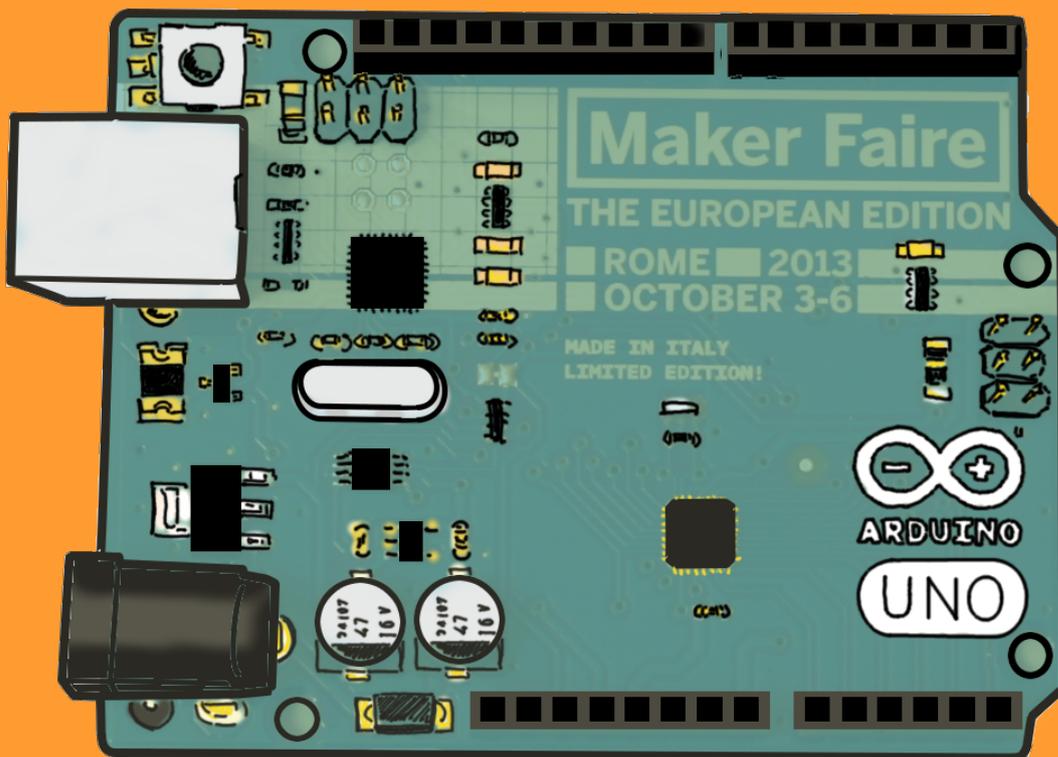


Paolo Aliverti

# Il manuale di Arduino

## Guida Completa

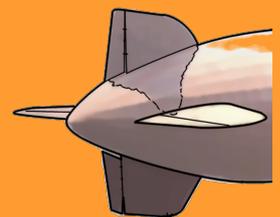


Guida completa per la progettazione e la programmazione >>

Sensori, attuatori, trasmettitori e Internet >>

Programmazione di base e avanzata >>

Spiegazioni semplici, esempi pratici >>

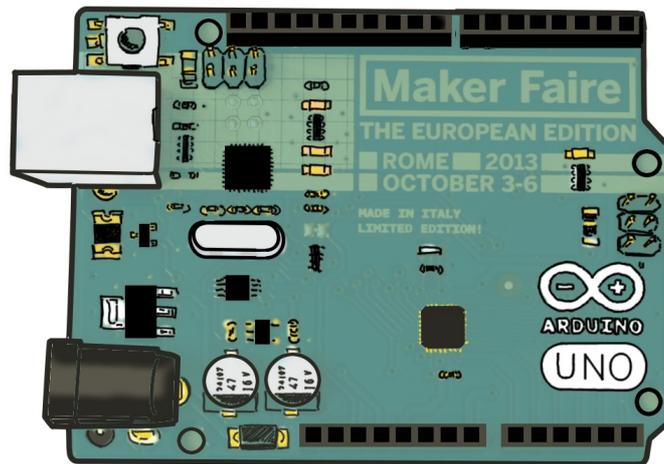


[zeppelinmaker.it](http://zeppelinmaker.it)

PAOLO ALIVERTI

# IL MANUALE DI ARDUINO

## GUIDA COMPLETA



Trovi altre informazioni, approfondimenti e libri su:

[www.zepelinmaker.it](http://www.zepelinmaker.it)

Se trovi degli errori, se il libro ti è piaciuto oppure no, se vuoi salutarmi:  
scrivimi!

[paolo@zepelinmaker.it](mailto:paolo@zepelinmaker.it)

Iscriviti alla mia Mailing List per ricevere tutti gli aggiornamenti  
e altre novità:

<http://www.zepelinmaker.it/ mailing-list/>

In autunno/inverno dovrei iniziare a pubblicare:

**“La guida galattica della stampa 3D”**

Non perdertela!

## IL MANUALE DI ARDUINO

1. Introduzione
  - La storia
  - Microcontrollori e microprocessori
  - Maker & Arduino
  - Siti di riferimento
2. Arduino
  - com'è fatto arduino UNO
  - le altre schede
  - shield, tinkerkit e groove
3. Installazione
  - download e installazione
  - il bootloader
  - lo sketch (loop e setup)
  - Hello Led! (programmazione del primo esempio)
4. Il linguaggio C
  - il C
  - come imparare a programmare
  - variabili
  - cicli e controllo di flusso
  - funzioni
  - programmi
5. Programmare Arduino
  - digitalWrite, Read, analogRead, analogWrite
  - tone → buzzer
  - millis
  - Serial
  - altri comandi
6. Programmazione Avanzata
  - interrupt
  - sleep mode
  - assembler
  - variabili persistenti (EEPROM)
7. Sensori
  - fotoresistenza
  - termistore
  - trimmer
  - lm35 temperatura
  - bussole, accelerometro, giroscopio
  - hall sensor
  - audio
  - IR
  - PIR
  - optoaccoppiatore
8. Attuatori
  - motori
  - transistor e mosfet
  - relè
  - motore passo passo
  - servo motore

- buzzer e speaker
- potenziometro digitale
- LCD

## 9. Comunicazioni

- porte seriali spi e i2c
- scheda SD
- blue tooth
- wifi
- zigbee
- comunicazione tra due Arduino (o raspberry)
- Arduino e Processing
- ethernet
  - richiamare pagine web
  - arduino as webserver

## 10. YUN

- descrizione della scheda
- installazione e configurazione
- Il bridge
- client web
- web server
- processi e scheduling su linux
- creazione di servizi web
- collegarsi con un terminale
- reset della scheda

## 11. Arduino DIY

- costruzione del circuito
- set dei fuses (i fuses)
- caricare il bootloader
- programmare firmware e caricarlo con ISP
- arduino as ISP
- programmazione attiny

# 1

## Introduzione

Prima di acquistare il mio primo Arduino, nel 2008, lessi il libro “Getting started with Arduino” di Massimo Banzi. Il libretto mi piacque molto perché ritrovai molte cose in comune con l’autore che era anche il «padre» della famosa schedina elettronica. Anch’io iniziai da bambino a interessarmi di elettronica e microcomputer. Negli anni ottanta possedevo uno Spectrum 48K. Quando aprii la scatola, all’interno vi trovai anche un manuale di programmazione, perché a quei tempi, era naturale programmare un computer. Erano gli anni delle riviste di elettronica in edicola, dei kit e dei negozietti di componenti per appassionati. Avevo circa una decina di anni quando incominciai a costruire circuiti elettronici traducendo dei segni sulla carta in oggetti fisici. Non so come imparai... mi aiutarono due libri che conservo ancora oggi: «Elettrotecnica illustrata» e il «Manuale dell’inventore». Erano libri illustrati, adatti alla mia età e che mi furono molto utili! Imparai a distinguere resistenze e condensatori grazie a quei libri e a tanti esperimenti. Avevo un piccolo laboratorio nello scantinato della casa di Ceriano Laghetto. Era lo studio di mio padre, che avevo completamente occupato e riempito con tutti i miei fili e computer. Passavo lì dentro, nel mio piccolo «bunker» giornate intere... e a volte anche le notti.

Nonostante i miei studi in ingegneria delle telecomunicazioni al politecnico di Milano, misi da parte il laboratorio per parecchi anni, fino a circa il 2006. Mi capitò di leggere «FAB» di Neil Gershenfeld, che riaccese la voglia di costruire circuiti elettronici. In una ventina d’anni molte cose erano cambiate: nuove potenti tecnologie erano ormai da tempo degli standard di mercato e molte altre cose si sono semplificate e i prezzi dell’elettronica sono scesi drasticamente.

In questo lungo periodo temporale ho assistito anche all’esplosione di Internet. Ricordo le mie prime «navigazioni» durante le pause tra le lezioni del politecnico: le sessioni erano in ftp e consistevano in esplorazioni manuali di enormi archivi online. Gli indirizzi si passavano a voce. Era come navigare bendati. Qualche anno dopo comparve il browser testuale Lynx per utilizzare il protocollo http: una grandissima novità.

Oggi Internet è diventato un mezzo di comunicazione comune e diffuso ed è di enorme aiuto per chi vuole imparare qualcosa o trovare informazioni. Nel mio laboratorio di Ceriano Laghetto, le uniche fonti disponibili erano le riviste che conservavo in grandi pile ordinate e che conoscevo a memoria e alcuni libri che mi ero fatto regalare con le caratteristiche dei transistor e dei circuiti integrati. Quando possibile recuperavo componenti smontando elettrodomestici e il più delle volte mi ritrovavo tra le mani parti di cui non conoscevo nulla. Oggi basta digitare la sigla di un qualsiasi componente in Google per avere un datasheet (cioè il foglio d’istruzioni). Su Internet possiamo trovare anche tutti gli schemi elettronici che desideriamo e anche tutorial e guide per apprendere qualsiasi nozione teorica e pratica. La vita per l’«hobbista elettronico» moderno è molto più semplice, anche se è un personaggio quasi estinto, perché chi s’interessa di questa materia, oggi, sono i «maker».

Nel 2008 m’inventai un App per partecipare a un concorso della provincia di Milano. L’App serviva agli sportivi per monitorare i loro allenamenti e per proporre loro delle sfide reali o virtuali con altri «runner». Per evitare alle persone di portarsi dietro il telefono, avevo ideato un aggeggio hardware con GPS, in grado di registrare i dati: un data-logger. Per realizzare il prototipo acquistai il mio primo Arduino, con l’intento di collegargli i sensori necessari.

Restai stupito dalla semplicità con cui riuscii a programmarlo la prima volta. Era sufficiente collegare la scheda al PC con un cavo USB, digitare qualche e caricare il tutto premendo un pulsante. Da allora ho utilizzato Arduino per molti altri prototipi e progetti. Nel 2011 ho iniziato a tenere corsi per imparare a usarlo. Credo fossi tra i primi a offrire a Milano dei corsi come “L’elettronica della sciura Maria” e “L’ABC di Arduino”.

## La storia<sup>1</sup>

Arduino nasce presso l’Interaction Design Institute di Ivrea, un istituto di formazione avanzata, post universitario, fondato da Olivetti e Telecom. Nasce in un posto particolare... in un paese che ha visto nascere (e tramontare) la grande industria elettronica italiana, che per un momento ci ha fatto sognare in grande. Purtroppo in Italia è rimasto poco di quelle grandi aziende e di quell’ingegno che sapeva competere con le grandi potenze mondiali e con le multinazionali. Abbiamo inventato il personal computer, siamo stati competitor di grandi aziende come IBM e poi qualcosa si è rotto. Il nostro sistema politico ed economico ha consumato lentamente, quanto costruito da quelle grandi aziende nate nel dopoguerra. Gli imprenditori hanno lasciato il posto a manager specializzati in finanza che hanno spolpato le nostre grandi aziende.

Non so bene come sia andata la storia, ma si dice che nel 2003, Hernando Barragan abbia sviluppato a Ivrea un progetto originale per avvicinare designer e creativi all’elettronica. Sarebbe meglio dire: per facilitare l’utilizzo dell’elettronica per la realizzazione di prodotti innovativi e interattivi. Il lavoro di Barragan si chiamava Wiring ed è un progetto ancora attivo. Wiring è una piccola scheda elettronica dotata di un chip che può essere programmato semplicemente con un computer e un programma derivato da Processing (un editor per semplificare la programmazione e l’apprendimento del linguaggio Java). Processing è un progetto Open Source che è stato modificato per programmare Wiring e per funzionare con il linguaggio C richiesto dalla scheda elettronica. Wiring e Processing «modificato» costituiscono un sistema o frame work per la progettazione rapida di circuiti elettronici. La scrittura del codice non è fatta in C «puro» ma con una forma semplificata, nascondendo i dettagli della programmazione a basso livello, tutto per facilitarne l’utilizzo da parte di persone non esperte.

Massimo Banzi si interessò al lavoro di Barragan e nel 2005 decise di sviluppare una nuova scheda partendo dall’idea molto originale di Wiring, cercando di semplificare maggiormente l’utilizzo del sistema e abbassando il costo della scheda. Il nuovo progetto fu battezzato Arduino, il nome del bar di Ivrea, dove il team del progetto si trovava per bere l’aperitivo. Il team che realizzò Arduino era composto da Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis. Da allora il progetto ha riportato notevole successo in tutto il mondo dei maker e non solo. La piccola schedina di Ivrea è stata adottata per innumerevoli progetti e ha permesso la realizzazione di cose fino a poco tempo prima impensabili... soprattutto da parte di persone che non avevano conoscenze elettroniche.

Nasce così Arduino, una piccola scheda con un microcontrollore, che chiunque può imparare a utilizzare in breve tempo per realizzare circuiti elettronici interattivi. Per programmare Arduino è sufficiente collegarlo al proprio computer con un cavo USB, scrivere qualche istruzione, collegare qualche componente elettronico e premere il tasto «Upload». Si procede per tentativi ed errori, correggendo il programma di volta in volta. Le istruzioni sono abbastanza semplici e comprensibili. Il software di programmazione, chiamato anche IDE (Ambiente Integrato di Sviluppo), contiene numerosi esempi da cui prendere spunto per scrivere i propri listati. Per facilitare la costruzione dei circuiti elettrici sono state inventate delle piccole schede chiamate «shield» che s’impilano su

<sup>1</sup> La storia riportata è frutto della mia immaginazione e dalle fonti reperibili in Rete. Non ho mai potuto intervistare i diretti interessati e spero che quanto riportato sia il più attendibile possibile. Mi perdonino Massimo Banzi e Hernando Barragan se ho scritto qualche bestialità! ;-)

Arduino. Esistono shield con display, con un modulo Bluetooth, pulsanti, controllori per motori, GPS, schede di rete, moduli audio... Senza neppure spelare un filo elettrico potete costruire un «panino» di shield infilate l'una sopra l'altra e realizzare dispositivi anche molto sofisticati.

Esistono kit modulari formati da una shield «madre» da collegare ad Arduino e tante piccole schede e cavetti che si connettono per comporre i circuiti che servono. Ci sono schede con pulsanti, sensori di vario tipo, motori, display, led, relè...

Arduino ha successo perché è semplice e funziona sempre. Perché trovate spesso quello che vi serve, senza dovervi scervellare troppo. Perché è un prodotto ben fatto e continuamente curato. La scheda è stata sempre prodotta in Italia e continua a esserlo e si può considerare un vero prodotto «made in Italy».

Il progetto didattico si è trasformato in prodotto mantenendo la caratteristica «Open Source». Non c'è nulla di segreto o di nascosto. Gli schemi dell'hardware sono liberamente scaricabili da Internet, così come tutto il software necessario per costruirsi in casa, autonomamente, il proprio Arduino. Le aziende sono spesso abbastanza paranoiche sul tema sicurezza e se possibile terrebbero tutto sotto chiave. È inimmaginabile diffondere su Internet i piani del proprio prodotto di punta. Arduino ha dimostrato che con le opportune accortezze si possono creare aziende di successo basate su prodotti Open Source. La tutela è sul marchio e la protezione non sta in un brevetto ma sulla rispettabilità del nome e del prodotto... Rispettabilità costruita on-line grazie alla collaborazione e al coinvolgimento di migliaia di maker.

## Microcontrollori e microprocessori

Che cos'è un microcontrollore? Prima dovrete sapere cos'è un microprocessore: un circuito integrato (anche chiamato chip), specializzato nell'esecuzione di operazioni matematiche, nel trasferimento e nella manipolazione d'informazioni. Questi chip sono il «cervello» dei nostri computer e per funzionare hanno bisogno di memorie, dischi, periferiche di vario tipo, mouse, tastiera, display. Il primo microprocessore è stato costruito attorno agli anni '70 da Intel. Il circuito prodotto da Intel si chiamava 4004, conteneva qualche manciata di transistor e sapeva svolgere semplici operazioni matematiche a quattro bit, con frequenze di qualche KHz. I microprocessori di oggi sono delle grandi metropoli se paragonate al 4004: hanno centinaia di migliaia di transistor e lavorano a frequenze di GHz.

Un microcontrollore è un chip che contiene un piccolo microprocessore e il minimo indispensabile per il suo funzionamento: della memoria, qualcosa che possa funzionare da disco (flash memory), una memoria a lungo termine (EEPROM), un generatore di frequenza, dei convertitori analogico-digitali e delle semplici periferiche per permettergli di interagire con il mondo. Molti microcontrollori hanno potenza limitata, ma anche costi molto bassi. Per questo motivo li troviamo in molti dispositivi che utilizziamo quotidianamente. Grazie a una serie d'innovazioni introdotte da ATMEL e Microchip attorno agli anni '90 il loro utilizzo è diventato molto semplice, contribuendo alla loro diffusione. Un microcontrollore moderno non richiede speciali apparati per essere programmato e il firmware può essere trasferito al suo interno anche se è già stato saldato su un circuito stampato. Prima di queste innovazioni per utilizzare un microcontrollore era necessario, esporlo a dei raggi UV per cancellarlo e poi programmarlo con kit di sviluppo molto costosi. I microcontrollori moderni hanno la possibilità di ospitare un piccolo programma, chiamato bootloader, residente in un'area di memoria speciale. Il bootloader è un programma che può scrivere direttamente le istruzioni nella memoria del chip. In questo modo non serve un programmatore particolarmente complicato: con una semplice porta seriale è possibile programmare il chip con il software che abbiamo scritto.

Spesso sentirete parlare di System On Chip. Con questo termine s'indicano dei chip ancora più complicati e completi di un microcontrollore. Sono circuiti integrati che contengono oltre a un microprocessore, anche la circuiteria per schede di rete o video e sono molto simili a un computer

vero e proprio.

Esistono diverse famiglie di microcontrollori, ognuna contraddistinta da un proprio nome e dall'organizzazione tipica dei suoi circuiti interni, cioè l'architettura del chip. La programmazione di questi chip era fatta in codice macchina, scrivendo lunghi programmi che poi erano convertiti in sequenze di numeri. Inizialmente il numero di comandi e di operazioni era limitato, si potevano eseguire semplici operazioni logiche, trasferimenti di dati e test, ma con l'aumentare della complessità dei chip, i progettisti iniziarono a inserire delle operazioni più specifiche e sempre più complesse.

Questi chip con un set d'istruzioni molto numeroso si chiamano CISC, che sta per Complex Instruction Set Computer. Sono circuiti di grosse dimensioni, molto costosi da progettare e da realizzare. I programmatori però usavano raramente tutte queste istruzioni complesse e a volte, compiere la stessa operazione in più passaggi e con comandi base era più veloce! Per questo motivo sono nati i chip RISC (Reduced Instruction Set Computer), in cui il set d'istruzioni è stato ridotto al minimo indispensabile.

I più noti processori RISC sono i PIC, gli AVR, gli ARM o gli SPARC. Il chip utilizzato da Arduino UNO è un ATmega328 della famiglia AVR. Avrete spesso sentito nominare i chip ARM (Advanced RISC Machine), che sono circuiti impiegati nella realizzazione di telefoni, tablet e dispositivi portatili con bassi consumi e ottime prestazioni.

## Shield

Arduino è modulare. Sulla scheda elettronica sono presenti dei piccoli connettori in cui infilare fili elettrici o componenti. Questi connettori presentano delle serie di sei o otto fori, chiamati anche pin. È possibile infilare su questi connettori delle schede di espansione che rispettino la stessa disposizione di contatti.

Le shield sono schede specializzate per realizzare specifiche funzioni. Per utilizzarne una non dovete fare altro che infilarla sopra ad Arduino, creando quello che si chiama un «panino» di schede. Si possono impilare più schede riuscendole a controllare tutte senza conflitti perché ogni scheda occupa un certo numero di pin. Le shield sono sviluppate e prodotte da Arduino, ma anche altri famosi produttori di elettronica per maker hanno creato delle schede compatibili. L'importante è rispettare la piedinatura dei pin. Abbiamo shield con display LCD e pulsanti, con Ethernet, per il WiFi, per il Bluetooth, per controllare motori elettrici, con il GPS, con il GSM, con display e-Ink, datalogger con SD card e clock e perfino shield per collegare dispositivi medicali. Non dovete fare altro che acquistare quello che vi serve e infilarlo sul vostro Arduino.

Un'altra possibilità per semplificare lo sviluppo elettronico di circuiti è di utilizzare kit modulari come TinkerKit o Groove. Questi sistemi sono costituiti da una shield dotata di molti piccoli connettori. La shield s'infilava su Arduino poi ci sono molte piccole schedine che si possono collegare alla scheda madre. I collegamenti si fanno con dei semplici cavetti dotati di connettori standard. Ogni piccola schedina monta un singolo componente o svolge una semplice funzione. Avete a disposizione schede con un semplice led, un pulsante, schede con un relè, un potenziometro, un sensore di temperatura, per il gas o per la luce... Per comporre i circuiti dovete semplicemente cablare quello che vi serve alla scheda madre.

## Maker & Arduino

I maker hanno subito apprezzato la versatilità di Arduino e l'hanno utilizzato per realizzare una moltitudine di progetti, condividendo i risultati raggiunti, gli schemi elettrici e il software. Alcuni di questi progetti sono molto noti su Internet, altri si sono trasformati in startup. Vi presento qui alcuni lavori che mi hanno particolarmente colpito. In ogni lavoro, anche se noterete una certa aria da garage e da prototipo, apprezzate lo spirito di condivisione e percepite la passione di chi vi ha

lavorato. Questi maker (tra cui il sottoscritto) perdono ore e ore del loro tempo per lavorare a progetti che non sempre hanno fini commerciali. Le luci delle nostre cantine (o della mia mansarda) stanno accese fino a tardi, per lavorare a quello che ci sta veramente a cuore.

## **Stampanti 3D**

Arduino ha avuto un ruolo importante nello sviluppo delle stampanti 3D Open Source. La maggior parte delle schede elettroniche utilizzate oggi nelle stampanti deriva da Arduino o da una delle sue numerose varianti. Per realizzare una stampante con Arduino sono necessari alcuni particolari circuiti di pilotaggio per dei motori passo-passo chiamati driver e una manciata di transistor e termistori per controllare la temperatura. Molte delle schede oggi in commercio come le RAMPS derivano da una scheda Arduino modificata, con integrata l'elettronica necessaria per il funzionamento della stampante.

## **CNC e lasercutter**

Se con Arduino posso costruire una stampante 3D, perché non una laser cutter o una macchina CNC? In effetti, c'è poca differenza e l'elettronica potrebbe, in alcuni casi, essere anche più semplice. Una CNC non ha delle parti che si riscaldano e al massimo prevede un controllo di velocità per un utensile. Una semplice fresa CNC potrebbe quindi essere costruita con tre motori per pilotare gli assi X, Y e Z e un controllo di velocità per l'utensile.

Il progetto Lasersaur utilizza una scheda da collegare ad Arduino per pilotare una macchina Open Source per il taglio laser.

## **Droni**

I droni sono piccoli elicotteri multi rotore (cioè dotati di più eliche) capaci di volare in modo molto stabile e controllato. Per volare serve una certa «intelligenza» per il pilotaggio dei motori e per il controllo della stabilità dell'elicottero. Arduino è perfetto per gestire i motori, l'assetto, controllare un accelerometro e altri sensori di orientamento, oltre che ricevere comandi da una radiotrasmittente.

Anche in questo caso dopo i primi prototipi creati con delle semplici schede Arduino, sono comparse delle «board» specializzate, ma sempre compatibili con la piattaforma originale. In questo modo è sempre possibile intervenire sul programma di pilotaggio, modificando a piacere parametri e comportamenti.

## **ArduFonino**

Con Arduino è possibile realizzare un telefonino personalizzato. Non potrà essere un telefono di dimensioni ridotte, ma è totalmente personalizzabile. Uno dei progetti che si possono trovare in Rete si chiama ArduFonino ed è stato realizzato da Davide Aloisi (<http://www.davidealoisi.it/>). Sul suo sito racconta così il suo progetto:

*“ArduFonino è un cellulare Open Source creato con Arduino, capace di effettuare e ricevere chiamate e di inviare e ricevere messaggi, inoltre con la realizzazione di questo progetto ho voluto gettare le basi per la costruzione di un sistema telefonico elementare. L'idea mi è venuta per semplice curiosità, perché volevo fare qualcosa che nessuno avesse mai fatto, dimostrando a tutti che volere è potere.”*

Il telefonino utilizza Arduino, una shield GSM con alloggiamento per la SIM e una shield «custom» con display, tastiera, microfono e altoparlante. È molto semplice collegare Arduino ad altri dispositivi o sistemi utilizzando delle shield Bluetooth, WiFi, GSM.

## **Robotica**

Per applicare al meglio tutte le conoscenze su Arduino, non c'è nulla di più emozionante che

applicarle nella costruzione di un robot. Per un progetto del genere, Arduino può svolgere la funzione di controllo, leggere informazioni dai sensori e trasmettere i movimenti a degli attuatori, come motori e servo comandi. Oggi è semplice costruirsi un robot DIY (Do It Yourself – Fai da te), soprattutto se avete a disposizione una stampante 3D per realizzare tutte le parti meccaniche. In alternativa esistono parecchi kit «meccanici» che si possono acquistare e completare aggiungendo sensori, motori e altri dispositivi. Il robot può essere controllato da remoto via radio, ma la vera sfida è di renderlo autonomo, programmando l'intelligenza artificiale necessaria per fargli prendere le corrette decisioni e per farlo muovere nel mondo in autonomia, evitando ostacoli e risolvendo problemi. Arduino ha creato uno speciale kit robotico: un robot completo formato da sue schede, di cui una con due ruote motorizzate. Non avete che da aggiungere qualche sensore e iniziare a programmare.

## **Garduino**

Sono veramente tanti i progetti in cui è stato utilizzato Arduino. Basta fare qualche ricerca su Google per restare stupiti dalla fantasia e ingegnosità di molte realizzazioni. Qualcuno ha pensato di utilizzarlo per controllare lo stato delle proprie piante domestiche. La piccola scheda può misurare l'umidità del terreno, la temperatura, l'esposizione solare e irrigare le piante avvisandoci con una mail. Esistono tanti progetti per controllare il proprio giardinetto; il più famoso è Garduino, oggi noto come Growerbot, un sistema per irrigare le piante quando necessario, che può anche accendere delle luci supplementari se il sole non è sufficiente e che ci può avvisare quando le condizioni ambientali non sono adeguate, perché fa troppo caldo o troppo freddo.

L'unico limite è a fantasia, perché poi, in effetti, realizzare il circuito non è poi così complicato. Arduino può essere utilizzato anche per riparare oggetti che non funzionano più. Mi è capitato di usarlo per riparare il boiler di un camper. La ditta produttrice è fallita dieci anni fa e non si trovano più le parti di ricambio. Ho sostituito la scheda di controllo della caldaia con Arduino, collegandolo al sensore di fiamma, ai sensori di temperatura dell'acqua e all'elettrovalvola per il controllo del gas.

## **Siti di riferimento**

Il principale sito di riferimento è <http://www.arduino.cc>, anche se recentemente, in seguito ad alcuni problemi legali dei fondatori, è stato aperto un sito «clone» [arduino.org](http://arduino.org). La disputa è aperta e non si sa come andrà a finire, ma il sito originale è [arduino.cc](http://www.arduino.cc). Sul sito potete accedere al forum e a numerose risorse online, come tutorial, istruzioni per la programmazione, esempi di circuiti e informazioni di utilizzo delle schede.

Una ricerca su Google ci farà scoprire numerose risorse, siti, blog, tutorial, video, dispense e libri. C'è solo l'imbarazzo della scelta. Se potete però, chiudete il vostro computer e visitate un makerspace o uno degli Arduino User Group sparsi in tutt'Italia. Probabilmente ne esiste uno anche nella vostra città. Troverete tanti appassionati come voi disposti a fare una chiacchierata e a condividere con voi le loro esperienze. Gli AUG organizzano periodicamente eventi e incontri per illustrare nuovi argomenti, progetti e approfondimenti. Di solito queste comunità hanno anche una forte presenza on-line sui social media come Facebook, Twitter e G+.

# 2

## Hardware

Arduino è una scheda a microcontrollore... ma che cos'è un microcontrollore? È un termine che assomiglia molto a microprocessore, ma non è proprio la stessa cosa. I computer che usiamo tutti i giorni, a casa o in ufficio, utilizzano un microprocessore, cioè un circuito integrato o chip, specializzato nell'esecuzione di calcoli e nell'elaborazione di numeri e informazioni. Un microprocessore, per funzionare, ha bisogno di una o più memorie, di un disco, di alcune periferiche, di un video, di una tastiera, un mouse. Un microcontrollore, invece, è un piccolo computer: al suo interno trovano posto un microprocessore e una serie di dispositivi «integrati» che funzionano da memoria, da disco, da periferiche per permettergli di comunicare con il modo esterno.

