ΚΕΦΑΛΑΙΟ 3: ΕΚΦΡΑΣΕΙΣ, ΑΝΑΜΟΝΕΣ ΚΑΙ ΗΧΟΙ

ΣΤΟΧΟΙ

Το πρώτο θέμα που θα καλύψουμε στο κεφάλαιο αυτό είναι να εμβαθύνουμε στις μεταβλητές, στις σταθερές και τις εκφράσεις. Αναφερθήκαμε σε αυτές στο προηγούμενο κεφάλαιο, αλλά τώρα θα τις δούμε αναλυτικότερα.

Επιπλέον, θα παρουσιάσουμε μερικές νέες συναρτήσεις της γλώσσας προγραμματισμού του Arduino, που θα σας επιτρέψουν να δημιουργήσετε αναμονές και ήχους. Χάρη σε αυτές θα είστε σε θέση να δημιουργήσετε περισσότερο ευέλικτα και ενδιαφέροντα προγράμματα.

ΕΝΟΤΗΤΑ ΘΕΩΡΙΑΣ

- ΕΚΦΡΑΣΕΙΣ
  - Σταθερές
  - Μεταβλητές
  - Εμβέλεια μεταβλητών
  - Πράξεις
- ΑΝΑΜΟΝΕΣ
  - Η συνάρτηση delayMicroseconds()
  - Η συνάρτηση delay()
  - Η συνάρτηση micros()
  - Η συνάρτηση millis()
- ΗΧΟΣ
- ΣΥΝΑΡΤΗΣΕΙΣ ΗΧΟΥ

ΕΝΟΤΗΤΑ ΕΞΑΣΚΗΣΗΣ

- EXAMPLE 1: Φώτα που αναβοσβήνουν
- EXAMPLE 2: Ομάδα από φώτα
- EXAMPLE 3: Φανάρι κυκλοφορίας (V1)
- EXAMPLE 4: Κουδούνι
- EXAMPLE 5: Μελωδίες
- EXAMPLE 6: Φανάρι κυκλοφορίας (V2)

ΑΠΑΙΤΟΥΜΕΝΑ ΥΛΙΚΑ

- Σταθερός ή φορητός υπολογιστής.

- Το Ολοκληρωμένο Περιβάλλον Ανάπτυξης Arduino –ή αλλιώς, το λογισμικό Arduino IDE– το οποίο περιλαμβάνει έναν επεξεργαστή κειμένου για τη σύνταξη κώδικα, μία περιοχή μηνυμάτων, μία κονσόλα κειμένου, μία μπάρα εργαλείων με κουμπιά για συχνές λειτουργίες και μία σειρά από μενού. Το Arduino IDE συνδέεται με το Arduino για το «ανέβασμα» προγραμμάτων και για επικοινωνία μαζί του. Το συμπληρωματικό υλικό περιλαμβάνεται στο πρόγραμμα και, θεωρητικά, έχει ήδη εγκατασταθεί και ρυθμιστεί.

- Αναπτυξιακή πλακέτα Arduino UNO.

- Πλακέτα δοκιμών (breadboard).

- Καλώδιο USB.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΝΟΤΗΤΑ ΘΕΩΡΙΑΣ

1. ΕΚΦΡΑΣΕΙΣ

Έως τώρα, πιθανότατα θα έχετε διαπιστώσει πως όλες οι συναρτήσεις της γλώσσας προγραμματισμού του Arduino που έχουμε εξετάσει χρειάζονται μία ή περισσότερες παραμέτρους για να λειτουργήσουν. Αυτές τοποθετούνται μέσα στις παρενθέσεις της συνάρτησης και διαχωρίζονται με κόμμα «,» μεταξύ τους. Ας τις ξαναδούμε:

- **pinMode(pin, mode)**: πρέπει να ορίσετε τον αριθμό του ακροδέκτη στον οποίο αναφέρεστε και το κατά πόσο αυτός θα χρησιμοποιηθεί σαν είσοδος, ή έξοδος.
- **digitalRead(pin)**: πρέπει να ορίσετε τον αριθμό του ακροδέκτη από τον οποίο θέλετε να διαβάσετε.
- **digitalWrite(pin,value)**: πρέπει να ορίσετε τον αριθμό του ακροδέκτη στον οποίο θέλετε να εγγράψετε μία τιμή.

Μπορείτε να εκφράσετε αυτές τις παραμέτρους με πολλαπλούς τρόπους, πράγμα το οποίο και κάνατε σε μερικά από τα παραδείγματα του προηγούμενου κεφαλαίου. Ο τρόπος με τον οποίο υποδεικνύετε τις παραμέτρους ονομάζεται «έκφραση» και υπάρχουν τρεις τύποι:

- **Σταθερά**: Καθορίζετε την τιμή της παραμέτρου απευθείας στη συνάρτηση. Για παράδειγμα, το **digitalRead(4)** διαβάζει την τιμή του ακροδέκτη 4.
- **Μεταβλητή**: Έχετε ήδη καθορίσει την τιμή της παραμέτρου με μία μεταβλητή. Για παράδειγμα, το **digitalRead(pushbutton**) διαβάζει τον ακροδέκτη ο αριθμός του οποίου έχει οριστεί στη μεταβλητή «pushbutton». Ο μικροελεγκτής θα αναζητήσει τη μεταβλητή στη μνήμη του και θα εξάγει από αυτή τον αριθμό του ακροδέκτη που θα διαβαστεί. Σε περίπτωση αποτυχίας, θα παραχθεί ένα μήνυμα σφάλματος.
- **Πράξη**: Η τιμή της παραμέτρου καθορίζεται ως αποτέλεσμα αριθμητικής ή λογικής πράξης ανάμεσα σε μεταβλητές και σταθερές. Για παράδειγμα, το

digitalRead((1+1)*2) θα διαβάσει τον ακροδέκτη 4, καθώς αυτό είναι το αποτέλεσμα της πράξης (1+1)*2.

Α. Σταθερές

Μία τιμή ενός προγράμματος που δεν αλλάζει, μπορεί να χρησιμοποιηθεί σαν «σταθερά». Εάν θέλετε να διαβάσετε ένα κουμπί που είναι μόνιμα συνδεδεμένο στον ακροδέκτη 4 μπορείτε να χρησιμοποιήσετε τη συνάρτηση **digitalRead(4)**.

