Εισαγωγή στον κόσμο των μικροελεγκτών (Arduino)

Σεβδυνίδης Μιχάλης
Φυσικός Ρ/Η

Περιεχόμενα

Τι είναι ο μικροελεγκτής;

Ο μικροελεγκτής (microcontroller) είναι ένας μικρός υπολογιστής, τον οποίο μπορούμε να προγραμματίσουμε ώστε να επικοινωνεί με το περιβάλλον του, δηλαδή να δέχεται πληροφορίες (εισόδους - inputs) από αισθητήρες (sensors) και να δίνει εντολές (εξόδους - outputs) σε συστήματα, όπως κινητήρες.

Ότι κάνει ένας μικροελεγκτής μπορεί να κάνει συνήθως κι ένα ηλεκτρονικό κύκλωμα. Η διαφορά είναι ότι το ηλεκτρονικό κύκλωμα του μικροελεγκτή μπορεί να προγραμματιστεί, ώστε να κάνει κάθε φορά μια άλλη εργασία, ενώ διαφορετικά θα χρειαζόταν ένα κύκλωμα αποκλειστικά για κάθε εργασία με σοβαρές συνέπειες στο κόστος και στο χρόνο, που χρειάζεται να αφιερώσει κάποιος.

Τι είναι ο μικροελεγκτής Arduino (και γιατί...Arduino);

Ο μικροελεγκτής Arduino είναι μια ευέλικτη πλατφόρμα hardware που σχεδιάστηκε για ανθρώπους χωρίς ιδιαίτερες γνώσεις προγραμματισμού Η/Υ και ηλεκτρονικών. Οι απαιτούμενες γνώσεις αποκτιούνται σταδιακά κατά τη χρήση του συστήματος. Έτσι το σύστημα αυτό είναι κατάλληλο και για ανθρώπους, που τους αρέσει απλώς να μαστορεύουν ή ακόμη και καλλιτέχνες.

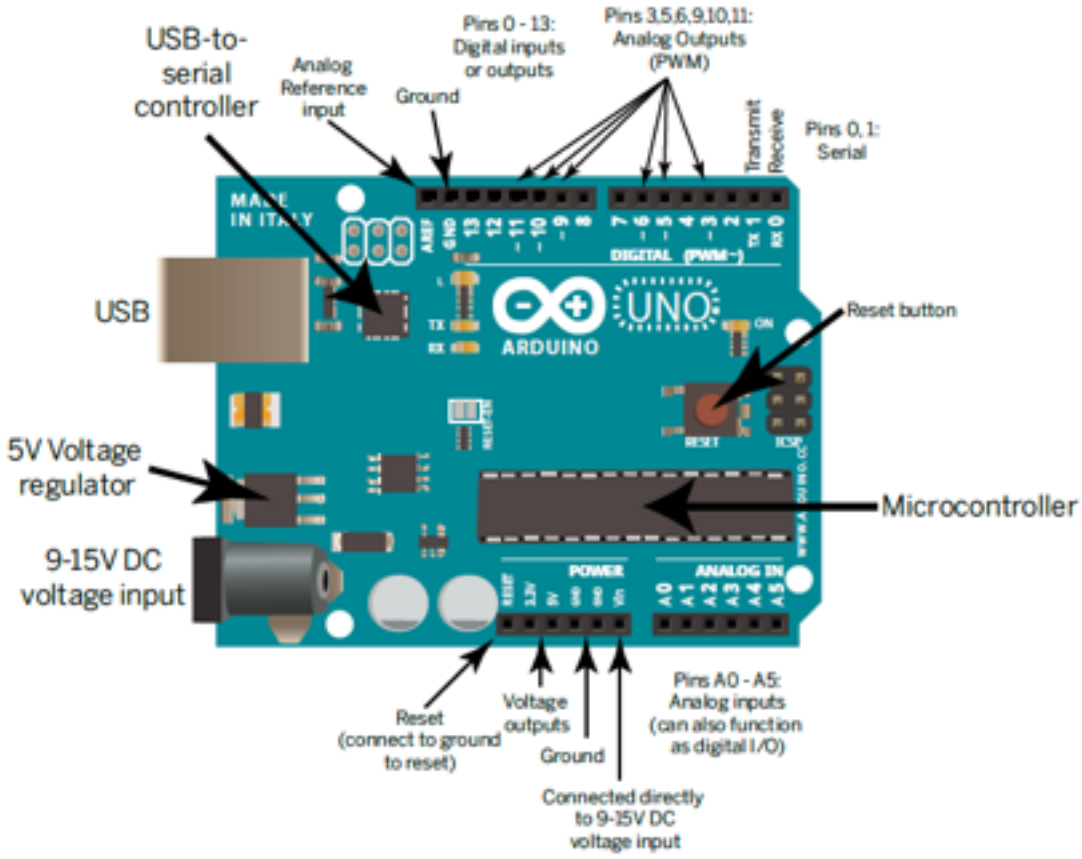
Η γνωστή μπλε πλακέτα του κυκλώματος του Arduino λέγεται ότι πήρε το όνομά της από μια τοπική καφετέρια της Ιταλίας και λόγω της απλότητάς της έχει δημιουργήσει μια ολόκληρη γενιά από «μάστορες», κάτι που φαίνεται από το πλήθος των σχετικών αναφορών και forums, που μπορεί εύκολα να βρει κάποιος στο διαδίκτυο.

Αυτό που έχει κάνει δημοφιλές το συγκεκριμένο σύστημα, πέρα από την απλότητά του, είναι ότι χαρακτηρίζεται ως hardware ανοικτού κώδικα (open source hardware), που σημαίνει ότι ο κατασκευαστής έχει δώσει τη δυνατότητα οποιοδήποτε χρήστης να έχει πρόσβαση στο σχέδιο του βασικού κυκλώματος, ώστε να μπορεί να κάνει βελτιώσεις και να τις γνωστοποιεί στους ενδιαφερόμενους. Γενικότερα όλα τα αρχεία με σχέδια κυκλωμάτων και το λογισμικό είναι ελεύθερα να τα «κατεβάσει» κάποιος στον υπολογιστή του, να τα χρησιμοποιήσει, τροποποιήσει ακόμα και να τα πουλήσει!

Τι έχει «μέσα» ο Arduino;

Στο κέντρο της πλακέτας υπάρχει ένα τσιπάκι που πρέπει να το βλέπουμε σαν ένα υπολογιστή! Το τσιπάκι αυτό είναι γνωστό ως AVR (το κατασκευάζει η εταιρεία Atmel). Είναι πολύ αργός δηλαδή «τρέχει» στα

16MHz με 8-bit core. Η διαθέσιμη μνήμη είναι 32 kilobytes με 2 kilobytes RAM.
Παρακάτω βλέπετε την πλατφόρμα του Arduino Uno (ένας από τη οικογένεια των μικροελεγκτών Arduino). Το σχέδιο δίνει όλες τις βασικές πληροφορίες, που μπορεί κάποιος να χρησιμοποιήσει για να κάνει το κύκλωμα ικανό να επικοινωνεί με το περιβάλλον του.



Δίπλα βλέπετε την ποικιλία των μικροελεγκτών Arduino, η επιλογή καθενός έχει να κάνει βεβαίως με τη χρήση δηλαδή το project στο οποίο θα ενταχθεί.