Un microcontrollore ha bisogno di ben poche cose per funzionare: potrebbe lavorare semplicemente con i due fili che gli forniscono l'alimentazione. La scheda Arduino UNO utilizza un microcontrollore chiamato ATmega328, prodotto da ATMEL.

Caricare un software in un computer è un'operazione abbastanza semplice, ma per farlo con un microcontrollore servono, di solito, degli strumenti speciali. Per programmare Arduino invece, serve un semplice cavo USB, perché della programmazione si occupano un piccolo circuito, che è stato posto sulla stessa scheda elettronica, e uno speciale programma pre-caricato nell'ATmega328. Così è tutto più comodo e la programmazione è semplice e veloce. Tutto qui!

### Arduino UNO

Non esiste un solo Arduino, ma molti modelli, ognuno adatto a particolari esigenze e con specifiche proprietà. Ora descriveremo il modello «Arduino UNO»: la scheda più diffusa e comune, grande poco più di un pacchetto di sigarette. Quasi tutti gli altri modelli di Arduino presentano la stessa disposizione delle connessioni, in modo che ci sia compatibilità con le schede di espansione. I piedini (pin) di Arduino sono accessibili semplicemente infilando dei cavi elettrici con le estremità spelate in piccole file di fori chiamate header.

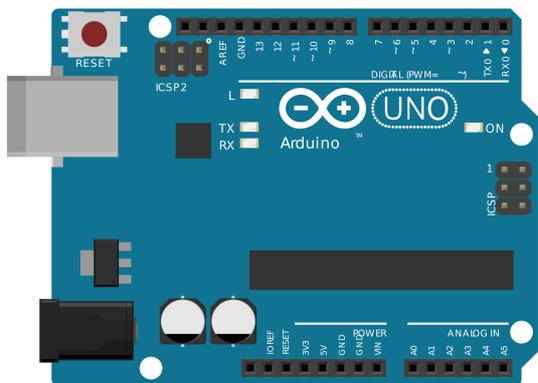


Fig 2.1 – La scheda Arduino UNO (Fritzing).

### Connettore USB

Ha un duplice scopo, infatti, si usa per alimentare la scheda utilizzando i cinque volt presenti nelle

porte USB, ma si può utilizzare anche per scambiare dati con Arduino (funzione molto utile per capire cosa accade a bordo).

## **Alimentazione**

Su un angolo del circuito stampato trovate un piccolo barilotto di colore nero. Vi potete inserire un jack da cinque millimetri (con diametro interno da 2,1 mm) per alimentare la scheda. La tensione fornita sarà poi livellata a cinque volt da un regolatore di voltaggio. Molte persone collegano una batteria da nove volt con una clip fornita di jack, per alimentare Arduino quando non c'è un computer nelle vicinanze.

## **GND**

GND significa Ground, cioè «terra» o «massa». È il polo negativo di alimentazione, anche chiamato il «comune» o «zero volt». Tra gli header di Arduino avete a disposizione ben tre posizioni in cui infilare delle connessioni a massa.

## **5V**

Questo pin fornisce la tensione a cinque volt regolata e stabilizzata da Arduino. Lo stabilizzatore interno può fornire fino a circa un ampere di corrente.

## **3.3V**

Molti chip e sensori moderni si alimentano a 3,3 volt, una tensione inferiore rispetto ai cinque volt, che produce meno calore. Non tutti hanno nel cassetto dei propri componenti un regolatore a 3.3 volt, ecco perché ne trovate uno su Arduino.

## **Vin**

Questo pin è collegato direttamente all'ingresso di alimentazione e si può utilizzare per prelevare una tensione di alimentazione più elevata necessaria per far funzionare componenti collegati ad Arduino. Potreste avere dei motori che funzionano a dodici volt: per utilizzarli, alimenterete Arduino a dodici volt attraverso la presa di alimentazione e porterete la tensione di dodici volt ai motori, prelevandola dal pin Vin.

## **AREF**

Arduino può leggere delle tensioni analogiche comprese tra zero e cinque volt che trasformerà poi in un numero tra zero e 1024. Il passo minimo, cioè la precisione della misura sarà pari a cinque volt diviso per 1024, cioè 4,88 millivolt. Arduino usa sempre una tensione di riferimento interna, prelevata dalla tensione di alimentazione che potrebbe non essere molto precisa e che dovrebbe valere cinque volt (ma potreste trovare 4,8 volt). Se volete misurare qualcosa con grande precisione, potreste usare una tensione di riferimento molto stabile e precisa che applicherete su AREF. Se la massima tensione analogica che volete leggere arriva a tre volt, potreste ottenere una maggiore precisione se, su AREF, applicaste tre volt. In questo modo il passo minimo sarà di tre volt diviso per 1024, e cioè di 2,93 millivolt. La tensione applicata a AREF non deve mai essere superiore a cinque volt, altrimenti si rischierebbe di danneggiare Arduino.

## **RESET**

La scheda è dotata di un tasto per eseguire il reset. Se lo premerete, l'esecuzione del programma si fermerà e tutto ripartirà da capo, come se la Arduino fosse stato appena acceso. Potete collegare un pulsante esterno per pilotare il RESET.

## **PIN 0-13**

Arduino ha quattordici pin che possono essere configurati per funzionare da ingressi o uscite. La configurazione si fa nel software. Arduino è un dispositivo digitale, quindi i pin possono generare o

leggere un valore alto o basso, e quindi pari a zero o cinque volt. Un pin digitale configurato come uscita non potrà mai fornire, per esempio, un valore di 2,5 volt, ma unicamente zero o cinque volt. Alcuni pin riportano accanto al numero identificativo una piccola «serpentina» chiamata «tilde» (~): questi pin possono generare un segnale particolare molto utile per pilotare motori elettrici o per regolare a piacere l'intensità luminosa di un LED. Questo segnale elettrico speciale è chiamato pulse width modulation (PWM) e lo descriveremo più avanti. I pin in grado di fornire un segnale PWM sono: 3, 5, 6, 9, 10, 11.

I pin 0 e 1 sono contrassegnati anche con TX0 e RX0 e sono collegati alla porta seriale del chip: possono essere utilizzati per collegare Arduino a qualsiasi altro dispositivo che disponga di una porta seriale.

## A0-A5

Arduino ha sei pin in grado di leggere livelli analogici e di convertirli in un numero che si potrà utilizzare all'interno degli sketch. Questi particolari ingressi sono posti a parte e contrassegnati dalle sigle: A0, A1, A2, A3, A4, A5.

## ICSP

Vicino alla piccola scritta ICSP trovate sei piedini che si possono utilizzare per comunicare con l'ATMega328. I piedini costituiscono un'interfaccia seriale usata da periferiche e altri microcontrollori. I pin ICSP si possono usare per programmare direttamente Arduino (usando una speciale «pennetta»). Il sistema di comunicazione si chiama SPI, cioè Serial Peripheral Interface, e prevede un circuito principale che conduce la comunicazione (master) e uno o più periferiche (slave). Due dei sei pin sono utilizzati per l'alimentazione (cinque e zero volt), una linea serve per selezionare la periferica con cui comunicare e gli altri tre fili sono:

- MISO (Master In Slave Out) – per inviare dati al master,
- MOSI (Master Out Slave In) - su cui passano i dati per lo slave,
- SCK (Serial Clock) – il segnale di sincronizzazione (clock) per la comunicazione.

In modo non troppo intuitivo, per comunicare con uno slave, si deve mettere il suo piedino a livello basso (zero volt o GND).

## ICSP2

Un secondo set di pin SPI è posto vicino al chip utilizzato per la gestione dell'interfaccia USB. Il chip è un piccolo microcontrollore e anche a lui serve un programma per pilotare la porta USB. Il programma di controllo è stato caricato utilizzando questi sei pin etichettati con "ICSP2".

## Una scheda non basta!

Arduino UNO è la scheda più famosa, ma non è l'unica, in questi anni Arduino ha creato molte altre schede, per coprire ogni necessità. Alcune di queste schede sono state progettate in collaborazione con altre realtà come SparkFun<sup>1</sup>, Intel o Texas Instrument e utilizzano a volte diversi tipi di microcontrollori o SoC, anche se la maggior parte delle schede impiega prodotti ATMEL. Le possibilità per i Maker sono veramente numerose: esiste una scheda per ogni necessità, con vari formati, costi e capacità. Di seguito presento una breve panoramica degli «Arduino» più famosi. Alcune di queste schede sono ormai fuori produzione, ma se ve ne servisse una, potreste sempre recuperare i progetti Open Source e replicarvela.

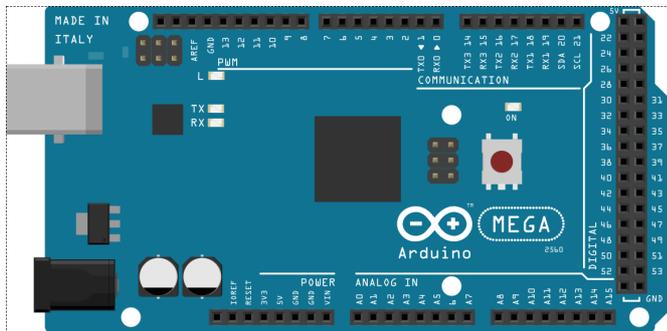
---

<sup>1</sup> SparkFun ([www.sparkfun.com](http://www.sparkfun.com)) è una società nata nel 2003 dalla passione di Nathan Seidle per l'elettronica e con l'intento di fornire schede, kit e conoscenze ai maker come lui. All'inizio SparkFun rivendeva schede prodotte da altre aziende, ma poi ha iniziato a progettare e realizzare propri prodotti.

## Arduino 2009, 10000, NG...

La scheda Arduino 2009, prodotta nell'anno 2009, è stata la progenitrice di Arduino UNO. Prima della 2009 ci furono altri modelli come la 10000 e la NG. Queste schede non sono state solo dei prototipi, ma erano in vendita fino a poco tempo fa. La «diecimila» prende il suo nome dal fatto di aver venduto diecimila schede. Un vero successo per una «piccola» start-up italiana di nome Arduino! Utilizzavano tutte un chip ATmega168 che fu sostituito solo nelle ultime serie delle «2009» con l'ATmega328, perché dotato di maggiori risorse. Le differenze maggiori sono nella comunicazione con il computer necessario per la programmazione e nei continui miglioramenti apportati al design del circuito e del prodotto. Le prime schede utilizzavano una porta seriale RS232 di quelle con il connettore a nove pin, una volta abbastanza diffusa su tutti i PC, ma ormai introvabile. La porta seriale è stata sostituita da una porta USB, che richiede un chip specializzato (FT232), impiegato nei primi modelli di Arduino e poi anch'esso rimpiazzato da un semplice microcontrollore ATMEL ATmega8U2, programmato opportunamente.

## Arduino Mega 2560

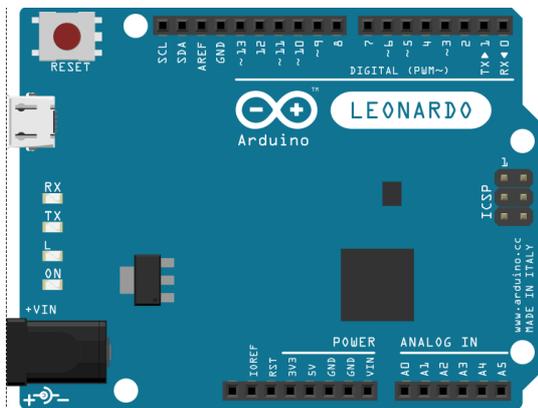


fritzing

Fig 2.2 – Arduino Mega 2560.

Quando Arduino non basta, si può ricorrere ad Arduino Mega: un Arduino UNO estrogenato! Ha ben cinquantaquattro porte d'ingresso/uscita, sedici ingressi analogici e quattro porte seriali. In più è compatibile nella forma, nelle tensioni e nella velocità (16MHz) con la scheda UNO. Per offrire così tante porte è stato utilizzato un microcontrollore ATmega2560. Esisteva una versione precedente di questa scheda, semplicemente chiamata “MEGA” su cui trovava posto un ATmega1280. Su queste schede la gestione della porta USB è affidata a un secondo microcontrollore, un ATmega16U2, ed è realizzata tutta da un software e non da un chip specializzato e più costoso come si usava fare una volta.

## Arduino Leonardo



fritzing

Fig 2.3 – Arduino Leonardo.

L'Arduino Leonardo è una scheda del tutto simile a Arduino UNO, solo che utilizza un microcontrollore differente, l'ATMega32u4 al posto dell'ATMega328P. La sua piedinatura e le tensioni di alimentazione sono del tutto compatibili con Arduino UNO. L'unica differenza è che mentre Arduino UNO utilizza un secondo chip per le comunicazioni tramite USB e la programmazione, sulla scheda Leonardo la comunicazione USB è gestita interamente dall'unico chip. La Leonardo, quando è collegata a un computer, può apparire come un dispositivo HID, cioè un mouse o una tastiera, oltre a presentare la porta seriale per le comunicazioni e la programmazione. Il costo della scheda è leggermente inferiore a quello di Arduino UNO.

## Arduino Ethernet

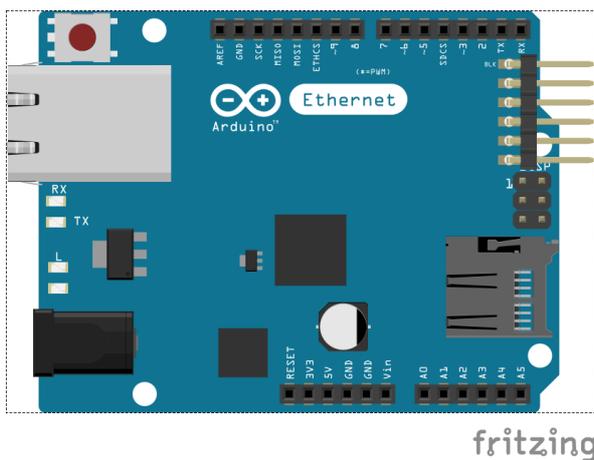


Fig 2.4 – Arduino Ethernet.

Arduino Ethernet è una variante del classico Arduino UNO, dotata di scheda di rete Ethernet integrata. La scheda di rete occupa i pin 10, 11, 12 e 13, che quindi non sono utilizzabili. Sul circuito stampato c'è un alloggiamento per una scheda microSD su cui si possono salvare file e pagine html che un web-server installato sulla scheda può pubblicare in rete.

Sulla scheda Arduino Ethernet non è presente il connettore USB e quindi per programmare il microcontrollore si deve utilizzare una piccola scheda «USB Serial» venduta separatamente che s'infilza in un connettore orizzontale formato da sei pin. La scheda elimina la necessità di aggiungere una Ethernet shield ad Arduino ed è stato il primo tentativo di creare un prodotto compatto in grado di comunicare e scambiare dati in Rete.

## Arduino YUN

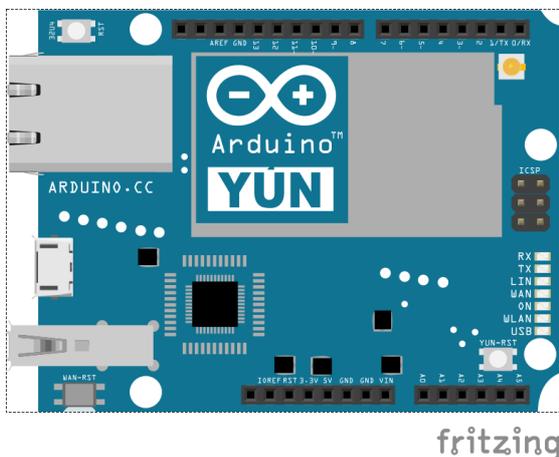


Fig 2.4 – Arduino YUN.

Arduino YUN è Arduino per Internet degli Oggetti. YUN unisce il classico Arduino con un piccolo chip, celato da una schermatura metallica, su cui gira un sistema operativo GNU/Linux. YUN ha una scheda WiFi integrata, una presa Ethernet e nasce per essere «connesso» e facilitare le comunicazioni. Il chip aggiuntivo è un SoC Atheros AR9331, cioè un System On Chip, un piccolo computer su cui è stata caricata una distribuzione GNU/Linux (OpenWrt). Il sistema operativo Linux può eseguire task, programmi, ospitare un web server, pubblicare servizi web, eseguire operazioni schedate e comunicare in modo semplice con l'ATMega con cui condivide la scheda. La scheda YUN, appena accesa, crea una rete WiFi a cui ci si può collegare con un computer per configurarla e per farla connettere alla rete WiFi esistente. YUN si programma con un cavo USB oppure, una volta connesso a una rete WiFi, direttamente da wireless e quindi da remoto!

L'ATMega e l'Atheros comunicano attraverso una speciale libreria chiamata «bridge», che permette ad Arduino di invocare o pubblicare servizi, inviare mail e interagire con la Rete con grande semplicità. Il bridge facilita anche la cooperazione tra ATMega e Linux: per esempio direttamente da uno sketch di Arduino potete eseguire programmi presenti su Linux. Il chip utilizzato è un ATMega32u4 simile a quello utilizzato sulle schede Leonardo. Come le Leonardo, anche YUN può essere visto da un computer come un mouse o una tastiera.

## Arduino DUE

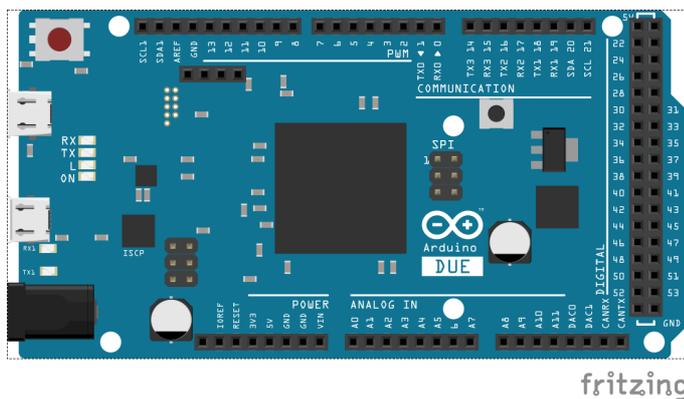


Fig 2.5 – Arduino DUE.

Arduino DUE assomiglia molto ad Arduino Mega, tanto che è facile confondersi perché hanno quasi lo stesso aspetto. Arduino DUE è molto più potente perché utilizza un microcontrollore di tipo ARM (Cortex-M3 SAM3X8E di Atmel). L'ARM lavora a 84 MHz contro i 16 MHz della scheda Mega e ha trentadue bit al posto di otto. Anche se molto simile al Mega, la differenza maggiore è nelle tensioni utilizzate: Arduino DUE lavora a 3.3 e non a cinque volt. Se non ci si accorge e gli si collegano dei dispositivi o delle shield a cinque volt, si rischia di danneggiarla irreparabilmente. A differenza delle altre schede Arduino, la DUE ha due uscite analogiche contrassegnate con DAC0 e DAC1. A queste uscite sono collegati dei convertitori digitali-analogici (DAC) con cui è possibile creare dei segnali analogici veri e propri per generare suoni o per controllare altri dispositivi analogici. Arduino DUE può utilizzare una speciale libreria chiamata «audio library» con cui riprodurre con facilità file audio presenti su una scheda SD.

## Arduino Micro

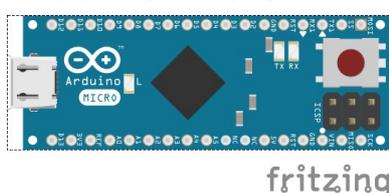
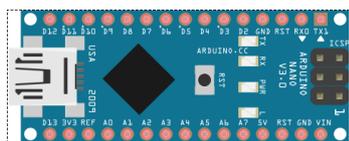


Fig 2.6 – Arduino Micro.

Arduino Micro utilizza un ATmega32u4 come la scheda Leonardo. Ha dimensioni molto ridotte. Offre venti pin digitali (in/out) e dodici ingressi analogici. A differenza dei classici «Arduini», ha una forma che lo rende adatto a essere inserito direttamente su delle breadboard. La piccola scheda ha una porta micro-USB che si utilizza per programmarlo. Sul circuito stampato trovano posto anche un tasto per il reset e i sei pin ICSP. La scheda è stata sviluppata in collaborazione con Adafruit ed è adatta per realizzare prototipi di piccole dimensioni.

## Arduino Nano

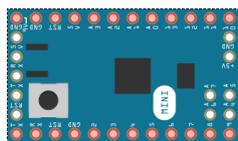


fritzing

Fig 2.7 – Arduino Nano.

Arduino Nano è ancora più piccolo di Arduino Micro. Inizialmente utilizzava l'ATmega168, poi sostituito con ATmega328. Può essere inserito su breadboard e si programma da USB. È comparabile con Arduino Duemilanove.

## Arduino Mini

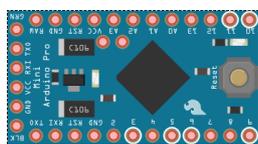


fritzing

Fig 2.8 – Arduino Mini.

Il Mini è ancora più piccolo di Arduino Nano. Utilizza un ATmega328 e ha quattordici pin digitali e otto ingressi analogici. Non ha nessuna porta USB e si programma utilizzando un cavo USB-Seriale che va collegato con dei jumper agli appositi pin. Si utilizza quando le dimensioni sono molto critiche. La scheda non ha molte protezioni e bisogna fare molta attenzione alla tensione di alimentazione che non deve mai oltrepassare i nove volt.

## Arduino Pro mini

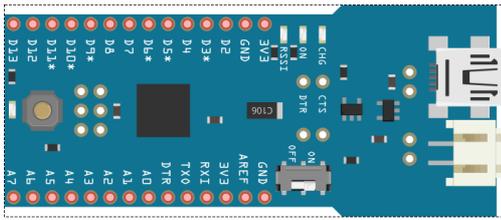


fritzing

Fig 2.9 – Arduino Leonardo.

Arduino Pro Mini impiega un ATmega328: ha quattordici pin digitali e sei ingressi analogici. Non ha una porta USB e si programma utilizzando un cavo USB-Seriale che va collegato con dei jumper agli appositi pin. Le sue dimensioni sono minime e si può utilizzare anche per prototipi e installazioni permanenti. Arduino Pro Mini si può acquistare in due versioni, una a 8 MHz e alimentata a 3,3 volt e l'altra a 16 MHz alimentata a cinque volt. La scheda è stata progettata da Sparkfun.

## Arduino FIO

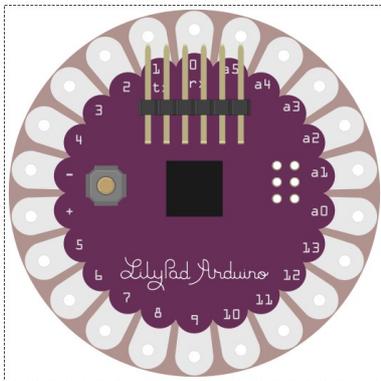


fritzing

Fig 2.10 – Arduino FIO.

Se desiderate un prototipo wireless, Arduino FIO è la scelta giusta. La scheda può ospitare un ricetrasmittitore XBee che può comunicare con una rete di ricetrasmittitori simili. FIO ospita un ATmega328P alimentato a 3,3 volt per essere compatibile con l'XBee. La FIO ha quattordici pin digitali, sei ingressi analogici e si programma con un cavo USB-Seriale (o FTDI Cable) oppure via radio utilizzando un secondo XBee connesso a un computer. È stata progettata da SparkFun.

## Lilypad Arduino



fritzing

Fig 2.11 – Arduino Lilypad.

Con Lilypad è possibile integrare Arduino con abiti e tessuti. La Lilypad nasce per realizzare prototipi e progetti “wearables”, dove al posto dei cavi elettrici si possono usare fili tessili formati da seta e argento: i circuiti si tracciano con ago e filo. Esistono componenti speciali che possono essere cuciti con facilità e anche tessuti particolari in grado di reagire al passaggio di corrente o capaci di funzionare come sensori: sono gli smart-textiles o e-textiles. La Lilypad utilizza un chip ATmega168 con consumi molto ridotti. La scheda ha quattordici porte digitali che possono essere utilizzate come ingressi o uscite e sei ingressi analogici. Le Lilypad si programmano utilizzando un cavo convertitore USB-Seriale (FTDI).

Esiste una versione chiamata **Lilypad Simple** che utilizza un microcontrollore ATmega328 ma con solo nove pin digitali. Un'altra variante è la **Lilypad Simple Snap** che al posto dei fori ha dei bottoni a clip per facilitare il cablaggio. Sul lato inferiore della “Snap” trova posto una piccola batteria ai polimeri di litio che si può ricaricare collegando la scheda a un cavo FTDI.

Le schede Lilypad sono state progettate da Leah Buechley e sono distribuite da SparkFun.

## LilyPad USB

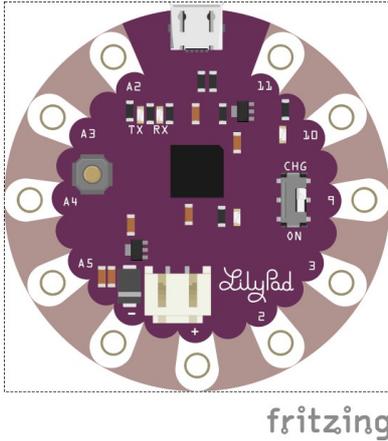


Fig 2.12 – Arduino LilyPad USB.

La scheda di prototipazione Arduino LilyPad USB include un connettore micro USB con cui collegarla direttamente a un pc: si può programmare senza bisogno del cavo FTDI. Questa variante delle LilyPad utilizza un ATmega32u4 che lavora a 8MHz.

## Mega ADK

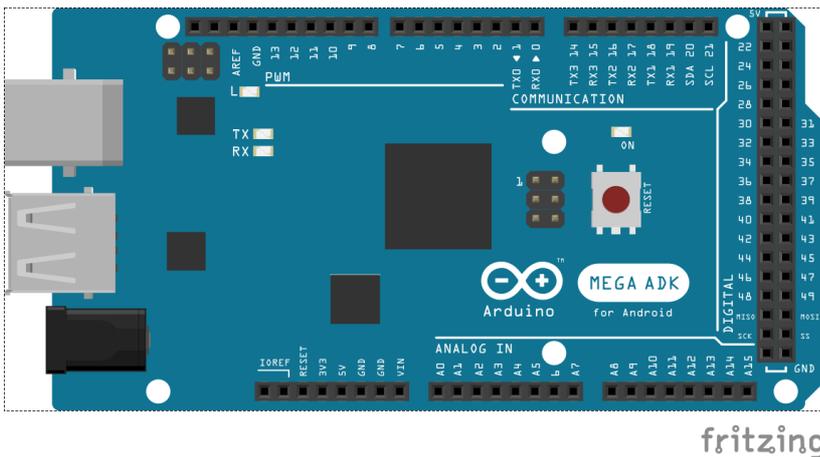


Fig 2.13 – Arduino Mega ADK.

La scheda Arduino Mega è realizzata anche in una versione per Android chiamata ADK (Accessory Development Kit), per facilitare lo sviluppo di accessori per i dispositivi con sistema Android. L'ADK comprende una serie di specifiche di Google per realizzare dispositivi compatibili con i telefoni Android<sup>2</sup>. Questa versione di Arduino utilizza un microcontrollore ATmega2560 e può facilmente essere collegata a un telefono Android utilizzando un cavo USB e grazie al chip MAX3421, in seguito sostituito da un microcontrollore ATmega8U2 (come sulle schede Leonardo). Arduino Mega ADK ospita un "USB host", così che vi si possa collegare qualsiasi periferica USB.

<sup>2</sup> <http://developer.android.com/tools/adk/index.html>

## Arduino Esplora

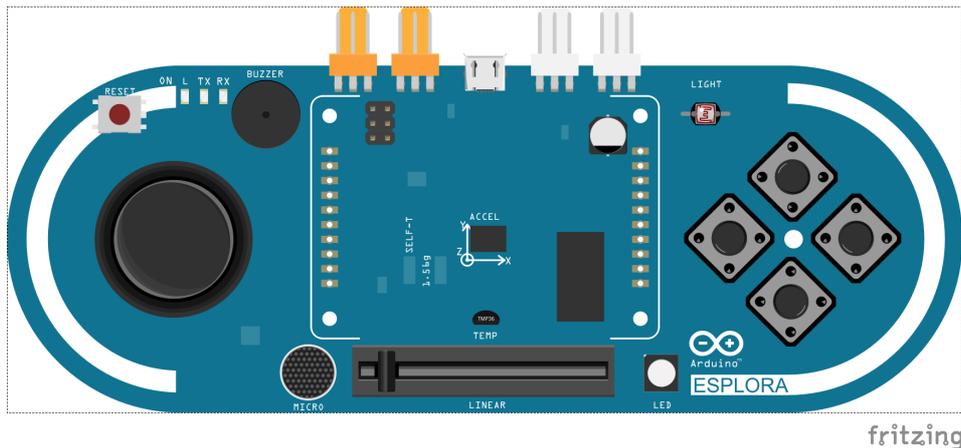


Fig 2.14 – Arduino Esplora.

Arduino Esplora assomiglia molto a un controller per videogiochi: ha un piccolo joystick resistivo, dei pulsanti, dei led, un microfono, un buzzer, un accelerometro e altri piccoli “accessori” cablati e pronti all’uso. Su Arduino Esplora si può aggiungere un display LCD a colori, sempre prodotto e fornito da Arduino, con cui creare una console personalizzata per video giochi. A differenza delle altre schede Arduino, è dotata di un completo set di sensori e attuatori che la rendono già pronta per l’uso.