Κάθε φορά που ο μικροελεγκτής εκτελεί αυτή τη συνάρτηση, διαβάζει πάντα τον ίδιο ακροδέκτη: τον 4. Δεν υπάρχει **ΚΑΝΕΝΑΣ** τρόπος αλλαγής αυτού **κατά τη διάρκεια** της εκτέλεσης του προγράμματος. Εάν αποφασίσετε να τον αλλάξετε θα πρέπει να τροποποιήσετε το πρόγραμμα και να το «ανεβάσετε» ξανά στη μνήμη του μικροελεγκτή. Ορίστε ένα ακόμα παράδειγμα: θέλετε να δημιουργήσετε ένα πρόγραμμα που θα υπολογίζει την περίμετρο βασιζόμενο σε γνωστή διάμετρο. Η εξίσωση θα ήταν **l = d * π**.

l = μήκος περιμέτρου.

d = διάμετρος περιμέτρου

π = 3.141592

Στο εν λόγω παράδειγμα…

Ποια είναι σταθερά; _____

Ποια είναι μεταβλητή; _____

Οι σταθερές είναι αναπόσπαστο κομμάτι του προγράμματος καθώς εμφανίζονται στις εντολές ή τις συναρτήσεις. Αποθηκεύονται στη μνήμη προγραμμάτων (Flash) του μικροελεγκτή κι έτσι δε μπορούν να τροποποιηθούν εκτός και αν περαστεί εκ νέου το

πρόγραμμα. Δεν είναι «πτητικές» ενώ κρατάνε την τιμή τους ακόμα και αν διακοπεί η τροφοδοσία.

Περισσότερα παραδείγματα:

digitalRead(12); // διαβάζει τον ακροδέκτη 12

digitalWrite(6,HIGH); δίνει στον ακροδέκτη 6 τιμή "1"

pinMode(9,OUTPUT); ορίζει τον εκροδέκτη 9 ως έξοδο

Β. Μεταβλητές

Οι μεταβλητές είναι τμήματα στη μνήμη RAM του μικροελεγκτή, ειδικά σχεδιασμένα για την αποθήκευση διαφορετικών τύπων δεδομένων. Φανταστείτε τα σαν σειρές από κουτιά ή δοχεία στα οποία μπορείτε να αποθηκεύσετε οποιοδήποτε είδος πληροφορίας και να τα ανακτήσετε αργότερα, όταν υπάρξει ανάγκη.

Μη ξεχνάτε πως μπορείτε να διαβάσετε και να γράψετε τη μνήμη RAM όσο συχνά επιθυμείτε, εφόσον το κάνετε με τον ενδεδειγμένο τρόπο. Μπορείτε να τροποποιήσετε τα περιεχόμενά της κατά τη διάρκεια εκτέλεσης ενός προγράμματος. Η μνήμη αυτή είναι πτητική και τα περιεχόμενά της χάνονται όταν διακοπεί η τροφοδοσία.

Χρησιμοποιήσατε μεταβλητές σε μερικά παραδείγματα στο προηγούμενο κεφάλαιο. Τις μεταβλητές πρέπει να τις δηλώσετε πριν τις χρησιμοποιήσετε. Για να δηλώσετε μία μεταβλητή, πρώτα απ' όλα πρέπει να της δώσετε ένα όνομα και, πιθανώς, περιεχόμενο. Η σύνταξη είναι η εξής:

type name = value;

type: Ορίζει τον τύπο δεδομένων που θα περιέχει η μεταβλητή. Αυτός μπορεί να είναι αριθμοί, μεμονωμένοι χαρακτήρες ή κείμενο. Τους τύπους δεδομένων θα τους εξετάσουμε στη συνέχεια.

name: Αυτό είναι το όνομα που αναθέτετε στη μεταβλητή. Αυτό θα χρησιμοποιείτε αργότερα όταν θα θελήσετε να προσπελάσετε τα περιεχόμενά της. Μία καλή ιδέα είναι να χρησιμοποιείτε περιεκτικά ονόματα. Τα ονόματα θα πρέπει επίσης να μην ξεκινούν με αριθμό, αλλά και να μην περιλαμβάνουν κενά.

value: Αυτή είναι η τιμή του περιεχομένου της μεταβλητής. Είναι προαιρετική: μπορούμε να δημιουργήσουμε μία μεταβλητή η οποία αρχικά δε θα περιέχει τίποτα. Σε αυτή την περίπτωση, θα πρέπει να φροντίσουμε ώστε η μεταβλητή να έχει πάρει κάποια τιμή πριν την προσπέλασή της.

Ο παρακάτω πίνακας περιέχει μία σύνοψη των συχνότερων τύπων μεταβλητών:

ΤΥΠΟΣ	BYTES	ΠΕΡΙΓΡΑΦΗ	ΠΑΡΑΔΕΙΓΜΑΤΑ
char	1	Μία μεταβλητή char καταλαμβάνει 1 byte μνήμης και αποθηκεύει μία τιμή χαρακτήρα (8 bits). Οι χαρακτήρες περικλείονται από μονά εισαγωγικά (‘). Ο τύπος char είναι προσημασμένος, που σημαίνει ότι κωδικοποιεί αριθμούς από το -128 έως το 127.	char N = ‘1’ char letter =‘A’
byte	1	Ένα byte αποθηκεύει έναν μη-προσημασμένο αριθμό 8-bit αποδίδοντας εύρος τιμών 0 έως 255 (28-1).	byte C = 23 byte result = C + 18
int	2	Ο int είναι ο βασικός τύπος δεδομένων για αποθήκευση αριθμητικών τιμών. Αποθηκεύει έναν προσημασμένο (2-byte) αριθμό 16-bit αποδίδοντας έτσι εύρος τιμών από -32768 έως +32767.	int C = 2453 int value = C * 10 number = 2453
unsigned int or word	2	Αποθηκεύει έναν μη-προσημασμένο αριθμό 16-bit αποδίδοντας εύρος τιμών από 0 έως 65535 (216-1)	unsigned int valor = 60000 word valor = 4328 * 10

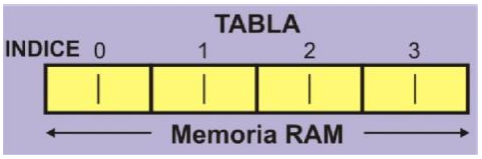
long	4	Προσημασμένες μεταβλητές που αποθηκεύουν 32 bits (4 bytes) και αποδίδουν εύρος τιμών από -2,147,483,648 έως 2,147,483,647.	long counter = -123456789
unsigned long	4	Μη-προσημασμένες μεταβλητές που αποθηκεύουν 32 bits (4 bytes) και αποδίδουν εύρος τιμών από 0 έως 4,294,967,295 (2^32 - 1).	unsigned long speedOfLight = 299792468
float	4	Οι αριθμοί κινητής υποδιαστολής μπορούν να είναι από 3.4028235E+38 έως -3.4028235E+38. Αποθηκεύονται σαν 32 bits (4 bytes) πληροφορίας.	float pi = 3.1416 float L = 4 * pi

