

ARDUINO



Prof.ssa Lucia Tattoli
Prof.ssa Maria Teresa Tattoli

Sommario

Introduzione	3
Cos'è un microcontrollore	3
Cos'è Arduino	3
Arduino Uno	6
Struttura della scheda.....	6
Progetti con Arduino: la breadboard.....	8
Strumenti per la programmazione.....	10
Arduino IDE.....	10
L'ambiente di sviluppo	11
Collegare la scheda.....	13
Il debug a runtime.....	16
Primi elementi di programmazione: lampeggio di un led	17
La struttura di uno sketch: le funzioni setup e loop	17
I pin di Arduino e la funzione pinMode.....	17
Funzione setup.....	17
Invio di valori ai pin: digitalWrite.....	18
Funzione loop	18
Temporizzare le operazioni: delay	18
Scrivere l'applicazione e caricarla nella scheda	19
Uso della breadboard: lampeggio di un led su circuito esterno	20
Approfondimento sui componenti	20
Schema elettrico	21
Sketch	21
Acquisire un input digitale: pulsante e led	22
Approfondimento sui componenti: il pulsante.....	22
Schema elettrico	22
Lettura dai pin: digitalRead.....	23
Sketch	23
Arduino, sensori e attuatori: il Physical Computing	24
Cos'è il Physical Computing	24
Sensori	24
Attuatori	25
Acquisire, elaborare e attuare	25
Sensori analogici e digitali	25
Arduino: leggere i movimenti da un potenziometro.....	25
Sketch per leggere il potenziometro.....	26

Introduzione

Cos'è un microcontrollore

Il termine Microcontrollore o MCU (Microcontroller unit) identifica un particolare circuito integrato che dispone nel suo interno di almeno cinque blocchi funzionali:

- un'**interfaccia d'ingresso**, composta di uno o più ingressi attraverso i quali acquisire dati e informazioni dal mondo esterno; agli ingressi possono venire applicati segnali digitali (livelli logici) o analogici (tensioni, frequenze).
- un'**interfaccia di uscita**, composta di una o più uscite, in grado di controllare attuatori di ogni genere, display, monitor, dispositivi di potenza, che trasformano la tensione in forza meccanica (motori).
- una **memoria EEPROM** (Electrically Erasable Programmable Read Only Memory, è una memoria di sola lettura cancellabile e riscrivibile elettricamente), solitamente chiamata "memoria programma", nella quale è caricato il programma da eseguire, che rimane registrato anche quando vien tolta l'alimentazione.
- una **CPU**, definita come l'unità principale di calcolo, con relativo clock, in grado di interpretare ed eseguire scrupolosamente il programma contenuto nella memoria; in base a questo programma, la CPU elabora i segnali d'ingresso e controlla le uscite.
- una **memoria RAM**, chiamata "memoria dati", nella quale la CPU scrive e legge le variabili, necessarie per contenere i dati presenti sugli ingressi e sulle uscite, i risultati delle operazioni ed altre informazioni; i dati presenti nella RAM vengono persi quando viene a mancare l'alimentazione.

Da quanto detto, è evidente quale sia la differenza tra un microprocessore e un microcontrollore:

- il processore contiene esclusivamente l'unità centrale di calcolo (CPU) e quindi, per poter funzionare, necessita, al suo esterno, di memorie ROM e RAM e di alcuni integrati per l'interfacciamento;
- nel caso del microcontrollore, tutto ciò è contenuto all'interno di un singolo chip.

Il principio di funzionamento di un microcontrollore coincide con quello di un computer e può essere riassunto in solo tre operazioni eseguite dalla CPU: leggere l'istruzione contenuta nella memoria programma, interpretarla ed eseguirla.

La programmazione dei microcontrollori può avvenire

- in linguaggio macchina
- in linguaggio assembler
- in linguaggio ad alto livello (per le ultime generazioni di microcontrollori).

Cos'è Arduino

Arduino è una serie di schede elettroniche dotate di un microcontrollore programmabile in un linguaggio molto simile al C, chiamato **Wiring** (cablaggio).

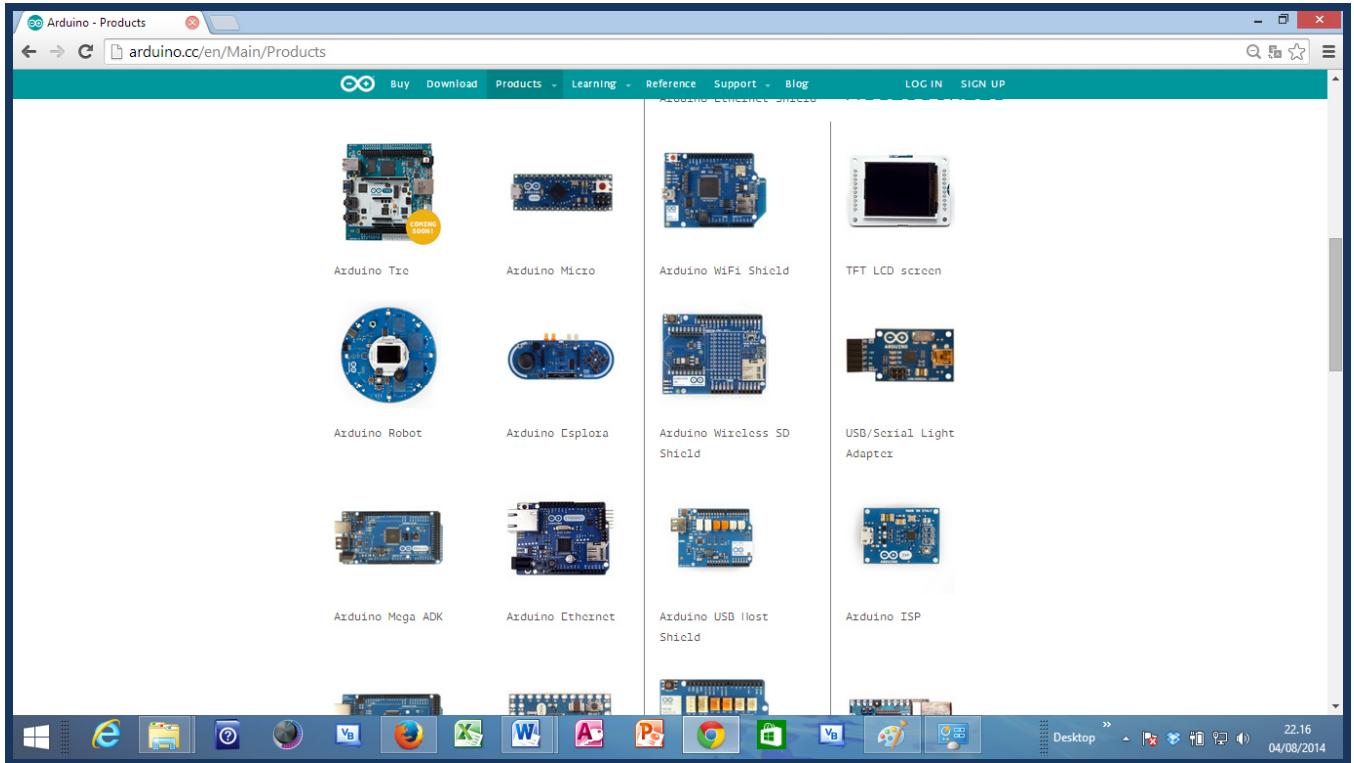
Arduino è una piattaforma open-source, nata e cresciuta in Italia, usata da milioni di utenti in tutto il mondo. Questo significa che è possibile scaricare direttamente dal sito di Arduino

<http://www.arduino.cc>

lo schema elettrico, per poi costruirsi il proprio modello.

Arduino è un sistema completamente Open Source, dal punto di vista sia hardware sia software. È il primo esempio di hardware open-source che abbia avuto successo. Il team di Arduino distribuisce sotto licenza **Creative Commons** le informazioni riguardo all'hardware e ai relativi progetti. In questo modo chiunque può utilizzare i dati e costruirsi da sè un clone, oppure modificare e innovare uno già esistente.

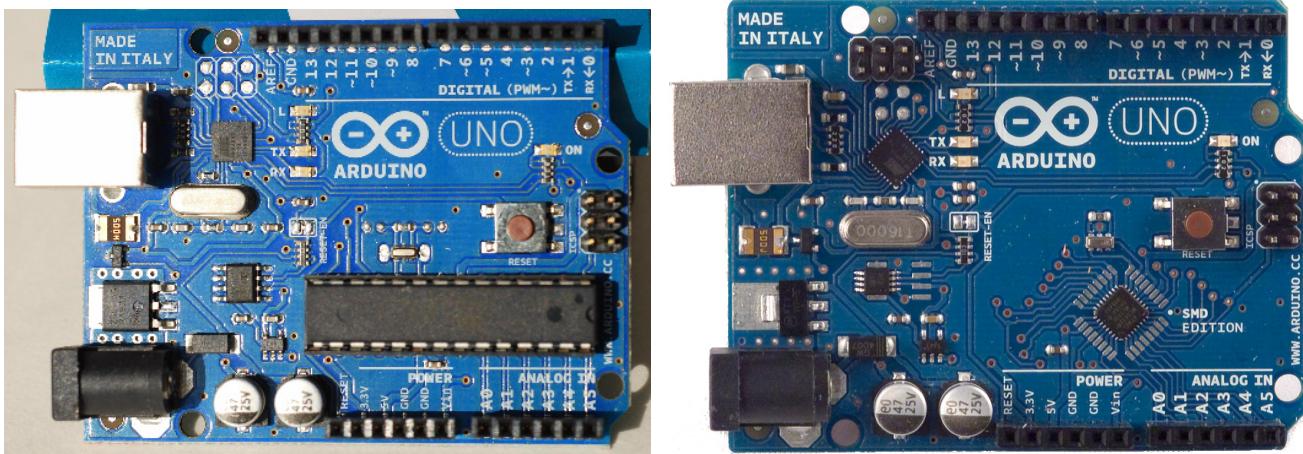
Ne deriva che di schede Arduino ve ne sono diverse, tra dispositivi ufficiali, che si possono visionare in <http://arduino.cc/en/Main/Products>, realizzate dal team Arduino, e cloni.