Esplora utilizza un microcontrollore ATmega32U4 a 16 MHz, lo stesso utilizzato sulla scheda Leonardo e che può essere visto da un PC come un mouse o una tastiera.

## Arduino Robot

Tra i tanti progetti che chi si avvicina ad Arduino vorrebbe realizzare c’è di sicuro un robot. Per farlo vi serviranno un po’ di competenze meccaniche (che potreste apprendere leggendo un libro come «Making Things Move - DIY Mechanisms for Inventors, Hobbyists, and Artists» di Dustyn Roberts), oltre che alle competenze elettroniche. Arduino vi facilita il compito offrendovi un robot pronto all’uso: Arduino Robot, un piccolo automa formato da due circuiti stampati, forniti di due ruote motrici e un assortimento di sensori e attuatori utili per realizzare un robot completo in poco tempo. Il robot è pronto all’uso, basta programmarlo e può subito partire per esplorare il mondo.

Arduino Robot ha due microcontrollori, uno per ogni scheda. La scheda inferiore, quella con i motori è chiamata la «Motor Board», quella superiore prende il nome di «Control Board». La scheda con i motori deve solo occuparsi della gestione dei movimenti e dei motori, La scheda di controllo ha il compito di gestire l’intelligenza artificiale necessaria alle strategie dell’automa. Ogni scheda si comporta come un singolo Arduino e si può programmare indipendentemente. Tutte e due le schede utilizzano un ATmega32u4. A bordo di Arduino Robot trovate un display LCD a colori, cinque pulsanti, una bussola, uno slider, un lettore per schede SD, un altoparlante e quattro zone per prototipazione in cui è possibile saldare componenti direttamente sulla scheda.

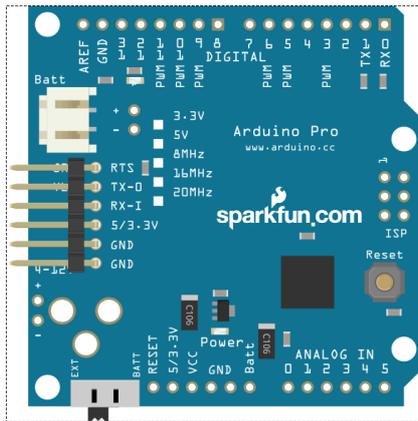
## Arduino TRE

Tra le novità in arrivo nei prossimi mesi, c’è la nuova scheda Arduino TRE, che utilizza un microcontrollore Sitara AM335x da 1GHz (ARM Cortex A8 di Texas Instruments) e un «classico» ATmega32u4. La scheda è molto più grande di Arduino perché si avvicina al mondo di Raspberry e BeagleBone (con cui, tra l’altro, Arduino ha collaborato per la realizzazione della TRE). Arduino TRE ha un sistema operativo (GNU/Linux) e un’uscita video HDMI, insomma è un piccolo computer con 512 Mb di memoria. L’integrazione con Arduino permette l’utilizzo delle shield già esistenti e apre le porte a molte nuove possibilità, soprattutto per il mondo delle stampa 3D e per i nuovi sviluppi nel campo di Internet degli oggetti.

## Arduino ZERO

Nel 2014, alla Maker Faire di San Mateo (California), Arduino ha presentato i prototipi di una nuovissima scheda: Arduino ZERO. La scheda sarà presto disponibile e porterà una ventata di novità alla storica piattaforma. La maggior parte dei prodotti basati su microcontrollori AVR lavorano a otto bit, mentre Arduino ZERO utilizzerà trentadue bit. Sarà più facile utilizzare dati di grandi dimensioni o con numeri decimali. La nuova scheda utilizza un microcontrollore ARM di ATMEL (ATSAMD21G18 ARM Cortex M0+). La scheda è più veloce del classico UNO, infatti lavora a 48 MHz, con una RAM di 32Kb e una flash di 256 Kb. Un'altra novità è la possibilità di debuggare gli sketch, utilizzando l'Atmel Embedded Debugger (EDBG). Sarà possibile seguire ancora meglio quello che accadrà all'interno del microcontrollore e quindi intervenire sugli errori.

## Arduino Pro

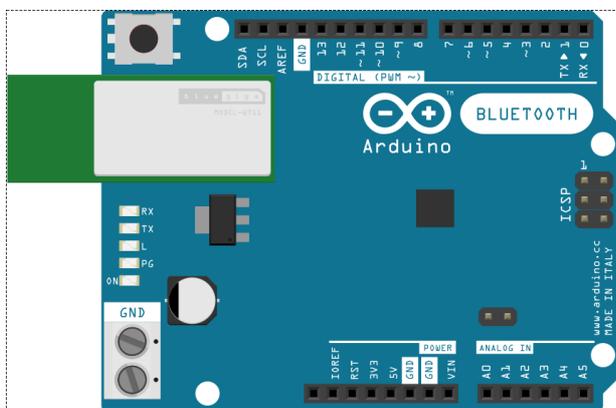


fritzing

Fig 2.15 – Arduino Pro.

SparkFun, in collaborazione con Arduino, ha creato una scheda dal nome Arduino Pro. La scheda utilizza il chip ATmega 168 oppure, i modelli più recenti, l'ATmega328. È alimentata a 3,3V a lavora a 8 Mhz oppure a 16 MHz a cinque volt. È compatibile con Arduino ma non ha il circuito USB-Seriale e per la programmazione si deve utilizzare una scheda a parte da collegare a una serie di pin orizzontali. La scheda non monta degli headers: ha solo dei fori in cui potete saldare direttamente dei fili alla scheda. Arduino Pro si può alimentare attraverso la connessione FTDI oppure utilizzando un connettore per una batteria ricaricabile. Uno switch permette di scegliere la fonte di alimentazione. Arduino Pro è distribuita da SparkFun.

## Arduino BT



fritzing

Fig 2.16 – Arduino BT.

Tra le tante varianti di Arduino, non poteva mancare quella con il BlueTooth integrato. Arduino BT è stata progettata e distribuita da SparkFun in collaborazione con Arduino. Attualmente è fuori produzione. Arduino BT utilizzava un ATmega168, in seguito sostituito da un ATmega 328. Il modulo BlueTooth, BlueGiga WT11, era applicato su un angolo della scheda e questa poteva essere programmata senza cavo, anche perché non è disponibile nessun connettore USB! Arduino BT è compatibile con tutte le altre schede Arduino, presentando la solita disposizione degli header. La BT è stata ben presto soppiantata da soluzioni più moderne e potenti come le YUN, che affianca il classico ATmega328 a un SoC con sistema operativo e semplifica la comunicazione della scheda con Internet.

## Galileo

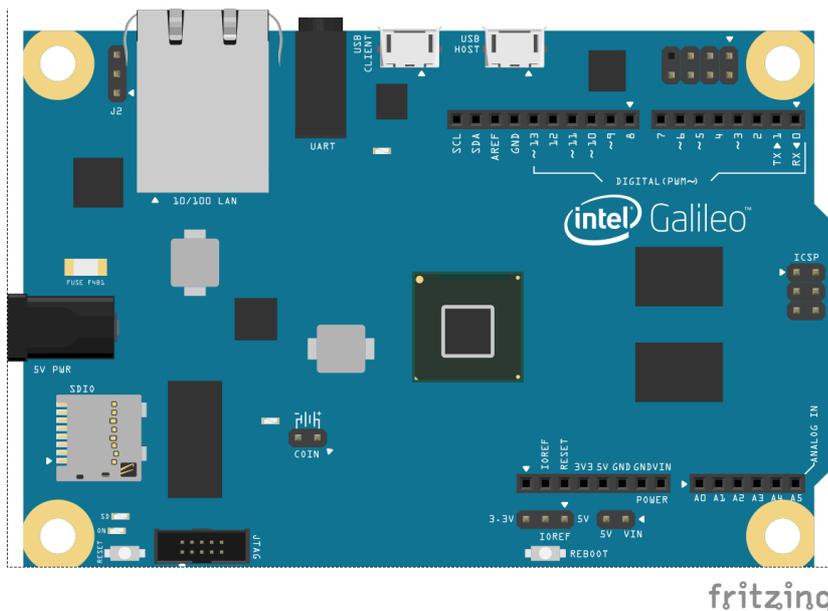


Fig 2.17 – Arduino Galileo.

Intel, leader nella produzione di microprocessori e chip, si è unita ad Arduino per realizzare una scheda con a bordo un microcontrollore della famiglia Pentium Intel Quark SoC X1000 a 32 bit. L'X1000 può lavorare a 400 MHz e 3,3 volt oppure a cinque volt, così che tutte le Shield per Arduino UNO siano compatibili anche con la Galileo. Per la programmazione si utilizza il software di Arduino. La Galileo dispone anche di una porta mini-PCI Express a cui potete collegare periferiche di solito utilizzate sui PC. Tra le altre caratteristiche trovate: una porta Ethernet 10/100Mb, uno slot per micro-SD, una porta seriale RS-232, delle porte USB Host e Client e una flash da otto Mb.

Intel sta sviluppando altre interessanti schede pensate per applicazioni di Internet degli Oggetti, come la piccola e potente Edison, anch'essa compatibile con il mondo Arduino. La Edison è una scheda grande come una memoria SD che può essere utilizzata sola oppure inserita su una speciale break-out board che la rende compatibile con il mondo Arduino e facilita l'accesso ai suoi pin. La piccola Edison utilizza un Intel Atom SoC Dual Core, affiancato da un modulo WiFi e un BlueTooth 4.0. Sulla Edison può essere installato Linux (Yocto Linux) o il nuovo Windows 10. La scheda si può programmare con l'IDE di Arduino o utilizzando Node.js, Python e molte altre soluzioni.

## Sanguino

Poco tempo dopo la comparsa e il successo di Arduino UNO, si affacciò sul mercato la scheda Sanguino, legata al mondo delle stampanti 3D (e da cui è derivata la scheda **Sanguinololu**).

Sanguino utilizza il microcontrollore ATmega644P, il più potente chip in formato DIL<sup>3</sup> prodotto da ATMEL. La scheda ha un maggior numero di contatti e si colloca a metà strada tra Arduino UNO e Arduino Mega. Può essere programmata con il software di Arduino. Lavora sempre a 16 MHz, ha 44 pin, 4 Kb di memoria RAM, 64 Kb di memoria Flash e 2 Kb di memoria EEPROM.

Recentemente si è reso disponibile anche l'ATmega1284P, ancora più potente del 644.

## Arduino è ATMEL

La maggior parte dei prodotti Arduino utilizza dei microcontrollori prodotti da ATMEL. Il più comune è l'ATmega328, utilizzato per produrre Arduino UNO. ATMEL ha un'intera famiglia di microcontrollori «ATmega», alcuni con prestazioni ridotte, altri con maggior numero di pin e funzioni integrate. Ho raccolto qui di seguito una tabella con le caratteristiche principali dei chip a confronto.

Nella tabella troverete riportate le seguenti caratteristiche dei chip:

- Velocità – la frequenza di lavoro massima del chip;
- RAM – la dimensione della memoria «volatile», quella usata per compiere i calcoli e le operazioni;
- Flash – è la memoria in cui è salvato il vostro programma e funziona un po' come un disco;
- EEPROM: la memoria «permanente» in cui salvare parametri o configurazioni che non si devono cambiare spesso... anche perché ha un numero molto limitato di scritture;
- GPIO – indica il numero di pin che possono essere utilizzati come ingressi o uscite digitali;
- Ingressi Analogici – è il numero d'ingressi analogici offerti dal chip. Questi pin corrispondono ad altrettanti convertitori analogico-digitali, cioè dei circuiti interni al microcontrollore che trasformano un segnale analogico in un numero intero che può essere utilizzato all'interno dei vostri programmi.;
- PWM (Pulse Width Modulation) – alcuni pin possono generare un particolare tipo di segnale a onda quadra, molto efficace per controllare il funzionamento di motori o la luminosità di LED;
- Porte seriali – alcuni microcontrollori offrono una o più linee seriali, utili per scambiare dati con altri dispositivi o per la programmazione dello stesso chip;
- Alimentazione – l'intervallo di tensioni di alimentazione ammissibile per il chip.

Questi chip sono acquistabili on-line in siti come RS-Components, Farnell, Mouser o DigiKey. Alcuni di questi, come l'ATmega328, sono ancora prodotti nel formato DIL, che può essere utilizzato con una breadboard. Aggiungendo pochi altri componenti potrete realizzare una scheda a microcontrollore personalizzata con minima spesa! Se v'interessa percorrere questa strada, troverete interessanti, oltre ai chip della famiglia ATmega, gli ATTiny, i fratelli minori... con caratteristiche contenute e prezzi molto più bassi. Sono ottimi per ingegnerizzare i vostri prototipi realizzati con Arduino, perché il costo di un ATTiny può essere anche inferiore a un euro.

---

3 Dual In Line, cioè con i pin in doppia fila e quindi semplice da usare anche su una breadboard

<b>Sigla</b>	<b>Clock (MHz)</b>	<b>RAM (Kb)</b>	<b>EEPROM (b)</b>	<b>Flash (Kb)</b>	<b>GPIO</b>	<b>Pin Analogici</b>	<b>PWM</b>	<b>Porte seriali</b>	<b>Alimentazione (V)</b>
<b>ATMega168</b>	20	1	512	16	23	8 in	6	1	1.8 - 5.5
<b>ATMega328</b>	20	2	1.024	32	23	8 in	6	1	1.8 - 5.5
<b>ATMega1284</b>	20	16	4.096	128	32	8 in	6	2	1.8 - 5.5
<b>ATMega2560</b>	16	8	4.096	256	86	16 in	16	4	1.8 - 5.5
<b>ATMega644P</b>	20	4	2.048	64	32	8 in	6	2	1.8 - 5.5
<b>SAM3X8E (ARM)</b>	84	64+32	16.384	2 x 256	103	16 in + 2 out	16	3/2	1.62 - 3.3

*Tab 2.1 – Tabella comparativa di alcuni microcontrollori ATMEL.*

## L'elettronica non è il mio forte: shield, TinkerKit e Groove

Per realizzare un prototipo con Arduino non è necessario conoscere a fondo l'elettronica. Avere qualche conoscenza non guasta e se siete a digiuno di teoria vi consiglio di dare un'occhiata al mio libro "Elettronica per Maker – Guida Completa".

Per iniziare vi servirà una breadboard e qualche componente elettronico come resistenze, led, potenziometri e qualche sensore. Se proprio volete evitare il contatto con fili e resistenze, potete acquistare un Arduino Esplora, già fornito di un assortimento minimo di componenti e sensori.

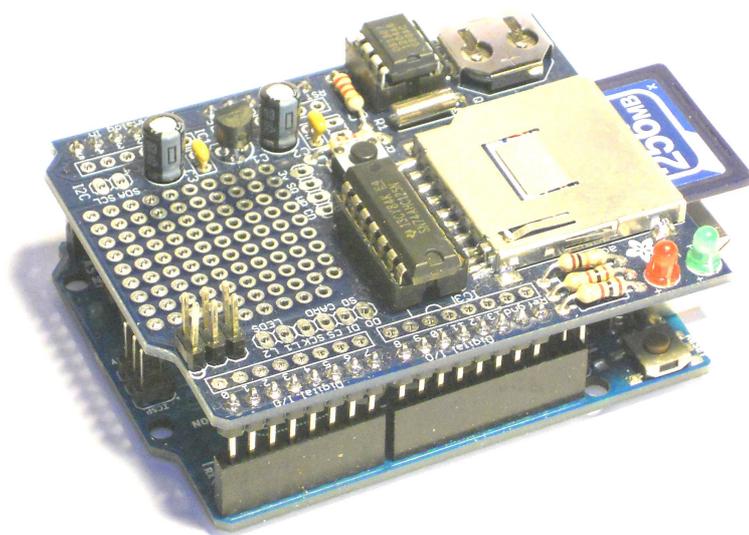


Fig 2.18 – Un «panino» composto da Arduino e un Data Logger Shield..

Arduino nasce come sistema modulare e tutte le schede hanno una piedinatura particolare, su cui sono montati degli header, cioè dei connettori per unire rapidamente tra di loro i circuiti stampati. Gli header permettono di raggiungere ogni pin di Arduino. Una scheda elettronica che rispetta la piedinatura degli header si chiama **shield** e si può infilare sopra alla scheda con il microcontrollore aggiungendo delle nuove funzionalità.

Esistono shield pre-assemblate in grado di offrirvi qualsiasi tipo di componente, sensore o sottosistema. Le prime shield sono state sviluppate e prodotte da Arduino, presto seguite da shield prodotte da SparkFun o Seeedstudio e da tanti altri produttori di hardware. Esistono anche delle shield «nude» dotate solo degli headers. Su queste shield potete saldare a piacere dei componenti elettronici per creare delle schede personalizzate. Potete impilare più shield, una sopra all'altra, ma avete un limite. Ogni shield impegna un certo numero di pin di Arduino e già con due schede potreste arrivare al limite delle possibilità. Quando impilate le schede una sull'altra formate un «panino». Tra le shield più comuni troviamo:

### Ethernet Shield

Alcuni anni fa quando non esistevano ancora schede come Arduino YUN e si parlava ancora poco di Internet degli Oggetti, questa shield, equipaggiata con un chip WizNet, permetteva di comunicare con Internet e di far twittare le prime caffettiere. La scheda occupa i pin «alti» dal numero «10» al

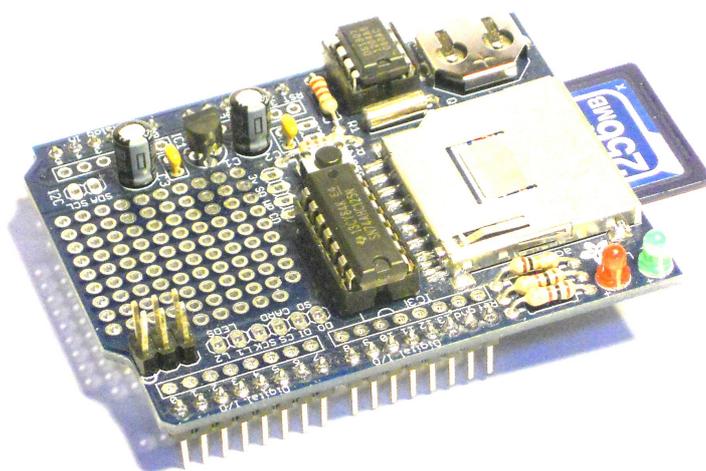
«13». Per utilizzare le funzioni di rete sono state create delle librerie e numerosi esempi, tutti inclusi nell'IDE. Le librerie operano a basso livello e vi troverete a processare pagine web carattere per carattere...

## **Wifi Shield**

Con questa scheda potrete collegare Arduino a una rete WiFi. La scheda è abbastanza costosa ed è l'«alternativa wireless» alla Ethernet Shield. Recentemente sono comparse delle piccole schede dal costo inferiore a cinque dollari, in grado di offrire connettività a qualunque microcontrollore dotato di una porta seriale.

## **Data Logger Shield**

Un primo esempio di shield complesso è quello offerto da questa scheda che unisce un orologio alimentato da una pila, che può mantenere data e ora anche se Arduino non è alimentato e un alloggiamento per una scheda di memoria di tipo SD. Collegando la shield potrete registrare i vostri dati su dei file salvati sulla scheda SD. Potrete esaminare il contenuto della scheda rimuovendola o collegandovi da remoto, magari aggiungendo una shield Ethernet, WiFi o BlueTooth.



*Fig 2.18 – Data Logger Shield.*

## **GPS Shield**

Se volete sapere dove siete, potete utilizzare il sistema satellitare GPS per localizzare con precisione i movimenti e la posizione del vostro Arduino. Esistono diversi moduli GPS, che comunicano i dati tramite una porta seriale. Le coordinate sono inviate in formato testuale. Il GPS richiede parecchia energia per funzionare, quindi prevedete un sistema di alimentazione o delle batterie di buona capacità.

## **GSM Shield**

Con una SIM GSM potete accedere a vari servizi di comunicazione, per lo scambio di dati o SMS. La GSM Shield si può usare anche per realizzare dei semplici sistemi telefonici.

## **LCD Shield**

Le più comuni Shield LCD hanno un display a cristalli liquidi retroilluminato con due righe da una

ventina di caratteri. Potete scrivere qualsiasi testo spostando un cursore e inviando sequenze di caratteri. Il display di solito è accompagnato da una serie di pulsanti per interagire con il sistema. Questo tipo di shield di solito sono in cima al panino. Per risparmiare pin digitali, i pulsanti possono essere collegati ai pin analogici, dove si leggono interpretando le tensioni analogiche prodotte.

## **e-Ink Shield**

Un'alternativa ai display a cristalli liquidi sono i display a inchiostro elettronico, che si «impressionano» con testi e immagini che mantengono nel tempo. Sono display molto utili per creare piccoli segnali o cartelli semi permanenti. Le dimensioni dei display per Arduino sono di solito abbastanza ridotte.

## **Motor Shield**

Non è possibile collegare direttamente un motore elettrico ai pin di Arduino. Il motore richiede troppa corrente e il microcontrollore non è in grado di fornirla senza danneggiarsi. Per gestire dei motori è necessario utilizzare dei sistemi di pilotaggio. Il sistema più semplice è un semplice transistor (bipolare o mosfet). Si possono usare soluzioni più elaborate che possono controllare direzione e velocità del motore. Ci sono schede specializzate per motori in corrente continua e per motori passo-passo. Queste schede semplificano le istruzioni necessarie per gestire i dispositivi collegati e si comportano un po' come il pilota automatico di un jet.

## **PWM shield**

Quando sei segnali PWM non sono sufficienti, potete utilizzare una scheda che può gestire decine PWM contemporaneamente. A bordo di queste schede troverete dei chip specializzati nella gestione dei segnali PWM. L'impostazione dei segnali si fa utilizzando una linea seriale e anche in questo caso, il microcontrollore non deve occuparsi della gestione continua dei segnali.

## **ZigBee Shield**

Il sistema ZigBee è una variante del bluetooth a bassa energia (LE o BT 4.0) ed è specializzato per la domotica e per le comunicazioni distribuite. Usando molte schedine ZigBee si possono creare delle reti di sensori distribuite. Le schede comunicano con una porta seriale e necessitano di un adattatore di voltaggio perché sono alimentate a 3,3 volt.

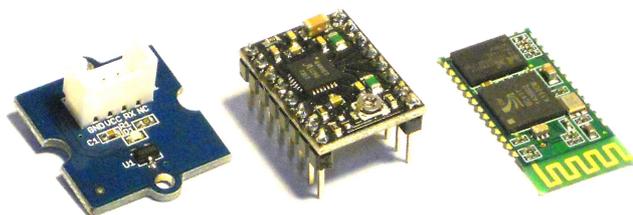
## **MIDI Shield**

La MIDI Shield ha due porte MIDI che si possono pilotare con Arduino, per creare eventi sonori legati a sensori, legando suoni e fenomeni fisici. Le nuove frontiere della musica! Se solo ne avesse avuta una tra le mani Giorgio Moroder!

## **MP3 Shield**

È possibile riprodurre dei file mp3 con un semplice microcontrollore: in Rete si trovano vari esperimenti. La decodifica dei dati di un mp3 non è una cosa semplice, per questo sono stati creati dei chip specializzati per la loro lettura. Una shield mp3 ha un alloggiamento per una scheda SD e un'uscita audio stereo con jack da 3,5 millimetri. Collegate Arduino, un display e avete realizzato il vostro lettore mp3 personalizzato.

Esistono anche delle soluzioni più minute: delle piccole schede su cui trovano posto solo un piccolo chip o un singolo sensore. Queste schede costano meno ma dovrete cablarle al vostro Arduino utilizzando fili elettrici jumpers. Se non volete avere problemi di cablaggio e preferite una soluzione Plug'n'Play, optate per una shield.



*Fig 2.19 – Piccole schede per microcontrollori: sensore di temperatura, driver per motori e modulo Bluetooth.*

Come avete visto, le shield sono delle soluzioni complete e pronte all'uso, ma abbastanza rigide. Se non trovate quello che vi serve, non riuscirete a realizzare il vostro progetto. Spesso potreste scontrarvi con delle incompatibilità tra le schede. Se due shield utilizzano gli stessi pin, dovrete eliminare una delle due... Per chi volesse una via di mezzo e desiderasse una certa flessibilità, sono state create delle soluzioni modulari come TinkerKit e Grove.

## **Tinkerkit**

Se volete sviluppare prototipi interattivi con facilità, senza dover utilizzare né saldatore, né breadboard, potete ricorrere a Tinkerkit, il sistema modulare per realizzare circuiti elettronici. Tinkerkit utilizza una scheda madre, una shield che si collega ad Arduino, dei cavetti di diverse dimensioni e tante schedine, ognuna specializzata per fornire una specifica funzione. La scheda madre è ricoperta da numerosi piccoli connettori standard a tre contatti. Per costruire un circuito dovrete collegare le piccole schede alla shield principale. Le schedine disponibili sono numerose: ci sono sensori e attuatori di ogni tipo. Nell'assortimento trovate pulsanti, led, slider, potenziometri, relè, display, sensori per la luce, per la temperatura, rivelatori di gas... tutto quello che può servire per realizzare un prototipo: non dovete fare altro che collegare tutto quello che vi serve utilizzando i cavi della lunghezza opportuna.

I connettori presenti sulla scheda madre sono contrassegnati con delle sigle, così che sia semplice riconoscerli. Ci sono sei ingressi analogici contrassegnati dalle scritte da I0 a I5 e sei pin digitali collegate ai connettori che vanno da O0 a O5. Le uscite «O» sono collegate ai pin PWM di Arduino.

## **Grove**

Simile a TinkerKit è Grove, sviluppato e distribuito da SeeedStudio (<http://www.seeedstudio.com/wiki/Category:Grove>), un sistema formato da numerosi piccoli blocchetti di espansione che si possono collegare a una shield per Arduino. Ogni elemento del sistema offre una sola funzione, come, per esempio, un pulsante o un sensore di movimento. Ogni parte è corredata da esempi e chiare spiegazioni sul suo funzionamento. Grove ha sviluppato due kit chiamati Mixer Pack<sup>4</sup> e Mixer Pack 2 che non richiedono un microcontrollore per funzionare e i cui elementi possono essere collegati direttamente tra di loro per realizzare semplici circuiti.

---

<sup>4</sup> L'indirizzo internet del Mixer Pack è: [http://www.seeedstudio.com/wiki/Grove\\_-\\_Mixer\\_Pack](http://www.seeedstudio.com/wiki/Grove_-_Mixer_Pack)

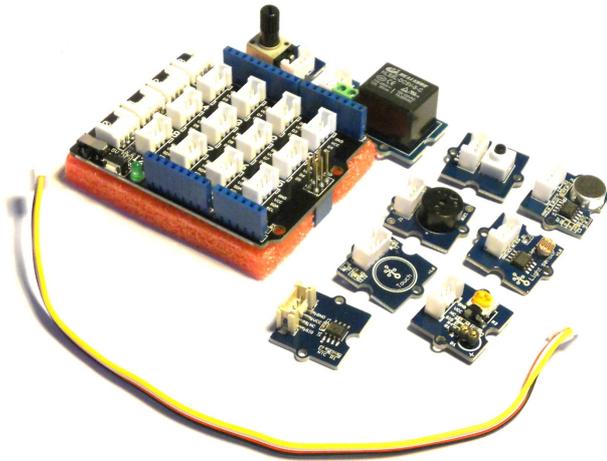


Fig 2.20 – Il kit Groove.

## Glossario

**Ambiente Integrato di Sviluppo o Integrated Development Environment (IDE):** è un programma, di solito grafico, per scrivere software in modo efficiente. L'editor per la scrittura del codice è specializzato per la scrittura del software e aiuta il programmatore con molte funzioni e strumenti, per assisterlo in ogni fase del lavoro. Di solito gli IDE offrono anche un debugger con cui analizzare l'esecuzione di un programma: molto utili per scovare errori e problemi. Esistono debugger in grado di collegarsi ai microcontrollori e seguire da remoto lo svolgimento di un firmware.

**Bootloader:** è un piccolo programma che può essere eseguito in particolari condizioni. Il bootloader di Arduino si risveglia ogni volta che arrivano dei dati dalla porta USB, e se riconosce l'invio di un nuovo programma, lo scrive nella flash del microcontrollore. Il bootloader può scrivere direttamente nella flash e quindi programmare il microcontrollore. Grazie al bootloader non è necessario utilizzare uno speciale programmatore.

**Convertitore Analogico Digitale (ADC):** è un circuito che può anche trovarsi all'interno di un microcontrollore e serve per trasformare un segnale analogico in uno, cioè in un numero o in una sequenza di bit.

**Convertitore Digitale Analogico (DAC):** è l'opposto di un ADC. Converte un numero binario in un valore analogico incluso in un certo intervallo di tensioni. Nei passaggi da digitale ad analogico (e viceversa) si perde sempre, inevitabilmente, una certa parte di segnale. Se si adottano delle precauzioni è possibile ridurre al minimo questi errori di conversione.

**Firmware:** è un programma caricato direttamente in un chip. Non è propriamente un software perché non è facilmente modificabile dagli utenti.

**FTDI (Future Technology Devices International):** è una società scozzese specializzata nella produzione di circuiti integrati per le tecnologie USB. È famoso il suo convertitore da USB a

seriale.

**Ingresso Analogico:** alcuni pin del microcontrollore sono collegati a dei convertitori analogico digitali e quindi possono fornire al software una lettura del segnale presente sul pin. Sulle schede Arduino questi pin sono di solito distinti dagli altri e contrassegnati da una lettera «A» seguita da un numero. I pin analogici di solito si possono configurare anche per essere usati come pin digitali.