Είναι προφανές πως ανάλογα με τον τύπο της μεταβλητής που δηλώνετε, δεσμεύεται και η αντίστοιχη ποσότητα από τη μνήμη RAM. Μία καλή συμβουλή είναι να δηλώνετε μεταβλητές ανάλογα με τον τύπο της πληροφορίας που πρόκειται να αποθηκεύσουν. Για παράδειγμα, εάν δηλώσετε μία “long” μεταβλητή, θα καταναλώσετε 4 bytes μνήμης RAM, αλλά δεν είναι ιδιαίτερα αποδοτικό να χρησιμοποιήσετε έναν τέτοιο τύπο εάν η προς αποθήκευση πληροφορία έχει εύρος τιμών 0-255. Σε αυτή την περίπτωση, είναι καλύτερα να χρησιμοποιήσετε τον τύπο byte, που καταναλώνει μόλις 1 byte μνήμης. Η ποσότητα της μνήμης RAM του μικροελεγκτή είναι περιορισμένη. Χρησιμοποιώντας την με λάθος τρόπο μπορεί να βρεθείτε μπροστά σε μία δυσάρεστη έκπληξη μόλις αυτή εξαντληθεί.

Από την άλλη, εάν δηλώσετε ένα “byte” και του δώσετε τιμή μεγαλύτερη από 255, η μεταβλητή υπερχειλίζει και χάνει την πληροφορία. Μοιάζει με το τι θα συμβεί στο οδόμετρο ενός αυτοκινήτου όταν ξεπεράσετε τη χωρητικότητά του: θα γυρίσει ξανά στο 0.

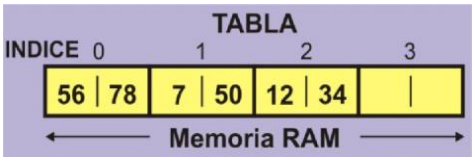
Οι **“Πίνακες”** είναι ακόμα ένα είδος μεταβλητών. Ένας πίνακας είναι μία ομάδα μεταβλητών και μπορεί να αποτελείται από οποιοδήποτε από τους τύπους που είδαμε νωρίτερα. Για να προσπελάσετε το περιεχόμενο αυτών των μεταβλητών, πρέπει να υποδείξετε το δείκτη ευρετηρίου τους. Δείτε τα παρακάτω παραδείγματα:

- **int Table[4]**



Δημιουργεί έναν άδειο πίνακα με όνομα “Table” και δεσμεύει χώρο για την αποθήκευση 4 μεταβλητών int. Κατά συνέπεια, δεσμεύονται 8 bytes από τη μνήμη RAM του μικροελεγκτή.

- **int Table[4]={5678,750,1234}**



Δημιουργεί έναν πίνακα “Table” με χώρο για αποθήκευση 4 int μεταβλητών. Εκεί αποθηκεύονται οι τρεις τιμές που δίνονται.

- **int Sum = Table[0] + Table[2]**

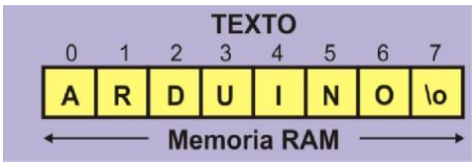
Ανακτά τις τιμές από τις θέσεις 0 και 2 του πίνακα “Table” και στη συνέχεια τις προσθέτει μεταξύ τους. Το αποτέλεσμα αποθηκεύεται στη μεταβλητή “Sum”.

- **unsigned int Values[12]**

Δημιουργεί έναν πίνακα με όνομα “Values” και δεσμεύει στη μνήμη χώρο για 12 μη-προσημασμένες μεταβλητές int. Συνολικά, δεσμεύει 24 bytes από τη μνήμη RAM.

- **Values[5] = 12345 * 3**

Το αποτέλεσμα της πράξης (37035) εισάγεται στην 5^{η} θέση του πίνακα “Values”.



- **char Text[8] = “ARDUINO”**

Δημιουργεί έναν πίνακα χαρακατήρων (τύπος char) με όνομα “Text” ο οποίος καταλαμβάνει 8 bytes στη μνήμη RAM και αποθηκεύει το αλφαριθμητικό “ARDUINO\0” (το ‘\0’ είναι ένας ειδικός χαρακτήρας που υποδηλώνει το τέλος της συμβολοσειράς).

char Letter = Text[2]

Ανακτά τα περιεχόμενα της 2ης θέσης του πίνακα “Text” (δηλαδή τον χαρακτήρα ‘D’) και τα αποθηκεύει στη μεταβλητή “Letter” που είναι τύπου char.

Γ. Εμβέλεια Μεταβλητών

Οι μεταβλητές μπορούν να είναι «καθολικές» ή «τοπικές» ανάλογα με το που έχουν δηλωθεί μέσα στο πρόγραμμα. Αυτό δεν θα σας επηρεάσει ιδιαίτερα προς το παρόν, αλλά θα είναι σημαντικό αργότερα, που θα αρχίσετε να δημιουργείτε μεγαλύτερα και περιπλοκότερα προγράμματα.

Καθολικές: Αυτές είναι οι μεταβλητές που έχουν δηλωθεί έξω από οποιαδήποτε συνάρτηση του προγράμματος, συμπεριλαμβανομένων των **setup()** και **loop()**. Μερικά από τα παραδείγματα του προηγούμενου κεφαλαίου χρησιμοποιούσαν τέτοιες μεταβλητές. Δείτε το επόμενο σχήμα για να διαπιστώσετε πως οι μεταβλητές “Data” και “Result” είναι καθολικές. Αυτές μπορούν να αξιοποιηθούν από όλες τις συναρτήσεις του προγράμματος.

Τοπικές: Αυτές είναι οι μεταβλητές που δηλώνονται και δημιουργούνται εντός κάποιας συγκεκριμένης συνάρτησης και μπορούν να χρησιμοποιηθούν μόνο από αυτήν. Κατ’ αυτό τον τρόπο αποφεύγουμε μία μεταβλητή που δηλώθηκε σε μία συνάρτηση να χρησιμοποιηθεί ή να τροποποιηθεί (ή και τα δύο) από μία άλλη.