Il team di Arduino ha realizzato anche alcune schede dedicate a utilizzo particolari. Ne è un esempio la scheda **LilyPad**, che permette di incorporare la scheda di controllo in un tessuto, da utilizzare in t-shirt interattive.

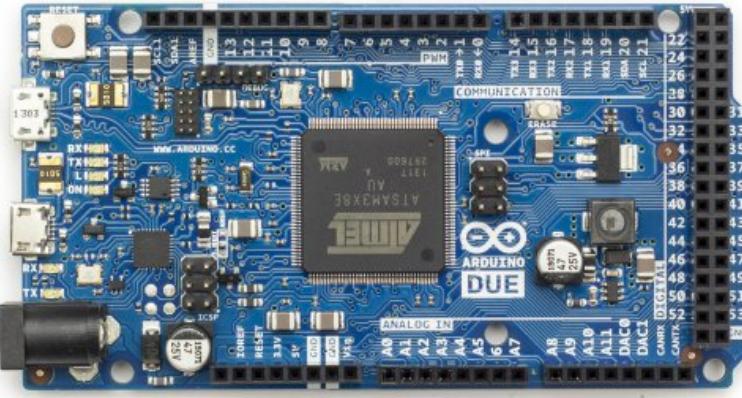
L'utilizzo e la modifica del progetto originale sono consentiti a chiunque, ma solo le schede ufficiali sono denominate con il nome Arduino, che è un marchio registrato, mentre i cloni hanno in genere un nome che termina con "duino": **Freeduino**, **Boarduino**, **Seeduino** e l'interessante **Paperduino**, un clone privo di circuito stampato e con tutti i componenti incollati su un foglio di carta.

Una delle prime e più vendute schede è **Arduino Uno**:

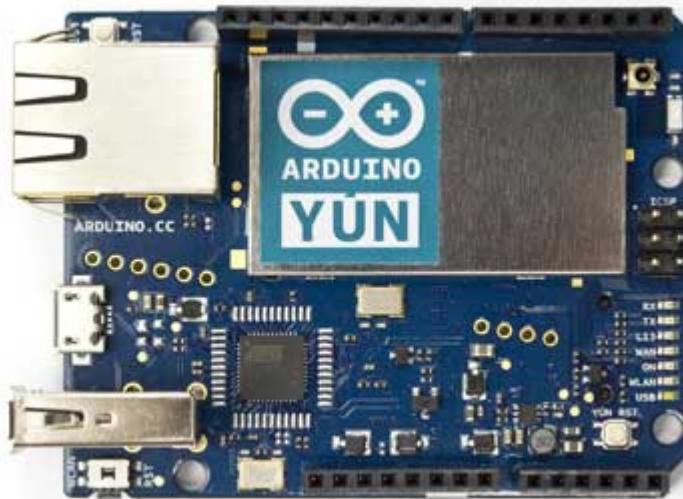


È stata formulata in due modelli diversi: la versione normale con microcontrollore **Atmega 328**, a 8 bit, montato su zoccolo, che può quindi essere estratto per essere usato in progetti che non richiedono l'uso della scheda Arduino. L'altra versione è chiamata **SMD**, perché presenta una versione miniaturizzata dell'Atmega 328; a differenza del precedente modello però questo chip è saldato direttamente sulla scheda, quindi non è possibile la sostituzione del chip in caso di malfunzionamento.

Più recente e potente, **Arduino DUE** è una scheda basata su un microcontrollore di tipo **ARM** a 32bit, il **SAM3X8E ARM Cortex-M3 di Atmel**, che migliora le funzionalità standard di Arduino e aggiunge nuove funzioni, oltre a disporre di un maggior numero di porte di I/O.



Particolare interesse sta riscontrando anche **Arduino YÚN**.



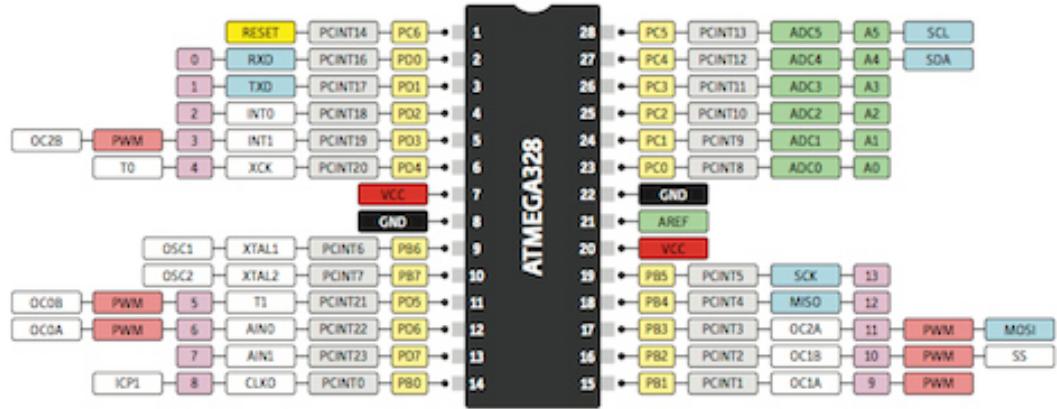
Se Arduino UNO ha consentito a migliaia di appassionati di creare facilmente applicazioni legate al mondo fisico per controllare luci, motori e altri attuatori, Arduino YÚN permette di fare interagire, con altrettanta facilità, le applicazioni con moltissimi servizi web. Non a caso Yún significa nuvola.

Arduino Uno

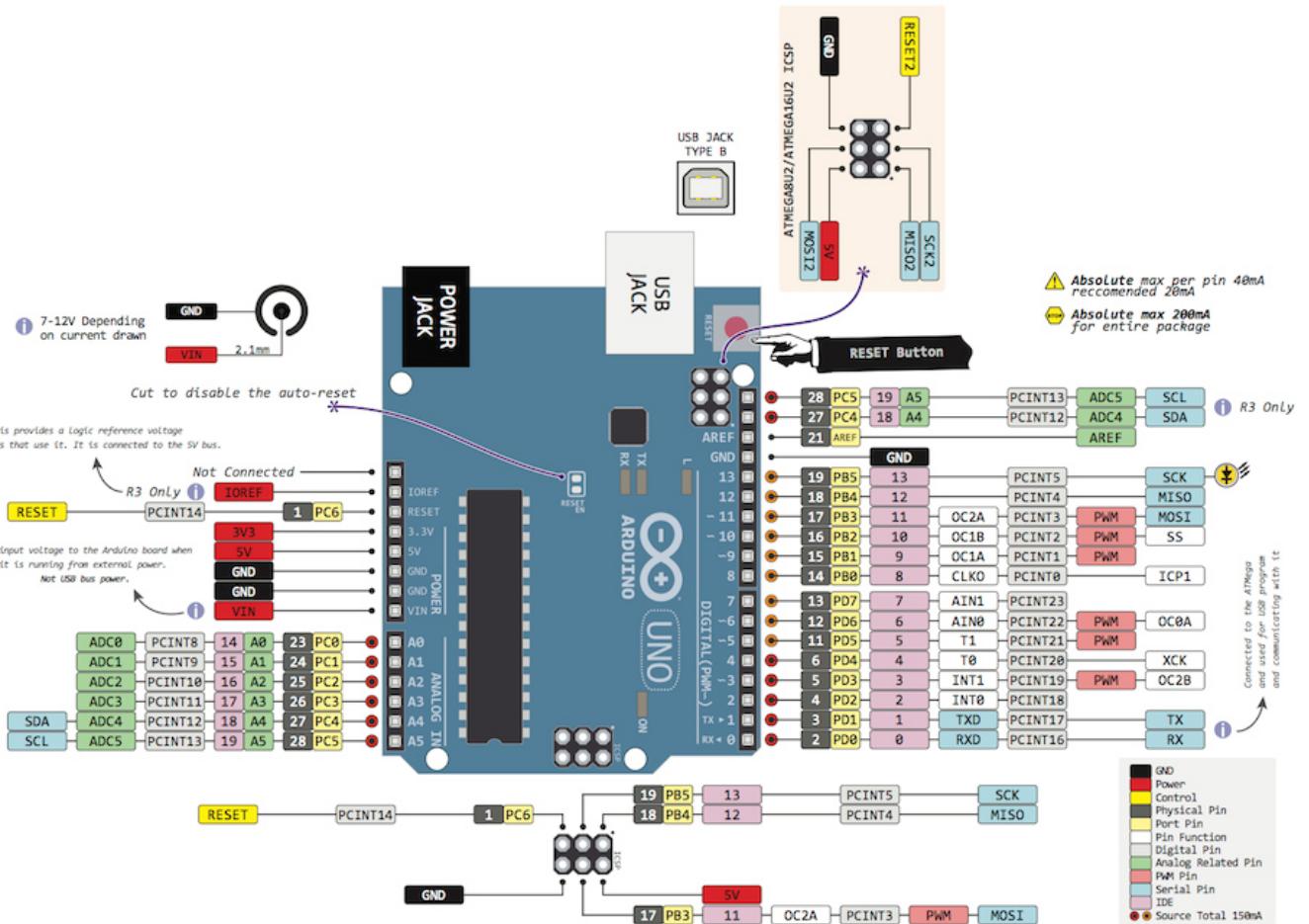
Passiamo ad analizzare la scheda (board) Arduino UNO con Atmega 328 su zoccolo.