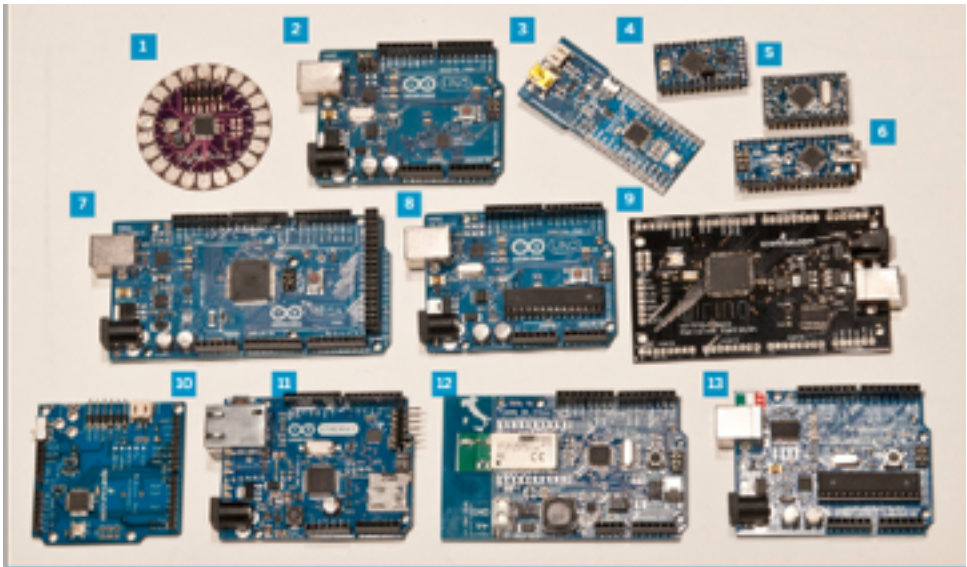


Figure 1-8. Varieties of Arduino, as well as a Wiring board: 1. LilyPad Arduino 2. Arduino Uno SMD 3. Arduino Fio 4. Arduino Pro Mini 5. Arduino Mini 6. Arduino Nano 7. Arduino Mega 2560 8. Arduino Uno 9. Wiring board 10. Arduino Pro 11. Arduino Ethernet 12. Arduino Bluetooth 13. Arduino Duemilanove.

Πως μπορεί να προγραμματιστεί ο Arduino;

Αρχικά πρέπει να εγκατασταθεί στον Η/Υ κατάλληλο λογισμικό (free downloading) ανάλογα με το υπάρχον λειτουργικό σύστημα, ώστε όταν συνδεθεί ο Arduino με αυτόν, να αναγνωριστεί και να μπορεί να επικοινωνήσει μαζί του. Στη συνέχεια ο προγραμματισμός γίνεται στον Η/Υ και τέλος το πρόγραμμα μεταφέρεται στον Arduino, οπότε ο τελευταίος λειτουργεί σύμφωνα με αυτό.

Εγκατάσταση του Arduino στον Η/Υ

Εγκατάσταση σε *Windows 7*

Πρώτα απ'όλα «κατεβάζουμε» στον υπολογιστή μας το κατάλληλο αρχείο που διατίθεται ελεύθερα από την ιστοσελίδα του Arduino και είναι συμπιεσμένο (zipped), οπότε το αποσυμπιέζουμε (unzip). Το **Program Files** είναι ένα καλό μέρος για το αποσυμπιεσμένο αρχείο. Στη συνέχεια πρέπει να εγκατασταθούν οι drivers, όποιο μικροελεγκτή και να διαθέτουμε.

Συνδέουμε τον Arduino και περιμένουμε τα Windows να ξεκινήσουν τη διαδικασία εγκατάστασης των drivers. Για τον Arduino Uno ή νεότερο πατάμε Start Menu και ανοίγουμε το Control Panel. Στη συνέχεια ανοίγουμε το tab 'System and Security'. Πατάμε στο System και ανοίγουμε το Device Manager. Στο Ports(COM&LPT) θα δούμε ένα port με όνομα Arduino UNO(COMxx). Κάνουμε δεξί κλικ σ' αυτό και επιλέγουμε Update Driver software. Μετά επιλέγουμε "Browse my computer for Driver software". Οδηγούμε με τις επιλογές μας και επιλέγουμε το αρχείο με όνομα ArduinoUNO.inf το οποίο βρίσκεται στο directory των drivers. Τα windows θα συνεχίσουν την εγκατάσταση από το σημείο αυτό.

Ορολογία Arduino (σύντομη)

Digital Output (ψηφιακή έξοδος)

Τη χρησιμοποιούμε για να ελέγξουμε ένα LED, αλλά με κατάλληλο κύκλωμα μπορεί να χρησιμοποιηθεί για να ελέγχει κινητήρες (έτσι λειτουργεί ένα ρομπότ), να παράγει ήχους και πολλά άλλα.

Analog Output (αναλογική έξοδος)

Αυτή μας δίνει τη δυνατότητα να ελέγχουμε τη φωτεινότητα ενός LED και όχι απλώς να το αναβοσβήνουμε. Μπορούμε ακόμα να ελέγχουμε την ταχύτητα ενός κινητήρα.

Digital Input (ψηφιακή είσοδος)

Αυτή μας επιτρέπει να «διαβάζουμε» σήματα απλών αισθητήρων (sensors) και διακοπτών(pushbuttons).

Analog Input (αναλογική είσοδος)

Μπορούμε να δεχτούμε σήματα από αισθητήρες, οι οποίοι στέλνουν συνεχές σήμα που δεν είναι σταθερό (δηλαδή «δεν υπάρχει» ή όταν υπάρχει να μην είναι το ίδιο κάθε φορά - on or off), αλλά τέτοιο που έρχεται από ένα ποτενσιόμετρο ή αισθητήρα φωτός.

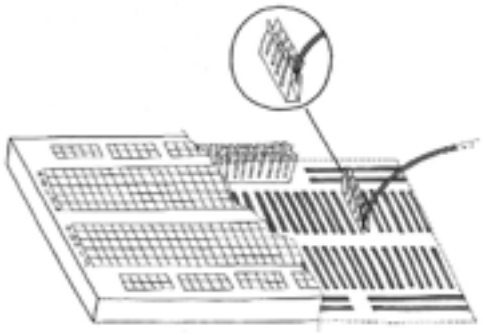
Τις παραπάνω εισόδους και εξόδους μπορείτε να βρείτε στη φωτογραφία ενός μικροελεγκτή Arduino στη σελίδα 4.

Serial Communication (σειριακή επικοινωνία)

Αυτή επιτρέπει στον μικροελεγκτή να επικοινωνεί με τον Η/Υ και να ανταλλάσει δεδομένα (data) ή απλά να δείχνει στην οθόνη τι συμβαίνει το πρόγραμμα (sketch) που «τρέχει» στον μικροελεγκτή.

Τι είναι το breadboard;

Το breadboard είναι μια βάση πάνω στην οποία γίνονται πρωτότυπα κυκλώματα ηλεκτρονικών χωρίς συγκολλήσεις (soldering). Επειδή αποφεύγονται οι συγκολλήσεις το breadboard μπορεί να χρησιμοποιείται συνεχώς για νέα κυκλώματα.