Μόλις η εκτέλεση μίας συνάρτησης ολοκληρωθεί, όλες οι τοπικές μεταβλητές εξαφανίζονται μέχρις ότου να χρησιμοποιήσουμε αυτή τη συνάρτηση ξανά. Σύμφωνα με το σχήμα, η συνάρτηση “My_Function_1” έχει τις παρακάτω τοπικές μεταβλητές: “Time”, “Hour” και “Result”.

Result

```
void setup()
(
)
```

```
void loop ()
(
My_function_1
(
Time
Hour
Result
)

My_function_2
(
Alarm
Temperature
Data
)

My_function_3
(
Data
Output
Light
)

My_function_4
(
Frequency
Duration
Result
)
}
```

Απαντήστε στις παρακάτω ερωτήσεις:

1. Μπορεί η “My_Function_2” να χρησιμοποιήσει την μεταβλητή “Light”; _____

2. Μπορεί η “My_Function_1” να χρησιμοποιήσει την μεταβλητή “Hour”; _____

3. Μπορεί η “My_Function_4” να χρησιμοποιήσει την μεταβλητή “Data”; _____

Δ. Πράξεις

Σίγουρα θα έχετε μαντέψει ήδη πως τόσο οι παράμετροι που χρησιμοποιούνται στις συναρτήσεις, όσο και οι τιμές που αποθηκεύονται σε μεταβλητές μπορούν να προέρχονται από αποτέλεσμα πράξεων ανάμεσα σε μεταβλητές και σταθερές.

Αυτές είναι οι συχνότερες μαθηματικές πράξεις:

= **Ισοδυναμία (ή ανάθεση τιμής)**

+ **Πρόσθεση**

- **Αφαίρεση**

***** **Πολλαπλασιασμός**

/ **Διαίρεση**

% **Υπόλοιπο (το υπόλοιπο της διαίρεσης μεταξύ δύο ακεραίων)**

Όλοι οι υπολογισμοί πραγματοποιούνται από τα αριστερά προς τα δεξιά. Ο πολλαπλασιασμός και η διαίρεση έχουν μεγαλύτερη προτεραιότητα από την πρόσθεση και την αφαίρεση. Μπορείτε να χρησιμοποιήσετε παρενθέσεις για να καθορίσετε την προτεραιότητα που θέλετε ανάμεσα στις πράξεις. Δείτε τα παρακάτω παραδείγματα:

-> 3 + 5 – 2 = 6

-> 20 – 2 × 3 = 14 και όχι 54

-> (20 - 2) × 3 = 54 και όχι 14

-> 20 / 2 × 3 = 30

-> 24 / (2 × 3) = 4

-> (12 - 2) + 5 × 3 = 25 και όχι 45

-> ((12 - 2) + 5) × 3 = 45 και όχι 25

->13 % 4=1

-> 7 % 5 =2

-> 5 % 5 = 0

Δείτε τα παρακάτω παραδείγματα και προσπαθήστε να απαντήσετε στις ερωτήσεις που ακολουθούν:

```
byte Light=3;

pinMode(Light×2,OUTPUT);

digitalRead(Light+1);

digitalWrite((Light-1)×3),HIGH);
```

Ποιος ακροδέκτης ορίζεται σαν έξοδος; _____

Ποιος ακροδέκτης διαβάζεται; _____

Ποιος ακροδέκτης παίρνει τιμή “1”; _____

```
byte A = 250;

byte B; byte C;

unsigned int D;

A = A + 3;

B = (A - 200) × 2 / 4;

C = A × 2 + 20;

D = (A × 100) % 2;
```

A = _____; B = _____; C = _____; D = _____

2. ΑΝΑΜΟΝΕΣ

Μπορεί αυτό να φαίνεται αντιπαραγωγικό, αλλά πολλές φορές είναι χρήσιμο ένας μικροελεγκτής να «περιμένει» χωρίς να κάνει απολύτως τίποτα.

Η γλώσσα προγραμματισμού του Arduino παρέχει έναν αριθμό συναρτήσεων οι οποίες «παγώνουν» την εκτέλεση του προγράμματος για έναν ορισμένο χρόνο. Μπορεί να μην κατανοείτε ακριβώς το λόγο τώρα, αλλά τα παραδείγματα θα ξεκαθαρίσουν το τοπίο.

A. Η συνάρτηση delayMicroseconds()

«Παγώνει» το πρόγραμμα για ορισμένο χρονικό διάστημα (σε μικροδευτερόλεπτα). Κάθε μιλιδευτερόλεπτο έχει χίλια μικροδευτερόλεπτα, ενώ κάθε δευτερόλεπτο έχει ένα εκατομμύριο μικροδευτερόλεπτα. Θυμηθείτε: 1 δευτερόλεπτο = 1000 μιλιδευτερόλεπτα (mS) and 1 μιλιδευτερόλεπτο = 1000 εκατομμυριοστό του δευτερολέπτου, με δεδομένο πως 1 δευτερόλεπτο = 1,000,000 εκατομμυριοστά του δευτερολέπτου (μS) και 1 μS = 0.000001 δευτερόλεπτα.

Σύνταξη:

delayMicroseconds(n);

n: υποδεικνύει τον αριθμό των μικροδευτερολέπτων για τα οποία θέλετε να αναμένει ο μικροελεγκτής. Είναι μη-προσημασμένος ακέραιος 16-bit (unsigned int). Μέχρι τη συγγραφή του παρόντος, η μεγαλύτερη τιμή που θα παράγει μία αναμονή με ακρίβεια είναι η 16383 (περίπου 16 mS ή 0.016 δευτερόλεπτα). Για μεγαλύτερες αναμονές, προτείνεται η χρήση της **delay()** που θα δούμε αμέσως μετά.

Παραδείγματα:

```
A = 100;

delayMicroseconds(A);  //παγώνει το πρόγραμμα για 100 μS = 0.1 mS = 0.0001 δευτ.
```

delayMicroseconds(A×10-100); //παγώνει το πρόγραμμα για 900 μS = 0.9 mS = 0.0009 δευτ.

Β. Η συνάρτηση delay()

Παγώνει το πρόγραμμα για ορισμένο αριθμό μιλιδευτερολέπτων (χιλιοστών του δευτερολέπτου). Θυμηθείτε πως: 1 δευτερόλεπτο = 1000 μιλιδευτερόλεπτα, ή αλλιώς, 1 μιλιδευτερόλεπτο (mS) = 0.001 δευτερόλεπτα.

Σύνταξη:

delay(n);

n: υποδεικνύει τον αριθμό των μιλιδευτερολέπτων για τα οποία θέλετε να αναμένει ο μικροελεγκτής. Είναι μη-προσημασμένος ακέραιος 32-bit (unsigned int) με εύρος τιμών από 0 έως 4,294,967,295 (2^{32}- 1).