**Jumper:** è un piccolo cavo dotato, alle estremità, di due piccole baionette che possono essere facilmente infilate negli «header» di Arduino.

**Memoria EEPROM o Electrically Erasable Programmable Read-Only Memory:** è una memoria che può mantenere piccole quantità di informazioni per lungo tempo, anche quando non è alimentata. Si può riscrivere utilizzando dei segnali elettrici.

**Memoria Flash:** nei microcontrollori la memoria flash è utilizzata un po' come il disco rigido di un classico computer. È una memoria a stato solido, particolarmente veloce, che può mantenere le informazioni per lungo tempo. Nei microcontrollori è dedicata alla scrittura del programma che sarà eseguito. Una parte della memoria flash può essere riservata per il bootloader.

**Memoria RAM o Random Access Memory:** è la memoria volatile, utilizzata dalla CPU del microcontrollore per eseguire l'operazione corrente del programma e per immagazzinare valori temporanei e variabili. Perde il suo contenuto quando il chip non è più alimentato.

**Microcontrollore,  $\mu$ C, MCU (Micro Controller Unit):** un microcontrollore è un piccolo computer racchiuso in un singolo chip. È un sistema autonomo, formato da memoria RAM, memoria Flash, EEPROM, un circuito di sincronizzazione, convertitori analogico-digitali e digitali-analogici, timer, porte di comunicazione di vario tipo. Le schede Arduino sono sostanzialmente dei supporti «pratici» che forniscono l'alimentazione e le comunicazioni di base per utilizzare i microcontrollori ATMEL.

**Pin:** è il nome inglese del «piedino» del chip.

**Pin digitali di I/O (GPIO – General Purpose I/O):** ogni microcontrollore comunica con il mondo esterno con dei «piedini» (pin) che possono comportarsi da ingressi o da uscite. Il comportamento dei pin si definisce nel software che si carica nel microcontrollore, così che la scheda possa avere un comportamento dinamico e flessibile. Alcuni pin hanno delle funzioni speciali, per esempio possono creare segnali PWM, esporre una porta seriale o svolgere altre specifiche funzioni.

**Pulse Width Modulation (PWM):** alcuni pin sono in grado di generare un segnale a onda quadra dove il rapporto tra i tempi “acceso” e “spento” può variare a piacere da zero al cento per cento. Quando il valore è zero il segnale è completamente nullo, quando il valore è pari a cento, il segnale è al massimo valore possibile. Se il segnale è al cinquanta per cento il tempo «acceso» equivale al tempo «spento». Il segnale PWM è adatto per pilotare motori elettrici o per controllare l'intensità di un LED in modo preciso.

**Serial Peripheral Interface o SPI:** è una porta seriale molto diffusa, utilizzata da un chip per comunicare con altri chip. È stata introdotta dalla Motorola e prevede un dispositivo master e tanti dispositivi client. La comunicazione può avvenire in due direzioni.

**Shield:** le schede di espansione di Arduino sono chiamate in questo modo. Presentano una piedinatura compatibile con i pin di Arduino e s'infilano una sopra all'altra. Ogni shield offre una funzionalità completa. Altre piattaforme, diverse da Arduino, utilizzano altri nomi, come per esempio i «capes» delle schede BeagleBone.

**Sincronia o Clock:** è un segnale digitale, formato da impulsi o da un'onda quadra con frequenza molto regolare. Questo segnale è utilizzato nei circuiti digitali per coordinare tutte le operazioni. Per i chip ATMEL utilizzati sulle schede Arduino è tipicamente pari a 16 MHz, anche se su alcune schede può essere pari a 8 MHz. Le schede che impiegano chip più evoluti, con tecnologia ARM arrivano a 40 MHz, mentre nelle soluzioni con SoC o sulle schede Intel si raggiungono i GHz..

**Universal Asynchronous Receiver-Transmitter (UART):** è una porta seriale che nella sua forma più semplice è formata da due semplici fili, uno per trasmettere e uno per ricevere dati. Alcuni microcontrollori possono avere più di una porta seriale.

# 3

## Installazione

Arduino non è solo una scheda che potete programmare a vostro piacimento, ma un sistema formato da:

- un circuito stampato con microcontrollore,
- un software per programmarlo,
- una vasta comunità on-line con cui interagire per ricevere e fornire supporto.

Il punto di partenza di è il sito internet [www.arduino.cc](http://www.arduino.cc) dove troverete molte informazioni utili per iniziare.

La prima cosa da fare dopo aver spaccettato il vostro Arduino è visitare il sito e scaricare il software necessario per far funzionare la scheda. Il programma per utilizzare Arduino con il vostro computer si chiama «Arduino IDE», o più brevemente «Arduino». Un IDE è un ambiente di sviluppo integrato (Integrated Development Environment), un nome altisonante per definire un editor avanzato arricchito da molti strumenti utili per scrivere del software, per testarlo e per pubblicarlo o impacchettarlo (deploy). Alcuni IDE «famosi», usati da molti programmatori sono Eclipse, NetBeans, Xcode o VisualStudio. Sono programmi complessi e specializzati per scrivere codice Java, C, php... Un IDE gestisce un compilatore, cioè il programma che riceve un listato e lo traduce nei bit comprensibili a microprocessori e microcontrollori.

### Java e Processing

L'IDE di Arduino è scritto in Java ed è basato su **Processing** ([www.processing.org](http://www.processing.org)). Java è un linguaggio di programmazione inventato da James Gosling nel 1992. Java nacque come linguaggio di programmazione destinato a essere utilizzato in piccoli dispositivi ed elettrodomestici, ma ben presto venne apprezzato per la sua versatilità e soprattutto per la possibilità di essere eseguito su ogni tipo di computer, grazie a un interprete chiamato «virtual machine». S'iniziò a utilizzarlo per realizzare siti e applicazioni per Internet. Oggi è considerato un linguaggio «storico» e ampiamente adottato da vaste schiere di programmatori.

Processing è un ambiente di sviluppo nato con scopi didattici. Si programma con un linguaggio molto simile a Java, arricchito di alcune funzioni ad alto livello per produrre grafica in due e tre dimensioni. Il software è Open Source con licenza GNU e si può utilizzare su Mac OSX, Windows e Linux. I programmi scritti in Processing possono dialogare con Arduino e si possono combinare le due piattaforme, per esempio creando un controller hardware che interagisce con un videogioco in esecuzione sul vostro computer..

Processing è stato utilizzato come punto di partenza per creare l'ambiente di programmazione per Arduino infatti, la finestra principale dei due programmi è molto simile. In Arduino non troverete il termine «listato», ma «sketch», che è stato ereditato da Processing. Dallo sketch di Processing quello di Arduino eredita anche la struttura caratterizzata da due blocchi ben distinti, chiamati **setup** e **loop**. Mentre in Processing non è obbligatorio utilizzare dei blocchi in cui inserire le istruzioni, negli sketch di Arduino dovrete sempre includere le due sezioni:

- **setup** in cui inserire tutte le istruzioni che devono essere eseguite una sola volta

- all'accensione di Arduino;
- **loop** con le istruzioni che saranno ripetute fino a che Arduino è alimentato.

*Lo sketch di Arduino ha sempre i blocchi loop e setup.*

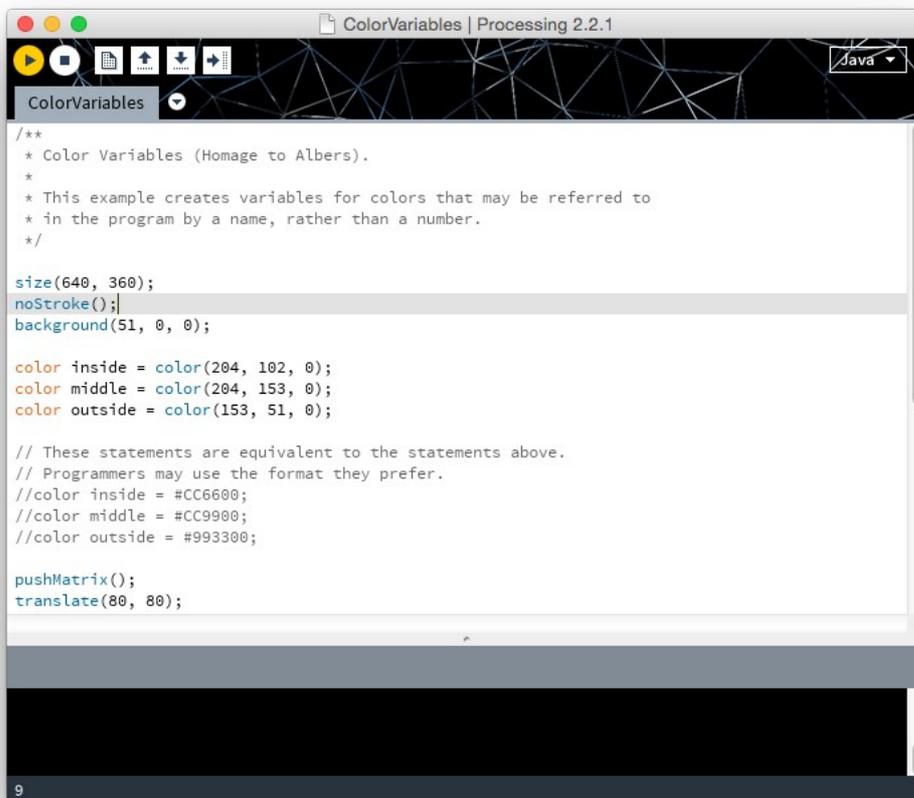


Fig 3.1 – La finestra principale di Processing.

## Download e installazione

Visitate l'indirizzo <http://www.arduino.cc/en/Main/Software> e scaricate l'ultima versione del software. Nel momento in cui scrivo la versione pubblicata è la 1.6.4 ed è disponibile per Windows, Mac OSX e Linux. Scegliete la versione compatibile con il vostro sistema e scaricatela in una cartella.

### Windows

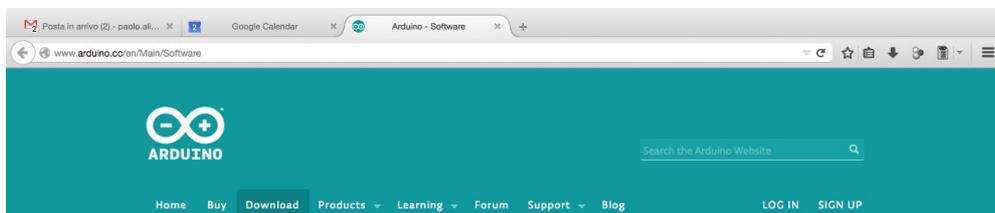
La procedura di installazione di Arduino è in genere abbastanza semplice. Se utilizzate Windows potete far riferimento alle istruzioni pubblicate sul sito di Arduino (<http://www.arduino.cc/en/Guide/Windows>).

Per iniziare procuratevi una scheda Arduino e un cavo per collegarla al vostro computer. Il cavo dipende dalla versione di Arduino che avete acquistato. Per la scheda più diffusa, Arduino UNO, vi servirà un cavo USB tipo A-B, di quelli che si utilizzano per le stampanti (fig. 3.3). Il connettore USB maschio tipo B è di forma rettangolare, mentre il connettore maschio di tipo A ha una forma quadrata, con due spigoli smussati.

Se avete un Arduino MEGA, un DUE o un Leonardo, dovrete invece utilizzare un cavo micro-USB:

il cavo utilizzato di solito per caricare i cellulari Android.

Se la scheda è nuova infilate il cavo e fate attenzione che il connettore entri bene nell'alloggiamento. Quando utilizzate i cavi micro-USB, fate molta attenzione perché, anche se la presa micro-USB è saldata sul circuito stampato, è molto facile spezzarla con una piccola flessione.



## Download the Arduino Software

**ARDUINO 1.6.4**  
The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software. This software can be used with any Arduino board. Refer to the [Getting Started](#) page for installation instructions.

**Windows Installer**  
**Windows ZIP file for non admin install**

**Mac OS X 10.7 Lion or newer**

**Linux 32 bits**  
**Linux 64 bits**

[Release Notes](#)  
[Source Code](#)  
[Checksums](#)

ARDUINO SOFTWARE  
**HOURLY BUILDS**

**LAST UPDATE**  
1 June 2015 15:32:15 GMT

ARDUINO 1.0.6 / 1.5.x / 1.6.x  
**PREVIOUS RELEASES**

Fig 3.2 – Download del software per Arduino.

Collegate Arduino al vostro computer: la scheda dovrebbe accendersi. Tutte le schede Arduino sono dotate di almeno un led che inizierà a lampeggiare o comunque a indicare qualche tipo di attività. Probabilmente Windows rileverà<sup>1</sup> la presenza di un nuovo hardware e segnalerà la ricerca o l'installazione di nuovi driver.



Fig 3.3 – Cavo USB di tipo A-B, adatto per Arduino UNO.

La descrizione ufficiale della procedura d'installazione, riportata sul sito di Arduino dice:

- Collegate Arduino al computer e attendete che Windows inizi il processo di installazione dei

<sup>1</sup> Il condizionale è d'obbligo vista l'eterogenea varietà di configurazioni hardware e versioni di Windows.

driver. L'installazione fallirà.

- Aprite il Pannello di controllo, quindi «Sistema e sicurezza» e infine su «Sistema».
- Nella finestra Sistema, aprite «gestione dispositivi».
- Nell'elenco dell'hardware cercate la voce «porte seriali e parallele» (COM e LPT). Tra le voci dovrete trovare una porta «Arduino UNO (COMxx)». Se non trovate la periferica, allora controllate in «Altri dispositivi» o «Dispositivi sconosciuti»;
- Cliccate con il tasto destro del mouse su «Arduino UNO (COMxx)» e scegliete «Aggiorna il software del driver»;
- Scegliete «Sfogliate le cartelle per cercare il driver»;
- Navigate tra le cartelle dove avete scaricato e spaccettato il software di Arduino e localizzate il file arduino.inf che troverete nella cartella «Driver» (non nella cartella FTDI USB Drivers«!). Nelle vecchie versioni di Arduino IDE (fino alla 1.0.3) troverete un file chiamato «Arduino UNO.inf»;
- Windows riconoscerà il driver e l'installazione si concluderà.

Ora potete avviare Arduino cliccando sull'icona apparsa sul Desktop o nel menu avvio.

## Mac OSX

Per OSX la procedura d'installazione è ancora più semplice. Potete far riferimento alle istruzioni pubblicate sul sito di Arduino (<http://www.arduino.cc/en/Guide/MacOSX>).

Dovreste aver scaricato un file compresso (.zip) sul vostro computer. Solitamente il file finisce nella cartella «Download» o sul «Desktop». Scompattate il file con un doppio clic. Vedrete apparire un file Arduino.app: trascinatelo nella cartella «Applicazioni».

Con le nuove schede in genere non avrete bisogno di fare altro, non dovrete installare nessun tipo di driver. I driver FTDI vi serviranno se per caso vorrete utilizzare una scheda «duemilanove». Trovate i driver sul sito FTDI (<http://www.ftdichip.com/Drivers/VCP.htm>).

Avviate Arduino con un doppio clic sull'icona che trovate nella cartella Applicazioni.

## GNU/Linux

Dalla versione dieci di Ubuntu, troverete Arduino nei repository di sistema («universe»). Utilizzate synaptic per individuare il pacchetto e installarlo. In alternativa aprite un terminale e digitate:

```
sudo apt-get update && sudo apt-get install arduino arduino-core
```

Dovrete inserire la vostra password perché state utilizzando il comando «sudo». Al termine dell'installazione troverete Arduino tra i programmi installati.

Se il software non dovesse avviarsi, potrebbe essere necessario installare Java. Installate il pacchetto open-jdk utilizzando synaptic o da terminale.

Verificate che il vostro utente sia nel gruppo «dialout», così che abbia i permessi per utilizzare le porte seriali. Se così non fosse (o in caso di dubbio) aggiungete il gruppo con:

```
sudo usermod -aG dialout paolo
```

Sostituite «paolo» con il nome del vostro utente. In alternativa potete lanciare Arduino da terminale usando sudo:

```
sudo ./arduino
```

## Hello Led!

Ora caricherete il vostro primo programma su Arduino. Non preoccupatevi se ora non vi darò molti dettagli. Nelle prossime pagine vi spiegherò tutto. Questo semplice esercizio pratico vi serve per capire se avete installato correttamente il software e se questo comunica con la scheda.

Caricheremo un listato che fa lampeggiare un LED<sup>2</sup>. Non dovrete neppure inserire un LED su Arduino, perché utilizzerete quello già a bordo della scheda, collegato al pin numero tredici.

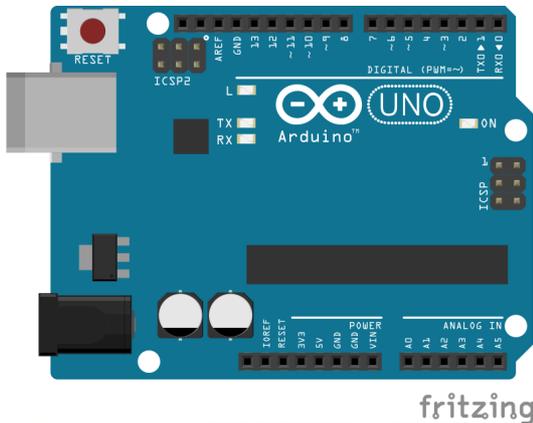


Fig 3.4 – Per il primo esperimento non vi serve nient'altro che Arduino: utilizzate il LED integrato sulla scheda (indicato con «L»).

Lo **sketch** che fa lampeggiare il LED è il più semplice programma che potete caricare su Arduino che faccia qualcosa di visibile, senza che voi dobbiate aggiungere delle parti elettroniche. Il titolo di questa sezione richiama l'«Hello World» dei programmatori software. Quando s'impara un nuovo linguaggio di programmazione si è soliti scrivere un breve programma che stampa a video il messaggio «Hello World!». In molti libri di informatica trovate nei primi capitoli un rapido esempio per realizzare un programma di questo tipo. In questo modo toccate subito con mano l'argomento e verificate che tutto funzioni correttamente. Con Arduino sarebbe difficile stampare un testo su un monitor... la cosa più semplice che potete fare e che possa assomigliare a un saluto è quella di far lampeggiare amichevolmente un LED. Vediamo come fare.

Non dovrete scrivere un programma, perché, per questa volta utilizzerete un esempio già pronto che eventualmente potrete modificare. Ecco cosa dovete fare:

- Avviate l'ambiente di programmazione di Arduino;
- Aprite il file d'esempio Blink che trovate sotto: File > Example > Basic > Blink

---

<sup>2</sup> Un LED è un componente elettronico che può emettere luce come se fosse una lampadina, ma sfruttando il principio fotoelettrico al posto dell'incandescenza di un filamento. La corrente all'interno di un LED circola in un solo senso. LED è una sigla che sta per Diodo Emittitore di Luce. Per maggiori dettagli si veda «Elettronica per Maker» - LSWR – P.Aliverti

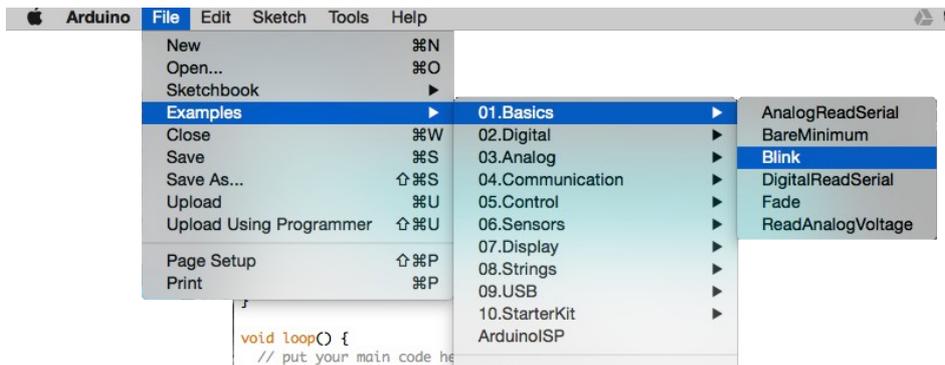


Fig 3.5 – Lo sketch per far lampeggiare il LED si trova negli esempi del menu File.

- Si aprirà una nuova finestra con un nuovo sketch. Come già detto, per ora non preoccupatevi troppo di quello che vedrete scritto.
- Collegare Arduino al vostro computer.
- Verificate che sia selezionata la porta corretta controllando in Tools > Port. Su Windows dovreste trovare una voce tipo COMx (Arduino UNO); su Mac avrete qualcosa tipo /dev/tty.usbmodemXXX, mentre su GNU/Linux troverete /dev/ttyACM0 o /dev/ttyS0 o S1.
- Verificate che anche l'hardware impostato in Tools > Board corrisponda alla vostra scheda (per esempio Arduino UNO).

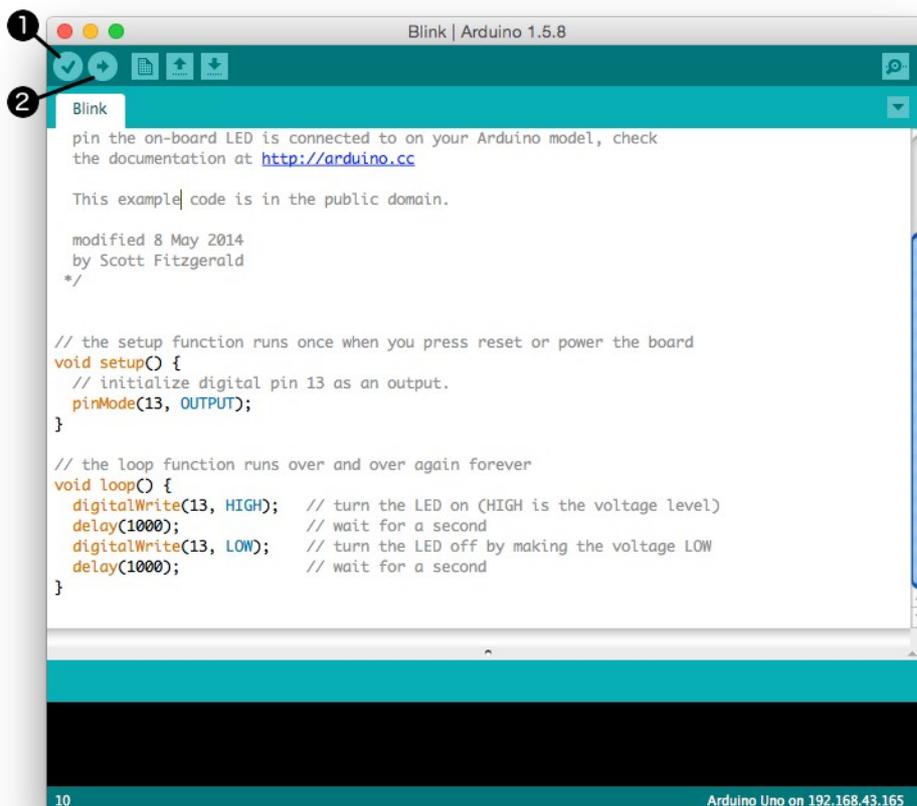


Fig 3.6 – Finestra di Arduino con caricato lo sketch Blink. Nella toolbar trovate i pulsanti «Verify» per controllare il listato (1) e «Upload» per caricarlo sulla scheda (2).

- Il computer frullerà per qualche secondo mentre controlla il vostro listato. Se tutto è andato bene e non ci sono errori, nella parte inferiore della finestra, nella console, dovrete vedere solo delle scritte di colore bianco.
- Ora premete pulsante «Verify» (il primo che trovate sulla toolbar).
- Premete «Upload», il secondo pulsante della toolbar, per trasferire il programma alla scheda.
- Se non ci sono stati problemi di comunicazione (di solito dovuti alla scelta della porta seriale sbagliata), dopo alcuni istanti il LED numero tredici di Arduino dovrebbe lampeggiare a ritmo di un secondo.

Visto che Arduino esce di fabbrica già programmato con lo sketch blink, non vedrete apprezzabili differenze. Ora però che sapete come caricare uno sketch, provate a editare il listato e a sostituire i numeri che trovate di fianco al comando delay. Sostituite «1000» con «100» e ricaricate lo sketch. Non c'è bisogno che scollegiate di nuovo Arduino: modificate lo sketch e che premete di nuovo «Verify» e poi «Upload».

Riporto qui di seguito, in grassetto, la parte di loop che dovrete modificare:

```
void loop() {
  digitalWrite(13, HIGH);
  delay(100);
  digitalWrite(13, LOW);
  delay(100);
}
```

Ora il LED dovrebbe lampeggiare più velocemente.

Se le cose vanno male:

- Verificate la porta seriale: è quella corretta? È stata installata correttamente?
- Il cavo è inserito bene nella scheda? I LED si accendono?
- Avete scelto la scheda corretta?

Per gli impazienti, ecco una breve spiegazione dello sketch.

Nel listato ci sono dei commenti, cioè delle note, che Arduino ignorerà, ma che vi servono per inserire delle informazioni di servizio nel programma o la descrizione del suo funzionamento. I commenti in Arduino sono colorati in grigio. Se un commento occupa una sola riga, è preceduto da una doppia sbarra //. Un commento che si estende su più righe inizia con /\* e si conclude con \*/. Tutto il testo che è tra i due delimitatori sarà ignorato. In testa all'esempio Blink trovate un commento su più righe:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the Uno and
  Leonardo, it is attached to digital pin 13. If you're unsure what
  pin the on-board LED is connected to on your Arduino model, check
  the documentation at http://arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
*/
```

Poco sotto troviamo un commento su una sola linea:

```
// the setup function runs once when you press reset or power the board
```

Lo sketch contiene due sezioni chiamate **setup** e **loop**. Nella sezione setup ci sono le istruzioni di inizializzazione della scheda, eseguite alla sua accensione; nella parte loop trovate le istruzioni che saranno ripetute all'infinito. Le sezioni sono delimitate da delle parentesi graffe.

```
void setup() {  
  //qui il codice di inizializzazione  
}  
  
void loop() {  
  //qui il codice da ripetere  
}
```

Nella sezione loop trovate una sola istruzione, `pinMode`, che serve per modificare il comportamento del pin numero tredici. Ogni pin può comportarsi come un ingresso o un'uscita: lo decidete voi. Per accendere un LED collegandolo al pin numero tredici è necessario specificare che il piedino si comporterà come un'uscita scrivendo :

```
pinMode(13, OUTPUT);
```

Fate molta attenzione a come scrivete i comandi, in fatti è necessario rispettare le maiuscole e minuscole e non dimenticarsi di mettere un punto e virgola dopo ogni comando. La direttiva `pinMode` serve per modificare il comportamento di un pin indicando il numero del pin e il comportamento che dovrà assumere: `INPUT` o `OUTPUT`. Le istruzioni all'interno di setup sono eseguite solo all'accensione di Arduino.

Nella sezione loop trovate quattro istruzioni che sono ripetute fino a che Arduino è alimentato. La prima istruzione è `digitalWrite`, che serve per far uscire la corrente da un pin o per «Accederlo». `DigitalWrite` richiede due parametri: il primo indica il pin su cui agirà e il secondo che tipo di comportamento dovrà assumere il pin, cioè se sarà acceso o spento. Un pin si accende con `HIGH` e si spegne con `LOW`. Quindi:

```
digitalWrite(13, HIGH);
```

accende il LED collegato al pin tredici e lo mantiene acceso fino a un nuovo ordine. La successiva istruzione è `delay`, che si utilizza per interrompere temporaneamente l'esecuzione del programma: è una pausa. Il tempo è espresso in millisecondi, così che un secondo è pari a mille millisecondi.