Στις παραπάνω εικόνες φαίνεται (αριστερά) πως είναι ένα απλό breadboard και (δεξιά) πως επικοινωνούν εσωτερικά οι τρύπες που φαίνονται στην επιφάνεια του breadboard.

Τι είναι το SHIELD ενός ARDUINO;

Τα Shields είναι πλακέτες με κυκλώματα (add-on modules) που συνδέονται πολλαπλά («κουμπώνουν» πάνω του) με έναν μικροελεγκτή Arduino σε τρόπο ώστε να αυξάνουν τις δυνατότητες επικοινωνίας του με το περιβάλλον. Τα κυκλώματα αυτά είναι ένας από τους λόγους της διάδοσης του συγκεκριμένου μικροελεγκτή. Στην παρακάτω εικόνα φαίνονται τέτοια shields.

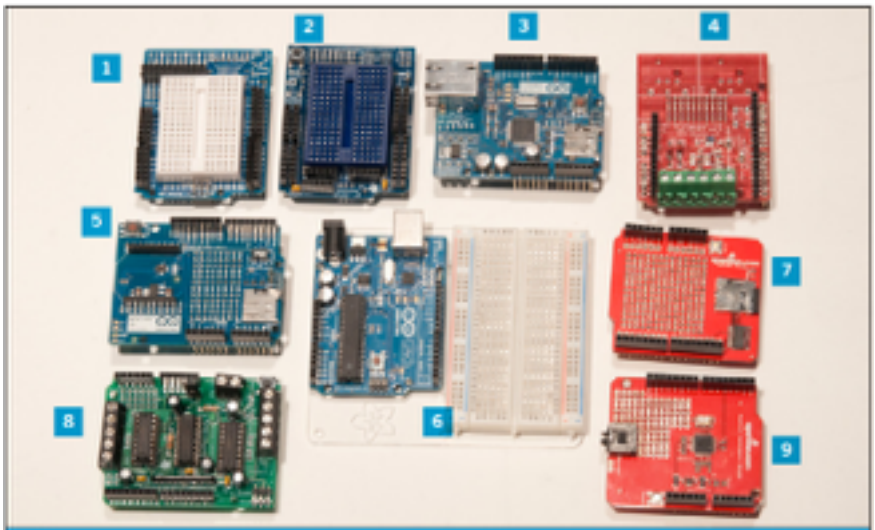
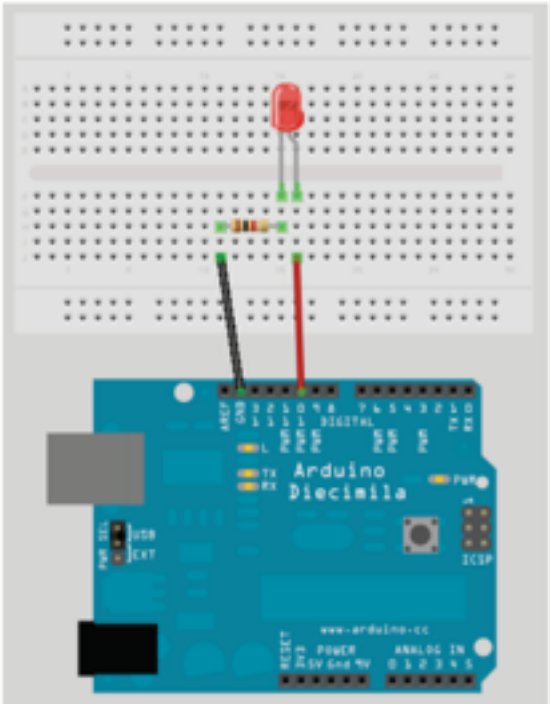


Figure 1-9. A sampling of shields for Arduino: 1. Arduino prototyping shield 2. Adafruit prototyping shield 3. Arduino Ethernet shield 4. TinkerKit DMX shield 5. Arduino wireless shield 6. Oomlout Arduino/breadboard mount, manufactured by Adafruit 7. Spark Fun microSD card shield 8. Adafruit motor driver shield 9. Spark Fun musical instrument shield.

Το πρώτο μου Arduino project:

Το LED που αναβοσβήνει!

Για το project αυτό θα χρειαστούμε *ένα breadboard, ένα LED, μία αντίσταση 220Ω και καλώδια.*



Η συνδεσμολογία που θα γίνει θα φαίνεται όπως στη διπλανή εικόνα. Χρειάζεται προσοχή για τον τρόπο σύνδεσης του LED. Το LED έχει δύο ποδαράκια, από τα οποία το οποία το μεγάλο πρέπει να έχει το μεγάλο δυναμικό (π.χ. το θετικό πόλο της μπαταρίας) και το μικρότερο το μικρότερο δυναμικό (π.χ. τον αρνητικό πόλο της μπαταρίας ή γείωση). Έτσι στην περίπτωσή μας πρέπει το μεγάλο ποδαράκι να συνδεθεί με την ψηφιακή έξοδο του arduino (αριθμό 10), που ουσιαστικά σημαίνει ότι τροφοδοτείται με +5V. Επίσης πρέπει πάντα να συνδέεται το LED με μια αντίσταση σε σειρά ώστε να μην μπορεί να περάσει μεγάλη ένταση ρεύματος που θα μπορούσε να βλάψει το κύκλωμα.

Ανοίγετε το Arduino IDE και πληκτρολογείτε τον παρακάτω κώδικα (Arduino **Sketch**).

Μετά την πληκτρολόγηση πατάτε *Verify/Compile* που θα βρείτε πάνω και αριστερά στο περιβάλλον του IDE. Αν έχουν γίνει όλα σωστά θα δείτε το LED να αναβοσβήνει κάθε δευτερόλεπτο.

Για να δούμε αναλυτικά τον κώδικα και πως λειτουργεί. Στην πρώτη γραμμή βλέπουμε:

```
// Project 1 - LED Flasher

int ledPin = 10;

void setup() {
      pinMode(ledPin, OUTPUT);
}

void loop() {
      digitalWrite(ledPin, HIGH);
      delay(1000);
      digitalWrite(ledPin, LOW);
      delay(1000);
}
```

// Project 1 - LED Flasher

Όταν μια γραμμή ξεκινάει με //, τότε ότι ακολουθεί δεν «μεταφράζεται» από τον compiler σε γλώσσα μηχανής. Πρακτικά μπορείτε να γράψετε ότι σχόλιο θέλετε! Συνήθως γράφουμε κάτι που θα μας βοηθήσει να θυμόμαστε τη λογική του κώδικά μας. Εναλλακτικά μπορεί κάποιος να περιλάβει σχόλια μεταξύ των συμβόλων /* και */. Για παράδειγμα:

/ καλή αρχή! */*

Στην επόμενη γραμμή διαβάζουμε:

int ledPin = 10;

Αυτό είναι που ονομάζουμε μεταβλητή (variable). Στην ουσία είναι το μέρος όπου αποθηκεύουμε δεδομένα. Εδώ συγκεκριμένα ορίζουμε το χώρο,

όπου θα αποθηκεύσουμε μεταβλητή τύπου int ή integer δηλαδή ακέραιο αριθμό που όμως κυμαίνεται μεταξύ των τιμών -32.768 και 32.767. Έτσι η μεταβλητή ledPin δεν μπορεί να πάρει τιμή π.χ. 40000. Αλλά προφανώς δεν θα χρειαστεί! Οι μεταβλητές είναι πολύ σημαντικό στοιχείο του προγραμματισμού ειδικά στον Arduino όπου τα διαθέσιμα bytes στης μνήμης είναι μετρημένα! Για πλήρη ενημέρωση για τις μεταβλητές διάβασε τις σελίδες 12 και 13.

