του αισθητήρα DS18B20 που συνδέθηκε στο δίαυλο1-wire. Αυτό μπορεί να γίνει αυτόματα με τη χρήση της συνάρτησης scan():

```
import machine
import onewire
import ds18x20
import utime

one_wire_bus = onewire . OneWire ( machine . Pin (13) )
ds18b20_sensor = ds18x20 . DS18X20 ( one_wire_bus )
device_addr = ds18b20_sensor . scan ()
```

e. Στη συνέχεια, θα πρέπει να δώσετε στον αισθητήρα μια εντολή να μετρήσει τη θερμοκρασία (συνάρτηση con vert_temp()) και μετά να περιμένετε περίπου 750 ms μέχρι να μετρήσει σύμφωνα με την τεκμηρίωση για τον αισθητήρα DS18B20. Στη συνέχεια, θα πρέπει να διαβάσετε τη μετρούμενη τιμή θερμοκρασίας σε βαθμούς Κελσίου χρησιμοποιώντας τη συνάρτηση read_temp(), η οποία παίρνει τη διεύθυνση του αισθητήρα ως όρισμα. Εφόσον η συνάρτηση scan() επιστρέφει έναν πίνακα με διευθύνσεις, θα πρέπει να εξαγάγετε το πρώτο στοιχείο του πίνακα. Οι πίνακες στην Python αριθμούνται από το 0 και για να εξαγάγετε ένα δεδομένο τμήμα του πίνακα, θα πρέπει να γράψετε *array_name[index]*:

```
import machine
import onewire
import ds18x20
import utime

one_wire_bus = onewire . OneWire ( machine . Pin (13) )
ds18b20_sensor = ds18x20 . DS18X20 ( one_wire_bus )
device_addr = ds18b20_sensor . scan ()

while True :
        ds18b20_sensor . convert_temp ()
        utime . sleep (0.75)

        temp = ds18b20_sensor . read_temp ( device_addr [0]) print
        ("T="+str ( temp ) )
```

```
4
5
6
7
8
9
10
11
12
13
14
15
```

f. Μπορείτε τώρα να δοκιμάσετε το πρόγραμμα. Αν διαβάζετε σωστά τη θερμοκρασία, μπορούμε να προχωρήσουμε στην εμφάνιση στην οθόνη LCD. Αρχικά, κατεβάστε τα δύο αρχεία: *lcd_api.py* (https: //github.com/T-622/RPI-PICO-I2C-LCD/blob/main/lcd_api.py) και *i2c_lcd.py* (https://github.com/T-622/RPI-PICO-I2C-LCD/blob/mai n/pico_i2c_lcd.py) και στη συνέχεια μεταφορτώστε τα στο Raspbery Pi Pico όπως φαίνεται στο σχήμα.

Σχήμα 7.26: Εγκατάσταση της βιβλιοθήκης για την εμφάνιση στην οθόνη LCD

g. Τώρα ας προσθέσουμε τις απαραίτητες βιβλιοθήκες:

```
import machine
import onewire
import ds18x20
import utime
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

one_wire_bus = onewire . OneWire ( machine . Pin (13) )
ds18b20_sensor = ds18x20 . DS18X20 ( one_wire_bus )
device_addr = ds18b20_sensor . scan ()
...
```

h. Στη συνέχεια, ας διαμορφώσουμε το δίαυλο I2C και ας τον σαρώσουμε προκειμένου να βρούμε τη διεύθυνση της οθόνης LCD. Εκτελέστε το πρόγραμμα και διαβάστε την τιμή.

```
import machine
import onewire
import ds18x20
import utime
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

one_wire_bus = onewire . OneWire ( machine . Pin (13) )
ds18b20_sensor = ds18x20 . DS18X20 ( one_wire_bus ) device_addr
= ds18b20_sensor . scan ()

i2c = machine . I2C (0 , sda = machine . Pin (16) , scl = machine . Pin
                                    (17) , freq =400000)
print ("LCD screen address ="+ str ( i2c . scan () ))
...
```