```
delay(1000);
```

Quando Arduino incontra `delay` si ferma per il tempo indicato. Nello sketch di esempio avete acceso il LED sul pin tredici, che ora resterà acceso per un secondo. Di seguito troverete l'istruzione per spegnere il LED:

```
digitalWrite(13, LOW);
```

e poi una pausa da un secondo:

```
delay(1000);
```

Dopodiché il ciclo si ripeterà. Riporto qui di seguito il listato completo:

```
void setup() {  
  pinMode(13, OUTPUT);
```

```
}  
  
void loop() {  
  digitalWrite(13, HIGH);  
  delay(1000);  
  digitalWrite(13, LOW);  
  delay(1000);  
}
```

Una volta caricato sulla scheda lo sketch accenderà e spegnerà regolarmente il LED a intervalli di un secondo, facendolo lampeggiare: Hello LED!

Il trasferimento dello sketch e la programmazione del microcontrollore sono gestiti da un piccolo programma residente in una porzione speciale della memoria di Arduino. Questo programma si chiama **bootloader** ed è immagazzinato in una parte della memoria flash del chip (la Boot Program Section). Lo sketch finirà nella Application Program Section della memoria Flash. Il bootloader si risveglia quando arriva un particolare codice dalla porta seriale. Il programma attende il nuovo sketch e come arriva, lo scrive nell'Application Program Section. Il bootloader può riscrivere anche se stesso!

Lo sketch trasferito nella memoria flash resterà anche dopo che Arduino sarà spento e lì lo ritroverete non appena lo riaccenderete e almeno fino a che non trasferirete un nuovo listato (sketch).

# 4

## Arduino e il linguaggio C/C++

Per utilizzare Arduino non serve sapere solo come collegare qualche componente e ricopiare qualche riga di codice. La maggior parte dei manuali presentano i comandi base tralasciando l'aspetto della programmazione. Ho deciso di inserire un capitolo dedicato alla programmazione pura, che potrebbe risultare difficile per qualche lettore, per coprire quest'aspetto importante di Arduino, spesso trascurato. Questo capitolo è solo una introduzione ai concetti essenziali.

Il linguaggio dei microcontrollori è il linguaggio C, un linguaggio definito ad «alto livello», ma non troppo semplice come punto di partenza per chi non si è mai avvicinato al mondo della programmazione. Alcuni microcontrollori utilizzano addirittura dei linguaggi ancora più difficili, come il linguaggio assembler che sono molto «vicini» al linguaggio nativo dei microprocessori. Una versione più moderna del linguaggio C è il C++, che ne condivide la maggior parte degli aspetti. Anche se il team di Banzi ha lavorato duramente per semplificare e facilitare il più possibile la programmazione di Arduino, il linguaggio da utilizzare è il C++ immerso in un ambiente che ne semplifica l'utilizzo.

Il linguaggio C risale agli anni 70, quando fu utilizzato per scrivere il sistema operativo UNIX. Si diffuse grazie al libro «The C Programming Language», scritto da Kernighan e Ritchie nel 1978. Questo linguaggio è molto potente e veloce, ma non molto semplice da imparare per chi è agli inizi. Sono richieste delle conoscenze sul funzionamento dei computer e dei loro circuiti, oltre a richiedere una certa «formalità» nella scrittura dei programmi. Il C non prevede molte istruzioni ma è molto flessibile, tanto che si può utilizzare per scrivere programmi per quasi ogni tipo di dispositivo. Per esempio, anche i moderni telefoni con sistema operativo Android possono essere programmati in C oltre che con il linguaggio Java. Il software di molti dispositivi hardware utilizzati anche in campo industriale, è scritto in questo linguaggio.

Un programma in C è contenuto in un file di testo che ha l'estensione .c; il file deve essere processato da un **compilatore**, che è uno speciale programma che legge l'elenco di istruzioni presenti nel file e le traduce nel linguaggio del processore. La sequenza di istruzioni è anche chiamata «codice sorgente». La traduzione da un codice a un altro è chiamata **compilazione**.

Il linguaggio C è un linguaggio molto preciso, che non vi perdonerà nulla: dovrete seguire le sue regole senza sbagliare, altrimenti otterrete degli errori di compilazione. Alcune regole sono:

- tutte le istruzioni devono terminare con un punto e virgola (;),
- si devono rispettare i caratteri maiuscoli e minuscoli,
- esiste un insieme di parole chiave con cui scrivere i programmi.

Le **parole chiave** sono le basi del linguaggio. Dovete intenderle come dei blocchi base da utilizzare per creare i vostri programmi. Ogni parola ha un preciso significato e delle regole che definiscono come va utilizzata. Le parole chiave (keyword) del C non sono molte:

auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while

Le tecniche di programmazione sono cambiate nel tempo per via delle evoluzioni tecnologiche e dell'aumento delle prestazioni dei computer. I primi programmi per elaboratore erano formati da elenchi di istruzioni da eseguire in serie. Nel tempo le esigenze sono aumentate e questo approccio ha dimostrato le sue debolezze. Uno dei maggiori problemi di un approccio così semplice è che si discosta dalla realtà. Perché? Perché chi programma cerca spesso di replicare meglio possibile gli aspetti del mondo... dalla realtà virtuale fino alle procedure bancarie. Il software cerca di replicare fatti o procedure del mondo reale con delle astrazioni, o modelli, che descrivano meglio possibile i fatti e le cose. È chiaro che questo approccio non è semplice da ottenere con una sequenza lineare di istruzioni. Un'approssimazione della realtà è quella in cui tutte le cose sono oggetti. Ogni oggetto ha delle proprietà e su di esso possiamo svolgere delle operazioni.

Queste riflessioni hanno fatto nascere i linguaggi «a oggetti» che permettono di creare dei programmi con cui è più semplice modellizzare gli aspetti del mondo reale.

La differenza principale del C++ rispetto al C è proprio questa: il C++ si definisce un linguaggio a oggetti.

Potete definire un oggetto «automobile» e assegnargli degli attributi, come: peso, colore, marca, cilindrata, numero di marce. Potete poi trattare con l'auto senza sapere veramente come è fatta la suo interno, cioè che tipo di software c'è al suo interno.

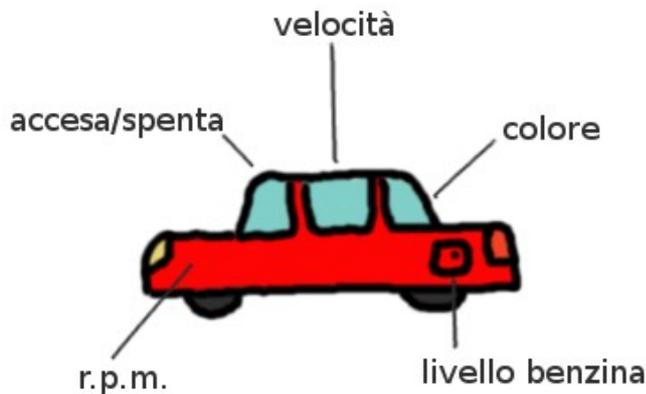


Fig 4.1 – L'oggetto «macchina» è dotato di varie proprietà.

L'oggetto «auto» vi mette a disposizione anche dei «metodi» per interagire con lei: potete accendere il motore, accelerare, frenare, cambiare marcia...

Come già detto, questo tipo di approccio facilita la separazione del codice. Potreste non sapere come è fatto l'oggetto «auto» e non volere neppure saperlo, perché è stato sviluppato da altri.

## Imparare a programmare

Imparare un linguaggio di programmazione non è poi così difficile. Serve un po' di esercizio per apprendere quello che si può fare con il linguaggio e prendere familiarità con la sua espressività, cioè il modo migliore per tradurre il progetto di un programma in una serie di istruzioni. Ci sono linguaggi più o meno semplici. Alcuni richiedono maggiori formalità come il rispetto di minuscole e maiuscole, l'utilizzo di parentesi particolari, di spaziature regolari o di segni di interpunzione. A

volte queste formalità sono comprensibili e altre volte lo sono di meno, ma alla fine ci si abitua e più si utilizza il linguaggio, più le cose sembrano naturali e familiari. Quello che è difficile da apprendere è l'approccio da utilizzare, anche perché è facile trovare libri che insegnino un linguaggio di programmazione, ma è raro trovare libri che si preoccupino di spiegare come programmare. La buona notizia è che una volta appreso «il metodo», potrete applicarlo a qualsiasi tipo di linguaggio.

Un programmatore deve capire il funzionamento di un processo, il comportamento di un fenomeno o il metodo con cui si devono modificare delle informazioni e poi tradurlo in qualcosa di formale, così che un programma di computer possa eseguirlo in modo corretto. Per riuscire bene in questo compito è utile magari sapere qualcosa di come funzionano i computer, ma a volte potrebbe non essere così importante.

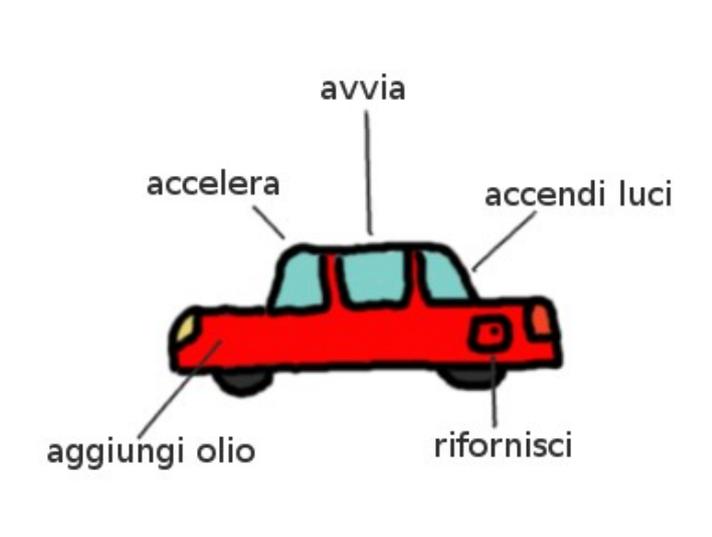


Fig 4.2 – Per interagire con l'oggetto «macchina» si utilizzano dei metodi forniti dal programmatore.

Per cominciare a scrivere un programma dovete prima avere un problema da risolvere. Vi assicuro che se accendete il computer e vi mettete lì davanti allo schermo, aspettando l'ispirazione, la vostra carriera di programmatore finirà rapidamente.

*Trovate un compito da svolgere.*

Potrebbe non essere così semplice, ma è di sicuro una cosa possibile. Per imparare a programmare in Android mi sono inventato un'App per ricordarmi i giorni in cui passa la nettezza urbana a ritirare i rifiuti. Mi sono trovato un semplice problema e ho cercato di risolverlo.

*Trovato l'obiettivo, cercate di comprenderlo.*

Sia che il vostro problema sia semplice, sia che sia complesso, cercate di definirlo e di limitarlo. Se dovete scrivere il programma di controllo per una macchina del caffè, limitatevi a quel compito e rispettate. Fissando dei limiti state definendo lo scopo del vostro lavoro. A volte lo scopo non lo scegliete voi, ma è imposto da un cliente, dal datore di lavoro o delle condizioni fisiche e tecnologiche.

## Come funziona un computer

Il modello di elaboratore (o computer) che tutti conosciamo è quello formato da una CPU (Central Processing Unit), delle memorie e una serie di unità periferiche di ingresso e uscita. La CPU o processore è quella parte del computer che si occupa di eseguire le operazioni che gli vengono sottoposte. La CPU recupera queste informazioni da un programma che di solito è caricato in memoria RAM, cioè la memoria volatile, che perde il contenuto quando il computer si spegne. La CPU utilizza un segnaposto per ricordarsi quale comando sta elaborando: questo segnaposto prende il nome di «program counter» ed è un contatore. Al termine dell'esecuzione di un'operazione, il valore del program counter è incrementato di una unità in modo che punti all'operazione successiva. Le operazioni richiedono che dei dati siano copiati in alcune celle di servizio interne alla CPU. Queste celle si chiamano registri. Se l'operazione produce un risultato, anche questo è salvato in un registro.

La CPU può eseguire operazioni aritmetiche, operazioni per copiare dati da una parte all'altra del computer e operazioni per comunicare con le periferiche, cioè qualsiasi aggeggio elettronico connesso al computer (come mouse, tastiera, scheda video, scheda di rete, stampante...). Le periferiche sono collegate tra di loro con un BUS, cioè un fascio ordinato di fili. Ogni periferica e anche le celle di memoria hanno un indirizzo per poter essere interrogate dalla CPU.

Definito lo scopo, cercate di comprendere bene tutti gli aspetti del vostro progetto. Aiutatevi scrivendo e disegnando. Annotate e ragionate sui vostri appunti, quindi quando è tutto chiaro iniziate a suddividere il lavoro in parti più piccole e poi suddividete ogni parte in sottoparti. Questo approccio si definisce «top-down» o anche «divide et impera» a memoria del modo di procedere in battaglia degli antichi romani, che invece che cercare di conquistare un intero paese, procedevano per passi, acquisendo piccole regioni per volta. L'approccio opposto si chiama «bottom-up» e prevede che si parta da un piccolo step e lo si completi, prima di passare al successivo per poi cercare di combinare il tutto. Una procedura del genere ha senso se avete dei dubbi sulla fattibilità di una cosa, anche se spesso si combinano i due approcci.

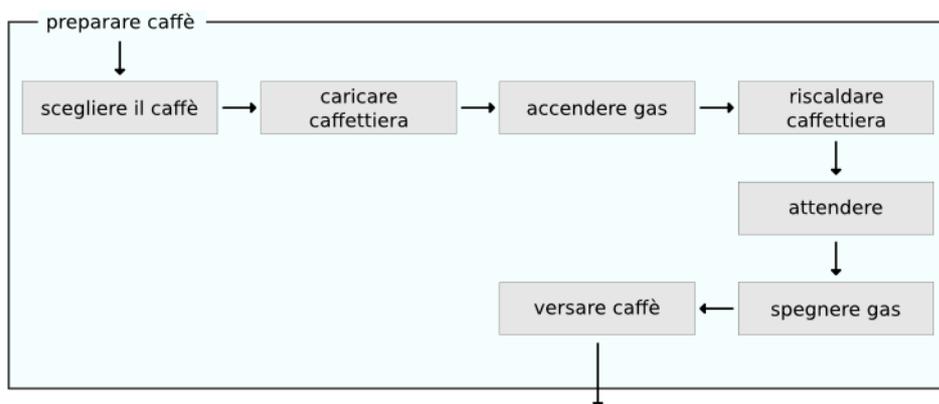


Fig 4.3 – Il dettaglio di attività che compongono il compito «preparare il caffè», che potrebbe essere parte della macro attività «preparare la colazione».

Tempo fa mi hanno chiesto di modificare un e-commerce inserendo un invio di mail automatico con la creazione di documenti in pdf. Avendo dei dubbi sulla fattibilità del lavoro, inizialmente sono partito dal basso, sviluppando subito la parte per generare un semplice documento pdf e per inviarlo per mail, aggiungendo in seguito tutto il resto.

### Suddividete il lavoro.

I blocchi che avete individuato possono essere disegnati utilizzando dei semplici blocchi collegati tra loro da delle linee o delle frecce. Esiste una formalità anche in questo tipo di rappresentazioni, ma non la utilizzerò in questi esempi. Disegnate un blocco per ogni attività e all'interno del blocco aggiungete un titolo o una breve descrizione del suo funzionamento. Potreste stupirvi, ma non ci sono poi molte possibilità per combinare i blocchi: le principali configurazioni possibili (o pattern) sono tre:

- sequenze,
- test,
- cicli.

L'organizzazione più semplice che può avere un lavoro è una sequenza di operazioni da svolgere una dopo l'altra. Per preparare un caffè dovreste:

1. scegliere il caffè,
2. caricare la caffettiera,
3. accendere il gas,
4. porre la caffettiera sul fornello,
5. attendere l'uscita del caffè,
6. versare il caffè nelle tazze.

Il processo di esecuzione di un lavoro, passo dopo passo, è definito «flusso».



Fig 4.4 – Una sequenza di operazioni.

Lavorare in questo modo è abbastanza naturale perché di solito riuscite a compiere «bene» una sola attività per volta. Vi comportate così ogni volta che cucinate seguendo una ricetta passo per passo, eseguendo le operazioni in fila, nell'ordine in cui sono presentate.

In una sequenza di operazioni è importante poter modificare il flusso di esecuzione quando si verificano certe condizioni. Potete disegnare un blocco in cui scrivete il tipo di test da compiere. Il blocco sarà fornito di due o più frecce di uscita, a seconda dei test che eseguirà. Il flusso d'esecuzione uscirà da una sola delle frecce. Se fa troppo caldo, allora si accenderà il ventilatore.

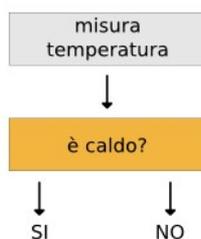


Fig 4.5 – Una semplice blocco per eseguire un test.

Il terzo tipo di configurazione è la ripetizione di gruppi di operazioni (ciclo). È per questo che sono nati i computer! Per indicare un ciclo di operazioni utilizzate una freccia che parte dall'ultimo blocco e punta al primo. Raramente un ciclo si ripete all'infinito, per questo includono un test che interrompe le ripetizioni dopo un certo numero di iterazioni o al verificarsi di una certa condizione.

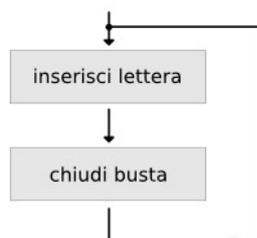


Fig 4.6 – Un semplice ciclo per ripetere due operazioni.

## Il linguaggio C/C++<sup>1</sup>

Per imparare a programmare in C/C++ vi propongo di utilizzare un ambiente di programmazione on-line: [www.ideone.com](http://www.ideone.com) dove potrete provare subito quanto troverete descritto qui. Non dovrete installare nulla sul vostro computer!

Ideone lavora con numerosi linguaggi di programmazione. Prima di iniziare a scrivere del codice nella casella di testo, impostate il linguaggio corretto, cioè il C++, utilizzando il menu a discesa che trovate nell'angolo inferiore sinistro.

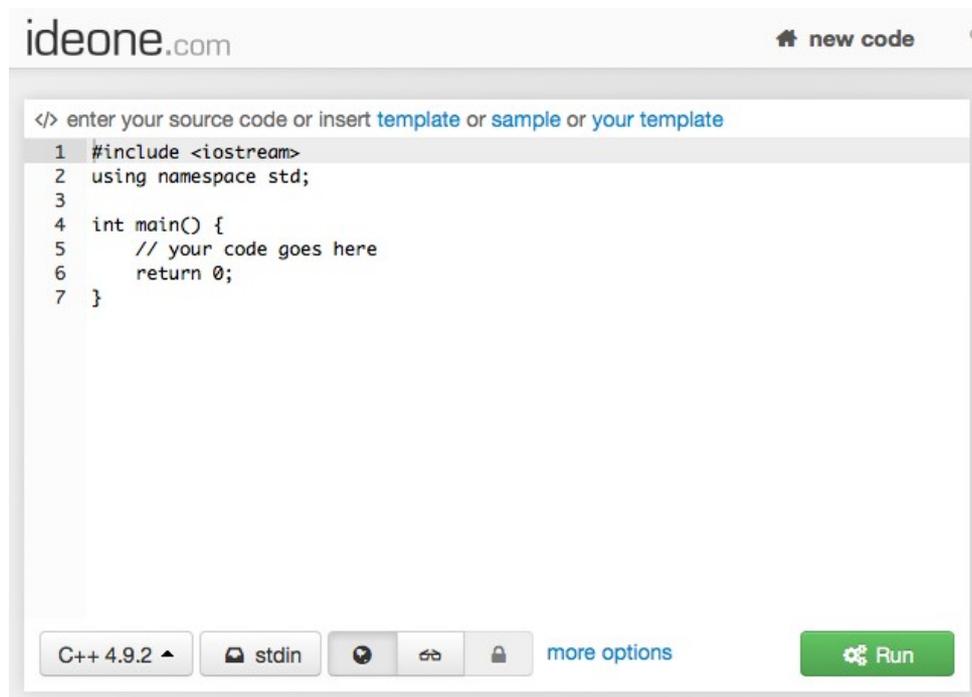


Fig 4.7 – Ideone è un ambiente di programmazione in C/C++ on-line.

1 - Tratterò di C/C++ perché di fatto non userete funzionalità specifiche del C++ e quanto scritto vale sia per il linguaggio C che per il C++. Arduino però lavora in C++. Nelle prossime pagine userò in modo intercambiabile C e C++.

## Il primo programma

Ora scriverete il vostro primo programma in C++: delle istruzioni per stampare a video un messaggio di saluto: «Hello World». Il programma è molto semplice, ma potrebbe già intimorirvi:

```
#include <stdio.h>

int main(void) {
    printf(«Hello C!»);
    return 0;
}
```

Ricopiate il listato nella casella di testo di Ideone. In C/C++ bisogna essere molto precisi nello scrivere i programmi. È necessario rispettare le lettere maiuscole e minuscole. Per un compilatore C/C++ scrivere:

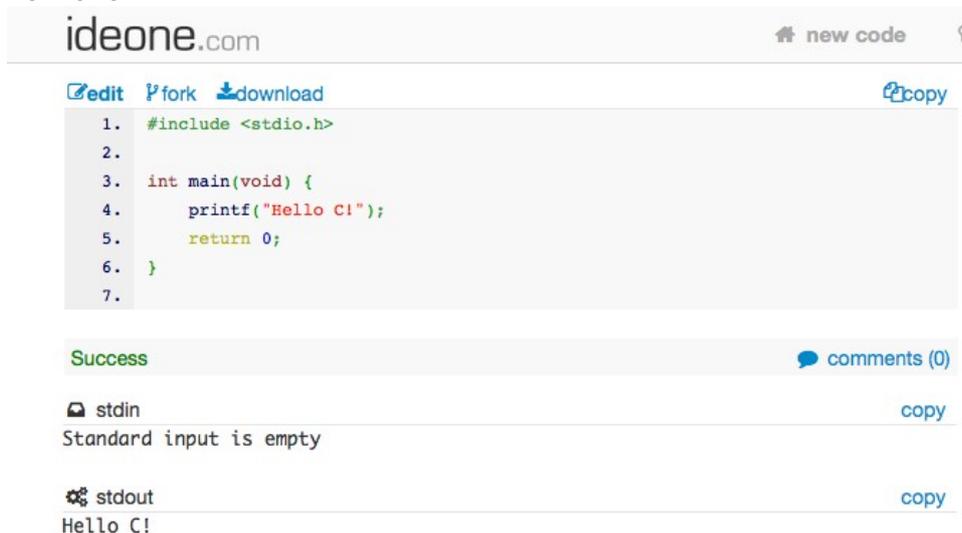
pinmode

è ben diverso da

PINMODE

Ogni istruzione deve essere completata con un punto e virgola, anche se intuitivamente sarebbe inutile. Verificate che tutte le parentesi siano sempre aperte e poi chiuse. Dovrete anche utilizzare delle parentesi graffe: le potete inserire premendo ALTGR + SHIFT + [ e ALTGR + SHIFT + ]. Per compilare ed eseguire il codice premete il pulsante di colore verde con la scritta «Run». Dopo qualche istante, poco sotto al riquadro con il codice, nella sezione «stdout», comparirà il messaggio:

Hello C!



```
ideone.com new code

edit fork download copy

1. #include <stdio.h>
2.
3. int main(void) {
4.     printf(«Hello C!»);
5.     return 0;
6. }
7.

Success comments (0)

stdin copy
Standard input is empty

stdout copy
Hello C!
```

Fig 4.8 – Il risultato prodotto dal vostro primo programma in C++: Hello C!..

La prima riga del listato è chiamata anche «direttiva»:

```
#include <stdio.h>
```

serve per aggiungere al programma una libreria (stdio.h) con delle funzioni aggiuntive per scrivere

a video. Il C/C++ è un linguaggio potente ma scarno... come avete visto è formato da poche parole chiave e per svolgere operazioni più complesse servono delle librerie. Le librerie sono per la maggior parte gratuite e le più comuni sono già incluse nel sistema.

Tutte le istruzioni del programma devono essere scritte entro le due parentesi graffe. La prima parentesi graffa è posta dopo la dicitura:

```
int main(void) {
```

**main** deve sempre essere presente in un programma C/C++ eseguibile. La parola **int** indica che il programma al termine dell'esecuzione restituirà un numero (di cui non ci interessa molto). La parola chiave **void**, racchiusa tra parentesi tonde, indica che il programma non riceverà dei dati in ingresso.

Un programma eseguibile deve obbligatoriamente restituire un numero. Per questo prima dell'ultima parentesi graffa dovete inserire:

```
return 0;
```

Restituirete zero, che per convenzione significa «tutto bene». In realtà potete utilizzare il numero che preferite. Nel caso che si siano verificati degli errori, si è soliti restituire un numero negativo: -1.

Per scrivere un saluto, si deve utilizzare **printf**, che non troverete tra le parole chiave del C/C++ perché è una **funzione**, cioè un frammento di codice che viene ripetuto spesso, a cui per comodità è stato dato un nome di comodo, per evitare di ripetere inutilmente insiem di operazioni. Una funzione riceve dei parametri, esegue un compito specifico e può restituire un risultato. Il parametro che riceve **printf** è la stringa da stampare a video.

```
printf("Hello C!");
```

Ricordatevi di terminare tutti i comandi con il punto e virgola, altrimenti il codice non funzionerà e otterrete degli errori.

Provate a stampare due righe di testo, utilizzando due printf, una di seguito all'altra.

```
#include <stdio.h>

int main(void) {
    printf(«Hello»);
    printf(«World!»);
    return 0;
}
```

Probabilmente il risultato non sarà quello atteso: le parole «Hello» e «World!» sono attaccate, mentre vi sareste immaginati di vederle su due righe. Per mandare a capo un testo si deve utilizzare il carattere speciale **\n** (il carattere è unico, ma dovete inserire la barra seguita dalla «n»). Inserite **\n** subito dopo la parola «Hello». Ora «Hello» e «World!» saranno stampate su due righe distinte.

```
#include <stdio.h>

int main(void) {
```

```
printf(«Hello\n»);
printf(«World!»);
return 0;
}
```

Quando un programma inizia a crescere e diventa complesso è buona cosa inserire delle descrizioni o dei commenti che vi aiuteranno a ricordare come funziona il tutto. Una scuola di programmazione ritiene invece che i commenti siano una perdita di tempo e che un buon codice non richieda l'utilizzo di note e descrizioni aggiuntive per spiegare quello che dovrebbe essere evidente. Probabilmente la verità sta nel mezzo...

Sentitevi liberi di inserire commenti utilizzando una doppia barra inclinata seguita dalle vostre note.

```
// qui posso scrivere un commento su una riga sola
```

Se vi servono più righe perché dovete scrivere molto testo, utilizzate la coppia `/*` e `*/`. Tutto quello che è incluso tra i due segni, sarà ignorato dal compilatore.

```
/*
Ho tante cose da
scrivere e non voglio
che siano lette dal compilatore...
*/
```

## Variabili e tipi numerici

I programmi immagazzinano dati entro delle celle di memoria. Come si fa a inserire un dato in una cella di memoria? Detto così sembra quasi intimidatorio... in realtà la faccenda è molto semplice: si utilizzano delle variabili. Una variabile è come un piccolo cassetto in cui custodire delle informazioni. Sono cassette «tipizzati», cioè possono contenere un tipo ben preciso di dato. Un cassetto (una variabile) ha un nome che deve rispettare delle regole: non può iniziare con dei numeri o dei caratteri speciali (es.: `*`, `+`, `-...`). Di solito si possono usare indifferentemente maiuscole e minuscole.

Nomi di variabili validi:

- temperatura
- i
- parametro123
- Livello\_della\_benzina
- VOLUME

Nomi di variabili non consentiti:

- 12gamma
- livello-del-gas
- \*pippo

Una volta scelto il nome, dovrete definire che tipo di informazione potrà contenere la vostra variabile. Una variabile creata per immagazzinare dei numeri interi potrà contenere solo numeri senza virgola, positivi o negativi : 0, 1, 2, -1...

Per definire una variabile in C/C++ scriverete:

```
int livello_benzina = 10;
```

Prima di tutto dovete indicare il tipo di numero che potrà essere contenuto nella variabile, che in questo caso è **int**, cioè numeri interi. Alla variabile assegno immediatamente il valore dieci, utilizzando il segno di uguale. Volendo è possibile creare una variabile senza pre-assegnare un valore:

```
int livello_acqua;
```

Il compilatore assegnerà un valore di default che potrebbe essere pari a zero, ma è sempre bene non fidarsi e assegnare alle variabili un valore iniziale noto (giusto per non trovare sorprese). Oltre al tipo **int** esistono anche **long** e **byte**. Un byte è un piccolo intero che può contenere valori che vanno da 0 a 255. Il tipo **long** invece contiene numeri molto grandi che vanno da -2.147.483.648 a 2.147.483.647.

È possibile usare numeri con la virgola? Sì, ma nei programmi scritti per Arduino è sconsigliabile perché consumano molta memoria! Per definire un numero con virgola si usa la parola chiave **float** seguita dal nome della variabile ed eventualmente dal valore che si vuole assegnare. Per indicare la virgola ricordatevi di usare il punto (.), come nel sistema anglosassone.