Παραδείγματα:

```
A=100; delay(A);

// «Παγώνει» το πρόγραμμα για 100 mS = 0.1 δευτερόλεπτο

delay(1000)

// «Παγώνει» το πρόγραμμα για 1000 mS = 1 δευτερόλεπτο

delay(A*600);

// «Παγώνει» το πρόγραμμα για 60000 mS = 60 δευτερόλεπτα = 1 λεπτό
```

Θυμηθείτε πως όταν «παγώνετε» το πρόγραμμα χρησιμοποιώντας είτε την συνάρτηση **delayMicroseconds()** ή την **delay()**, ο μικροελεγκτής αναμένει μέχρι να περάσει αυτό το χρονικό διάστημα πριν εκτελέσει οποιαδήποτε επόμενη εντολή.

Γ. Η συνάρτηση micros()

Επιστρέφει τον αριθμό των μικροδευτερολέπτων (μS) που πέρασαν από όταν ξεκίνησε η εκτέλεση του προγράμματος από το Arduino. Επιστρέφει τιμή τύπου unsigned long ενώ ο αριθμός θα υπερχειλίσει (θα γυρίσει πίσω στο μηδέν) μετά από περίπου 70 λεπτά. Θυμηθείτε πως υπάρχουν 1,000 μS (μικροδευτερόλεπτα) σε ένα μιλιδευτερόλεπτο (mS) και 1,000,000 μS σε ένα δευτερόλεπτο.

Σύνταξη:

var = micros();

var: Αυτή είναι η μεταβλητή που θα αποθηκεύσει τον αριθμό των μικροδευτερολέπτων (μS) που πέρασαν από την εκκίνηση της εκτέλεσης του προγράμματος.

Δ. Η συνάρτηση millis()

Επιστρέφει τον αριθμό των μιλιδευτερολέπτων (mS) που πέρασαν από όταν ξεκίνησε η εκτέλεση του προγράμματος από το Arduino. Επιστρέφει τιμή τύπου unsigned long ενώ ο αριθμός θα υπερχειλίσει (θα γυρίσει πίσω στο μηδέν) μετά από περίπου 50 ημέρες. Θυμηθείτε πως υπάρχουν 1,000 μιλιδευτερόλεπτα (mS) σε ένα δευτερόλεπτο.

Σύνταξη:

var millis();

var: Αυτή είναι η μεταβλητή που θα αποθηκεύσει τον αριθμό των μιλιδευτερολέπτων (mS) που πέρασαν από την εκκίνηση της εκτέλεσης του προγράμματος.

3. ΗΧΟΣ

Τι συμβαίνει όταν ρίχνετε μία πέτρα μέσα σε μία λίμνη; Η πέτρα χτυπάει το νερό και προκαλεί αλλαγές στην πίεσή του. Αυτό δίνει εκκινεί μία σειρά κυμάτων που εξαπλώνονται από το σημείο που χτύπησε η πέτρα και σιγά-σιγά χάνονται.

Ο ήχος μοιάζει εξαιρετικά με το παραπάνω. Το μόνο που χρειάζεται είναι μία συσκευή ικανή να παράγει δονήσεις ώστε να προκαλέσει αλλαγές στην πίεση του αέρα και να δημιουργηθούν κύματα. Τα κύματα αυτά, φτάνουν στο τύμπανο του αυτιού μας και ο εγκέφαλός μας τα ερμηνεύει ως ήχο. Όταν μιλάμε, οι φωνητικές μας χορδές είναι υπεύθυνες για την παραγωγή αυτών των αλλαγών στην πίεση του αέρα –κυμάτων- που εν τέλει φτάνουν στο τύμπανο του αυτιού των γύρω μας.

Υπάρχουν πολλές γνωστές συσκευές ικανές να προκαλέσουν αλλαγές στην πίεση του αέρα και άρα να δημιουργήσουν ηχητικά κύματα: ηχεία, ακουστικά, βομβητές, σειρήνες, κλπ.

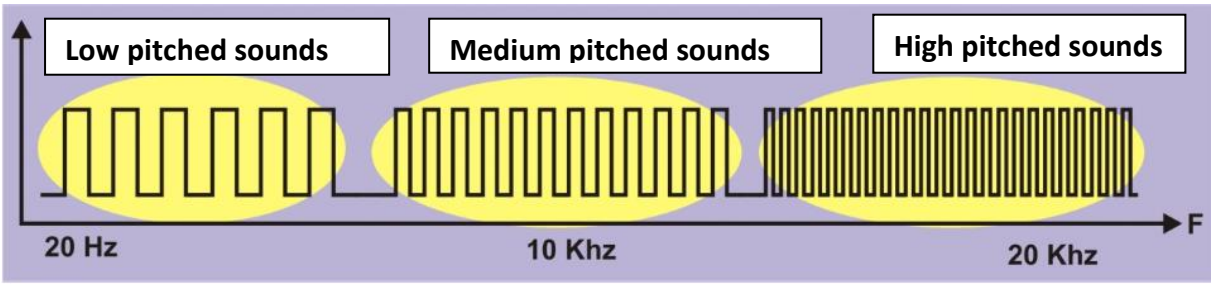
Όλες οι συσκευές αυτές βασίζονται σε τρία κοινά εξαρτήματα: ένα διάφραγμα, ένα πηνίο και ένα μόνιμο μαγνήτη. Το διάφραγμα μπορεί να είναι κατασκευασμένο από χαρτί, χαρτόνι, πλαστικό ή οποιοδήποτε άλλο κατάλληλα επεξεργασμένο υλικό. Το διάφραγμα ενώνεται με το πηνίο, το οποίο με τη σειρά του είναι τοποθετημένου πάνω στο μαγνήτη.

Ένα ηλεκτρικό σήμα εφαρμόζεται στο πηνίο, το οποίο δημιουργεί ένα μαγνητικό πεδίο. Αυτό με τη σειρά του αλληλοεπιδρά με το μαγνητικό πεδίο του μαγνήτη. Τα δύο μαγνητικά πεδία μπορεί να έλκονται, ή να απωθούνται μεταξύ τους. Αυτή η κίνηση ανάμεσα στο πηνίο και το μαγνήτη «τραβάει» ή «σπρώχνει» το διάφραγμα, που βρίσκεται ενωμένο με το πηνίο. Οι κινήσεις του διαφράγματος προκαλούν αλλαγές στην πίεση του αέρα, οι οποίες με τη σειρά τους δημιουργούν τα ηχητικά κύματα. Αυτές οι δονήσεις –κύματα- μάλιστα, είναι ακριβείς αναπαραστάσεις του ηλεκτρικού σήματος που τις δημιούργησε.