Struttura della scheda

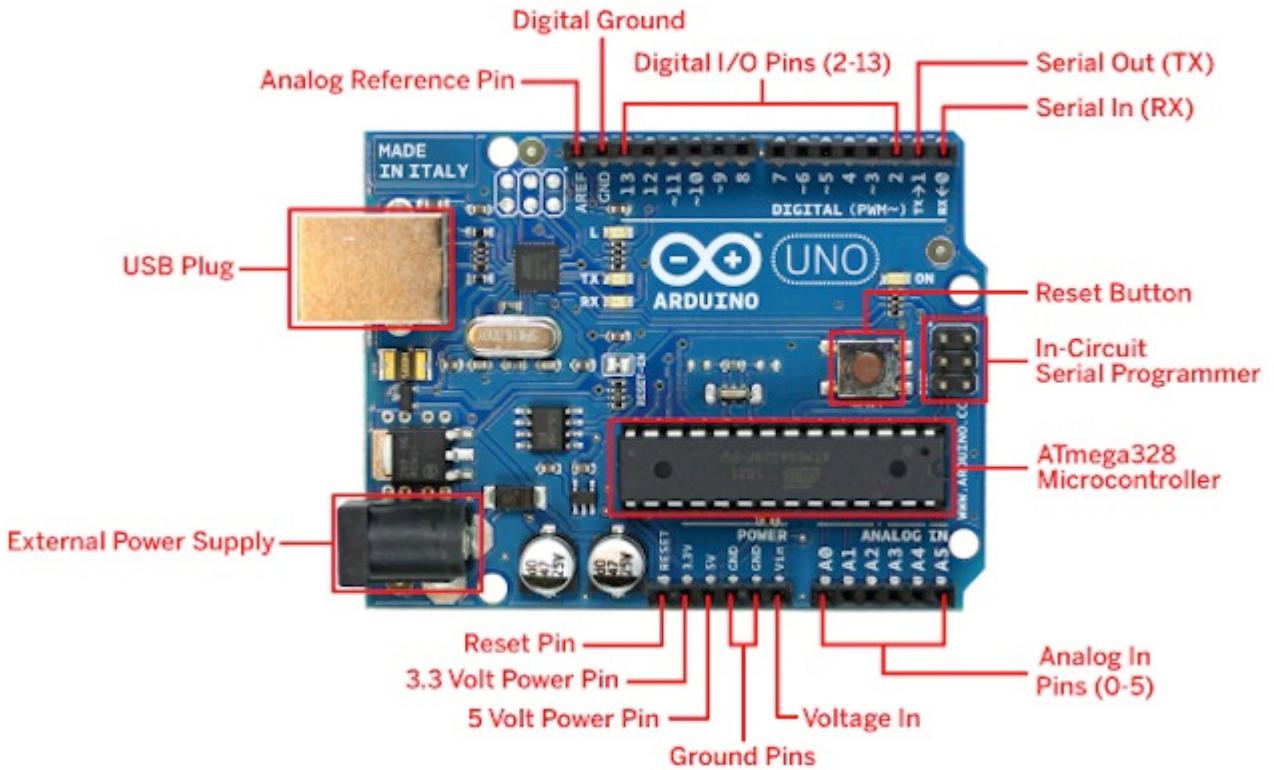
Il pin out diagram del microcontrollore è il seguente:



Il pin out diagram della scheda Arduino Uno è invece il seguente:



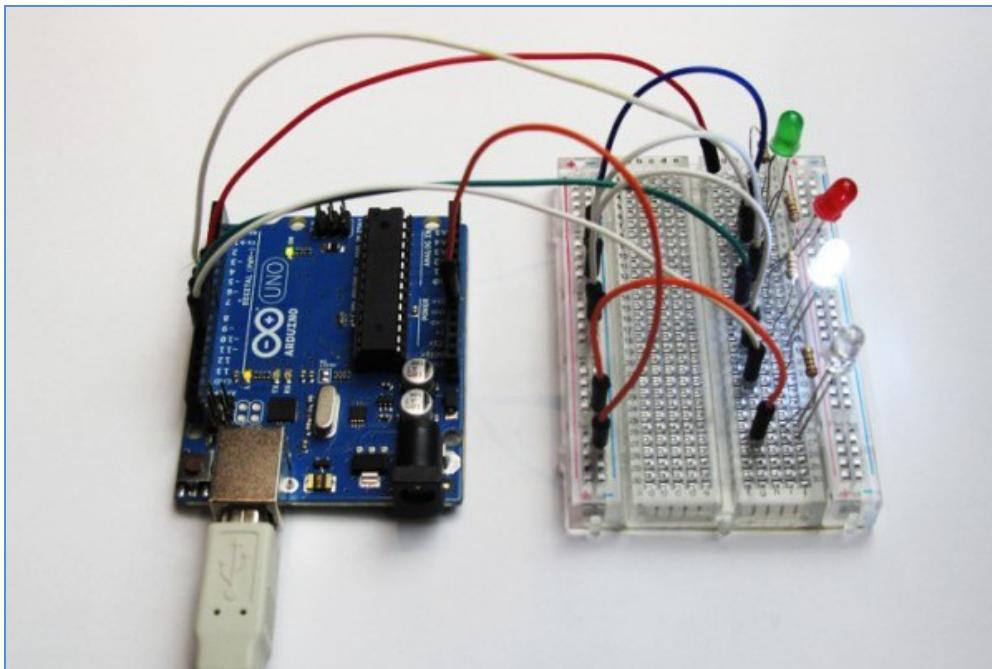
Molto più semplicemente, lo schema della scheda, con i principali contatti, è il seguente:



- **Porta USB:** è una porta di tipo B, come quelle presenti sulle stampanti. È di fondamentale importanza, perchè consente di collegare Arduino al PC in modo da trasferire il programma nella sua memoria.
- **Porte (pin) digitali:** in totale sono 14 (di cui 6 con segnale PWM, il carattere ~), numerate da 0 a 13. Le porte digitali sono usate per inviare o ricevere un segnale digitale, che può assumere solo due valori come HIGH (Alto) o LOW (Basso), altrimenti indicati con 1 e 0.
I segnali PWM (Pulse-width modulation) assumono invece valori da 0 a 255 e quindi forniscono un controllo molto più specifico dei segnali. Per esempio, se si collega un LED ad una porta digitale, lo si potrebbe solo accendere o spegnere, mentre, collegandolo alle porte PWM, si potrebbe ottenere un graduale cambiamento da acceso a spento e viceversa. Sulle porte digitali è possibile collegare componenti che assorbono al massimo 40mA. Se fossero richiesti valori superiori, si potrebbe rischiare di bruciare il microcontrollore. Per dare più corrente ai componenti, si dovrà ricorrere a un'alimentazione esterna tramite transistor, che possono sostenere una corrente maggiore.
- **Porte (pin) analogiche:** in questa scheda sono 6, contrassegnati da A0, A1, A2, A3, A4, A5. Tramite queste porte è possibile inviare/ricevere solo segnali analogici, ad esempio collegando oggetti come potenziometri, sensori crepuscolari, sensori di temperature.
- **Connettori di alimentazione:** la scheda dispone di un jack per l'alimentazione esterna, che non deve superare i 20V, ma è raccomandato immettere non più di 12V e non meno di 7V. Troviamo poi uscite di tensione 3.3V e 5V, la prima non può superare i 50mA di corrente, la seconda i 40mA. È presente anche un'entrata VIN alla quale è possibile collegare direttamente un pacco batterie tramite il cavo positivo in VIN e negativo in GND, che andranno ad alimentare la scheda. Sono infine disponibili diversi punti di terra (GND o Ground) e un pulsante di reset, da premere se dovesse bloccarsi la scheda.

Progetti con Arduino: la breadboard.

Nella realizzazione di progetti con Arduino è consigliato l'uso di una breadboard (anche detta basetta sperimentale), che rende molto più facile montare i componenti che devono interfacciarsi con il microcontrollore.