```
float tensione_sensore = 1.12;
```

I numeri float assumono valori da  $3.4028235 \cdot 10^{+38}$  fino a  $-3.4028235 \cdot 10^{+38}$ . Se serve una maggior precisione si possono usare i numeri con virgola di tipo **double**.

## Testi e caratteri

Testi e caratteri si possono memorizzare utilizzando delle variabili. Potete utilizzare una variabile che conterrà un solo **carattere** oppure un testo intero o «stringa». Per carattere s'intende un simbolo solo, come il simbolo<sup>2</sup> utilizzato per la «a» o il simbolo che rappresenta un numero (per es.: «1»). Una **stringa** è invece una sequenza di caratteri. Nel linguaggio C i caratteri s'indicano usando le virgolette singole. Ecco la definizione di una variabile di tipo carattere:

```
char un_carattere = 'a';
```

Al posto del tipo int, questa volta utilizzerete il tipo **char**.

Per stampare un carattere usate la funzione **printf** con il segnaposto %c e la variabile con il carattere da visualizzare. Ecco come fare:

```
#include <iostream>

int main() {
    char il_mio_carattere = 'a';
    printf("Il carattere è: %c", il_mio_carattere);
    return 0;
}
```

La variabile «il\_mio\_carattere» è di tipo char ed è inizializzata con il carattere «a». La funzione printf riceve due parametri: il testo che volete visualizzare, al cui interno c'è il segnaposto %c e la variabile da sostituire al segnaposto. %c serve a printf per trattare nel modo corretto la variabile che

2 - Il «simbolo 1» è ben diverso dal «numero uno». Il simbolo è solo il segno grafico che usiamo per rappresentare la grandezza matematica di valore «uno». Mi rendo conto che questo è un concetto un po' filosofico...

gli state passando. Per stampare dei numeri interi potete utilizzare il segnaposto %d.

Eseguite il programma e otterrete:

Il carattere è: a

Una sequenza di caratteri forma un testo, che i programmatori chiamano anche **stringa**. Il concetto di «stringa», in origine, non era presente nel linguaggio C, che usava (e usa ancora oggi) delle sequenze di caratteri: immaginate un cassetto fatto di tanti scomparti ordinati. Una stringa si definisce così:

```
char testo[] = «Non ho molte parole da dire.»;
```

Subito dopo il nome della stringa dovete inserire una coppia di parentesi quadre; il testo va racchiuso entro una coppia di virgolette doppie. Per stampare una stringa potete quindi scrivere un programma come il seguente:

```
#include <iostream>

int main() {
    char parole[] = "mela pera miele zucchero";
    printf("parole a caso: %s", parole);
    return 0;
}
```

Eseguite il programma per stampare a video il contenuto della stringa. Per stampare una stringa utilizzerete la funzione printf adottando, come segnaposto, %s.

Si può dichiarare una stringa anche indicando la sua lunghezza a priori:

```
char testo[5] = «ciao»;
```

La lunghezza utile deve sempre essere maggiorata di un carattere rispetto a quelli necessari. Questo perché l'ultimo posto deve sempre essere occupato dal carattere speciale «\0» (detto carattere nullo). Quando non indicate esplicitamente la lunghezza di una stringa, ci penserà il compilatore a inserire il carattere nullo in fondo alla sequenza di caratteri. Il carattere nullo serve al computer per capire quando finisce la stringa.

Visto che una stringa è un insieme ordinato di caratteri, potete accedere a ogni singolo carattere indicando la sua posizione. Potete accedere al singolo carattere scrivendo:

```
#include <iostream>

int main() {
    char parole[5] = "ciao";
    printf("il terzo carattere: %c", parole[2]);
    return 0;
}
```

In questo programma è stata creata una stringa, lunga cinque caratteri, di nome «parole». La printf stampa solo il terzo carattere della stringa.

L'indice per accedere ai singoli caratteri di una stringa parte da zero e arriva al massimo valore, meno uno. Nell'esempio precedente l'indice va da zero a tre.

Arduino prevede un tipo **String** che facilita molto le operazioni: non è necessario utilizzare

parentesi o strane funzioni per modificarle. Vedremo in seguito come fare, però vi lascio un'anticipazione su come definire una stringa:

```
String testo = «Così è tutto più facile!»;
```

Per combinare due stringhe in Arduino è sufficiente sommarle:

```
String txt1 = «Hello »;  
String txt2 = «World!»;  
String testo = txt1 + txt2;
```

Su Arduino però non potete stampare a video le stringhe con printf...

## Operazioni

Potete utilizzare le variabili per fare dei calcoli matematici utilizzando gli operatori di somma, sottrazione, moltiplicazione e divisione. Sommate due numeri interi in questo modo<sup>3</sup>:

```
int somma = 10 + 2;  
printf(«Il risultato è: %d», somma);
```

Potete svolgere altre operazioni utilizzando gli operatori di sottrazione (-), moltiplicazione (\*) e divisione (/). Quando si svolgono delle operazioni matematiche bisogna fare molta attenzione ai tipi numerici che si utilizzano. Se utilizzate solo interi (int), il risultato sarà a sua volta intero. Provate a fare la seguente divisione:

```
int res = 10 / 3;
```

Il risultato è pari a tre: vi siete persi tutte le virgole perché i numeri interi non le prevedono. Per avere un risultato più preciso dovete utilizzare il tipo float.

```
float res = 10 / 3;
```

Anche così il risultato non sarà corretto e varrà sempre tre. Perché? Perché state dividendo due numeri senza virgola (10 e 3 sono due interi), che non possono produrre un risultato con virgola. Per avere il risultato preciso dovete far capire al compilatore che tutti i numeri che state utilizzando sono dei float: specificate la virgola e i decimali anche se sono pari a «0».

```
float res = 10.0 / 3.0;  
printf("Il risultato è: %f", res);
```

In questo caso il risultato è pari a 3.3333.

Per forzare un tipo numerico potete fare un **type casting** (una forzatura di tipo) scrivendo davanti al numero o alla variabile il tipo in cui vorreste trasformarlo.

```
float res = (float)10 / (float)3;
```

## Array

A volte una semplice variabile non è sufficiente e vi potreste trovare con la necessità di dover creare

3 - Per brevità negli esempi seguenti, non riporterò il listato intero: completatelo voi aggiungendo l'include, il metodo main, le parentesi graffe e l'istruzione return.

tanti «cassetti» per memorizzare tante informazioni diverse ma molto simili tra di loro nel significato.

Proviamo a vedere un caso semplice ma concreto: calcolare la media di cinque numeri. Per calcolare la media definite cinque variabili differenti, una per ogni numero, poi le sommate e dividete il risultato per cinque. Chiamate le variabili: n1, n2, n3, n4, n5.

```
float n1 = 1.1;
float n2 = 2.3;
float n3 = 3.7;
float n4 = 4.1;
float n5 = 5.8;
float media = (n1 + n2 + n3 + n4 + n5) / 5.0;
```

Certo non è molto pratico dover creare cinque variabili differenti. Un array è come un cassetto dotato di scomparti. Il numero di scomparti è predefinito. Create un cassetto «temperature» in grado di memorizzare tre diversi valori:

```
float temperature[3];
```

Indicherete il numero di posti disponibili (o scomparti) aggiungendo il numero, racchiuso da parentesi quadre, subito dopo il nome della variabile. Potete anche non indicare il numero di posti e lasciare vuote le parentesi quadre, in questo caso, subito dopo l'uguale, elencherete gli elementi dell'array racchiusi da una coppia di parentesi graffe e separati da virgola (,).

```
float temperature[] = {32.1, 30.0, 33.0};
int stati[] = {0, 10, 2, 3, 1, 9};
```

È possibile creare un array vuoto e poi modificare in seguito la sua lunghezza con delle funzioni speciali.

L'esempio per calcolare la media di cinque numeri può essere riscritto così:

```
#include <iostream>

int main() {
    float nn[] = {12.2, 1.2, 324.5, 34.8, 45.9};
    float media = (nn[0]+nn[1]+nn[2]+nn[3]+nn[4])/5.0;
    printf("media = %f", media);
    return 0;
}
```

Per accedere a un argomento dell'array dovete indicare la sua posizione all'interno del «cassetto» utilizzando l'indice che parte da zero. Se l'array ha cinque posizioni, allora l'indice andrà da zero a quattro. Per specificare l'elemento desiderato si usano le parentesi quadre, poste dopo il nome dell'array.

```
float nn[] = {12.2, 1.2, 324.5, 34.8, 45.9};
//per stampare il terzo elemento
printf("terzo elemento = %f", nn[2]);
//per stampare il primo elemento, l'indice è 0
printf("primo elemento = %f", nn[0]);
//per stampare l'ultimo elemento
printf("ultimo elemento = %f", nn[4]);
```

Per operare sugli array esistono una serie di funzioni «specifiche» che non tratterò in queste pagine (vi consiglio di approfondire on-line o con un libro sul C o C++).

## Sarà vero?

Una delle funzioni principali e più importanti di un linguaggio di programmazione è quella di poter «prendere delle decisioni»... che in linguaggio meno fantascientifico significa «fare dei test». È raro che un programma abbia un flusso continuo e regolare dall'inizio alla fine: spesso si devono misurare delle variabili o valutare delle condizioni per poi prendere delle decisioni. Per valutare una condizione si deve eseguire un test che restituisce un risultato che può essere «vero» o «falso». Non sono ammesse situazioni intermedie, perché stiamo parlando di logica «da calcolatori» o «booleana». Esistono molti tipi di logiche, tra cui anche alcune particolari, più simili al nostro modo di ragionare, chiamate logiche Fuzzy, dove non c'è solo vero o falso ma sono ammesse anche situazioni intermedie o sfumate.

In C/C++ potete usare delle variabili di tipo booleano, che ammettono solo true e false come valori, oppure usare delle variabili numeriche intere. Il valore true corrisponde infatti a 1 e false a 0.

```
#include <stdio.h>
```

```
int main(void) {
    bool test = true;
    printf("test = %d \n", test);
    printf("test = %d \n\n", !test);
    return 0;
}
```

Nell'esempio ho usato una variabile chiamata test di tipo booleano (bool) e le ho assegnato il valore true. Ho poi stampato il valore con printf. Il risultato a video sarà: 1. La seconda printf scrive il valore di test, negandolo (cioè invertendolo). Per negare il valore di una variabile booleana antepone il simbolo «!» che significa «NON». La scrittura !test significa quindi «non test» o «l'opposto del valore di test».

Potete fare dei test chiedendo al programma di verificare se una variabile assume un preciso valore: basta usare un **doppio segno di uguale**:

```
(i == 10)
```

Questa scrittura restituisce true solo quando la variabile i vale dieci. Potete utilizzare un test per far compiere un'azione specifica al vostro programma. Se si verifica una certa condizione, allora il programma farà qualcosa di differente. Utilizzate **if** (se) per verificare una condizione. La condizione da valutare deve essere compresa tra due parentesi tonde e l'azione (o il gruppo di operazioni) che sarà eseguita vanno racchiuse tra due parentesi graffe.

```
int i = 10;
if (i == 10) {
    printf(«i vale 10»);
}
```

Se l'istruzione da eseguire al verificarsi della condizione è una sola, potete omettere le parentesi graffe e utilizzare una struttura più compatta:

```
int i = 0;
if (i == 0) printf(«i vale 0»);
```

Ovviamente potete combinare più condizioni, racchiudendole tra parentesi tonde e utilizzando gli

operatori logici «e» (&&) e «o» (||). Per stampare un certo messaggio se la variabile i vale dieci e se j è pari a cinque scriverete:

```
int i = 10;
int j = 5;
if ((i == 10) && (j == 5)) {
    printf(«OK!»);
}
```

*Utilizzate le parentesi tonde per separare le condizioni e per fare ordine*

Per svolgere una certa operazione se una tra più condizioni è vera, fate così:

```
int i = 10;
int j = 5;
int k = 6;
if ((i == 9) || (j == 5) || (k == 2)) {
    printf(«OK!»);
}
```

Il programma stamperà «OK!» se i sarà pari a nove, cinque o due. Ovviamente potete combinare tra di loro più operatori logici.

Un'operazione potrebbe includere un'alternativa: **se** si verifica una certa cosa faccio in un modo, **altrimenti** faccio altro. Questo comportamento si traduce in una struttura **if..then** ed è come se il vostro programma fosse a un bivio: se accade una cosa, vado da una parte, altrimenti dall'altra.

```
int i = 10;
if (i == 5) {
    printf(«i vale 5»);
} then {
    printf(«i non vale 5»);
}
```

Se if..then è comparabile a un bivio, potete creare più alternative con if..else if..then:

```
int i =5;
if (i == 0) {
    printf("A \n");
} else if (i == 5) {
    printf("B \n");
} else {
    printf("C \n");
}
```

Nella programmazione incontrerete diversi tipi di condizioni, per esempio vorrete verificare quando il valore contenuto in una variabile è maggiore di un numero:

```
if (temperatura > 30) {
    //accendi il ventilatore
}
```

Il ventilatore si accenderà quando la temperatura sarà superiore a trenta... quindi a partire da trentuno.

Potete verificare quando una variabile è maggiore o uguale ad un numero ponendo il simbolo di uguale (=) dopo il maggiore (>):

```
if (temperatura >= 30) {  
    //accendi il ventilatore  
}
```

Il ventilatore, in questo caso, si accenderà quando la temperatura sarà uguale o maggiore a trenta. A volte potrebbe capitarvi di avere un lungo elenco di else if. In questi casi potete utilizzare il comando **switch**. Provate a eseguire quest'esempio:

```
#include <iostream>  
  
int main () {  
    int marcia = 1;  
  
    switch(marcia) {  
  
        case 1 :  
            printf("Si parte");  
            break;  
  
        case 2 :  
            printf(«hai messo la seconda»);  
            break;  
  
        case 3 :  
        case 4 :  
        case 5 :  
            printf("Vrooom!");  
            break;  
  
        default :  
            printf("In che marcia siamo?");  
    }  
  
    return 0;  
}
```

**Switch** esegue un test iniziale sulla variabile marcia, quindi fa saltare l'esecuzione del programma nell'apposita sezione. Ogni sezione è delimitata da case..break. Come potete vedere, si possono raggruppare più sezioni, tralasciando «break» (caso con 3, 4 e 5). L'ultima sezione del blocco switch è default, in cui capiterete nel caso di situazioni non previste. Se marcia valesse «-1», finireste nella sezione «default». Switch sembrerebbe del tutto equivalente a un elenco di if..then, ma a differenza delle if, è più veloce perché esegue un solo test, mentre l'elenco di if richiede tanti test quante sono le if presenti.

## Cicli: ripetere operazioni

I programmi per computer sono specializzati nel ripetere operazioni lunghe e noiose. Per ripetere una serie di operazioni non dovrete eseguire più volte un programma, ma potete realizzare dei cicli che ripetono un gruppo di istruzioni per un certo numero di volte oppure fino a che non si verifica una certa condizione. Il tipo più semplice di ripetizione è quella legata al verificarsi di una certa condizione, dove il numero di ripetizioni non è predefinito. Potete realizzare un ciclo di questo tipo utilizzando l'istruzione **while**. Dopo alla parola chiave while, si aggiunge la condizione da valutare, racchiusa tra parentesi e quindi una coppia di parentesi graffe che racchiudono le istruzioni da

ripetere. Fino a che la condizione indicata tra le due parentesi è vera, il ciclo è ripetuto. Ecco un esempio:

```
int i = 1;
while (i <= 5) {
    i = i + 1;
    printf(«i vale: %d», i);
}
```

La variabile `i` è impostata a uno. L'istruzione `while` verifica se `i` è minore o uguale a cinque, e se questa condizione è verificata, il programma «entra» nel ciclo ed esegue il codice presente tra le parentesi graffe che seguono `while`. La prima istruzione incrementa il valore di `i`, che ora varrà due. Il programma stampa un messaggio a video e l'esecuzione ritorna al comando `while`. Ora `i` vale due: la condizione è verificata e il ciclo si può ripetere. Quando `i` varrà cinque, il programma entrerà per l'ultima volta nel blocco-`while` dove il valore di `i` sarà incrementato a sei, stamperà il messaggio «`i vale: 6`» e poi tornerà all'istruzione `while` che verificherà che la condizione non è più verificata e quindi il programma non entrerà più nel blocco `while`, e proseguirà con le istruzioni seguenti.

Per brevità spesso incontrerete l'operazione di incremento della variabile `i` in una forma «contratta» che piace tanto ai programmatori:

```
i++
```

Il ciclo dell'esempio precedente potrebbe essere riscritto così:

```
int i = 1;
while (i <= 5) {
    i++;
    printf(«i vale: %d», i);
}
```

Con un ciclo `while` il numero di ripetizioni non è prefissato. Per questo motivo a volte s'introduce una variabile contatore che memorizza il numero di ripetizioni:

```
int contatore = 0;
bool CONDIZIONE = true;
while (CONDIZIONE) {
    //nel ciclo eseguo delle operazioni
    //...

    //quindi chiamo una funzione che verifica se proseguire o no
    CONDIZIONE = possoProseguire();

    contatore++;
}

printf(«Numero ripetizioni: %d», contatore);
```

A volte è preferibile eseguire un gruppo di istruzioni e poi valutare solo al termine dell'esecuzione se ripeterle oppure no. In questi casi potete usare il ciclo **do-while**:

```
int i = 1;
do {
    i++;
    printf(«i vale: %d», i);
}
```

```
} while (i <= 5);
```

Questa volta il programma entra subito nel corpo del ciclo (il blocco che sarà ripetuto), incrementa la variabile *i* e stampa un messaggio, quindi verifica se eseguire di nuovo il gruppo di istruzioni.

Un altro costrutto molto utilizzato è **for**, con cui potrete ripetere delle operazioni per un numero predefinito di volte. Per usare il ciclo for dovete specificare:

- la variabile da usare come contatore,
- un test per capire quando terminare le ripetizioni,
- un modo per incrementare la variabile contatore.

Queste tre informazioni si «codificano» in C e seguono la parola chiave **for**:

```
for (int i = 1; i < 3; i++) {  
    printf(«i vale: %d», i);  
}
```

Subito dopo «for», trovate una coppia di parentesi tonde in cui, separate da punti e virgola trovate le tre informazioni necessarie a far funzionare il ciclo. Prima di tutto si definisce una variabile *i* che sarà utilizzata come contatore e le si assegna il valore iniziale uno; poi si aggiunge la condizione di fine ciclo: le ripetizioni continueranno fino a che il contatore *i* sarà minore di tre. Infine trovate la modalità di incremento: la variabile *i* sarà incrementata di un'unità per ogni giro. Le istruzioni da ripetere sono racchiuse tra due parentesi graffe.

Il programma raggiunge l'istruzione for e crea una variabile *i*<sup>4</sup> e vi memorizza il valore uno. Quindi rileva la condizione di fine ciclo (terminerà quando *i* sarà maggiore o pari a tre) e la modalità di incremento della variabile (*i++*). Visto che *i* è minore di tre, il blocco di istruzioni racchiuso tra le due parentesi graffe può essere eseguito (corpo del ciclo), dopodiché il programma incrementa la variabile *i*, che ora varrà «due», verifica la condizione ( $2 < 3$ ) e esegue di nuovo il corpo del ciclo. Al passo successivo *i* varrà «tre» e la condizione non sarà più verificata ( $3 < 3$ ), quindi il programma salta alla fine del ciclo for e prosegue.

Potete modificare il «passo» sostituendo *i++* con la modalità di incremento che preferite. Potete avanzare a passi di due unità utilizzando:

```
#include <iostream>  
  
int main() {  
    for (int i = 1; i < 10; i = i + 2) {  
        printf("i vale: %d\n", i);  
    }  
    return 0;  
}
```

Ora la modalità di incremento è più comprensibile: la variabile *i* ad ogni passo sarà calcolata aggiungendo «due» al suo valore.

La variabile «contatore» può anche essere una variabile pre-esistente e non è necessario definirla all'interno di for, ma dovete sempre inizializzarla impostando un valore iniziale. Potete per esempio scrivere:

```
int i = 0;
```

4 - La variabile che il ciclo for utilizza come contatore viene creata solo la prima volta che il programma entra nel ciclo.

```
for (i = 1; i < 10; i = i + 2) {
    //corpo del ciclo for
}
```

Potete modificare le ripetizioni di un ciclo utilizzando i comandi **break** e **continue**. Il primo, **break**, interrompe il ciclo, mentre **continue** interrompe l'esecuzione dell'iterazione corrente. Provate a eseguire il seguente programma:

```
#include <stdio.h>
int main(void) {
    for (int i = 0; i < 10; i++) {
        if (i == 5) break;
        printf(" i: %d \n", i);
    }
    return 0;
}
```

Il ciclo for dovrebbe stampare il valore della variabile *i* per dieci volte. All'interno del corpo del ciclo, c'è un test che verifica se *i* vale cinque e quindi chiama **break**. In questo modo le dieci iterazioni si interromperanno alla quinta e a video potrete leggere:

```
i: 0
i: 1
i: 2
i: 3
i: 4
```

L'istruzione **continue** interrompe solo l'esecuzione dell'iterazione corrente, senza interrompere del tutto l'esecuzione del ciclo. Provate questo esempio:

```
#include <stdio.h>
int main(void) {
    for (int i = 0; i < 10; i++) {
        if (i == 5) continue;
        printf(" i: %d \n", i);
    }
    return 0;
}
```

Quando *i* sarà pari a cinque, il programma salterà all'inizio del ciclo e proseguirà con le rimanenti iterazioni. Il risultato a video sarà:

```
i: 0
i: 1
i: 2
i: 3
i: 4
i: 6
i: 7
i: 8
i: 9
```

Le istruzioni **break** e **continue** funzionano con ogni tipo di ciclo, non solo con il ciclo **for**.

## Funzioni

Un modo intuitivo di programmare è quello di iniziare a scrivere i comandi uno dopo l'altro, in sequenza. Così è come si faceva una volta, per esempio utilizzando linguaggi come il BASIC, in cui dovevate anche inserire i numeri di linea:

```
10 REM programma basic
20 LET A = 10
30 PRINT A
40 GOTO 30
```

Anche nel BASIC erano previsti cicli FOR o WHILE, ma avevate anche la possibilità di eseguire dei salti utilizzando le GOTO. Questa pratica non è molto ben vista dai programmatori più esperti, perché l'uso delle GOTO complica il flusso del programma.

Scrivere programmi come lunghi elenchi di istruzioni si è rivelata una pratica poco efficiente: può funzionare per casi molto semplici, ma non appena le cose si complicano, gestire un programma con questa struttura diventa impossibile. Avrete di sicuro delle parti ripetute più volte, che non potrete gestire con dei semplici «copia e incolla». Immaginate di aver replicato alcune volte un gruppo di righe che si occupano di un'operazione ripetitiva: leggono la data di sistema e la presentano in un modo particolare (giorno/mese/anno).

Dopo qualche tempo vi chiedono di modificare il modo in cui stampate la data: anno/mese/giorno. Dovrete passarvi tutto il listato modificando tutte le sezioni in cui avete composto la data. Per questo motivo hanno introdotto le **funzioni**, con cui potete isolare un gruppo di istruzioni, attribuirgli un nome di comodo e definire cosa gli serve per funzionare (i parametri) e che risultato produrrà.

Per svolgere un'operazione matematica potreste definire una funzione che riceve due numeri come parametri e restituisce il risultato. In C definite una funzione in questo modo:

```
int operazione_matematica(int a, int b) {
    int c = a * 10 + b;
    return c;
}
```

Per creare una funzione dovete:

- indicare il tipo di dato che la funzione restituirà - in questo caso trovate **int**;
- battezzare la funzione con un nome «univoco» e significativo – il nome assegnato è operazione\_matematica;
- dichiarare i parametri che vi serviranno come se fossero delle variabili e racchiuderli tra due parentesi subito dopo al nome – int a e int b;
- aggiungere il corpo della funzione, cioè le operazioni che dovrà svolgere;
- restituire il calcolo eseguito utilizzando la parola chiave return – nell'esempio è restituito il valore della variabile c.

Potrete richiamare la funzione ovunque, indicando semplicemente i suoi parametri e salvando il risultato in una nuova variabile. La funzione sarà memorizzata in un'area di memoria e quando il computer arriverà a **operazione\_matematica** si segnerà il punto in cui è arrivato e salterà alla porzione di memoria in cui è salvata la funzione, la eseguirà e poi tornerà al punto in cui era rimasto, restituendo il risultato calcolato dalla funzione.

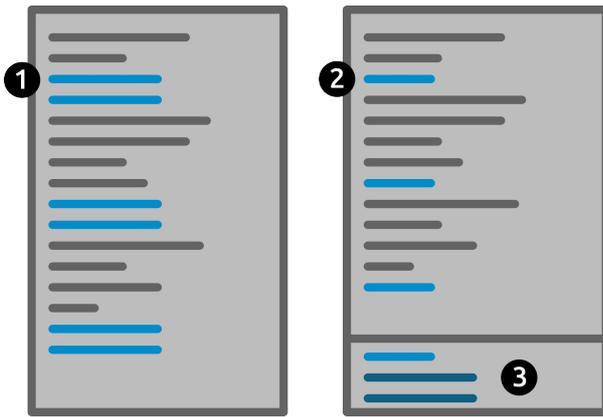


Fig 4.9 – Rappigurazione grafica di un programma con codice ripetuto (1) e di un listato che utilizza una funzione (3) richiamata in più punti del codice (2).

Nella figura 4.9 ho cercato di rappresentare graficamente il caso di codice senza funzioni e codice che utilizza delle funzioni. Nel listato di destra c'è un gruppo di linee (1) che sono ripetute più volte in diversi punti del codice. La situazione si può migliorare introducendo una funzione (3) che conterrà le linee di codice ripetuto. La funzione sarà richiamata da più parti all'interno del listato (2).

Per utilizzare la funzione è sufficiente scrivere il suo nome e fornire i parametri necessari. Potete passargli direttamente numeri e stringhe, oppure delle variabili.

```
int valore = operazione_matematica(4, c);
```

Se la funzione non restituisce un valore di ritorno è sufficiente invocarla:

```
accendi_stampante("stampante in corridoio");
```

Una funzione può anche non prevedere dei parametri.

```
int numero_vincente() {
    int c = numero_a_caso();
    return c;
}
```

Una funzione può non restituire nessun risultato e in questi casi si chiamerebbe **procedura**. In questo caso la prima parola da scrivere, prima del nome della funzione, è void che indica che non restituirete risultati. Nel corpo della funzione non serve usare return... perché non restituirete nulla.

```
void invia_ping (String str) {
    WebClient.ping(str);
}
```

Dove si mettono le funzioni? In C/C++ le potete definire dove volete, all'inizio o alla fine del codice, a patto che all'inizio del listato ci sia sempre la definizione della funzione, cioè la descrizione della funzione senza il «corpo». Potete inserire la funzione completa all'inizio del listato, come nell'esempio seguente:

```
#include <stdio.h>
```

```

int somma(int a, int b) {
    return a + b;
}

int main(void) {
    int res = somma(10, 5);
    printf("somma = %d \n", res);

    return 0;
}

```

La funzione «somma» è posta all'inizio del listato, esternamente al metodo principale «main». Se preferite, per maggior «pulizia», potete inserire solo la dichiarazione della funzione e poi porre il corpo dove preferite, come nell'esempio seguente:

```

#include <stdio.h>

int somma(int a, int b);

int main(void) {
    int res = somma(10, 5);
    printf("somma = %d \n", res);

    return 0;
}

int somma(int a, int b) {
    return a + b;
}

```

La funzione «somma» non è stata ripetuta: il compilatore deve sapere che la utilizzerete e si accontenta di avere una semplice dichiarazione di come sarà la funzione. Il corpo completo della funzione si può mettere in fondo al listato o anche in un file esterno.

# 5

## Programmare Arduino

In questo capitolo imparerete le basi di Arduino, cioè i comandi essenziali per creare la maggior parte dei progetti. Resterete stupiti, ma i comandi fondamentali sono solo cinque:

- `pinMode` – per configurare i pin della scheda,
- `digitalWrite` – per accendere un pin,
- `digitalRead` – per leggere lo stato di un pin,
- `analogWrite` – per creare dei segnali variabili,
- `analogRead` – per leggere segnali analogici.

### Lo sketch

Il punto di partenza di ogni progetto con Arduino è lo sketch, cioè il listato, che come sapete è organizzato in due sezioni principali:

- `setup` – eseguito solo all'accensione della scheda o dopo la pressione del tasto RESET;
- `loop` – che contiene le istruzioni che saranno ripetute in continuazione fino che Arduino è alimentato.

Impostate lo sketch in questo modo:

```
void setup() {  
  //codice eseguito all'avvio  
}  
void loop() {  
  //codice ripetuto  
}
```

Dovete assicurarvi che siano sempre presenti le due sezioni `setup` e `loop`, altrimenti appariranno degli errori. `Setup` e `loop` sono due funzioni che non restituiscono nessun risultato e non richiedono parametri, per questo vanno precedute dalla parola «chiave» **void**.

Potete caricare uno sketch «vuoto» cliccando sulla voce del menu `File: Esempi > Basic > BareMinimum`.

È obbligatorio creare uno sketch con `loop` e `setup`, ma non è necessario inserire del codice all'interno delle due sezioni. Potreste avere uno sketch in cui è presente del codice solo nella sezione `loop`, oppure solo nella parte `setup` (così che sia eseguito una volta sola). Potreste anche caricare uno sketch «vuoto» su Arduino, per ripulirlo da programmazioni precedenti.

### Come caricare uno sketch

Riporto in questo paragrafo la procedura da seguire per caricare uno sketch su Arduino. Ecco la sequenza di operazioni che dovrete ripetere ogni volta:

- avviate l'ambiente di programmazione di Arduino;
- scrivete il vostro sketch nella finestra del programma;
- componete il circuito elettronico e collegatevi Arduino;
- connettete Arduino al vostro computer utilizzando il cavo USB;

- verificate che sia impostata la porta corretta controllando in Tools > Port. Su Windows dovreste trovare una voce tipo COMx (Arduino UNO); su Mac avrete qualcosa tipo /dev/tty.usbmodemXXX, mentre su GNU/Linux troverete /dev/ttyACM0 o /dev/ttyS0 o S1.
- verificate che anche l'hardware impostato in Tools > Board corrisponda alla vostra scheda (per esempio Arduino UNO);
- premete il pulsante «Verify» (il primo della toolbar);



Fig. 5.1 – La finestra di Arduino con evidenziato il pulsante per la verifica dello sketch.

- il computer frullerà per qualche secondo mentre controlla il vostro listato. Se tutto è andato bene e non ci sono errori, nella parte inferiore della finestra (la console), dovreste vedere solo delle scritte di colore bianco.
- premete «Upload» (il secondo pulsante della toolbar), per trasferire il programma alla scheda;

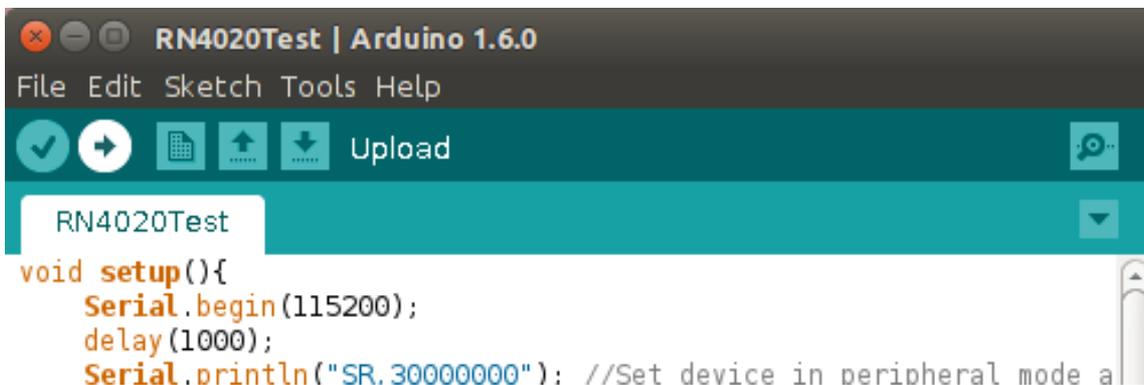


Fig 5.2 – La finestra di Arduino con evidenziato il pulsante «upload» per caricare lo sketch.

- i LED TX e RX lampeggeranno per qualche istante fino a che tutto il programma non è stato trasferito sulla scheda.

Non preoccupatevi se i passi della procedura di upload vi sembrano tanti, perché una volta presa confidenza, questa richiederà solo pochi istanti. Al termine del trasferimento, nella parte inferiore della finestra, troverete indicate la dimensione in byte dello sketch, l'occupazione di memoria espressa in percentuale e altre informazioni simili.

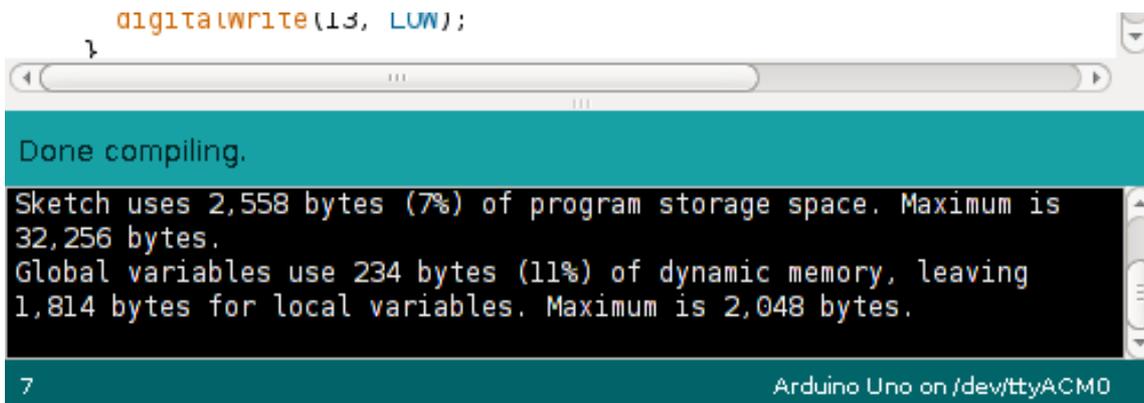


Fig 5.3 – Messaggi presenti nell'area inferiore della finestra di Arduino a seguito di un upload senza errori.

Se ci sono stati errori fase di verifica, oppure durante l'upload, nell'area inferiore appariranno dei messaggi di colore arancione che forniranno delle indicazioni utili per risolvere il problema. Nel caso di errori di scrittura del codice troverete indicata la riga in cui è stato rilevato l'errore. Caricando uno sketch su Arduino potreste incappare in problemi di trasferimento. Solitamente questi errori sono dovuti alla porta seriale che potrebbe bloccarsi, oppure al cavo che non fa bene contatto o anche a qualche problema con il circuito che avete collegato ad Arduino. In questi casi provate a scollegare la scheda, salvare il listato, chiudere del tutto Arduino e ripartire da capo. A volte potrebbe anche essere necessario riavviare il computer per risolvere la situazione. Un errore comune si ha quando il vostro circuito utilizza i pin numero 0 e 1, che sono condivisi con la porta seriale collegata al computer. Per caricare correttamente lo sketch dovrete scollegare temporaneamente i componenti che avete collegato a questi pin.

*Tutto quello che è connesso ai pin 0 e 1 può interferire con il caricamento degli sketch.*



Fig 5.4 – Messaggi di errore nell'area inferiore della finestra di Arduino: è stato rilevato un errore alla linea 8, colonna 5 dello sketch.

## BreadBoard

Una breadboard è una speciale tavoletta per costruire rapidamente prototipi elettronici. Non è necessario utilizzare un saldatore. La tavoletta è realizzata in materiale plastico ed è fornita di numerosi fori in cui s'inseriscono componenti elettronici e fili di collegamento (chiamati jumper). I fori sono raggruppati per file e per righe e sono in contatto elettrico tra di loro, così che se s'inseriscono i piedini di due componenti sulla stessa riga, questi saranno in contatto.

Le breadboard sono fabbricate in vari formati ma sono sempre divise in due metà da un solco centrale. Richiamando l'analogia con l'acqua, in cui i fili elettrici sono come tubi idraulici, i cinque fori di una riga della breadboard sono paragonabili a un tubo di raccordo con cinque vie: introducendo l'acqua da un foro potete prelevarla da uno dei rimanenti.

Il nome breadboard deriva dall'analogia con tipo di tagliere per il pane formato da un vassoio sopra di cui è posta una griglia con dei fori rettangolari. Quando si taglia il pane, le briciole si raccolgono nel vassoio sottostante passando per i fori.

Le breadboard sono molto utili per realizzare circuiti rapidi, ma non sempre garantiscono realizzazioni stabili e affidabili: basta poco per creare un falso contatto o un corto circuito. Un circuito dotato di numerosi «fili volanti» non ha buone prestazioni ed è affetto da rumore e disturbi che lo possono rendere instabile. Il circuito sarà tanto più instabile quanto maggiore sarà la sua frequenza di funzionamento.

*Fig 5.5 – Una Breadboard «half-size»: ha un solco centrale e due coppie di file laterali per distribuire l'alimentazione.*

Per maggiori informazioni si veda il libro «Elettronica per Maker» P.Aliverti – ed. LSWR

## pinMode

L'aspetto «magico» di Arduino<sup>1</sup> è la possibilità di modificare il comportamento di un pin direttamente e semplicemente con un'istruzione software. I pin digitali possono funzionare come

1 - Il fatto di poter modificare la destinazione, o mappatura, di un pin a vostro piacere non è un'esclusiva di Arduino ma di tutti i microcontrollori moderni. Alcuni circuiti integrati (chiamati FPGA), possono modificare le connessioni al loro interno come se fossero una breadboard.

ingressi o uscite. Li possiamo utilizzare per accendere LED, inviare segnali verso il mondo esterno, rilevare la pressione di un pulsante o ricevere informazioni. Prima di utilizzare un pin dovete dichiarare che tipo di comportamento assumerà: sarà un ingresso o un'uscita? Quest'operazione è necessario farla una sola volta, quando la scheda si accende e l'ATmega328 inizia a leggere lo sketch. Il comando da utilizzare è:

```
pinMode( numero_pin, direzione );
```

pinMode è una funzione a cui dovrete passare due parametri:

- il numero del pin che volete configurare (per Arduino UNO vanno da 0 a 13);
- la direzione o il comportamento del pin, usando le «parole chiave» (in realtà sono delle variabili «costanti»), INPUT o OUTPUT.

Per utilizzare il pin undici come un'uscita, nel blocco setup scrivete:

```
void setup() {  
    pinMode(11, OUTPUT);  
}
```

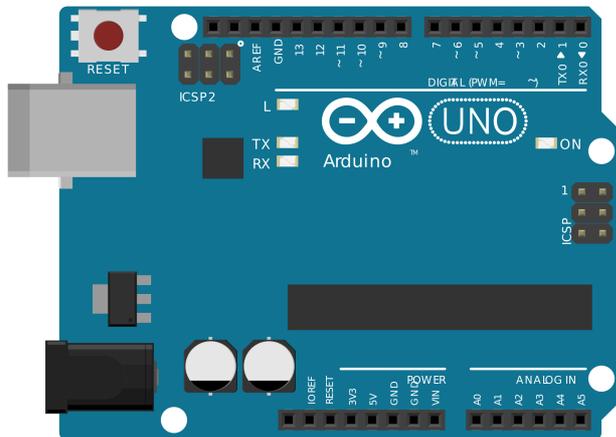


Fig 5.6 – Arduino UNO ha 14 pin digitali che possono comportarsi come ingressi o uscite.

Per utilizzare il pin come un ingresso digitale:

```
void setup() {  
    pinMode(11, INPUT);  
}
```

Nel capitolo due ho descritto in dettaglio la scheda Arduino UNO, elencando anche una serie di ingressi speciali, destinati alla lettura di segnali analogici. Questi pin sono contrassegnati con le sigle da A0 ad A5. Vedrete tra poco come utilizzare questi ingressi, ma per farlo non dovrete configurarli con pinMode.

Se vi servono molti canali digitali, probabilmente vi converrà impiegare un ArduinoMega che non presenta fondamentali differenze rispetto a un semplice Arduino, senonché ha un maggior numero di pin, porte e funzioni. Tutti i pin sono numerati, quindi per utilizzarne uno, verificate il suo «nome» sulla scheda e configuratelo con pinMode. Impostate il pin 32 di Arduino Mega scrivendo:

```
void setup() {
```

```

    pinMode(32, OUTPUT);
}

void loop() {
    //il resto delle istruzioni
}

```

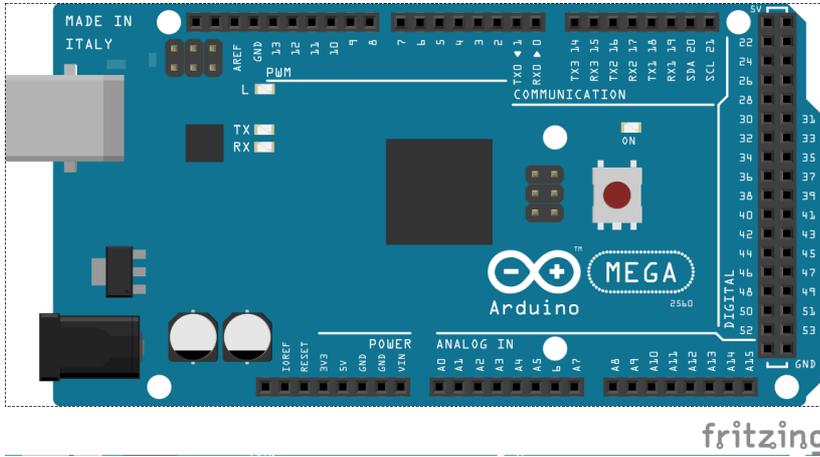


Fig 5.7 – Arduino Mega ha 54 pin di ingresso/uscita digitali

Non dimenticatevi di dichiarare i pin nel setup! Se non dichiarerete i pin in modo corretto o se vi scorderete di farlo, osserverete dei comportamenti strani e inspiegabili. Per esempio: se collegate un LED a un pin e poi vi dimenticate di impostarlo come «OUTPUT», il LED non si accenderà correttamente. In circuiti più complessi potreste osservare dei malfunzionamenti generali e inspiegabili.

### Quante volte si può utilizzare Arduino?

La memoria flash di Arduino ha un numero di scritture limitato. Gli sketch, durante l'upload sono copiati in quest'area di memoria che mantiene le informazioni anche quando Arduino è spento. La memoria flash è simile a una chiavetta USB e può essere scritta un numero limitato di volte. Questo può sembrare un grosso limite, ma il numero reale di cicli di cancellazione e scrittura della memoria flash è pari a 10.000.

Non ci sono limiti sulle operazioni di lettura.

## delay

L'istruzione delay introduce una pausa nel flusso di esecuzione delle istruzioni. Quando Arduino incontra questo comando, si mette in pausa, cioè si blocca per il tempo specificato dall'istruzione.

```
delay ( durata_della_pausa );
```

L'istruzione delay accetta quindi un solo parametro di tipo long (senza segno) che corrisponde alla pausa desiderata, espressa in millisecondi. Può sembrare esagerato usare una tale precisione ma Arduino è in grado di svolgere 16 milioni di operazioni al secondo e quindi di poter lavorare tranquillamente con tempi di microsecondi! Per introdurre una pausa di un secondo dovrete

## I LED

Il LED è un componente elettronico che può emettere della luce: una specie di lampadina. Ha la particolarità di generare la luce in modo molto efficiente, consumando poca corrente. Le comuni lampadine «a incandescenza» creano la luce surriscaldando un sottile filo metallico che diventa incandescente ed emette luce e calore, consumando molta energia (la maggior parte della quale finisce in calore). I LED utilizzano un differente processo fisico in cui un elettrone libera dell'energia che si trasforma in fotoni, cioè in particelle di luce. Questo processo ha un alto rendimento e non produce molto calore.

I LED hanno anche una particolarità: possono essere attraversati dalla corrente elettrica in un solo verso. Il loro nome è in realtà una sigla che significa Diodo Emittitore di Luce.

Per accenderli correttamente dovrete fornirgli una tensione e una corrente adeguate. La tensione dipende dal colore del LED e varia tra 1,8 e 3,5 volt. La corrente può variare da dieci a quindici milliampere.

Colore	Tensione (V)
rosso	1,8
giallo	1,9
verde	2,0
bianco	3,0
blu	3,5

*Tabella 5.1 – Relazione tra il colore del LED e la tensione di alimentazione.*

I LED sono dotati di polarità e quindi hanno un terminale positivo (anodo) e un terminale negativo (catodo). I terminali si riconoscono osservando il LED in trasparenza. Il terminale negativo si riconosce perché:

- è quello più corto tra i due,
- il bordo del LED presenta una smussatura sul lato del catodo,
- il catodo, all'interno della calotta del LED, ha la forma di specie di mazza da golf.

*Fig 5.8 – Un LED con indicati l'anodo e il catodo. Di fianco al LED è rappresentato il suo simbolo elettrico.*

Il simbolo elettrico del LED è una specie di freccia che ricorda il senso in cui scorre la corrente.

scrivere:

```
delay(1000);
```

un secondo è infatti pari a mille millisecondi. Per ottenere delle pause molto lunghe, oltre le decine di secondi, si preferiscono utilizzare altri metodi che evitano di bloccare il microcontrollore troppo a lungo.

## digitalWrite

Per attivare un'uscita digitale si utilizza il comando digitalWrite indicando il numero del pin e lo stato da assegnargli: alto o basso (HIGH o LOW).

```
digitalWrite( numero_del_pin, stato );
```

Un pin, utilizzato come uscita digitale può assumere solo due valori, corrispondenti ad «acceso» e «spento», cioè cinque o zero volt. Quando è configurato in questo modo, può anche fornire una corrente in uscita. Un pin impostato come ingresso può assorbire una corrente. Quando la corrente che attraversa il pin è eccessiva, si corre il rischio di danneggiarlo irrimediabilmente. Con Arduino il limite di corrente è di qualche decina di milliampere, per questo motivo non è possibile collegare direttamente un relè o un motore, che richiedono centinaia di milliampere, ma si devono utilizzare transistor o MOSFET di pilotaggio.

Potete «accendere» o «spegnere» tutti i pin che vanno dal numero 0 al numero 13<sup>2</sup>. Per verificare subito il comportamento di digitalWrite, provate ad accendere il LED sul pin 13 (è integrato sulla scheda).

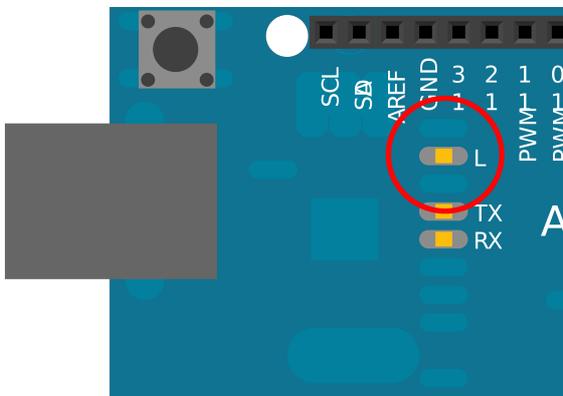


Fig 5.9 – Il LED integrato sulle schede Arduino è collegato al pin 13.

Per accenderlo aprite un nuovo sketch e scrivete:

```
void setup() {  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    digitalWrite (13, HIGH);  
}
```

2 - se utilizzate Arduino UNO che ha quattordici pin digitali.

Nel setup dello sketch avete configurato il pin 13 come un'uscita. Nel loop lo accendete impostando lo stato HIGH con digitalWrite. Caricate lo sketch premendo «verify» e poi «upload»: vedrete il LED «L» accendersi e restare acceso. Per spegnere il LED modificate lo sketch modificando i parametri dell'istruzione digitalWrite:

```
digitalWrite (13, LOW);
```

Per spegnere il LED l'uscita deve essere posta al valore LOW. In questi sketch l'istruzione digitalWrite è posta nella sezione loop. Potete riscriverli in modo più furbo:

```
void setup() {  
    pinMode(13, OUTPUT);  
    digitalWrite (13, HIGH);  
}  
  
void loop() {  
}
```

Le istruzioni nel blocco loop sono ripetute in continuazione, mentre non è necessario ripetere digitalWrite una volta che è stato modificato lo stato dell'uscita. Per questo potete mettere i due comandi nel blocco setup e lasciare il loop completamente vuoto.

*I pin digitali mantengono il loro stato fino a che non lo modificherete con un nuovo comando.*

## LED lampeggiante montato su una breadboard

Vi propongo un semplice esercizio per far lampeggiare un LED posto su una breadboard. Dovrete comporre un circuito utilizzando Arduino, un LED e una resistenza. La resistenza serve per limitare la corrente in uscita dal microcontrollore e alimentare correttamente il LED senza danneggiare Arduino. In alcuni esempi si vede spesso un LED inserito direttamente nei pin di Arduino, senza l'uso di una resistenza. Potete permettervi di collegare un LED in questo modo solo se infilate l'anodo nel pin 13 e il catodo nel pin GND (posto di fianco). Questo è possibile perché il pin 13 è collegato al LED «L» presente sulla scheda tramite una resistenza e quindi non rischierete di danneggiare Arduino.

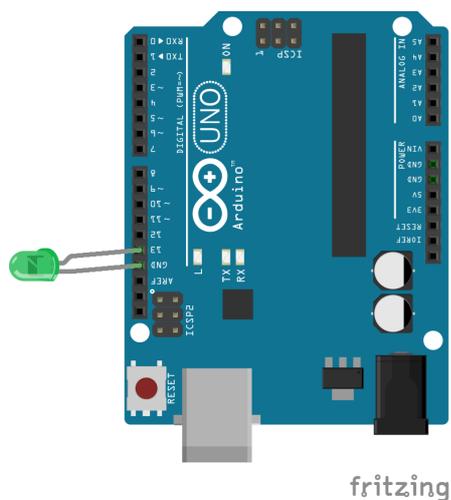


Fig 5.10 – LED collegato senza resistenza sui pin 13 e GND.

In ogni altro caso dovrete fare un semplice calcolo per determinare la resistenza da collegare in serie al LED. Immaginate di avere un LED verde che richiede una tensione di alimentazione di 2 volt e una corrente di 10 mA. I pin di Arduino però, quando sono configurati come uscite e posti a livello alto, producono 5 volt.

Immaginate di sostituire al pin che utilizzerete per accendere il LED, una batteria da 5 volt. La batteria sarà collegata alla resistenza e quindi al LED (l'ordine non è importante perché la corrente non si comporta proprio come l'acqua che scorre in un tubo!).

Dovrete scrivere un bilancio in cui da una parte avete la batteria e dall'altra la resistenza con il LED.

$$5V = V_{LED} + RI$$

Risolvendo questa formula in R:

$$RI = 5V - V_{LED}$$

$$R = (5V - V_{LED}) / I$$

Sostituite 2 a  $V_{LED}$  e 10 mA (0,010 A) a I:

$$R = (5V - 2V) / 0,010A = 3V / 0,010A = 300 \text{ ohm}$$

La resistenza da utilizzare ha un valore di 300 ohm, che purtroppo non troverete in commercio. Dovrete accontentarvi di una resistenza da 270 o 330 ohm.

Procediamo con l'esercizio. Procuratevi il necessario: il LED, la resistenza, la breadboard e dei fili (o jumper) per i collegamenti.

Ecco cosa dovete fare:

- assicuratevi che Arduino sia scollegato dal computer;
- avviate l'IDE di Arduino;
- copiate lo sketch per far lampeggiare il led sul pin 11 che trovate riportato qui di seguito.

```
void setup() {
    pinMode(11, OUTPUT);
}

void loop() {
    digitalWrite (11, HIGH);
    delay(1000);
    digitalWrite (11, LOW);
    delay(1000);
}
```

- con Arduino scollegato dal computer, componete il circuito presentato in figura 5.11;
- collegate Arduino al computer utilizzando il cavo USB;
- caricate lo sketch premendo «verify» e quindi «upload»;
- dopo qualche istante, terminata la procedura di upload, il LED inizierà a lampeggiare.

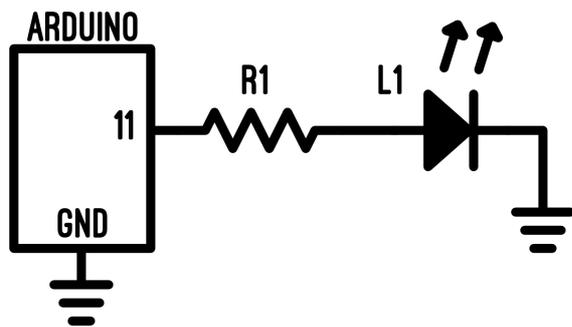


Fig 5.11 - Schema elettrico del circuito con il LED e Arduino.

### Descrizione dello sketch

Nel setup, pinMode, configura il pin 11 come uscita. La prima operazione che s'incontra nel loop è `digitalWrite(11, HIGH)`, che accende il LED. Subito dopo, l'istruzione `delay`, impone una pausa della durata di un secondo: il LED rimarrà acceso per tale tempo. Al termine della pausa, `digitalWrite(11, LOW)`, spegne il LED e poi un nuovo `delay` impone una pausa di un secondo. Al termine della pausa le quattro istruzioni sono ripetute: il LED lampeggerà con intervalli di un secondo.

### Descrizione del circuito

Su una breadboard inserite un LED e una resistenza da 330  $\Omega$ . Collegate il pin undici di Arduino all'anodo del LED e un terminale del resistore al pin GND, come indicato nella figura 5.12.

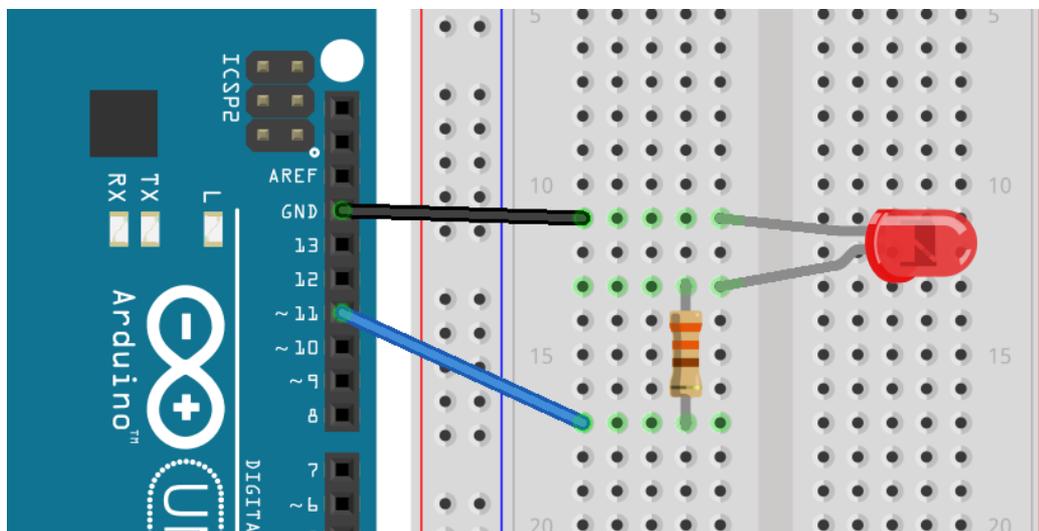


Fig 5.12 - Composizione del circuito elettrico con Arduino, una resistenza e un LED sul pin 11.

*Non collegate direttamente ai pin di Arduino dispositivi e componenti elettronici che richiedono correnti maggiori di 40/50 mA. I dispositivi da evitare sono tutti quelli che includono delle bobine o avvolgimenti, come motori elettrici, relè e magneti e dispositivi che producono calore.*

## I sei pin segreti di Arduino

Arduino UNO ha ben venti pin digitali. Avete capito bene: venti, non quattordici come trovate stampigliato sulla scheda. I sei pin «segreti» non sono altro che gli ingressi analogici contrassegnati con le sigle da A0 ad A5. I pin analogici corrispondono ai numeri: 14, 15, 16, 17, 18 e 19. Quando utilizzate uno di questi pin come «digitale», non potete utilizzarlo anche come ingresso analogico. Per configurare i pin non dovete fare altro che dichiararne l'utilizzo con `pinMode`.

Ecco come far lampeggiare un LED sul pin A0:

- inserite un LED e una resistenza da 270 ohm su una breadboard (vedi figura);
- collegare l'anodo (+) del LED sul pin A0;
- collegate il terminale «libero» della resistenza a GND;
- caricate il seguente sketch:

```
void setup() {
  pinMode(14, OUTPUT); //il pin A0
}

void loop() {
  digitalWrite(14, HIGH);
  delay(1000);
  digitalWrite(14, LOW);
  delay(1000);
}
```

*Fig 5.13 – All'occorrenza potete configurare gli ingressi analogici come porte digitali.*

## digitalRead

Per leggere lo stato di pin digitale utilizzato come ingresso si utilizza `digitalRead`, indicando come parametro il numero della linea da leggere.

```
digitalRead( numero_del_pin );
```

La funzione restituirà un valore numerico di tipo intero (`int`) che potrà valere solo HIGH o LOW. Solitamente il risultato della funzione è salvato in una variabile di tipo `int`:

```
int tasto1 = digitalRead( 7 );
```

Per utilizzare un pin come ingresso digitale, dovete ricordarvi di dichiararlo come tale nella sezione setup dello sketch. Usate pinMode per configurare il pin numero 7:

```
void setup() {  
    pinMode(7, INPUT);  
}
```

Il caso più semplice di utilizzo di digitalRead è per rilevare la pressione di un pulsante. Il pin configurato come ingresso digitale può rilevare se la tensione applicata è pari a zero o cinque volt. È necessario costruire un piccolo circuito con un pulsante e una breadboard. Lo schema elettrico è illustrato nella figura seguente.

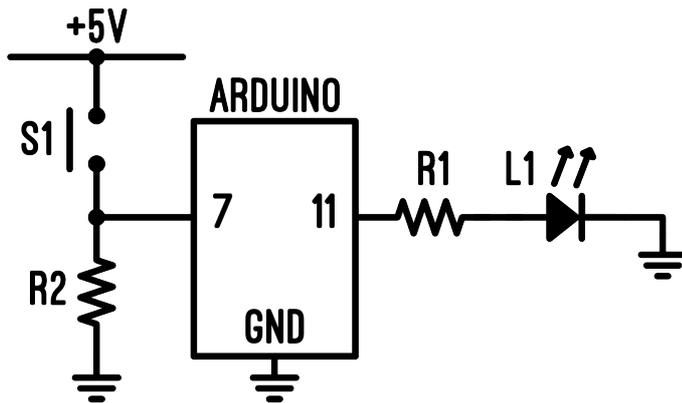


Figura 5.14 – Schema elettrico di collegamento di un pulsante ad Arduino.

Vi dovrete procurare un pulsante del tipo normalmente aperto, una resistenza da 10000 ohm, un LED verde e una resistenza da 330 ohm, oltre a una breadboard e alcuni jumper per realizzare i collegamenti. La figura 5.15 illustra la disposizione dei componenti sulla breadboard.

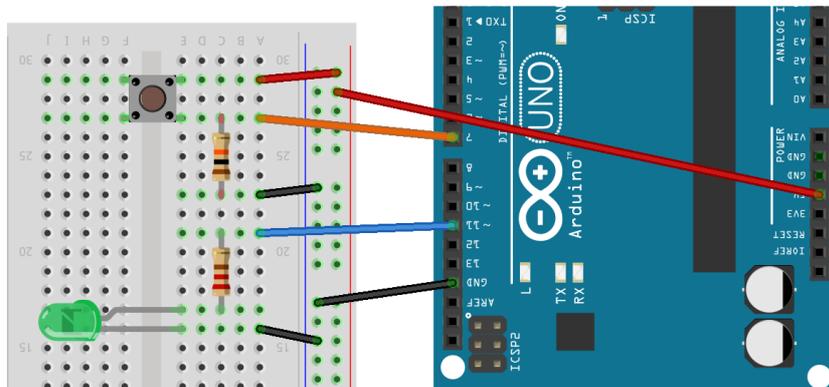


Figura 5.15 – Arduino collegato a un pulsante sul pin 7 e a un LED sul pin 11.

Il pulsante ha due terminali<sup>3</sup>: uno dei due è collegato a massa tramite la resistenza da 10 kΩ e anche al pin 7. L'altro contatto è collegato ai cinque volt dell'alimentazione. In questo modo l'ingresso di Arduino è collegato stabilmente a zero volt tramite la resistenza. Non circola corrente e quindi il pin 7 si può considerare a livello basso. Quando premiamo il pulsante, colleghiamo i cinque volt alla

<sup>3</sup> In realtà questo tipo di pulsante ha quattro terminali per ragioni di stabilità meccanica, che sono collegati a coppie.

resistenza e quindi al pin 7. Se non ci fosse la resistenza, premendo il tasto uniremmo direttamente i cinque volt con la massa, creando un corto circuito!

Lo sketch non è molto complicato:

```
void setup() {  
  pinMode(7, INPUT);  
  pinMode(11, OUTPUT);  
}  
  
void loop() {  
  int tasto1 = digitalRead(7);  
  if (tasto1 == HIGH) {  
    digitalWrite (11, HIGH);  
    delay(1000);  
  }  
  digitalWrite (11, LOW);  
}
```

Nel setup troviamo la configurazione dei pin: il pin 7 è impostato come ingresso e il pin 11 come uscita. Nella sezione loop, il programma legge lo stato del pin 7 con `digitalRead` e lo salva nella variabile di tipo `int` di nome «pulsante». La funzione `digitalRead` richiede un solo parametro, il numero del pin da leggere e fornisce lo stato del pin che può essere HIGH o LOW. La prima riga della sezione loop legge lo stato del pin e lo salva nella variabile «tasto1». Il contenuto della variabile è controllato dall'istruzione `if`, e se questo è pari a HIGH perché qualcuno ha premuto il pulsante, vengono eseguite le istruzioni racchiuse tra le due parentesi graffe: `digitalWrite` accende il pin 11 e poi c'è una pausa di un secondo. Dopo la pausa, il programma esce dal «blocco if» e trova l'istruzione `digitalWrite` che spegne il LED.

Se nessuno preme il tasto, la lettura del pin restituisce LOW e quindi Arduino non esegue le istruzioni all'interno dell'if, passando direttamente all'ultima `digitalWrite` che spegne il LED.

Potrebbe venirvi un dubbio su come collegare il pulsante ad Arduino. Non collegatelo mai direttamente al pin. Un errore comune è quello di collegare un terminale del pulsante ai 5 volt e l'altro terminale ad Arduino, così quando premerete il tasto applicherete direttamente 5 volt al microcontrollore. Purtroppo non accade così: il pin di Arduino quando il pulsante non è premuto non è collegato a nulla e assume degli stati a caso.

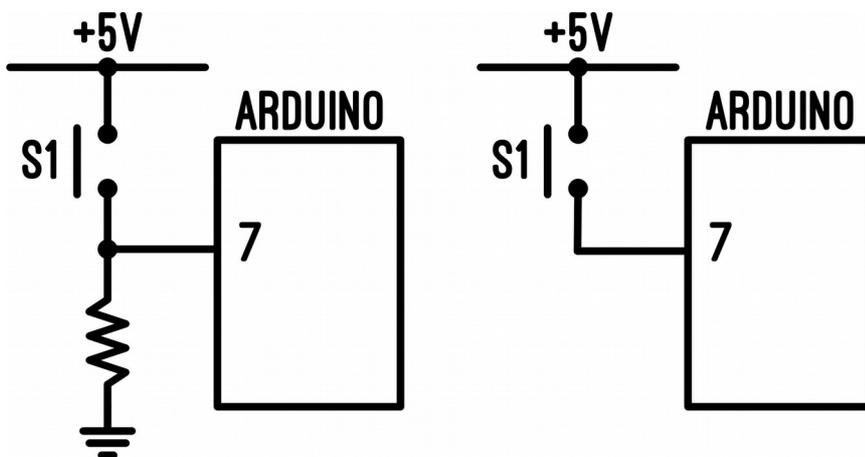


Fig. 5.16 – nello schema a sinistra il pulsante è collegato correttamente ad Arduino, lo schema si destra mostra come non dovreste mai collegare un pulsante a un microcontrollore.

## analogWrite

La funzione `analogWrite` produce un segnale PWM, cioè un'onda quadra cui potete modificare a piacere la percentuale di segnale acceso e spento. `AnalogWrite` richiede due parametri: il pin di uscita e un numero che indica il rapporto tra acceso e spento. Il numero non è una percentuale, ma un valore che va da zero a 255. Non tutti i pin di Arduino possono generare un segnale di questo tipo, quelli abilitati sono: 3, 5, 6, 9, 10 e 11. Li riconoscerete perché sulla scheda, di fianco al numero c'è una piccola tilde (~).