Το ηλεκτρικό σήμα σε όλη αυτή τη διαδικασία πρέπει να είναι μεταβαλλόμενο και να εναλλάσσεται ανάμεσα στις τιμές «1» και «0» με συγκεκριμένη ταχύτητα ή συχνότητα. Εάν το σήμα δεν εναλλάσσεται, δεν θα δημιουργηθεί ήχος. Φανταστείτε ένα σήμα που είναι μόνιμα στο «1» ή στο «0» -- το πηνίο θα λάβει μία συγκεκριμένη θέση και έπειτα θα μείνει σταθερό. Το διάφραγμα δεν θα πάλλεται, συνεπώς δε θα δημιουργούνται ηχητικά κύματα.

Οι άνθρωποι είναι ικανοί να αντιληφθούν ηχητικές συχνότητες από περίπου 20 Hz έως και 20,000 Hz (20 KHz) – ή «κύκλους» ανά δευτερόλεπτο. Η συχνότητα αυτή είναι το πόσες φορές κάθε δευτερόλεπτο το ηλεκτρικό σήμα μεταβάλλεται από «1» σε «0» ή αντίστροφα.

Αντιλαμβανόμαστε τις συχνότητες ως «τόνους». Ένας ήχος χαμηλής συχνότητας παράγει ένα βαρύ, χαμηλό τόνο. Όσο υψηλότερη είναι η συχνότητα, τόσο πιο υψηλός και ο τόνος του ήχου.



Οι δονήσεις με συχνότητα χαμηλότερη από 20 Hz ονομάζονται «υπόηχοι» ενώ αυτές άνω των 20 KHz «υπέρηχοι». Οι άνθρωποι γενικά δεν είναι ικανοί να αντιληφθούν είτε τους υπόηχους, ή τους υπέρηχους, ενώ κάποια άλλα είδη τους αντιλαμβάνονται.

Ας κάνουμε μερικούς υπολογισμούς. Υποθέστε ότι θέλουμε να παράγουμε ένα ηχητικό σήμα με συχνότητα 100 Hz:

1. Βασιζόμενοι στο **F**, την επιθυμητή συχνότητα (100 Hz), υπολογίζουμε την περίοδο (**T**), ή αλλιώς, το χρονικό διάστημα που διαρκεί ένας «κύκλος».

$$T = \frac{1}{F} = \frac{1}{100} = 0.01\ s = 10\ mS = 10000\ \mu S$$

2. Πλέον μπορούμε να υπολογίσουμε την ημι-περίοδο (**S**) ή αλλιώς, το χρόνο για τον οποίο το σήμα θα παραμείνει είτε στο επίπεδο «1», ή στο «0».

$$S = \frac{T}{2} = \frac{0.01}{2} = 0.005\ s = 5\ mS = 5000\ \mu S$$

3. Με όσα ήδη γνωρίζουμε, θα μπορούσαμε να δημιουργήσουμε ένα πρόγραμμα το οποίο θα δημιουργεί την επιθυμητή συχνότητα με την οποία θα τροφοδοτούμε το ηχείο που θα έχουμε συνδέσει πάνω στην πλακέτα δοκιμών (breadboard) μας. Δείτε την παρακάτω λύση:

```
void setup()
{
pinMode(2,OUTPUT);                    //Έξοδος στον ακροδέκτη 2 (ηχείο)
}
// Περίοδος (T) για συνολικά 10000 μS. Δηλαδή συχνότητα ίση με 100 HZ .
void loop()
{
digitalWrite(2,HIGH);                 // Ο ακροδέκτης 2 παίρνει τιμή “1”
delayMicroseconds(5000);              // Παύση για 5000 μS
digitalWrite(2,LOW);                  // Ο ακροδέκτης 2 παίρνει τιμή “0”
delayMicroseconds(5000);              // Παύση για δηλαδή 5000 μS
}
```

Σημειώστε πως ο ακροδέκτης 2 παίρνει τιμή «1» (HIGH) για 5000 μS και τιμή «0» (LOW) για το ίδιο διάστημα. Με άλλα λόγια, ο ακροδέκτης εναλλάσσεται ανάμεσα στις τιμές «1» και «0» με περίοδο ("**T**") που διαρκεί για 10000 μS. Μιας και αυτή η διαδικασία επαναλαμβάνεται συνεχώς, σε κάθε δευτερόλεπτο έχουμε 100 «κύκλους» (1000000/10000). Αυτή είναι η συχνότητα **F**.

«Ανεβάστε» το πρόγραμμα και βεβαιωθείτε ότι λειτουργεί σωστά. Θα πρέπει να παράγεται ένας συνεχόμενος (και ενοχλητικός) ήχος με συχνότητα 100 Hz.

Τροποποιήστε το πρόγραμμα ώστε να παράγει ήχους με διαφορετικούς τόνους (συχνότητες). Υπολογίστε τις περιόδους τους (**T**) και τις ημι-περιόδους τους (**S**) χρησιμοποιώντας τον παρακάτω πίνακα.

F	T	S	F	T	S
50 Hz			2.5 KHz		
200 Hz			5 KHz		
500 Hz			10 KHz		
1 KHz			15 KHz		
2 KHz			20 KHz		

4. ΣΥΝΑΡΤΗΣΕΙΣ ΗΧΟΥ

Η γλώσσα προγραμματισμού του Arduino κάνει πολύ εύκολη τη δημιουργία διαφόρων ήχων. Δε χρειάζεται να υπολογίσετε είτε την περίοδο (**T**) ή την ημι-περίοδο (**S**). Το μόνο που χρειάζεται είναι να υποδείξετε τη συχνότητα (**F**) και το Arduino θα αναλάβει τα υπόλοιπα.

Α. Η συνάρτηση tone()

Παράγει έναν ήχο με ορισμένη συχνότητα (**F**) στον υποδειχθέντα ακροδέκτη και για δοθείσα διάρκεια.