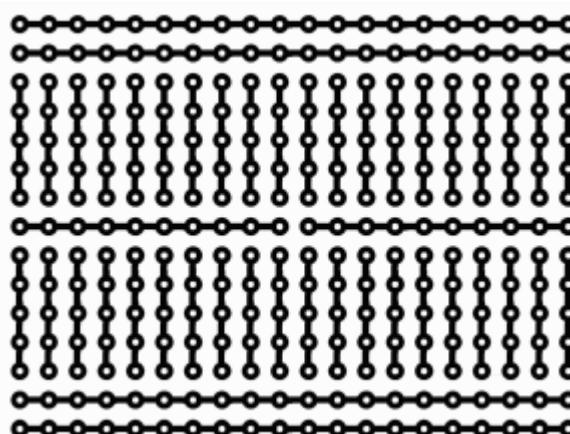


Una breadboard è uno strumento utilizzato per creare prototipi di circuiti elettrici. A differenza della basetta millefori, che è un circuito stampato (su basetta ramata) su cui è possibile saldare i componenti e i collegamenti che formano il prototipo (e che dunque non è riusabile), la breadboard non richiede saldature ed è completamente riusabile; è perciò utilizzata soprattutto per circuiti temporanei e per la prototipazione di circuiti semplici e anche molto complessi.

Una breadboard moderna consiste in una base in plastica con numerosi fori, nei quali inserire i **reofori** (i fili conduttori terminali dei componenti). Questi sono fermati meccanicamente e collegati elettricamente mediante clip metalliche presenti alla base dei fori. La distanza tra i fori è tipicamente di 2,54 mm.

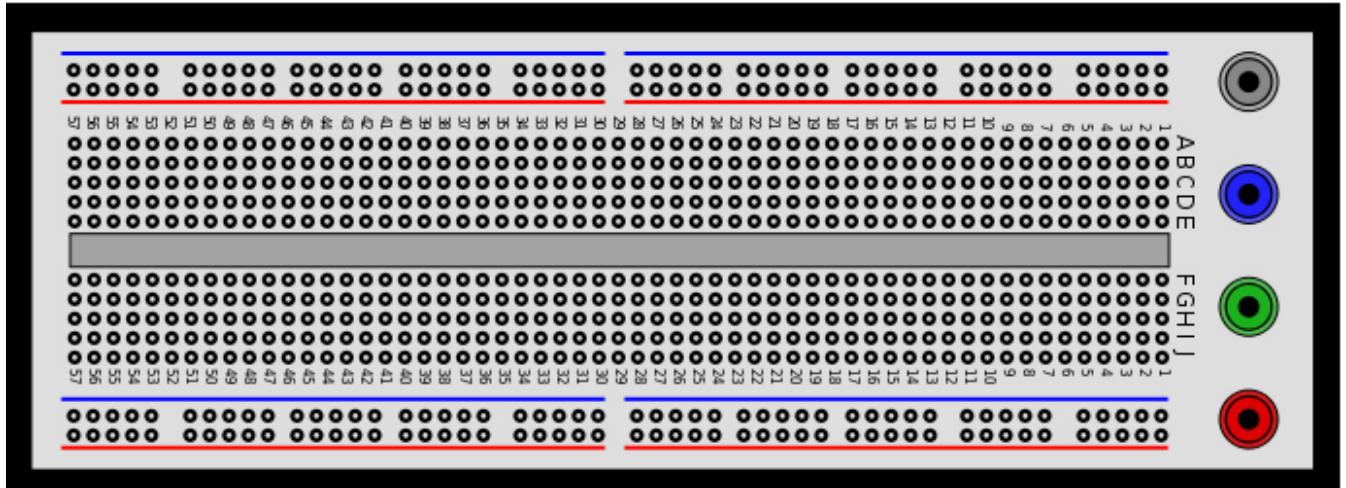
Tutte le breadboard hanno generalmente una struttura simile, composta di linee di trasmissione (strips) che consistono in collegamenti elettrici tra i fori.

La seguente immagine mostra la struttura dei collegamenti di una breadboard, da tenere ben presenti in fase di montaggio dei componenti.



Si distinguono, in genere:

- le linee di alimentazione, generalmente poste ai lati e collegate lungo tutto l'asse;
- le linee dedicate ai componenti, collegate in posizione perpendicolare alle linee d'alimentazione e più corte.



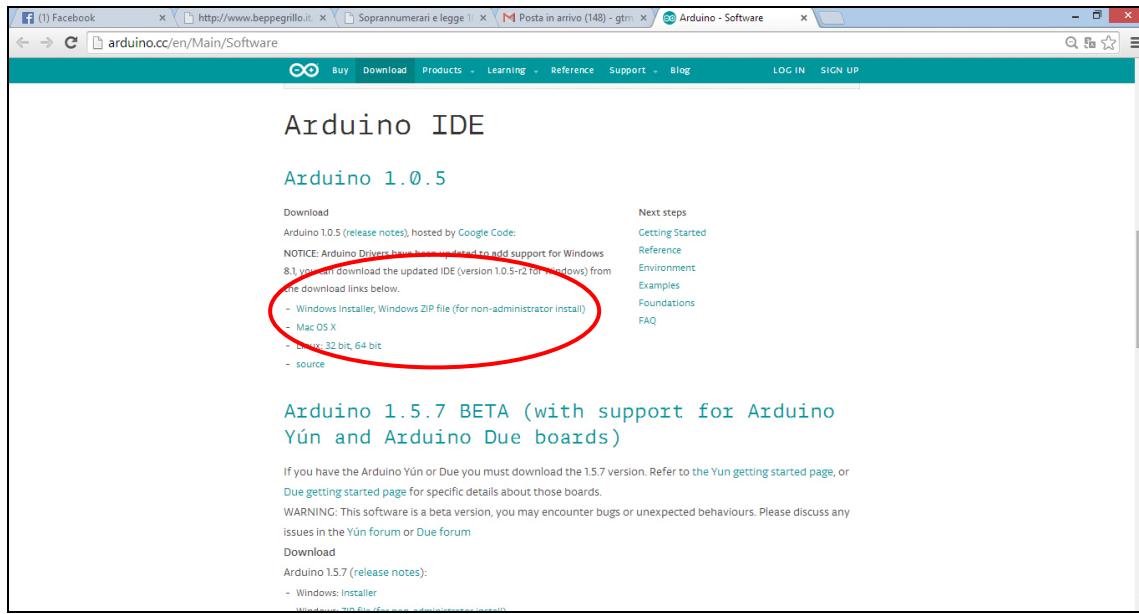
Spesso le breadboard sono componibili, cioè è possibile collegare più basette tra di loro per ampliare le linee a disposizione.

Strumenti per la programmazione

Arduino IDE

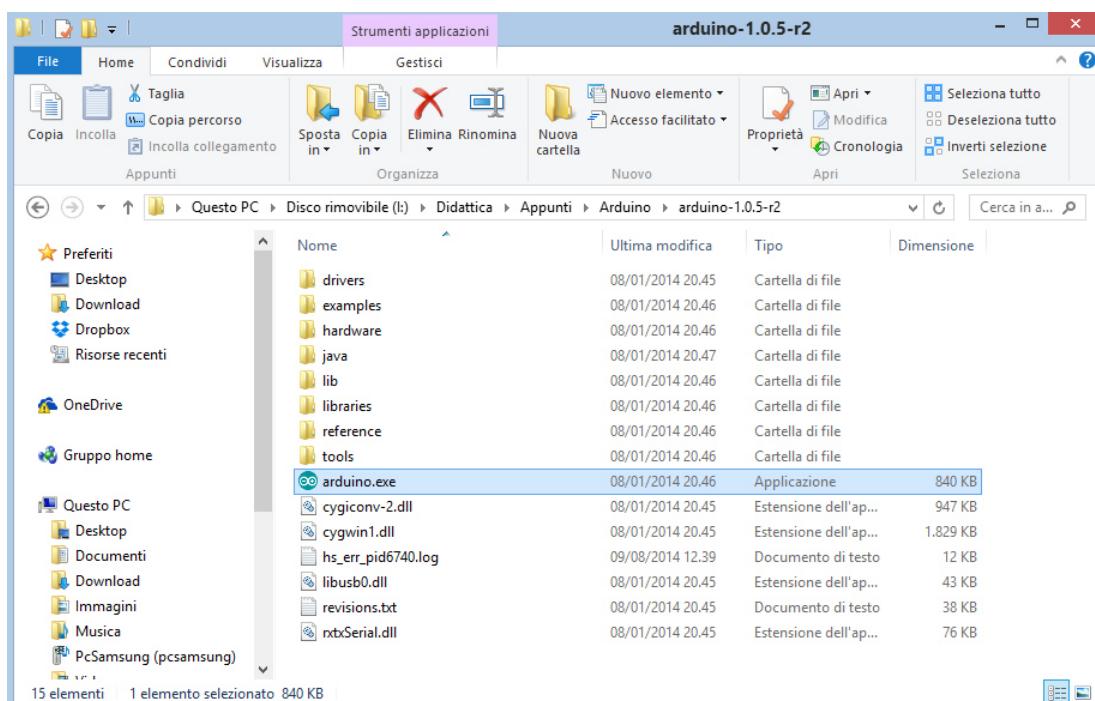
Per programmare Arduino è necessario aver scaricato nel PC il programma **Arduino IDE**, disponibile, nella versione più recente, all'indirizzo:

<http://arduino.cc/en/main/software>

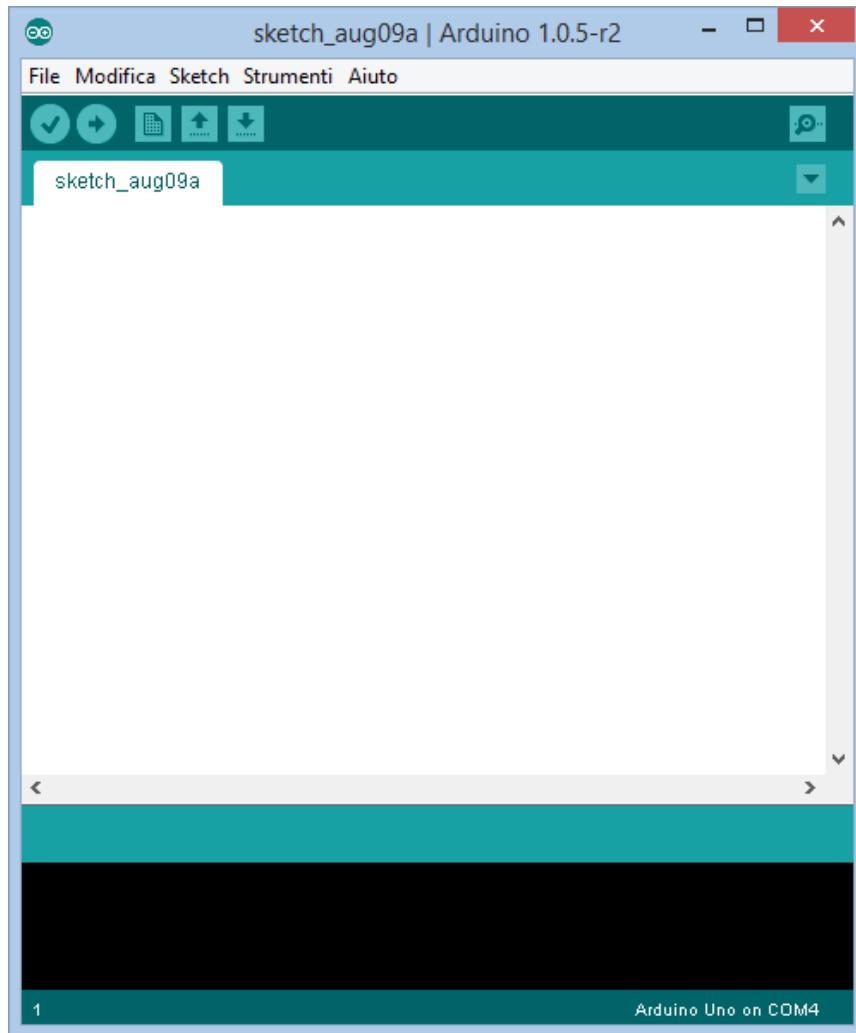


Si può scegliere, per i sistemi operativi Windows, la versione installabile o quella portable, in formato zip. Per evitare un inutile processo d'installazione, la seconda soluzione è senza dubbio la migliore; in questo modo, basterà semplicemente cancellare la cartella nel caso in cui si volesse, in seguito, aggiornare l'IDE oppure non volerlo più utilizzare.

Decompressa la cartella, si può avviare il programma cliccando due volte sul file arduino.exe:

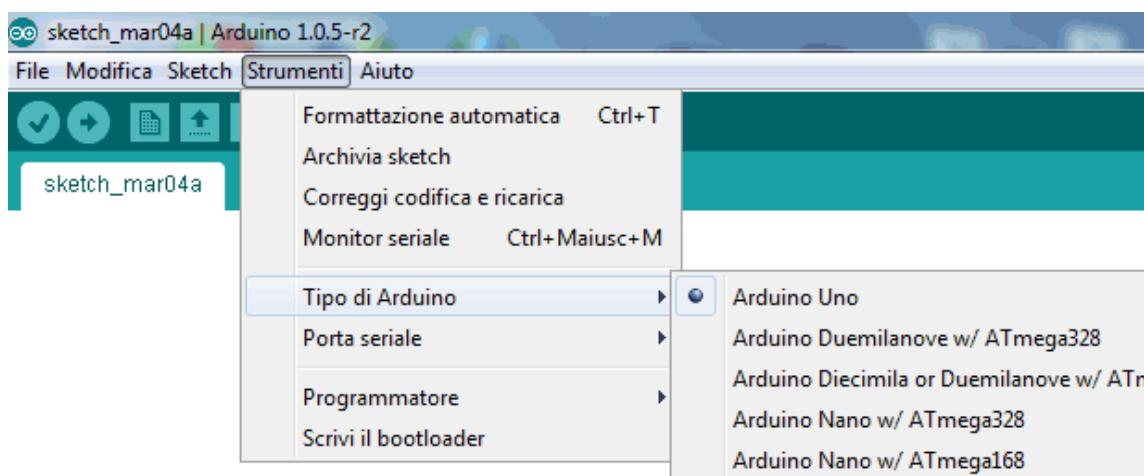


Compare la finestra dell'IDE:



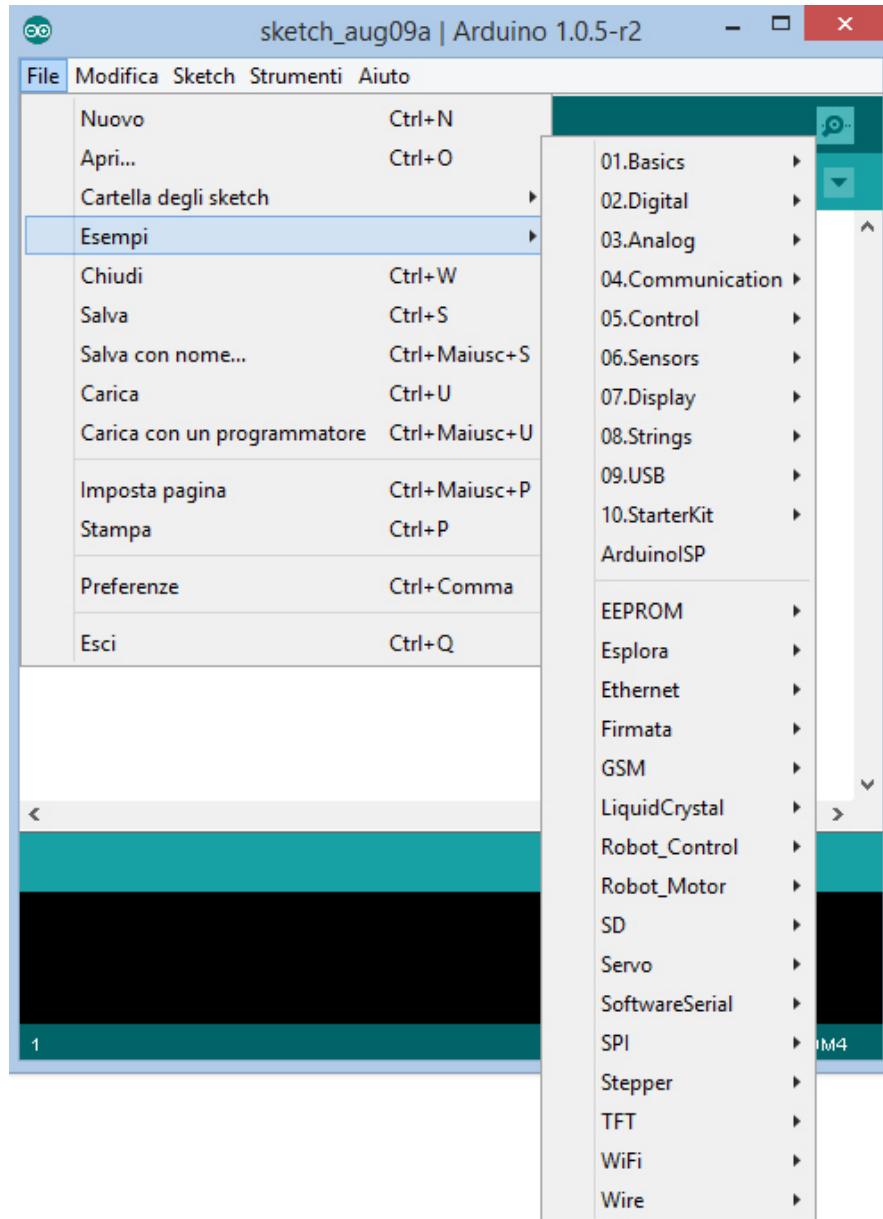
L'ambiente di sviluppo

La prima operazione da fare consiste nel selezionare quale board della famiglia Arduino sarà utilizzata (nel nostro caso "Arduino Uno"), attraverso l'opzione del menu **Strumenti - Tipo di Arduino**.



Librerie ed esempi

Nel menu **File - Esempi**, si può accedere direttamente a tutti i programmi di esempio che sono disponibili nella cartella **examples** dell'IDE e suddivisi per categoria in base alla funzionalità di Arduino che vogliamo esplorare.



Ci sono esempi sulla gestione dei pin della board, sull'uso del convertitore analogico/digitale, su tutti i tipi di connessione dalla porta seriale alla Ethernet e così via. Inoltre, attraverso il menu **Sketch - Importa libreria ...** possiamo aggiungere al nostro programma una o più delle tante librerie che il progetto Arduino mette a disposizione.