Ορίσατε λοιπόν μια περιοχή στη μνήμη στην οποία έχετε αποθηκεύσει ένα αριθμό τύπου int και συγκεκριμένα τον αριθμό 10. Φανταστείτε τη μεταβλητή σαν ένα κουτί στο οποίο φυλάγετε πράγματα. Τώρα είναι ο αριθμός 10. Μετά όμως μπορείτε να βάλετε κάτι άλλο. Γι'αυτό ονομάζεται μεταβλητή. Μεταβάλλεται αυτό που φυλάγετε!

Στην επόμενη γραμμή υπάρχει τη *συνάρτηση setup():*

```
void setup() {
        pinMode(ledPin, OUTPUT);
        }
```

Σε ένα πρόγραμμα για τον Arduino (sketch) πρέπει πάντα να υπάρχει μια συνάρτηση *setup*() και μια *loop*(), διαφορετικά δεν λειτουργεί.

Η συνάρτηση *setup()* «τρέχει» μια φορά στην εκκίνηση του προγράμματος. Σε αυτήν υπάρχουν γενικές οδηγίες όπως να οριστούνε pin modes ή serial baud rates (σηριακός ρυθμός μεταφοράς δεδομένων) για να προετοιμαστεί το πρόγραμμα πριν «τρέξει» η βασική συνάρτηση *loop()*. Μέσα σε αυτή τη συνάρτηση μπορεί να καλούνται άλλες συναρτήσεις οι οποίες είτε βρίσκονται πιο κάτω στον κώδικα, είτε βρίσκοται σε βιβλιοθήκες (libraries) οι οποίες είναι ελεύθερα διαθέσιμες από το διαδίκτυο και αποθηκεύονται σε ομώνυμο directory.

Στην *setup()* υπάρχει μόνο μία δήλωση. Με αυτήν λέμε ότι το pin με αριθμό 10 (η τιμή του *ledPin* δηλαδή) ορίζεται ως έξοδος (το λογισμικό του arduino καταλαβαίνει ως έξοδο ότι χαρακτηρίζεται OUTPUT και ως είσοδο ότι χαρακτηρίζεται INPUT). Η μεταβλητή *pinMode* χρησιμοποιείται γι'αυτή τη δήλωση.

Τι είναι όμως συνάρτηση (function); Είναι ένα σύνολο από γραμμές κώδικα που κάνει μια συγκεκριμένη δουλειά. Αν για παράδειγμα έχουμε κάνει μια σειρά από γραμμές κώδικα που κάνουν κάποιους συγκεκριμένους μαθηματικούς υπολογισμούς, αυτοί μπορούν να μπουν κάτω από τον ορισμό μιας συνάρτησης. Όποτε μέσα στο πρόγραμμα χρειάζεται να γίνουν οι υπολογισμοί αυτοί δε χρειάζεται να γράψουμε ξανά τις ίδιες γραμμές εντολών αλλά μπορούμε να καλούμε τη συνάρτηση.

Ας δούμε ένα παράδειγμα συνάρτησης και το νόημα των γραφόμενων:

```
int myMultiplyFunction(int x,int y)
{
int result;

result = x * y;
return result;
}
```

Η παραπάνω συνάρτηση με όνομα *myMultiplyFunction* επιστρέφει μια τιμή στο όνομά της σα να είναι η ίδια μεταβλητή και μάλιστα ακέραιη (αφού

γράφεται int μπροστά από το όνομά της). Η συνάρτηση δέχεται ως δεδομένα (εισόδους) δύο ακέραιους (ή καλύτερα τύπου int) δηλαδή τους x και y. Ότι υπάρχει μεταξύ των αγκύλων {...} είναι όλη η επεξεργασία που εκτελεί η συνάρτηση. Η μεταβλητή result που ορίζεται μέσα στη συνάρτηση είναι γνωστή μόνο στη συνάρτηση και θα ήταν ακατανόητη (θα έβγαζε λάθος ο compiler!) στο υπόλοιπο πρόγραμμα.

Όταν μπροστά από το όνομα της συνάρτησης υπάρχει ο προσδιορισμός *void* (αντί για το *int* στην περίπτωση της *myMultiplyFunction*) τότε η συνάρτηση δεν επιστρέφει κάποια τιμή. Παραπάνω το είδαμε στη συνάρτηση *setup()*.

Μετά τη συνάρτηση *setup()* έρχεται η σειρά να «τρέξει» τη συνάρτηση *loop()*. Σε αυτή, η πρώτη εντολή που βλέπουμε είναι η ακόλουθη:

digitalWrite(ledPin,HIGH);

Με την digitalWrite στην πραγματικότητα λέμε στον arduino να θέσει το pin με αριθμό ledPin δηλαδή 10 σε κατάσταση υψηλού δυναμικού που είναι 5V. Μέχρι πότε όμως θα είναι το LED αναμμένο; Την απάντηση δίνει η εντολή που ακολουθεί:

delay(1000);

Η *delay* λέει στο μικροελεγκτή να καθυστερήσει την εκτέλεση της επόμενης εντολής του κώδικα κατά 1000 μονάδες. Η μονάδα που καταλαβαίνει είναι 1/1000 του δευτερολέπτου ή 1 msec. Έτσι 1000msec είναι 1sec. Μετά από 1 sec εκτελείται η επόμενη εντολή που λέει στον arduino να θέσει το pin 10 σε χαμηλό δυναμικό (LOW) δηλαδή 0V ή γείωση (ground). Με βάση και τα προηγούμενα έχουμε:

digitalWrite(ledPin,LOW);

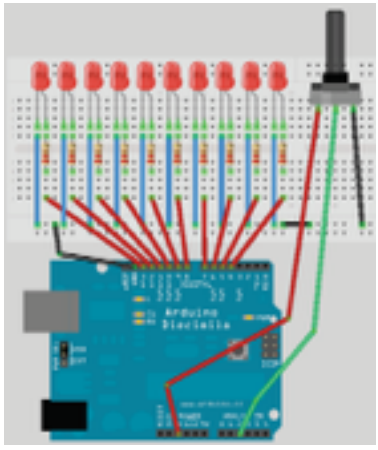
Ακολουθεί ακόμη μία καθυστέρηση κατά 1000msec και η συνάρτηση τελειώνει. Επειδή όμως είναι η βασική συνάρτηση *loop()*, θα εκτελεστεί ξανά από την αρχή. Αυτό θα συνεχίζεται μέχρι να θέσουμε τον Arduino εκτός λειτουργίας ή αν ανεβάσουμε σε αυτόν άλλο πρόγραμμα (sketch) ή πατήσουμε το διακόπτη Reset που βρίσκεται πάνω στην πλακέτα.

Συνοψίζοντας τη λειτουργία του προγράμματος λέμε ότι ξεκινάει ορίζοντας τη μεταβλητή ledPin στην οποία δίνει την τιμή 10. Στη συνέχεια πάμε στη συνάρτηση setup() στην οποία ορίζεται το pin 10 ως έξοδο (OUTPUT).