Σύνταξη:

tone(pin, frequency, duration);

pin: *Υποδεικνύει τον ακροδέκτη πάνω στον οποίο θα δημιουργηθεί ο ήχος.*

frequency: *Μη-προσημασμένος ακέραιος (unsigned int) που αναπαριστά τη συχνότητα του ήχου σε Hz (χερτζ). Θυμηθείτε πως ο άνθρωπος είναι ικανός να ακούσει ήχους ανάμεσα σε 20 και 20,000 Hz (ή «κύκλους» ανά δευτερόλεπτο).*

duration: *Προαιρετική παράμετρος. Μη-προσημασμένος αριθμός (unsigned long) που αναπαριστά τη διάρκεια του ήχου σε μιλιδευτερόλεπτα. Εάν αυτή η παράμετρος δεν οριστεί, ο ήχος θα παράγεται συνεχόμενα έως την κλήση της συνάρτησης noTone().*

Παραδείγματα:

```
int Pin=11;
int F = 1200;
int D = 3000;
tone(2,F,D);    // Δημιουργεί ένα τόνο 1200 Hz για 3 δευτερόλεπτα στον ακροδέκτη 2.
tone(6,F*10,500); // Δημιουργεί ένα τόνο 12 KHZ για 0.5 δευτερόλεπτα στον ακροδέκτη 2.
tone(Pin,200);          //Δημιουργεί έναν τόνο 200 Hz.
```

Β. Η συνάρτηση noTone():

Σταματάει την παραγωγή ήχου στον υποδειχθέντα ακροδέκτη.

Σύνταξη:

noTone(pin);

pin: *Υποδεικνύει τον ακροδέκτη στον οποίο θα σταματήσει ο ήχος.*

Παράδειγμα:

```
tone(2,13000);        // Παράγει έναν τόνο 13 KHz στον ακροδέκτη 2.
....
....                  // Ο τόνος συνεχίζει να παράγεται.
....
noTone(2);            // Ο τόνος σταματάει να παράγεται.
```

ΕΝΟΤΗΤΑ ΕΞΑΣΚΗΣΗΣ

1. ΠΑΡΑΔΕΙΓΜΑ 1: Φώτα που αναβοσβήνουν

Αυτό είναι από τα απλούστερα προγράμματα που μπορούμε να γράψουμε: ένα λευκό LED συνδεδεμένο στον ακροδέκτη 6. Δείτε τη λύση στο πλαϊνό σχήμα.

Η μεταβλητή “Time” περιέχει το χρόνο της παύσης, 500 mS = 0.5 δευτερόλεπτα. Ο ακροδέκτης 6, στον οποίο έχουμε συνδέσει το λευκό LED, είναι ορισμένος ως έξοδος.

Το κύριο σώμα του προγράμματος που περιέχεται στη συνάρτηση **loop()** περιορίζεται στο να αναβοσβήνει τον ακροδέκτη 6 στο διάστημα που περιέχει η μεταβλητή “*Time*”, 500 mS.

- **Η σειρά σας**

Μπορείτε να πραγματοποιήσετε αρκετές τροποποιήσεις στο παραπάνω παράδειγμα. Μερικές προτάσεις είναι:

1. Να δοκιμάσετε να μικρύνετε το χρονικό διάστημα της παύσης χρησιμοποιώντας τη μεταβλητή “*Time*”. Μειώστε το αρκετά φτάνοντας έως τα 10 mS. Τι παρατηρείτε;

   Φαίνεται το LED να μένει αναμμένο συνέχεια, αλλά στην πραγματικότητα δε συμβαίνει αυτό. Το LED συνεχίζει να αναβοσβήνει, αλλά πάρα πολύ γρήγορα. Τόσο γρήγορα που ο αμφιβληστροειδής του ματιού μας δεν καταφέρνει να αντιληφθεί την

εναλλαγή. Έτσι νομίζουμε πως το LED είναι αναμμένο συνεχώς.

Παρ' όλα αυτά, αυτό που δείχνει συνεχόμενο σε εμάς, δεν είναι το ίδιο και για το μικροελεγκτή. Το Arduino συνεχίζει να ανάβει το LED για 10 mS και να το κρατά σβηστό για άλλα 10 mS, απλώς δεν ήμαστε ικανοί να το αντιληφθούμε.

2. Αντί να χρησιμοποιήσετε τη συνάρτηση **delay()** δοκιμάστε να κάνετε το ίδιο με την **delayMicroseconds()**. Με αυτή τη συνάρτηση, μπορείτε να δημιουργήσετε εξαιρετικά μικρές και ακριβείς αναμονές, της τάξης των εκατομμυριοστών του δευτερολέπτου (μS).

3. Μπορείτε επίσης να τροποποιήσετε το πρόγραμμα έτσι ώστε να προσομοιώνει ένα φάρο. Για παράδειγμα, μπορείτε να κάνετε το LED να ανάβει για 0.1 δευτερόλεπτα κάθε 2 δευτερόλεπτα.

2. ΠΑΡΑΔΕΙΓΜΑ 2: Ομάδα από φώτα

Ένα ωραίο παράδειγμα. Τα τέσσερα LED στην πλακέτα δοκιμών (breadboard) θα ανάψουν και θα σβήσουν, το ένα μετά το άλλο, για την ίδια χρονική διάρκεια. Θα μας δώσει μία αίσθηση κίνησης από τα δεξιά προς τα αριστερά.

Το πρόγραμμα είναι λίγο μεγάλο, αλλά είναι απλό. Ανάβει το λευκό LED, κάνει παύση, το σβήνει, και μετά ανάβει το πράσινο. Μετά κάνει παύση, το σβήνει, και ανάβει το πορτοκαλί. Μετά κάνει παύση, το σβήνει, και ανάβει το κόκκινο. Τέλος, κάνει παύση, το σβήνει και ο κύκλος ξεκινάει από την αρχή.

- **Η σειρά σας**

1. Αλλάξτε το χρόνο: η ιδέα είναι οι εναλλαγές να γίνονται πιο γρήγορα, ή πιο αργά, καθώς μετακινείστε από δεξιά προς τα αριστερά.

2. Κάντε τα LED να ανάβουν πρώτα από τα δεξιά προς τα αριστερά και έπειτα από τα αριστερά προς τα δεξιά.

3. ΠΑΡΑΔΕΙΓΜΑ 3: Φανάρι κυκλοφορίας (V1)

Πως θα σας φαινόταν να κάνουμε ένα μικρό project με όσα μάθαμε; Μία ιδέα είναι η προσομοίωση ενός φαναριού κυκλοφορίας, το οποίο θα ξεκινήσουμε τώρα και θα βελτιώσουμε στην πορεία.