Sketch (schizzo) è il nome con cui Arduino indica i programmi.

Nel caso in cui sia stata sviluppata da terze parti (oppure da noi stessi) una libreria per la gestione di un particolare device, è possibile aggiungerla attraverso la voce **... Add Library...**, sempre in **Sketch - Importa libreria**.

L'operazione d'importazione di una libreria non fa altro che aggiungere all'interno del file sorgente tutte le direttive **#include** necessarie per l'utilizzo delle classi e delle funzioni che la libreria stessa ci mette a disposizione. Sarà l'IDE a farsi carico, al momento della compilazione, di includere, nel codice oggetto generato, quello delle librerie utilizzate.

Compilazione

Sempre nel menu **Sketch** è disponibile la voce **Verifica/Compila**, che ci permette di verificare la correttezza del codice (ovviamente da un punto di vista sintattico e dei riferimenti alle funzioni utilizzate) e quindi compilarlo, ma senza caricarlo immediatamente sulla board.

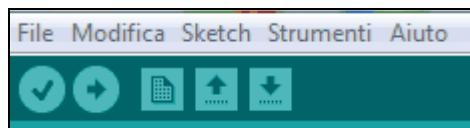
Modifica

Nel menu **Modifica** troviamo una serie di funzioni strettamente legate all'editor per eseguire le operazioni di commento del codice, di ricerca, di copia/incolla e d'indentazione.

I tasti di scelta rapida

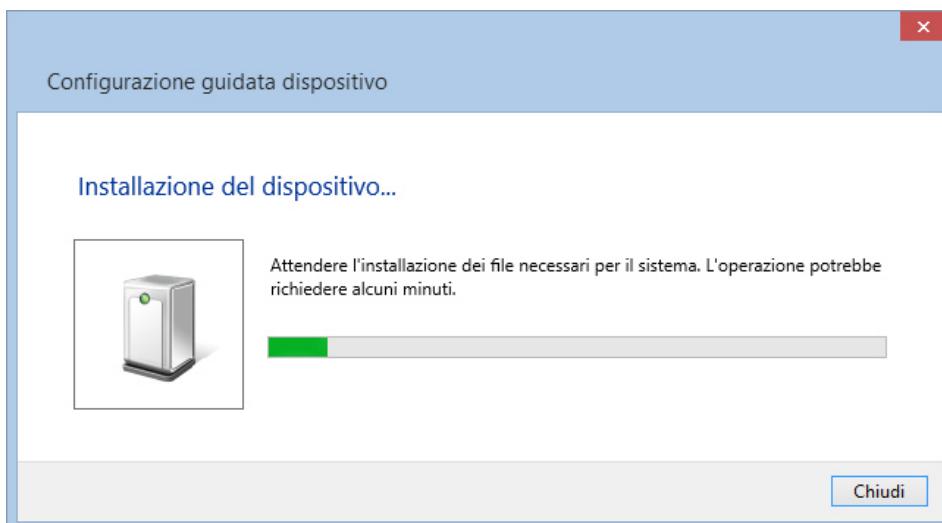
Alcune delle operazioni più comuni sono accessibili attraverso una serie di pulsanti posti immediatamente sotto la barra dei menu e che rispettivamente indicano:

- *Verifica*: esegue la verifica del codice scritto e la relativa compilazione;
- *Carica*: esegue il caricamento del firmware compilato nell'EEPROM della board;
- *Nuovo*: permette di creare un nuovo sketch;
- *Apri*: permette di aprire uno sketch esistente;
- *Salva*: permette di salvare lo sketch correntemente aperto.



Collegare la scheda

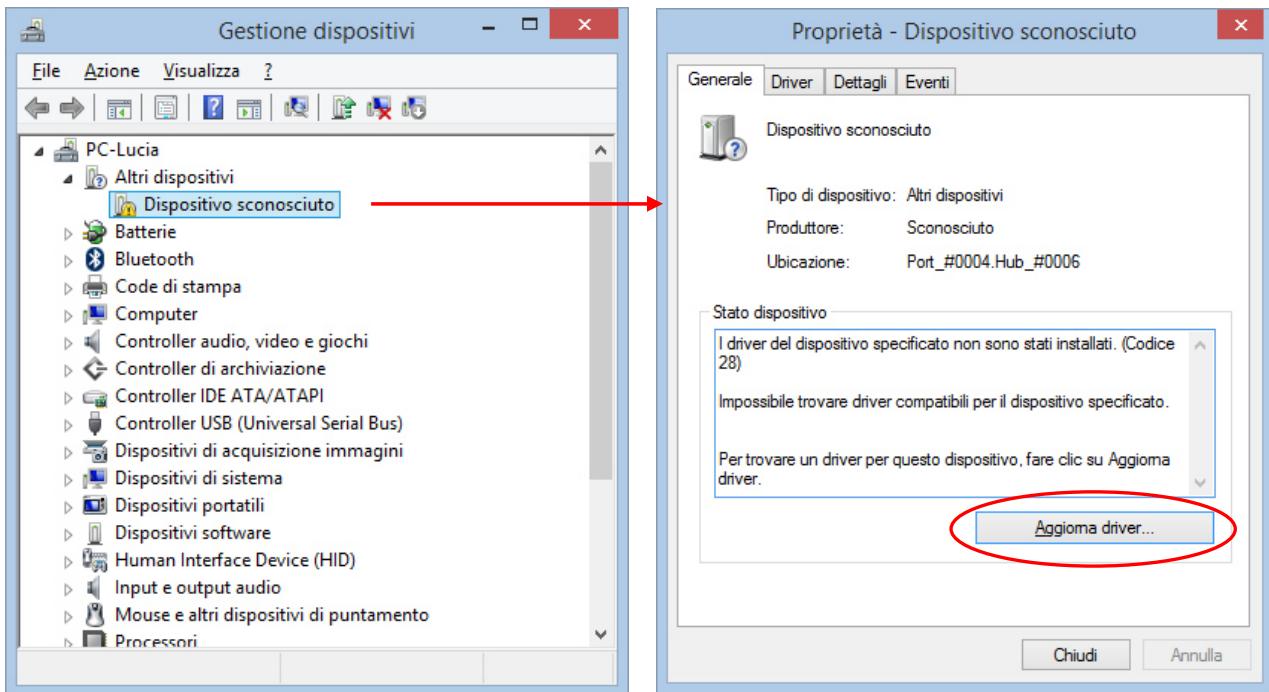
Una volta collegata e alimentata la scheda (anche solo attraverso il cavo USB), il PC dovrebbe riconoscere la sua presenza, ma ha bisogno dei driver per una corretta installazione:



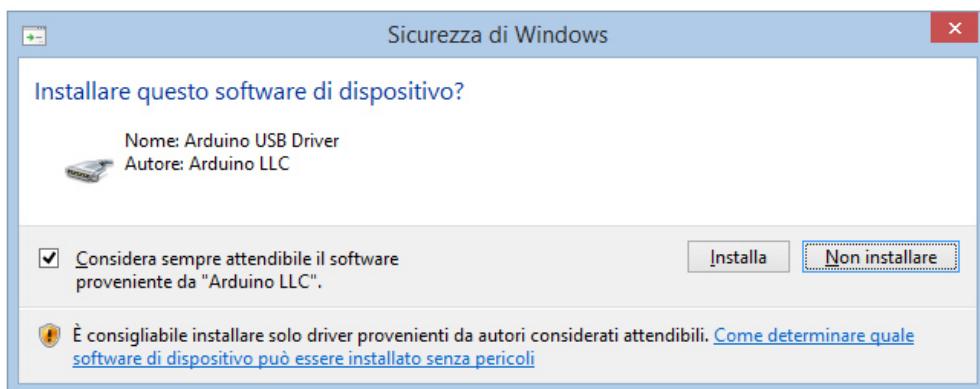
Se il programma d'installazione non dovesse riuscire a installare correttamente il dispositivo perché non ne individua i drivers, è possibile trovarli all'interno della cartella **drivers** della cartella contenente l'IDE che abbiamo scaricato; i due file eseguibili **dpinst-x86.exe** e **dpinst-amd64.exe** permettono l'installazione della scheda su un sistema operativo Windows rispettivamente a 32 e 64 bit

Basta cliccare due volte sul file compatibile con il proprio PC per avviare l'installazione dei driver. In alternativa, si può procedere nel modo seguente.