Στο βασικό πρόγραμμα loop θέτουμε το pin 10 σε κατάσταση HIGH στέλνοντας 5V. Μετά περιμένουμε 1 sec και σβήνουμε το LED θέτοντας το pin 10 σε κατάσταση LOW δηλαδή στέλνοντας 0V. Περιμένουμε ακόμη 1 sec και ξαναρχίζοντας τον κύκλο από την αρχή με συνέπεια το LED να αναβοσβήνει συνεχώς.

Κάποιος μπορεί εύκολα να κάνει κάποιες τροποποιήσεις στο πρόγραμμα αλλάζοντας την ταχύτητα που αναβοσβήνει το LED αλλάζοντας πολύ απλά το όρισμα της εντολής delay. Για παράδειγμα αν το 1000 γίνει 50msec θα αναβοσβήνει πολύ γρήγορα, ενώ αν γίνει 2000msec πολύ αργά!

Ένα πιο προχωρημένο project:
Το ελεγχόμενο κυνηγητό των LED!



Γι' αυτό το project θα χρειαστούμε *ένα breadboard, δέκα LED και ίσο αριθμό αντιστάσεων 220Ω, ένα ποτενσιόμετρο 4Κ7 και καλώδια.*

Στην πρώτη γραμμή του sketch βλέπουμε:

```
byte      ledPin[ ]={4,5,6,7,8,9,10,11,12,13};
```

Πρόκειται για τον ορισμό μεταβλητής τύπου «πίνακα». Με απλά λόγια η μεταβλητή *ledPin* μπορεί να πάρει οποιαδήποτε από τις τιμές 4 έως 13 ανάλογα με τι περιέχει το όρισμά της. Για παράδειγμα *ledPin[2]* είναι το 6. Αυτό συμβαίνει γιατί ο πίνακας έχει 10 στοιχεία με αρίθμηση από το 0 έως το 9, όπως συμβαίνει σε κάθε πίνακα με 10 στοιχεία. Το στοιχείο 0 του πίνακα είναι το πρώτο κ.τ.λ.

Αν θέλουμε να δηλώσουμε έναν πίνακα με 10 στοιχεία χωρίς να δώσουμε αρχικές τιμές σε αυτά θα γράψουμε:

```
byte ledPin[10];
```

και κάποια στιγμή αργότερα θα δώσουμε τιμές σε αυτά τα 10 στοιχεία.

Για να δώσουμε μια τιμή από τον πίνακα σε μια μεταβλητή x γράφουμε:

```
x = ledPin[5];
```

οπότε η μεταβλητή x παίρνει την 6η τιμή του πίνακα δηλαδή την τιμή 9!

Στο πρόγραμμά μας με τον πίνακα *ledPin* έχουμε αποθηκεύσει 10 τιμές οι οποίες είναι τα ψηφιακά pin που χρησιμοιούνται για τα 10 LED μας.

Κάτι ακόμα καινούργιο σ'αυτόν τον κώδικα είναι η συνάρτηση *millis()* (η οποία είναι μια συνάρτηση φτιαγμένη γι'αυτόν τον μικροελεγκτή) που επιστρέφει τον αριθμό των msec που πέρασαν από τη στιγμή που άρχισε να τρέχει ο κώδικας που την περιέχει!

Επιστρέφοντας στον κώδικά μας και στη συνάρτηση *loop* έχουμε φροντίσει ώστε να περάσουν τουλάχιστον ledDelay (στον αριθμό!) msec από την τελευταία αλλαγή στα LED και τότε περνάει η ροή του προγράμματος στη συνάρτηση *changeLED()*. Παρατηρούμε μέσα στην *changeLED()* ότι στη μεταβλητή *currentLED* προστίθεται η μεταβλητή direction. Η τελευταία μπορεί να πάρει τιμές μόνο 1 ή -1, οπότε είτε αυξάνει την *currentLED* κατά 1 είτε την μειώνει κατά 1.

Στη συνέχεια υπάρχει μια δήλωση if για δούμε αν έχουμε φτάσει στο τελευταίο LED, ώστε να αλλάξουμε την τιμή της μεταβλητής *direction* και να αλλάξει η κατεύθυνση που ανάβουν τα LED.

Στο κύκλωμά μας υπάρχει ένα ποτενσιόμετρο με σκοπό να αυξομειώνει την ταχύτητα του φαινομένου! Θα δούμε τις εντολές που εντάσσουν το ποτενσιόμετρο στη λειτουργία. Πρώτα δηλώνεται η μεταβλητή που δείχνει τη θέση της σύνδεσης με τον Arduino:

```
int potPin = 2;
```

```
// Create array for LED pins
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10,
11, 12, 13};
int ledDelay; // delay between changes
int direction = 1;
int currentLED = 0;
unsigned long changeTime;
int potPin = 2;    // select the input
pin for the potentiometer

void setup() {
  // set all pins to output
  for (int x=0; x<10; x++) {
    pinMode(ledPin[x], OUTPUT); }
    changeTime = millis();
}

void loop() {
// read the value from the pot
ledDelay = analogRead(potPin);
  // if it has been ledDelay ms since
last change
  if ((millis() - changeTime) >
ledDelay) {
    changeLED();
    changeTime = millis();
  }
}

void changeLED() {
  // turn off all LED's
  for (int x=0; x<10; x++) {
    digitalWrite(ledPin[x], LOW);
  }
  // turn on the current LED
  digitalWrite(ledPin[currentLED],
HIGH);
  // increment by the direction value
  currentLED += direction;
  // change direction if we reach the
end
  if (currentLED == 9) {direction =
-1;}
  if (currentLED == 0) {direction = 1;}
}
```

Σύμφωνα με την παραπάνω δήλωση το ποτενσιόμετρο συνδέεται στην αναλογική είσοδο του pin 2. Για να διαβαστεί η τιμή από αναλογικό pin χρησιμοποιείται η εντολή *analogRead*.Το αναλογικό pin μπορεί να δεχτεί τάσεις μεταξύ 0 και 5V σε ακέραιες τιμές μεταξύ 0 (0 Volts) και 1023 (5 Volts). Αυτό έχει σαν αποτέλεσμα μια ακρίβεια 5 Volts/ 1024 μονάδες ή 0.0049 Volts (4.9mV) ανά μονάδα.
Για να εξαρτήσουμε την καθυστέρηση από το ποτενσιόμετρο θα χρησιμοποιήσουμε τιμές απευθείας από το αναλογικό pin, ώστε να κυμαίνεται η καθυστέρηση μεταξύ 0 και 1023 msec. Αυτό γίνεται εισάγοντας απευθείας την τιμή από το pin του ποτενσιόμετρου στη μεταβλητή ledDelay.

ledDelay = analogRead(potPin);

Στα αναλογικά pin δε χρειάζεται να ορίσουμε είσοδο ή έξοδο όπως κάνουμε στα ψηφιακά.