```
analogWrite ( numero_del_pin, valore_pwm );
```

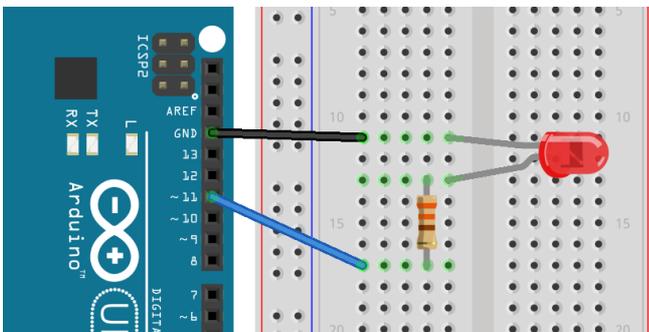
Il nome di questa funzione trae in inganno, perché non produce un segnale analogico, ma un'onda quadra variabile (cioè un segnale PWM)!

*Arduino legge segnali analogici ma non può produrli.*

Da uno dei suoi 14 pin non potrete far uscire 2,25 volt, ma solo 0 o 5. Applicando un segnale PWM a un LED potrete però modulare la sua luminosità con grande precisione. Quando il valore del PWM è a 0, il LED è spento perché l'onda quadra in uscita è sempre «a livello basso». Provate il seguente sketch:

```
void setup() {  
    pinMode(11, INPUT);  
    analogWrite(11, 0);  
}  
void loop() {}
```

Per utilizzare lo sketch dovrete costruire un semplice circuito, come illustrato nella figura seguente. Utilizzate un LED che collegherete al pin 11 di Arduino. Vi serviranno anche una breadboard, dei jumpers e una resistenza da 330 ohm. Caricatelo e... non vedrete nulla: il LED sarà spento.



*Fig 5.17 – collegamento di un LED al pin 11 per l'esperimento con il segnale PWM.*

Quando il valore del PWM è pari a 255, dal pin uscirà un segnale costante, pari a 5 volt e il LED sarà completamente acceso:

```
void setup() {  
    pinMode(11, INPUT);  
    analogWrite(11, 255);  
}  
void loop() {}
```

Caricate lo sketch e osserverete che il LED rimarrà sempre acceso. Ora provate a modificare il

valore fornito ad `analogWrite`, inserendo un numero compreso tra 0 e 255. Per accendere il LED a metà potenza inserite, per esempio, 127.

```
void setup() {
    pinMode(11, INPUT);
    analogWrite(11, 127);
}
void loop() {}
```

Il LED ora sembra accendersi a metà della sua potenza ma è un'illusione ottica. In realtà dal pin 11 esce un segnale velocissimo che passa da 0 a 5 volt numerose volte in un secondo. Modificando il valore passato alla funzione, modificherete il rapporto tra il tempo in cui il segnale è nello stato alto e quello in cui è nello stato basso, dando l'impressione che il LED si accenda di più o di meno.

Un segnale PWM può essere utilizzato anche per controllare la velocità di un motore elettrico. Non collegate direttamente il motore ad Arduino! Serve un transistor per evitare di danneggiare il microcontrollore.

Se per impostare la luminosità, utilizzare un numero da 0 a 255, risulta un po' inusuale, potete adottare la funzione **map**, che esegue una proporzione e ri-mappa i numeri da 0 a 255 su quelli da 0 a 100.

```
void setup() {
    pinMode(11, OUTPUT);
}

void loop() {
    int lum = 10;
    int pwm = map(lum, 0,100, 0,255);
    analogWrite (11, pwm);
}
```

Nello sketch trovate due variabili di tipo intero chiamate: `lum` e `pwm`.

La funzione `map` riceve come parametro la variabile **lum** che rappresenta la luminosità desiderata espressa in percentuale e la ri-mappa tra 0 e 255. Se `lum` vale 10, allora `map` restituirà 25,5, ma visto che deve essere un numero intero, avremo 25.

Caricate lo sketch su Arduino e osservate il comportamento del LED. Modificate il valore di `lum` e osservate come cambia la luce emessa dal LED.

## Serial Monitor

Non è facile capire quello che accade all'interno di Arduino. Se lo sketch è semplice e tutto funziona, non ci sono problemi, ma se qualcosa va storto non è immediatamente possibile comprendere cosa stia capitando. È possibile far trasmettere delle informazioni attraverso la porta seriale di Arduino e quindi leggerle dal vostro PC<sup>4</sup>. Quando la porta seriale non è impegnata nella programmazione del microcontrollore, potete utilizzarla per scambiare informazioni che si possono visualizzare su una finestra del computer. Per utilizzare la porta seriale di Arduino è necessario configurarla nel `setup`:

<sup>4</sup> Solo quando Arduino e il PC sono collegati con un cavo USB.

```
void setup() {  
  Serial.begin(9600);  
}
```

La configurazione della porta seriale richiede un solo parametro, la velocità di comunicazione, che di solito s'impone a 9600 baud<sup>5</sup>. Ora potete scrivere sulla seriale con:

```
Serial.println("Hello World!");
```

Ecco lo sketch completo:

```
void setup() {  
  Serial.begin(9600);  
  Serial.println("Hello World!");  
}  
  
void loop() {  
}
```

Dove finiscono questi messaggi? Sono inviati verso il computer e si possono leggere aprendo il «Serial Monitor» di Arduino, cioè una finestra che visualizza i messaggi ricevuti (e può anche trasmetterli!).



Fig. 5.18 – Per aprire il Serial Monitor premete il pulsante sulla toolbar di Arduino.

Ecco uno sketch che, ogni secondo, invia un messaggio al PC:

<sup>5</sup> Un baud indica il numero di simboli, cioè gruppi di bit, trasmessi in un secondo attraverso una linea digitale.

```

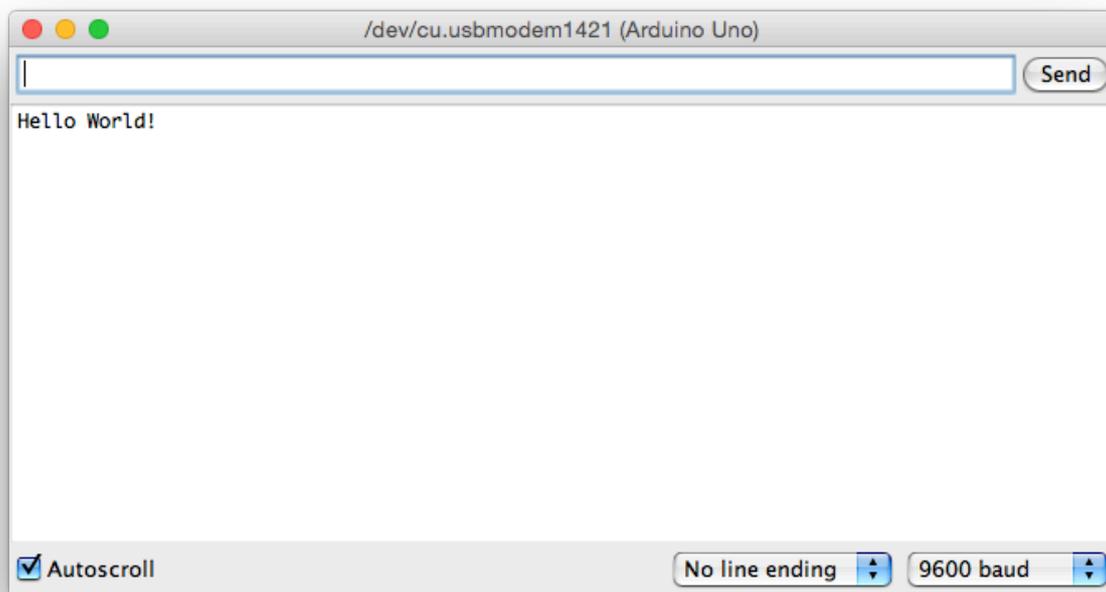
void setup() {
    Serial.begin(9600);
}

void loop() {
    Serial.println("Hello World!");
    delay(1000);
}

```

Nel setup si configura la porta seriale, indicando la sua velocità di trasmissione. Nella sezione loop, l'istruzione `Serial.println` invia un testo alla porta seriale, quindi `delay` imposta la pausa di un secondo.

Caricate lo sketch e poi aprite il Serial monitor per ricevere il messaggio. Noterete che quando aprite il serial monitor, Arduino si resetterà<sup>6</sup>.



*Fig 5.19 – Il serial monitor riceve i messaggi inviati da Arduino.*

Con questo metodo potete controllare il valore di una variabile o una condizione presente in uno sketch e inviarla al Serial Monitor, per capire come si sta comportando lo sketch. Provate a caricare lo sketch seguente:

```

int i = 0;

void setup() {

```

<sup>6</sup> Il Serial Monitor invia un segnale «standard» chiamato RTS che sulla scheda è collegato a «reset». Inevitabilmente, ogni volta che aprite il serial monitor, Arduino si resetterà. Potete evitarlo con un terminale che vi permette di non inviare il segnale di RTS, disattivando il controllo di flusso della trasmissione seriale.

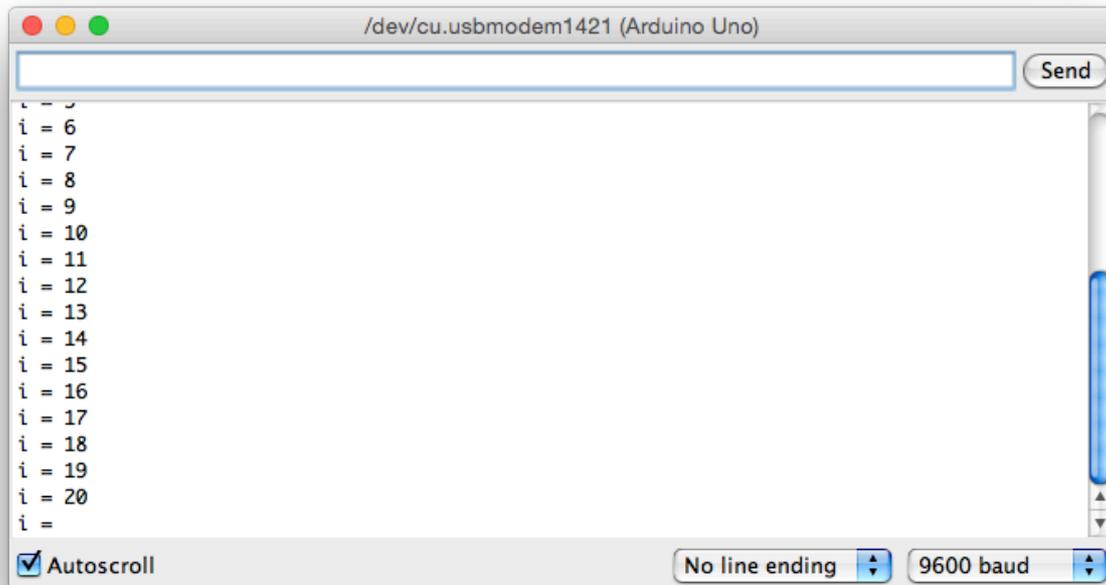
```

    Serial.begin(9600);
}

void loop() {
    Serial.print("i = ");
    Serial.println(i);
    delay(1000);
    i++;
}

```

Caricate lo sketch su Arduino e poi aprite il serial monitor: vedrete una sequenza di numeri crescere a intervalli di un secondo.



*Fig. 5.20 – Serial monitor con la sequenza progressiva di numeri trasmessi da Arduino.*

Lo sketch usa una variabile esterna alle sezioni loop e setup e che quindi è visibile in entrambi i blocchi. Nel loop l'istruzione print stampa un testo senza andare a capo. L'istruzione seguente, println, stampa la variabile «i». Println a differenza di print aggiunge un carattere di “a capo”. L'ultima istruzione del loop incrementa la variabile i di un'unità.

## analogRead

Arduino non può generare dei segnali analogici, ma può leggerli con la funzione:

```
analogRead ( numero_porta_analogica );
```

AnalogRead converte il valore di tensione presente su uno degli ingressi analogici dedicati (i pin da A0 a A5) in un numero compreso tra 0 e 1023<sup>7</sup>. Potete collegare un qualsiasi tipo di sensore analogico che produca una tensione compresa tra zero e cinque volt. Provate a collegare una foto-

<sup>7</sup> Il valore minimo di segnale che potete leggere è pari a  $5 \text{ V} / 1024$  e cioè 0,0048 V, cioè circa 5 millivolt.

resistenza ad Arduino e a leggerne il valore. La foto resistenza è componente speciale, sensibile alla luce. Ha una resistenza di qualche kilo-ohm che collegherete in serie con una resistenza da 10 kilo-ohm, formando una partitore di tensione. Collegate il pin A0 al punto centrale del partitore.

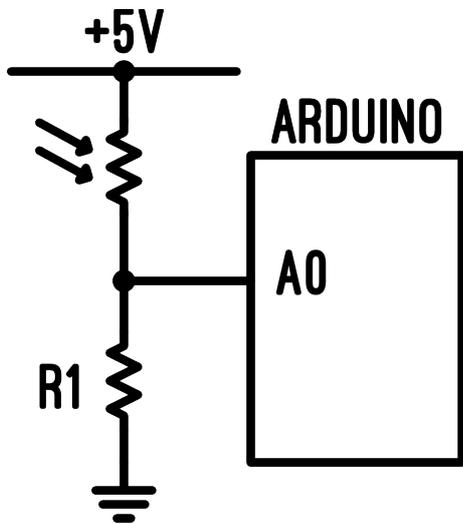


Fig. 5.21 – Schema elettrico del circuito per collegare una foto resistenza ad Arduino.

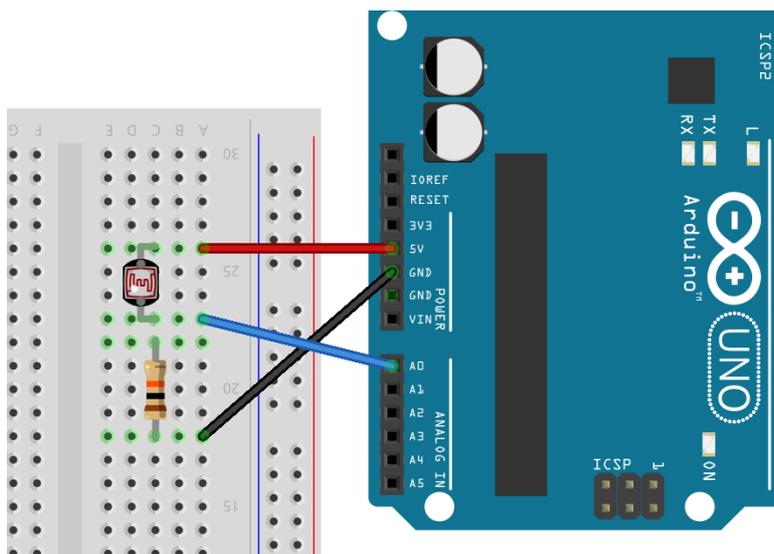


Fig. 5.22 – realizzazione del circuito su una breadboard.

*Non è necessario dichiarare gli ingressi analogici.*

Ecco lo sketch per controllare la quantità di luce che raggiunge la foto resistenza:

```
void setup() {
    Serial.begin(9600);
}

void loop() {
    int luce = analogRead(A0);
    Serial.print("livello luce: ");
    Serial.println(luce);
}
```

```
    delay(200);  
}
```

Nel setup impostate solo la porta seriale, perché non è necessario dichiarare gli ingressi analogici come tali. Nel loop il valore prelevato dall'`analogRead` è conservato nella variabile intera «luce». Il valore della variabile è stampato sul serial monitor. Infine trovate una pausa da 200 millisecondi, per evitare che Arduino «vomiti» una valanga di letture.

Collegate Arduino e caricate lo sketch, quindi aprite il serial monitor per leggere la sequenza di numeri. Dovreste vedere cambiare i valori quando farete ombra alla foto resistenza (o se la illuminerete).

Provate a scrivere uno sketch per accendere un led solo quando è buio.

Trovi altre informazioni, approfondimenti e libri su:

[www.zeppelinmaker.it](http://www.zeppelinmaker.it)

Se trovi degli errori, se il libro ti è piaciuto oppure no, se vuoi salutarmi:  
scrivimi!

[paolo@zeppelinmaker.it](mailto:paolo@zeppelinmaker.it)

Iscriviti alla mia Mailing List per ricevere tutti gli aggiornamenti  
e altre novità:

<http://www.zeppelinmaker.it/ mailing-list/>

In autunno/inverno dovrei iniziare a pubblicare:

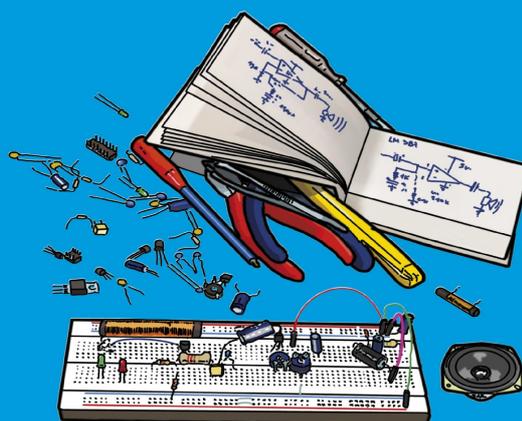
**“La guida galattica della stampa 3D”**

Non perdertela!

Paolo Aliverti

# Elettronica per maker

Guida completa



Tutte le informazioni per essere autonomo nei tuoi progetti >>

Transistor, chip analogici e digitali, microcontrollori >>

Impara a realizzare circuiti elettronici completi >>

Spiegazioni semplici ed esempi pratici >>

MADE FOR  
MAKERS

LSWR

## Elettronica per Maker - Guida completa

Il movimento dei maker, le stampanti 3D e Arduino hanno portato nuovo interesse per l'hobbistica elettronica. Sempre più nuovi appassionati, curiosi, inventori e innovatori si avvicinano a nuove e potenti tecnologie per creare prototipi e circuiti complessi. Le potenzialità offerte dai nuovi strumenti sono innumerevoli e a volte strabilianti. Chiunque può programmare una scheda Arduino usando un semplice cavo USB e costruire droni, robot e stampanti 3D. Per poter essere indipendenti servono un po' di esperienza e alcune conoscenze di base che non sempre sono facilmente reperibili in Rete. Questo libro non vuole essere un nuovo testo su Arduino, trattandolo solo in modo marginale, ma propone al lettore una serie di approfondimenti teorici e pratici per comprendere questa affascinante materia e per essere autonomi nello sviluppo dei propri progetti. Il testo include delle sezioni teoriche necessarie per spiegare e capire gli esperimenti oltre che a esercizi e applicazioni pratiche. Che componenti si possono usare oltre a LED e pulsanti? Come funziona un transistor e a cosa serve? Come si amplifica un segnale? Come si alimenta un prototipo?

PER MAGGIORI INFORMAZIONI: [HTTP://WWW.ZEPPELINMAKER.IT/PUBBLICAZIONI/](http://www.zepelinmaker.it/pubblicazioni/)

# 123Design

per la Stampa 3D



## 123Design per la Stampa 3D

Tutto quello che serve sapere per passare dal disegno all'oggetto stampato  
Come utilizzare 123Design per creare disegni adatti ad essere stampati con una stampante 3D.  
Scopri cos'è la stampa 3D e come creare file adatti ad essere stampati. Un libro per chi si avvicina al mondo della stampa 3D e non ha familiarità con un programma CAD.

PER MAGGIORI INFORMAZIONI: [HTTP://WWW.ZEPPELINMAKER.IT/PUBBLICAZIONI/](http://www.zepelinmaker.it/pubblicazioni/)



## Il Manuale del Maker

Cosa significa essere un maker? Come si può partire da un'idea e renderla realtà? Quali sono gli strumenti a nostra disposizione? E come possiamo evitare di sprecare un anno di lavoro per costruire qualcosa che nessuno desidera?

«Il Manuale del Maker» risponde a queste e a tante altre domande sul fenomeno che sta rivoluzionando il modo di progettare e produrre gli oggetti, dal pezzo di ricambio ormai introvabile per la vecchia lavatrice alle più complesse e fantasiose macchine interattive.

In 324 pagine, con oltre 300 immagini a colori, tutte le tecniche e le pratiche per trasformare immediatamente le proprie idee in progetti concreti, utilizzando le nuove tecnologie digitali, l'elettronica e la programmazione.

Spiegazioni chiare e puntuali per liberare la creatività, gestire progetti di successo e dare vita a un business sostenibile.

PER MAGGIORI INFORMAZIONI: [HTTP://WWW.ZEPPELINMAKER.IT/PUBBLICAZIONI/](http://www.zepelinmaker.it/pubblicazioni/)

Paolo Aliverti

## Stampa 3D Stazione futuro



*Con una storia italiana di successo: Sharebot*

MICROSCOPI



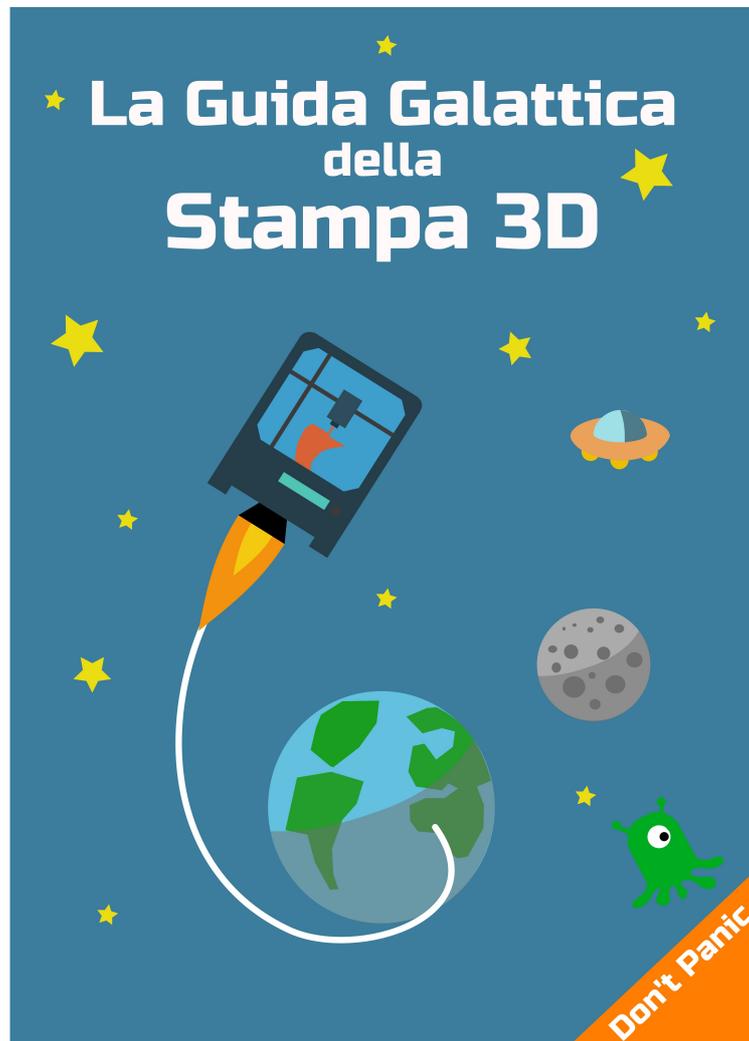
HOEPLI

## Stampa 3D - Stazione Futuro

La tecnologia della stampa 3D si sta diffondendo sempre più velocemente nella nostra società. Progettare e produrre oggetti con queste macchine, oggi, è semplice ed economico. E così, le stampanti 3D, da prodotto di nicchia e per pochi iniziati, entreranno nelle nostre case e nei nostri uffici, per affiancare gli altri strumenti tecnologici che già caratterizzano la nostra vita. Queste macchine possono realizzare (quasi) ogni tipo di oggetto: basta sapere usare semplici programmi di modellazione 3D per trasformare le nostre idee in prodotti tridimensionali. Dall'idea ai bit e dai bit agli atomi: questa è la potenza della digital fabrication. Una nuova e rivoluzionaria tecnologia sta arrivando sulle nostre scrivanie e potrebbe stravolgere il modo con cui produciamo e compriamo gli oggetti.

**PER MAGGIORI INFORMAZIONI: [HTTP://WWW.ZEPPELINMAKER.IT/PUBBLICAZIONI/](http://www.zepelinmaker.it/pubblicazioni/)**

**Di prossima uscita:**



La guida completa e gratuita sul mondo della stampa 3D: tecnologie, teoria, software e programmi.

**PER MAGGIORI INFORMAZIONI: [HTTP://WWW.ZEPPELINMAKER.IT/PUBBLICAZIONI/](http://www.zepelinmaker.it/pubblicazioni/)**

# Il manuale di Arduino

## Guida Completa

Arduino è una piccola scheda elettronica dotata di un microcontrollore, cioè un piccolo computer, che chiunque può imparare a utilizzare in breve tempo per realizzare circuiti elettronici interattivi. Per sviluppare un progetto è sufficiente collegare Arduino con un cavo USB al proprio computer, aggiungere qualche sensore e pochi componenti elettronici. Il sistema è molto semplice, rapido e intuitivo.

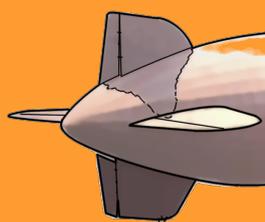
Questo manuale nasce dall'esperienza pluriennale dell'autore nell'utilizzo di Arduino e dei microcontrollori. Il libro è una guida semplice e completa per principianti e maker poco pratici di elettronica e programmazione.

## Argomenti del libro

- > Introduzione ad Arduino
- > Installazione e utilizzo delle schede
- > Elementi di programmazione
- > Elettronica, sensori e attuatori
- > Internet, trasmissione dati e comunicazioni
- > Arduino YUN e Internet of Things

## L'autore

Paolo Aliverti, ingegnere delle telecomunicazioni, artigiano digitale, maker e imprenditore. Utilizza Arduino dal 2008 con cui ha sviluppato vari progetti amatoriali e professionali. Ha scritto vari testi su elettronica, digital fabrication e stampa 3D. Il suo blog è [www.zeppelinmaker.it](http://www.zeppelinmaker.it)



[zeppelinmaker.it](http://zeppelinmaker.it)