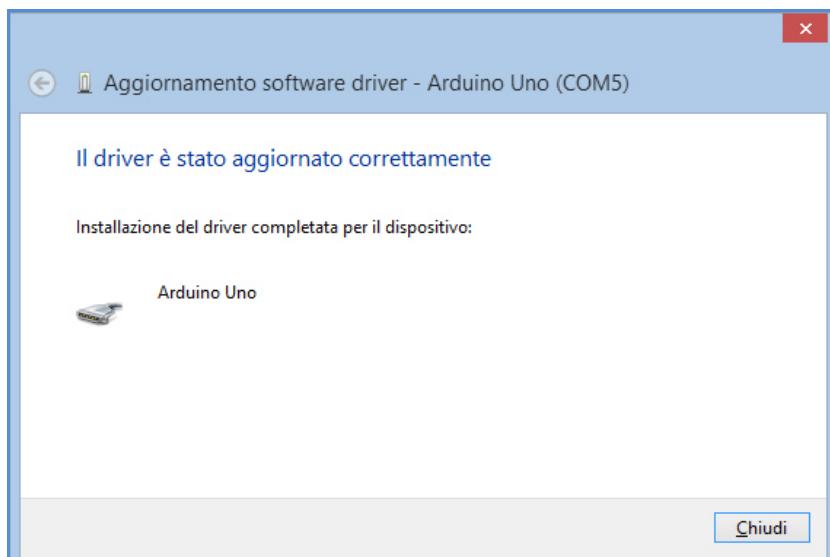
È bene innanzitutto controllare quanto riportato in **Pannello di controllo - Sistema - Gestione dispositivi**: Se compare la voce **Altri dispositivi - Dispositivo sconosciuto**, cliccare due volte su di essa per procedere all'installazione dei driver.



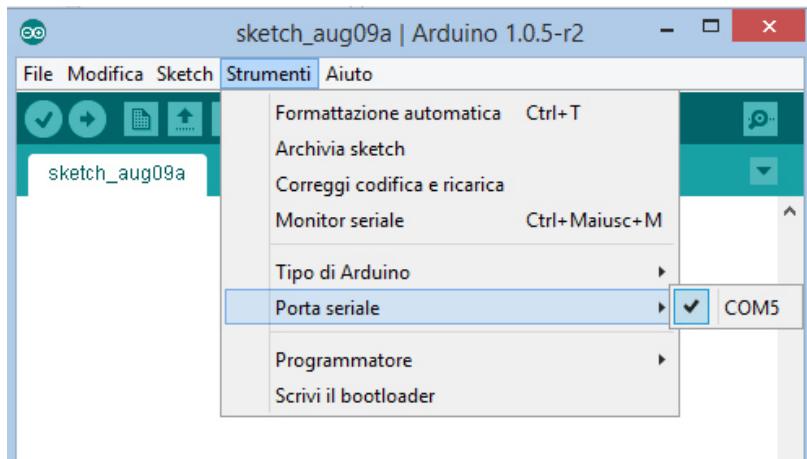
Cliccare su **Aggiorna driver**, scegliere di selezionare i driver manualmente, cercando nel computer, e selezionare la cartella **Drivers**, dove sono presenti i file da installare. Comparirà la finestra:



Vuol dire che il dispositivo è stato riconosciuto. Occorre ovviamente cliccare su **Installa** per avviare l'installazione dei drivers. Al termine, comparirà la finestra:

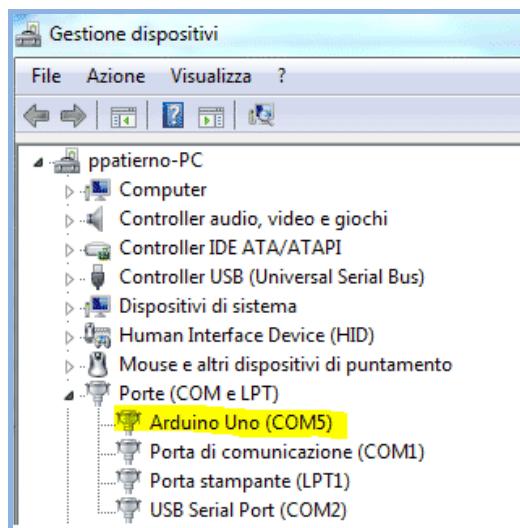


Una volta completata l'installazione, le porte seriali già disponibili sul PC più quella appena installata e relativa alla board Arduino saranno visibili nel menu **Strumenti - Porta seriale** dell'IDE.

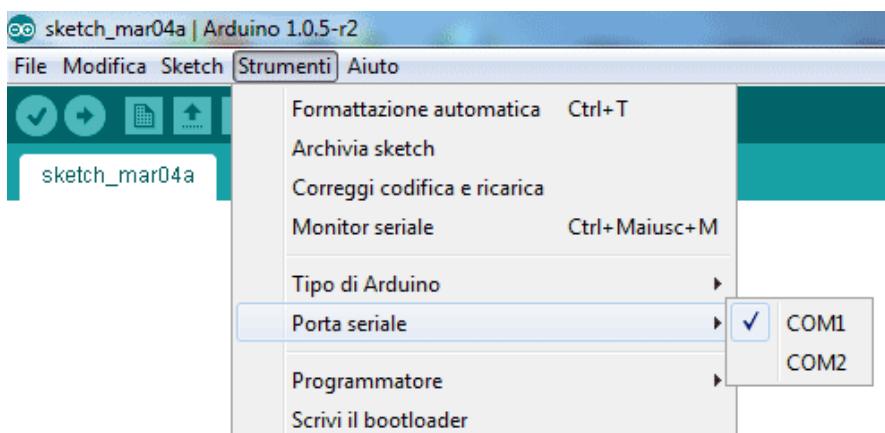


Selezioniamo la porta alla quale è collegata la scheda.

Se compare più di una porta, possiamo individuare la porta seriale giusta in **Pannello di controllo - Sistema - Gestione dispositivi**, dove troviamo, alla voce **Porte (Com e Lpt)**, una porta del tipo *Arduino Uno (COMx)*, che è una porta seriale virtuale associata al convertitore USB-seriale a bordo della scheda.



Adesso possiamo impostarla nell'IDE.



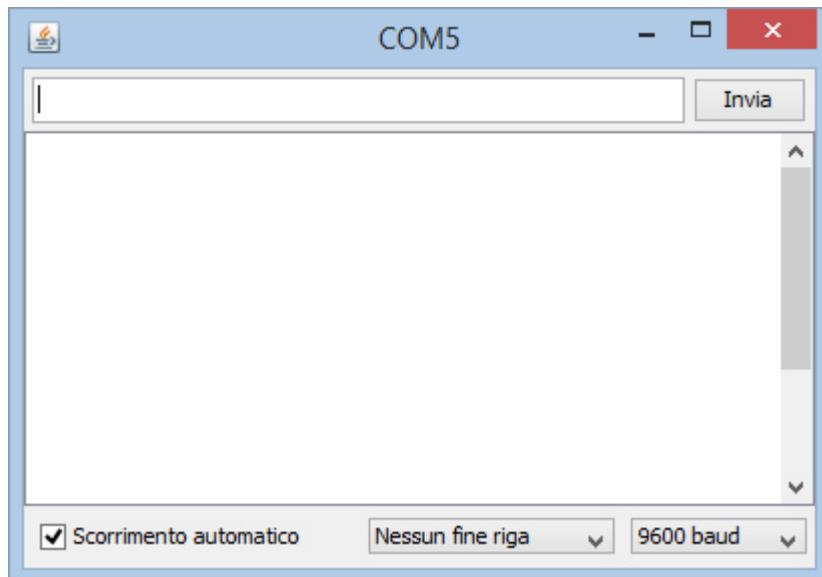
Il debug a runtime

Un'utilissima funzionalità dell'IDE è il **monitor della porta seriale** disponibile nella voce di menu **Strumenti - Monitor seriale**, molto importante per **il debug dei programmi a runtime** (runtime = fase di esecuzione).

Per limiti dovuti in primo luogo al tipo di microcontrollore, non abbiamo a disposizione le funzioni di debugging che utilizziamo di solito per altri tipi di applicazione. Quindi non si dispone di breakpoints, esecuzione step by step e visualizzazione in tempo reale del valore delle variabili.

Utilizzando però la libreria di gestione della seriale, c'è la possibilità di inviare messaggi dal microcontrollore e visualizzarli attraverso il monitor; solo in questo modo si potrà sapere, durante l'esecuzione di un programma, in quale punto ci troviamo oppure qual è il valore assunto da una o più variabili.

Anche se può apparire un po' rudimentale, rappresenta l'unica soluzione possibile.



Primi elementi di programmazione: lampeggio di un led

Come prima applicazione, realizziamo un programma per far lampeggiare un led con periodo pari a 2 secondi (1 secondo di accensione e 1 secondo di spegnimento).

La struttura di uno sketch: le funzioni setup e loop

Un programma Arduino è caratterizzato dalle due seguenti routine principali che sono assolutamente necessarie:

setup()	È eseguita una sola volta, quando parte lo "sketch", immediatamente dopo l'accensione (il reset della board). È tipicamente utilizzata per eseguire le inizializzazioni necessarie al programma, come ad esempio l'impostazione dei valori iniziali delle variabili, la specifica delle modalità di uso dei pin (input, output, etc.) e l'inizializzazione delle librerie utilizzate.
loop()	È la routine che contiene le istruzioni del programma, che si ripetono ciclicamente (dalla prima all'ultima, per poi ripartire dall'inizio), per tutta la fase di accensione del sistema. Loop racchiude il ciclo del programma.

Ovviamente, a queste due funzioni, che costituiscono la struttura minima di uno "sketch", possiamo aggiungerne altre per rendere il programma modulare.

I pin di Arduino e la funzione pinMode

L'obiettivo è far lampeggiare un led, ossia accenderlo e tenerlo acceso per un certo tempo (1 secondo), spegnerlo e tenerlo spento per un certo tempo (sempre 1 secondo) e ripetere ciclicamente queste azioni.