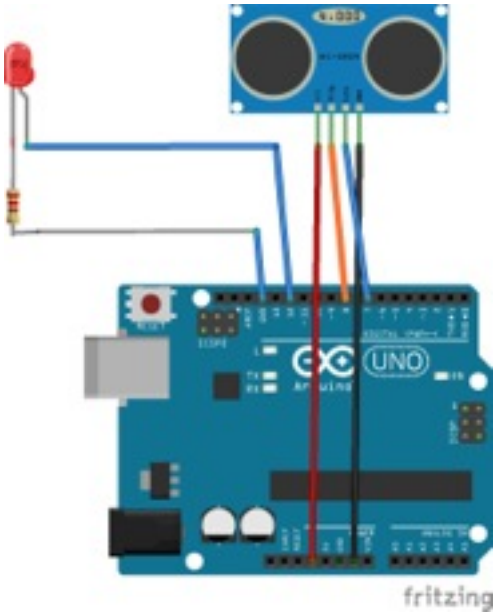
Ελέγχω την απόσταση των αντικειμένων με ένα ραντάρ ηπερήχων και τη δείχνω αλλάζοντας το ρυθμό που αναβοσβήνει ένα LED.

Ένα ρομπότ πρέπει τουλάχιστον όταν κινείται να αποφεύγει τα εμπόδια για να έχει την ελάχιστη αυτονομία, ώστε μετά να έχουμε και άλλες απαιτήσεις από αυτό.

Ο οικονομικότερος τρόπος για να το πετύχει αυτό είναι να χρησιμοποιήσει έναν αισθητήρα που εκπέμπει και λαμβάνει πίσω τους ανακλώμενους υπέρηχους.

Σ' αυτό το project θα χρησιμοποιήσουμε τον αισθητήρα HC-SR04 με εύρος δράσης από 4cm έως 4 m. Βέβαια επειδή πάντα η ποιότητα είναι ανάλογη του κόστους, τα μεγέθη δεν είναι ακριβώς τα ίδια σε όλους τους HC-SR04 και ακόμη πρέπει να ελέγξει κάποιος και τη γωνία υπό την οποία βλέπει ο δικός του αισθητήρας.

Στην εφαρμογή μας ο αισθητήρας υπερήχων θα αντιλαμβάνεται την απόσταση ενός αντικειμένου και ανάλογα με αυτήν θα αλλάζει ο ρυθμός που θα αναβοσβήνει ένα LED.



Συγκεκριμένα ο HC-SR04 έχει τέσσερεις εισόδους και εξόδους. Δέχεται τροφοδοσία 5V DC στην είσοδο Vcc, ένα pin εισόδου το trigPin (pin 8) με το οποίο δίνει εντολή το arduino στον αισθητήρα να κάνει μέτρηση, ένα pin εξόδου το echoPin (pin 7) με το οποίο στέλνει ο αισθητήρας στο arduino την πληροφορία «απόσταση» και φυσικά η σύνδεση με τη γείωση Gnd.

Συνδέουμε ακόμη σε σειρά ένα LED και έναν αντιστάση 220Ω. Ο αντιστάτης γειώνεται και το LED μπορεί να δέχεται τάση από το pin 12.

Βλέποντας τον κώδικά μας μετά τον ορισμό των μεταβλητών, που γίνεται όπως περιγράψαμε σε προηγούμενα project, υπάρχει στη ρουτίνα setup μια καινούργια συνάρτηση:

Serial.begin (9600);

Η Serial.begin ξεκινάει μια επικοινωνία μεταξύ του arduino και του αισθητήρα, ενώ καθορίζει και το ρυθμό μεταφοράς δεδομένων στα 9600 baud ή bits/sec. Οι ρυθμοί μετάδοσης δεδομένων για το μικροελεγκτή arduino UNO κυμαίνονται μεταξύ 300 και 115200 bits/sec, ανάλογα με τις προδιαγραφές λειτουργίας της συσκευής επικοινωνίας. Για το HC-SR04 είναι 9600 bits/sec.

Στη συνέχεια ορίζουμε ως pin εξόδου το digital pin 8, ενώ ως pin εισόδου το digital pin 7. Το LED φυσικά ορίζεται ως έξοδος (digital pin 12) αφού θα κάνουμε να ανάβει τη στιγμή και για τη διάρκεια που θέλουμε.

```
#define echoPin 7 // Echo Pin
#define trigPin 8 // Trigger Pin
#define LEDPin 12
long duration, distance; int k;

void setup() {
 Serial.begin (9600);
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
 pinMode(LEDPin, OUTPUT);
}

 void loop() {
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 duration = pulseIn(echoPin, HIGH);

 //Calculate the distance (in cm) based on the speed of sound.
 distance = duration/58.2;
 //****************************************** d<30cm
 if (distance <=30)
{
 k=50;
 Serial.print("distance under 30cm =  ");
 Serial.println(distance);
 Blink();
 }
 //****************************************** 30<d<60cm
 if (distance <=60 && distance>=30)
{
 k=150;
 Serial.print("distance under 60cm =  ");
 Serial.println(distance);
 Blink();
 }
 //****************************************** 60<d<120cm
if (distance >60 && distance<=120)
{
 k=250;
 Serial.print("distance between 60cm and 120cm =  ");
 Serial.println(distance);
 Blink();
 }
 //****************************************** 120cm<d
 if (distance >120)
{
 k=500;
 Serial.print("distance over 120cm =  ");
 Serial.println(distance);
 Blink();
  }
 delay(1000);
}

void Blink()
{
  digitalWrite(LEDPin,HIGH);
  delay(k);
  digitalWrite(LEDPin,LOW);
  delay(k);
}
```

```
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
pinMode(LEDPin, OUTPUT);
```

Οι εντολές που ακολουθούν κάνουν το HC-SR04 να στείλει έναν παλμό υπερήχων διάρκειας 10 μsec, ενώ με την εντολή pulseIn δέχονται τον επιστρεφόμενο παλμό υπερήχων ως είσοδο. Αυτός μετατρέπεται σε απόσταση στη μεταβλητή distance.

```
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH);
distance = duration/58.2;
```

Ακολουθούν τέσσερεις λογικές εντολές **if,** που η κάθε μια εκτελείται ανάλογα με την απόσταση που έχει ανιχνεύσει ο HC-SR04 .Η μεταβλητή **k** πέρνει διαφορετική τιμή κάθε φορά και αυτή είναι ο χρόνος που είναι αναμμένο και στη συνέχεια σβηστό το LED. Η μεταβλητή k έχει οριστεί ως γενική μεταβλητή, οπότε η τιμή της μεταφέρεται και στην υπορουτίνα Blink, την οποία καλούμε να εκτελεστεί μετά από κάθε λογικό έλεγχο **if.**

Για παράδειγμα αν η απόσταση (distance) είναι πάνω από 120cm το LED θα ανάψει επί 500msec.

Σύντομη εξήγηση όλων των βασικών οδηγιών που υποστηρίζονται από τη γλώσσα του Arduino.

Στην πραγματικότητα η γλώσσα προγραμματισμού που χρησιμοποιείται είναι η C με ιδιαίτερες προσθήκες για τον συγκεκριμένο μικροελεγκτή.

Structure
An Arduino sketch runs in two parts:

void setup()

This is where you place the initialisation code—the instructions that set up the board before the main loop of the sketch starts.

void loop()

This contains the main code of your sketch. It contains a set of instructions that get repeated over and over until the board is switched off.

Special symbols
Arduino includes a number of symbols to delineate lines of code, comments and blocks of code.