i. Ας αποθηκεύσουμε τη διεύθυνση της οθόνης LCD που διαβάσαμε σε μια μεταβλητή και ας αποθηκεύσουμε τις διαστάσεις της οθόνης (αριθμός γραμμών και στηλών):

```
import machine
import onewire
import ds18x20
import utime
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

one_wire_bus = onewire . OneWire ( machine . Pin (13) )
ds18b20_sensor = ds18x20 . DS18X20 ( one_wire_bus )
device_addr = ds18b20_sensor . scan ()

i2c = machine . I2C (0 , sda = machine . Pin (16) , scl = ,→machine .
    Pin (17) , freq =400000)
print ("LCD screen address ="+ str ( i2c . scan () ))

I2C_ADDR =63
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16
...
```

j. Ας αρχικοποιήσουμε την οθόνη LCD καις ας περιμένουμε 1δευτερόλεπτο. Στη συνέχεια, ας καθαρίσουμε την οθόνη, ας τοποθετήσουμε τον κέρσορα στην αρχή και ας εμφανίσουμε το κείμενο καλωσορίσματος:

```
import machine
import onewire
import ds18x20
import utime
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

one_wire_bus = onewire . OneWire ( machine . Pin (13) ) 
ds18b20_sensor = ds18x20 . DS18X20 ( one_wire_bus ) device_addr
= ds18b20_sensor . scan ()

i2c = machine . I2C (0 , sda = machine . Pin (16) , scl = ,→machine . Pin
                                (17) , freq =400000)
```

```
13 print ("LCD screen address ="+ str ( i2c . scan () )) 14
15 I2C_ADDR =63

16 I2C_NUM_ROWS = 2
17 I2C_NUM_COLS = 16
18
19 lcd = I2cLcd ( i2c , I2C_ADDR , I2C_NUM_ROWS , I2C_NUM_COLS ,→)
20 utime . sleep (1)
21
22 lcd . clear ()
23 lcd . move_to (0 ,0)
24 lcd . putstr (" Hello ")
25 ...
```

k. Το επόμενο βήμα είναι να εμφανίσουμε τη θερμοκρασία στην επόμενη γραμμή της οθόνης LCD μετά τη μέτρηση της θερμοκρασίας:

```
import machine
import onewire
import ds18x20
import utime
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

one_wire_bus = onewire . OneWire ( machine . Pin (13) )
ds18b20_sensor = ds18x20 . DS18X20 ( one_wire_bus )
device_addr = ds18b20_sensor . scan ()

i2c = machine . I2C (0 , sda = machine . Pin (16) , scl = ,→machine .
    Pin (17) , freq =400000)
print ("LCD screen address ="+ str ( i2c . scan () ))

I2C_ADDR =63
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

lcd = I2cLcd ( i2c , I2C_ADDR , I2C_NUM_ROWS , I2C_NUM_COLS ,→)
utime . sleep (1)

lcd . clear ()
lcd . move_to (0 ,0)
lcd . putstr (" Hello ")

while True :
        ds18b20_sensor . convert_temp ()
        utime . sleep (0.75)

        temp = ds18b20_sensor . read_temp ( device_addr [0]) print
        ("T="+str ( temp ) )

        lcd . move_to (0 ,1)
        lcd . putstr ("T="+ str ( temp )+" C")
```

Το πρόγραμμα είναι έτοιμο και μπορούμε να το δοκιμάσουμε.

> Συμβουλή 7.6.1
>
> Εάν θέλετε να χρησιμοποιήσετε την οθόνη LCD χωρίς μετατροπέα I2C, μπορείτε να χρησιμοποιήσετε τη βιβλιοθήκη που είναι διαθέσιμη εδώ: https://github.com/gusandrio li/liquid-crystal-pico.

8. Ενεργοποιητές (Actuators)

Οι ενεργοποιητές είναι στοιχεία που κάνουν μια κίνηση (εκτελεστικά στοιχεία). Οι βασικοί ενεργοποιητές είναι: σερβοκινητήρας, βηματικός κινητήρας και κινητήρας συνεχούς ρεύματος.

8.1 Παράδειγμα 13: Σερβοκινητήρας (Servo motor)

Ο servo είναι ένας κινητήρας με γρανάζι που περιστρέφεται κατά μια δεδομένη γωνία, συνήθως στην περιοχή (0;180) μοίρες ή (-90;90) μοίρες. Η κατασκευή του φαίνεται στο

Σχ. 8.1. Ο σερβοκινητήρας μπορεί να συνδεθεί απευθείας στην πλακέτα Raspberry Pi Pico. Τα τρία καλώδια βγαίνουν από τον σερβομηχανισμό. Το μεσαίο (συνήθως κόκκινο) πρέπει να συνδεθεί στο τροφοδοτικό (+5V = VBUS). Το μαύρο είναι GND (γείωση) και το τελευταίο καλώδιο (συνήθως ανοιχτόχρωμο: λευκό/κίτρινο) είναι το χειριστήριο που πρέπει να συνδεθεί στην ακίδα PWM.