Το πρόγραμμα δεν έχει ουσιαστικά κάτι καινούριο. Περιλαμβάνει άναμμα του πράσινου, πορτοκαλί και κόκκινου LED με τη σωστή σειρά, για το χρόνο που πρέπει. Υπάρχει όμως μία ειδοποιός διαφορά…

- **Συμπέρασμα**

Παρατηρείστε πως όλες οι λειτουργίες που απαρτίζουν το πρόγραμμα που προσομοιώνει το φαναριού περιλαμβάνονται στη συνάρτηση **setup()**. Η κύρια συνάρτηση, η **loop()**, είναι άδεια.

Αυτό είναι απόλυτα φυσιολογικό όπως είδαμε και σε προηγούμενα κεφάλαια. Πρέπει, φυσικά, να περιλάβουμε τη συνάρτηση **loop()** ακόμα και αν δεν επιτελεί κάποια λειτουργία. Από την άλλη, ο κώδικας που υπάρχει στη συνάρτηση **setup()** εκτελείται μία μόνο φορά. Έτσι, κάθε φορά που θέλετε το φανάρι να εκτελέσει την ακολουθία, θα πρέπει να πατήσετε το πλήκτρο **RESET** στο Arduino. Δοκιμάστε το.

4. ΠΑΡΑΔΕΙΓΜΑ 4: Κουδούνι

Ένα ακόμα απλό project. Αυτή τη φορά θα προσομοιώσουμε ένα ηλεκτρικό κουδούνι. Κάθε φορά που πατάμε το πλήκτρο, το κουδούνι θα παράγει έναν ήχο που αποτελεί από τρεις διαφορετικούς τόνους.

Αυτή θα είναι η πρώτη φορά που θα χρησιμοποιήσουμε τη συνάρτηση **tone()**. Θα παρατηρήσετε ότι η χρήση της είναι εξαιρετικά απλή. Το μόνο που έχετε να κάνετε, είναι να ορίσετε τον ακροδέκτη εξόδου, τη συχνότητα και τη διάρκεια (ο ορισμός της διάρκειας είναι προαιρετικός).

Η υλοποίηση του προγράμματος παρουσιάζεται στο διπλανό σχήμα. Αρχικά, παράγεται ένας τόνος 1000 Hz για 0.2 δευτερόλεπτα. Έπειτα, ύστερα από μία παύση 0.4 δευτερολέπτων, παράγεται ένας δεύτερος τόνος των 15,000 Hz για 0.3 δευτερόλεπτα. Τέλος, ύστερα από ακόμα μία παύση 0.4 δευτερολέπτων, παράγεται ο τρίτος και τελευταίος τόνος των 2,000 Hz για 0.4 δευτερόλεπτα.

Παίρνουμε τους τόνους στον ακροδέκτη 2 του Arduino.

Σημειώστε πως όλες οι λειτουργίες περιλαμβάνονται στη συνάρτηση **setup()**, οπότε η ακολουθία εκτελείται κάθε φορά που πιέζουμε το πλήκτρο **RESET**.

- **Η σειρά σας**

Βελτιώστε το πρόγραμμα προσθέτοντας ένα ενδεικτικό LED. Το λευκό LED θα πρέπει να ανάβει στην αρχή της ακολουθίας και να σβήνει αφού ολοκληρωθεί η παραγωγή όλων των τόνων.

5. ΠΑΡΑΔΕΙΓΜΑ 5: Μελωδίες

Σε αυτό το παράδειγμα θα συνδυάσουμε μία σειρά από μουσικές νότες έτσι ώστε να δημιουργήσουμε μία γνωστή και διαδεδομένη μελωδία. Οι μουσικές νότες αναπαριστούν διαφορετικές ηχητικές συχνότητες. Οι συνδυασμοί τους, παράγουν τις διέσεις και υφέσεις. Δοκιμάστε το παράδειγμα και στη συνέχεια, δοκιμάστε να παράγετε διαφορετικές μελωδίες!

6. ΠΑΡΑΔΕΙΓΜΑ 6: Φανάρι κυκλοφορίας (V2)

Ακόμα ένα μικρό project. Είναι μια βελτιωμένη έκδοση του φαναριού κυκλοφορίας που κατασκευάσαμε προηγουμένως η οποία προσθέτει ακουστικές ειδοποιήσεις για όσους έχουν μειωμένη όραση.

Το πρόγραμμα ίσως να φαίνεται κάπως μεγάλο, αλλά με προσεκτικότερη εξέταση θα διαπιστώσετε πως δεν είναι ιδιαίτερα περίπλοκο. Είναι ιδιαιτέρως γραμμικό. Ξεκινάει φωτίζοντας το πράσινο LED ενώ ταυτόχρονα παράγει έξι τόνους των 1,000 Hz διάρκειας 0.5 δευτερολέπτων έκαστος. Έπειτα, φωτίζει το πορτοκαλί LED ενώ ταυτόχρονα παράγει έξι τόνους των 1,000 Hz διάρκειας 0.4 δευτερολέπτων έκαστος. Τέλος, το κόκκινο LED φωτίζεται, ενώ παράγεται συνεχόμενος τόνων 6 δευτερολέπτων στα 1,000 Hz.

Καθώς όλες οι λειτουργίες του προγράμματος περιλαμβάνονται στη συνάρτηση **setup()** εκτελούνται μονάχα μία φορά: κάθε φορά που πιέζετε το πλήκτρο **RESET**.

- **Η σειρά σας**

Παρατηρείστε ένα φανάρι κυκλοφορίας στην περιοχή σας και μετρήστε το χρόνο που μένει αναμμένη κάθε λυχνία. Παρατηρείστε επίσης πιθανούς προειδοποιητικούς ήχους που μπορεί να παράγει και το κατά πόσο αυτοί είναι συνεχόμενοι ή διακοπτόμενοι. Η ιδέα είναι να τροποποιήσετε το πρόγραμμα έτσι ώστε να προσομοιώνει ένα πραγματικό φανάρι δρόμου, με όσο το δυνατόν πιο ρεαλιστικό τρόπο.

ΠΑΡΑΠΟΜΠΕΣ

ΒΙΒΛΙΑ

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. **Practical ARDUINO**, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. **Programing Arduino, Next Steps,** Simon Monk
[4]. **Sensores y actuadores, Aplicaciones con ARDUINO,** Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. **ARDUINO: Curso práctico de formación,** Oscar Torrente Artero, Ed.RC libros
[6]. **30 ARDUINO PROJECTS FOR THE EVIL GENIUS,** Simon Monk, Ed. TAB
[7]. **Beginning C for ARDUINO,** Jack Purdum, Ph.D., Ed.Technology in action
[8]. **ARDUINO programing notebook,** Brian W.Evans

ΙΣΤΟΣΕΛΙΔΕΣ

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com