; (semicolon)

Every instruction (line of code) is terminated by a semicolon. This syntax lets you format the code freely. You could even put two instructions on the same line, as long as you separate them with a semicolon. (However, this would make the code harder to read.)
Example:

```
delay(100);
```

{} (curly braces)

This is used to mark blocks of code. For example, when you write code for the loop() function, you have to use curly braces before and after the code.
Example:

```
void loop() {
Serial.println("ciao");
}
```

Constants
Arduino includes a set of predefined keywords with special values.
HIGH and LOW are used, for example, when you want to turn on or off

an Arduino pin. INPUT and OUTPUT are used to set a specific pin to be either and input or an output

true and false indicate exactly what their names suggest: the truth or falsehood of a condition or expression.

Variables

Variables are named areas of the Arduino's memory where you can store data that you can use and manipulate in your sketch. As the name suggests, they can be changed as many times as you like.
Because Arduino is a very simple processor, when you declare a variable you have to specify its type. This means telling the processor the size of the value you want to store.
Here are the datatypes that are available:
boolean
Can have one of two values: true or false.
char
Holds a single character, such as A. Like any computer, Arduino stores it as a number, even though you see text. When chars are used to store numbers, they can hold values from –128 to 127.

NOTE: There are two major sets of characters available on computer systems: ASCII and UNICODE. ASCII is a set of 127 characters that was used for, among other things, transmitting text between serial terminals and time-shared computer systems such as mainframes and minicomputers. UNICODE is a much larger set of values used by modern computer operating systems to represent characters in a wide range of languages. ASCII is still useful for exchanging short bits of information in languages such as Italian or English that use Latin characters, Arabic numerals, and common typewriter symbols for punctuation and the like.
byte
Holds a number between 0 and 255. As with chars, bytes use only one byte of memory.
int
Uses 2 bytes of memory to represent a number between –32,768 and 32,767; it's the most common data type used in Arduino.
unsigned int
Like int, uses 2 bytes but the unsigned prefix means that it can't store negative numbers, so its range goes from 0 to 65,535.
long
This is twice the size of an int and holds numbers from –2,147,483,648 to 2,147,483,647.
unsigned long
Unsigned version of long; it goes from 0 to 4,294,967,295.

float
This quite big and can hold floating-point values, a fancy way of saying that you can use it to store numbers with a decimal point in it. It will eat up 4 bytes of your precious RAM and the functions that can handle them use up a lot of code memory as well. So use floats sparingly.
double
Double-precision floating-point number, with a maximum value of 1.7976931348623157 x 10308. Wow, that's huge!
string
A set of ASCII characters that are used to store textual information (you might use a string to send a message via a serial port, or to display on an LCD display). For storage, they use one byte for each character in the string, plus a null character to tell Arduino that it's the end of the string.
The following are equivalent:

```
char string1[] = "Arduino"; // 7 chars + 1 null char
char string2[8] = "Arduino"; // Same as above
```

array
A list of variables that can be accessed via an index. They are used to build tables of values that can easily be accessed. For example, if you want to store different levels of brightness to be used when fading an LED, you could create six variables called light01, light02, and so on. Better yet, you could use a simple array like:

```
int light[6] = {0, 20, 50, 75, 100};
```

The word "array" is not actually used in the variable declaration: the symbols [] and {} do the job.

Control Structures
Arduino includes keywords for controlling the logical flow of your sketch.
if . . . else
This structure makes decisions in your program. if must be followed by a question specified as an expression contained in parentheses. If the expression is true, whatever follows will be executed. If it's false, the block of code following else will be executed. It's possible to use just if without providing an else clause.
Example:

```
if (val == 1) {
digitalWrite(LED,HIGH);
}
```

for
Lets you repeat a block of code a specified number of times.
Example:

```
for (int i = 0; i < 10; i++) {
Serial.print("ciao");
}
```

switch case

The if statement is like a fork in the road for your program. switch case is like a massive roundabout. It lets your program take a variety of directions depending on the value of a variable. It's quite useful to keep your code tidy as it replaces long lists of if statements.

Example:

```
switch (sensorValue) {
        case 23:
                digitalWrite(13,HIGH);
                break;
        case 46:
                digitalWrite(12,HIGH);
                break;
        default: // if nothing matches this is executed
        digitalWrite(12,LOW);
        digitalWrite(13,LOW);
        }
```

while

Similar to if, this executes a block of code while a certain condition is true.

Example:

```
// blink LED while sensor is below 512

        sensorValue = analogRead(1);
while (sensorValue < 512) {
        digitalWrite(13,HIGH);
        delay(100);
        digitalWrite(13,HIGH);
        delay(100);
        sensorValue = analogRead(1);
        }
```

do . . . while

Just like while, except that the code is run just before the the condition is evaluated. This structure is used when you want the code inside your block to run at least once before you check the condition.

Example:

```
        do {
                digitalWrite(13,HIGH);
                delay(100);
```

```
        digitalWrite(13,HIGH);
        delay(100);
        sensorValue = analogRead(1);
    } while (sensorValue < 512);
```

break

This term lets you leave a loop and continue the execution of the code that appears after the loop. It's also used to separate the different sections of a switch case statement.

Example:

```
                    // blink LED while sensor is below 512
    do {
                    // Leaves the loop if a button is pressed
        if (digitalRead(7) == HIGH)
        break;
        digitalWrite(13,HIGH);
        delay(100);
        digitalWrite(13,LOW);
        delay(100);
        sensorValue = analogRead(1);
    } while (sensorValue < 512);
```

continue

When used inside a loop, continue lets you skip the rest of the code inside it and force the condition to be tested again.

Example:

```
    for (light = 0; light < 255; light++)
        {
        // skip intensities between 140 and 200
        if ((x > 140) && (x < 200))
        continue;
        analogWrite(PWMpin, light);
        delay(10);
        }
```

return

Stops running a function and returns from it. You can also use this to return a value from inside a function. For example, if you have a function called computeTemperature() and you want to return the result to the part of your code that invoked the function you would write something like:

```
    int computeTemperature() {
            int temperature = 0;
            temperature = (analogRead(0) + 45) / 100;
            return temperature;
            }
```

Arithmetic and formulas

*You can use Arduino to make complex calculations using a special syntax. + and – work like you've learned in school, and multiplication is represented with an * and division with a /. There is an additional operator called "modulo" (%), which returns the remainder of an integer division. You can use as many levels of parentheses as necessary to group expressions. Contrary to what you might have learned in school, square brackets and curly brackets are reserved for other purposes (array indexes and blocks, respectively).*

Examples:

```
a = 2 + 2;
light = ((12 * sensorValue) - 5 ) / 2;
remainder = 3 % 2; // returns 1
```

Comparison Operators

When you specify conditions or tests for if, while, and for statements, these are the operators you can use:

```
== equal to
!= not equal to
< less than
> greater than
<= less than or equal to
>= greater than or equal to
```

Boolean Operators

These are used when you want to combine multiple conditions. For example, if you want to check whether the value coming from a sensor is between 5 and 10, you would write:

```
if ((sensor => 5) && (sensor <=10))
```

There are three operators: and, represented with &&; or, represented with ll; and finally not, represented with !.

Compound Operators

These are special operators used to make code more concise for some very common operations like incrementing a value.