Σχήμα 8.1: Αρχή λειτουργίας ενός σερβομηχανισμού. Πηγή: https://ai.thestempedia.com/docs/quarky/quarky-technical-specifications/servo-motor-wi th-quarky/.

Ο έλεγχος του servo βασίζεται στο σήμα PWM. Ο ελεγκτής, που βρίσκεται στον σερβομηχανισμό, διαβάζει το σήμα PWM και βάσει αυτού καθορίζει τη γωνία με την οποία θα περιστραφεί το γρανάζι. Η ιδέα της λειτουργίας σερβομηχανισμού φαίνεται στο Σχ. 8.2.

Σχήμα 8.2: Αρχή λειτουργίας του ελέγχου servo control. Πηγή: https://howtomechatronics.com/wp -content/uploads/2018/03/RC-Servo-Motor-Control-Signal.png.

Ας δημιουργήσουμε ένα πρόγραμμα που θα περιστρέφει τον σερβομηχανισμό από την ελάχιστη στη μέση θέση και μετά στη μέγιστη και πάλι πίσω. Για να γίνει αυτό, ας συνδέσουμε πρώτα το σύστημα όπως φαίνεται στο Σχ. 8.3 (κόκκινο καλώδιο- VBUS, κίτρινο καλώδιο- GP18, μαύρο καλώδιο- GND).

Σχήμα 8.3: Σύνδεση του ηλεκτρονικού κυκλώματος για το παράδειγμα 13.

Ανοίξτε τον επεξεργαστή Thonny και ακολουθείστε τα εξής βήματα:

1. Προσθέστε τις απαραίτητες βιβλιοθήκες:

```
import machine
import utime
```

2. Σύμφωνα με την τεκμηρίωση για τον σερβομηχανισμό, η ελάχιστη θέση είναι όταν το υψηλό σήμα διαρκεί 0,9 ms (μερικές φορές 0,5 ms). Η μεσαία θέση είναι όταν το υψηλό σήμα διαρκεί 1,5 ms και η μέγιστη όταν διαρκεί 2,1 ms (μερικές φορές 2,5 ms). Η περίοδος σήματος PWM πρέπει να είναι 20 ms (50 Hz). Ας δημιουργήσουμε μεταβλητές που θα αποθηκεύουν αυτές τις τιμές σε ns:

```
import machine
import utime

MIN = 900000
MID = 1500000
MAX = 2100000
```

3. Διαμορφώστε τον ακροδέκτη GP18 ως PWM και ρυθμίστε τη συχνότητα σήματος του PWM σε 50 Hz.

```
import machine
import utime

MIN = 900000
MID = 1500000
MAX = 2100000

pwm = machine . PWM ( machine . Pin (18) )
pwm . freq (50)
```

4. Θα ορίσουμε τις θέσεις για το servo ρυθμίζοντας τον κύκλο λειτουργίας. Στο προηγούμενο κεφάλαιο, παρουσιάστηκε η συνάρτηση *duty_u16(),* η οποία ορίζει τον κύκλο λειτουργίας που δίνεται στην περιοχή από 0 έως 65535 (100%). Θα είναι πιο βολικό εδώ να χρησιμοποιήσουμε τη συνάρτηση *duty_ns()*, η οποία ορίζει τον κύκλο λειτουργίας σε έναν καθορισμένο χρόνο σε νανοδευτερόλεπτα. Ας ορίσουμε την αρχική εκτροπή στη μέση:

```
import machine
import utime

MIN = 900000
MID = 1500000
MAX = 2100000

pwm = machine . PWM ( machine . Pin (18) )
pwm . freq (50)
pwm . duty_ns ( MIN )
```

5. Στον κύριο βρόχο, ας κάνουμε το σερβοκινητήρα να περιστρέφεται από την ελάχιστη θέση μέχρι τη μέση, στη μέγιστη θέση και πάλι πίσω σε 1 δευτερόλεπτο:

```
import machine
import utime

```