For example, to increment value by 1 you would write:

```
value = value +1;
```

but using a compound operator, this becomes:

```
value++;
```

increment and decrement (— and ++)

These increment or decrement a value by 1. Be careful: if you write i++ this increments i by 1 and evaluates to the equivalent of i+1; ++i evaluates to the value of i and then increments i. The same applies to —.

```
+= , –=, *= and /=
```

These make it shorter to write certain expressions. The following two expressions are equivalent:

```
a = a + 5;
a += 5;
```

Input and output functions

Arduino includes functions for handling input and output. You've already seen some of these in the example programs throughout the book.

pinMode(pin, mode)

Reconfigures a digital pin to behave either as an input or an output.

Example:

pinMode(7,INPUT); // turns pin 7 into an input

digitalWrite(pin, value)

Turns a digital pin either on or off. Pins must be explicitly made into an output using pinMode before digitalWrite will have any effect.

Example:

digitalWrite(8,HIGH); // turns on digital pin 8

int digitalRead(pin)

Reads the state of an input pin, returns HIGH if the pin senses some voltage or LOW if there is no voltage applied.

Example:

val = digitalRead(7); // reads pin 7 into val

int analogRead(pin)

Reads the voltage applied to an analog input pin and returns a number between 0 and 1023 that represents the voltages between 0 and 5 V.

Example:

val = analogRead(0); // reads analog input 0 into val

analogWrite(pin, value)

Changes the PWM rate on one of the pins marked PWM. pin may be 11,10, 9, 6, 5, 3. value may be a number between 0 and 255 that represents the scale between 0 and 5 V output voltage.

Example:

analogWrite(9,128); // Dim an LED on pin 9 to 50%

shiftOut(dataPin, clockPin, bitOrder, value)

Sends data to a shift register, devices that are used to expand the number of digital outputs. This protocol uses one pin for data and one for clock. bitOrder indicates the ordering of bytes (least significant or most significant) and value is the actual byte to be sent out.

Example:

shiftOut(dataPin, clockPin, LSBFIRST, 255);

unsigned long pulseIn(pin, value)

Measures the duration of a pulse coming in on one of the digital inputs. This is useful, for example, to read some infrared sensors or accelerometers that output their value as pulses of changing duration.

Example:

time = pulsein(7,HIGH); // measures the time the next

// pulse stays high

Time functions

Arduino includes functions for measuring elapsed time and also for pausing the sketch.

unsigned long millis()

Returns the number of milliseconds that have passed since the sketch started.

Example:

```
duration = millis()-lastTime; // computes time elapsed since "lastTime"
```

delay(ms)

Pauses the program for the amount of milliseconds specified.

Example:

```
delay(500); // stops the program for half a second
```

delayMicroseconds(us)

Pauses the program for the given amount of microseconds.

Example:

```
delayMicroseconds(1000); // waits for 1 millisecond
```

Math functions

Arduino includes many common mathematical and trigonometric functions:

min(x, y)

Returns the smaller of x and y.

Example:

```
val = min(10,20); // val is now 10
```

max(x, y)

Returns the larger of x and y.

Example:

```
val = max(10,20); // val is now 20
```

abs(x)

Returns the absolute value of x, which turns negative numbers into positive. If x is 5 it will return 5, but if x is –5, it will still return 5.

Example:

```
val = abs(-5); // val is now 5
```

constrain(x, a, b)

Returns the value of x, constrained between a and b. If x is less than a, it will just return a and if x is greater than b, it will just return b.

Example:

```
val = constrain(analogRead(0), 0, 255); // reject values bigger than 255
```

map(value, fromLow, fromHigh, toLow, toHigh)

Maps a value in the range fromLow and maxLow to the range toLow and toHigh. Very useful to process values from analogue sensors.

Example:

```
val = map(analogRead(0),0,1023,100, 200); // maps the value of
// analog 0 to a value
// between 100 and 200
```

double pow(base, exponent)

Returns the result of raising a number (base) to a value (exponent).

Example:

```
double x = pow(y, 32); // sets x to y raised to the 32nd power
```

double sqrt(x)

Returns the square root of a number.

Example:

```
double a = sqrt(1138); // approximately 33.73425674438
```

double sin(rad)

Returns the sine of an angle specified in radians.

Example:

```
double sine = sin(2); // approximately 0.90929737091
```

double cos(rad)

Returns the cosine of an angle specified in radians.

Example:

```
double cosine = cos(2); // approximately -0.41614685058
```

double tan(rad)

Returns the tangent of an angle specified in radians.

Example:

```
double tangent = tan(2); // approximately -2.18503975868
```

Random number functions

If you need to generate random numbers, you can use Arduino's pseudorandom number generator.

randomSeed(seed)

Resets Arduino's pseudorandom number generator. Although the distribution of the numbers returned by random() is essentially random, the sequence is predictable. So, you should reset the generator to some random value. If you have an unconnected analog pin, it will pick up random noise from the surrounding environment (radio waves, cosmic rays, electromagnetic interference from cell phones and fluorescent lights, and so on).

Example:

```
randomSeed(analogRead(5)); // randomize using noise from pin 5
```

long random(max)

long random(min, max)
Returns a pseudorandom long integer value between min and max – 1. If min is not specified, the lower bound is 0.
Example:

```
long randnum = random(0, 100); // a number between 0 and 99
long randnum = random(11); // a number between 0 and 10
```

Serial communication
As you saw in Chapter 5, you can communicate with devices over the USB port using a serial communication protocol. Here are the serial functions.

Serial.begin(speed)
Prepares Arduino to begin sending and receiving serial data. You'll generally use 9600 bits per second (bps) with the Arduino IDE serial monitor, but other speeds are available, usually no more than 115,200 bps.
Example:

```
Serial.begin(9600);
```

Serial.print(data)
Serial.print(data, encoding)
Sends some data to the serial port. The encoding is optional; if not supplied, the data is treated as much like plain text as possible.
Examples:

```
Serial.print(75); // Prints "75"
Serial.print(75, DEC); // The same as above.
Serial.print(75, HEX); // "4B" (75 in hexadecimal)
Serial.print(75, OCT); // "113" (75 in octal)
Serial.print(75, BIN); // "1001011" (75 in binary)
Serial.print(75, BYTE); // "K" (the raw byte happens to
// be 75 in the ASCII set)
```

Serial.println(data)
Serial.println(data, encoding)
Same as Serial.print(), except that it adds a carriage return and linefeed (\r\n) as if you had typed the data and then pressed Return or Enter.
Examples:

```
Serial.println(75); // Prints "75\r\n"
Serial.println(75, DEC); // The same as above.
Serial.println(75, HEX); // "4B\r\n"
Serial.println(75, OCT); // "113\r\n"
Serial.println(75, BIN); // "1001011\r\n"
Serial.println(75, BYTE); // "K\r\n"
```

int Serial.available()
Returns how many unread bytes are available on the Serial port for

reading via the read() function. After you have read() everything available, Serial.available() returns 0 until new data arrives on the serial port.
Example:
int count = Serial.available();

int Serial.read()
Fetches one byte of incoming serial data.
Example:
int data = Serial.read();

Serial.flush()
Because data may arrive through the serial port faster than your program can process it, Arduino keeps all the incoming data in a buffer. If you need to clear the buffer and let it fill up with fresh data, use the flush() function.
Example:
Serial.flush();