Per quanto riguarda il led, sulla scheda Arduino Uno ne è montato uno, collegato al pin numero 13 del microcontrollore, che è un pin digitale. Ogni pin digitale può assumere solo due stati:

HIGH (alto)	1	> 3.3V
LOW (basso)	0	< 3.3V

Se lo stato del pin è alto, il led si accende, se è basso si spegne.

Tutti i pin digitali di Arduino sono configurabili come input oppure come output. In questo caso il pin 13 deve essere impostato come output. Per farlo possiamo avvalerci della procedura `pinMode`, la cui sintassi è:

pinMode(pin, mode)

<i>pin</i>	È il numero del pin da impostare (nel nostro caso 13).
<i>mode</i>	È la modalità di utilizzo del pin e può assumere i valori INPUT e OUTPUT, che sono due costanti incluse nella libreria di base.

Funzione setup

Poiché l'inizializzazione di un pin è un'operazione da eseguire una sola volta nel corso di un programma, possiamo eseguirla all'interno della funzione `setup()` nel modo seguente.

```
void setup()
{
    // inizializzazione del pin in uscita
    pinMode(13, OUTPUT);
}
```

Per comprendere meglio il codice scritto ed evitare di utilizzare direttamente valori numerici nelle istruzioni (in questo caso il valore 13), è utile definire una variabile globale con un nome significativo, ad esempio "led", e assegnarle il valore 13, per poi utilizzarla come primo parametro per la funzione pinMode().

```
int led = 13;

void setup()
{
    // inizializzazione del pin in uscita
    pinMode(led, OUTPUT);
}
```

All'avvio della board, il led risulta inizialmente spento, per cui la prima operazione da eseguire è la sua accensione. Accendere il led significa impostare il valore "alto" sul pin corrispondente; viceversa, occorre impostare il valore "basso" per spegnerlo; in pratica è necessaria un'istruzione che permetta di modificare il valore di uscita del pin (e quindi la corrispondente tensione).

Invio di valori ai pin: digitalWrite

La funzione a disposizione è **digitalWrite(pin, value)** che prevede in ingresso i seguenti parametri:

<i>pin</i>	il numero del pin di cui modificare il valore (nel nostro caso 13, tramite la variabile led).
<i>value</i>	il valore da assegnare al pin che può essere HIGH o LOW che rappresentano due costanti corrispondenti rispettivamente ai valori 1 e 0.

Funzione loop

Detto ciò, quindi, la funzione loop() inizia a prendere forma nel modo seguente :

```
void loop()
{
    digitalWrite(led, HIGH); // accende il led
    digitalWrite(led, LOW); // spegne il led
}
```

Le due istruzioni sono però eseguite in sequenza alla velocità del microcontrollore; per questioni di propagazione dei segnali, il risultato potrebbe non essere predicibile: potremmo avere un led sempre spento o sempre acceso.

Temporizzare le operazioni: delay

Ciò che manca è garantire che il led rimanga acceso oppure spento per un certo tempo e per fare questo è necessario utilizzare la funzione **delay()**, che garantisce un ritardo tra un'istruzione e l'altra. La sintassi è:

delay(ms)

Il suo compito è mettere in "pausa" il programma per un certo tempo (in millisecondi) specificato come parametro. In realtà, il microcontrollore non va mai in "pausa"; all'interno della funzione delay esegue ciclicamente delle istruzioni in linguaggio macchina che non producono risultato ma che hanno lo scopo di far trascorrere un numero ben definito di cicli di clock in modo da ottenere il ritardo desiderato.

Tornando al programma, l'istruzione delay (utilizzando un secondo di ritardo, cioè 1000 ms) è aggiunta immediatamente dopo ciascuna delle due istruzioni di set del pin:

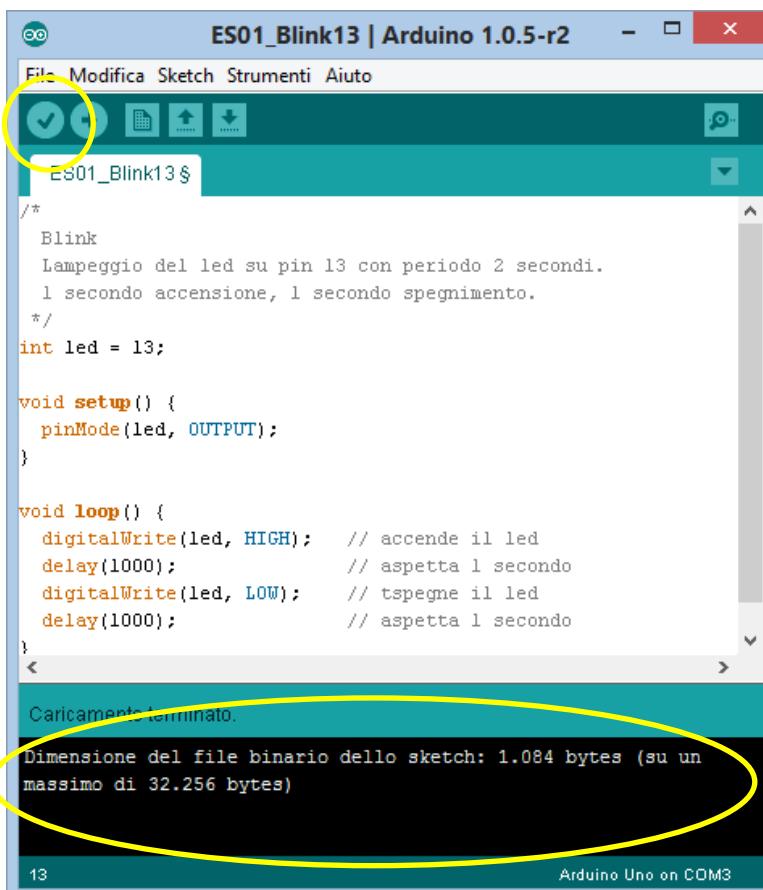
```
void loop() {
    digitalWrite(led, HIGH); // accende il led
    delay(1000);

    digitalWrite(led, LOW); // spegne il led
    delay(1000);
}
```

Il programma è ora terminato.

Scrivere l'applicazione e caricarla nella scheda

Scritta l'applicazione nella finestra dell'editor (l'area bianca) dell'IDE, si può eseguire la compilazione attraverso l'opzione **Sketch - Verifica/Compila**, oppure cliccando sul primo pulsante della barra superiore. Se non ci sono errori di battitura, compare, in basso, nell'area a sfondo nero, l'esito della compilazione. Si osservi che il programma occupa poco più di 1 KB!



Si può ora collegare la scheda Arduino Uno (accertandosi che sia correttamente riconosciuta dal sistema operativo grazie ai driver già installati), selezionare la porta seriale cui è collegata e cliccare sul pulsante **Carica** o selezionare l'opzione del menu **File - Carica**. Dopo pochi secondi, vedremo lampeggiare il led!

Lo sketch, una volta caricato, è ovviamente salvato nella memoria Flash (EEPROM) del microcontrollore, per cui possiamo tranquillamente chiudere l'IDE e riavviare la board, utilizzando eventualmente un alimentatore esterno in luogo della connessione al PC mediante il cavo USB. A ogni riavvio, lo sketch presente in memoria sarà eseguito fino allo spegnimento della board.

Nella finestra dell'IDE, lo sketch può essere salvato in un'unità a disco, selezionando **File - Salva** o **File - Salva con nome...**, come in tutte le applicazioni.

Uso della breadboard: lampeggio di un led su circuito esterno

Scopo	fare lampeggiare un LED
Componenti	LED, resistore da 220Ω

Approfondimento sui componenti



Il **LED** (light emitting diode) è un particolare tipo di diodo che emette luce quando attraversato da corrente.

Il terminale contraddistinto dalla lettera A è l'anodo e, per distinguerlo, risulta sempre più lungo del Catodo, contraddistinto dalla lettera K. Il catodo è identificato anche da una smussatura alla base del corpo cilindrico del led.

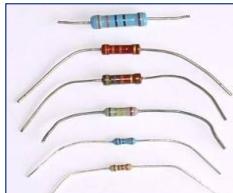
Per accendere il led è necessario collegare l'anodo al positivo e il catodo verso la massa o al negativo di alimentazione. Se accidentalmente s'invertisse il collegamento, il led rimarrebbe spento.

E' assolutamente necessario interporre in serie all'alimentazione, indifferentemente dal lato del catodo o dell'anodo, una resistenza che limiti il passaggio della corrente, in quanto il valore ottimale per il corretto funzionamento del led è compreso fra 15 e 17 milliampere (mA). Correnti superiori a 25 mA metterebbero presto il led fuori uso, mentre con correnti molto basse (ma non inferiori a 3 mA) è comunque assicurato un corretto funzionamento, ma con minore luminosità.

La semplice formula per calcolare il valore della resistenza da collegare al circuito è la seguente:

$$R = (V_{cc} - 1.5)/0.016 \text{ Ohm.}$$

V_{cc} è la tensione continua di alimentazione, indicata in Volt; 1.5 è il valore caratteristico della caduta di tensione interna al led; 0.016 rappresenta il valore medio di corrente, espresso in Ampere.



Le **resistenze elettriche**, nella loro forma piu' comune, sono caratterizzate da bande colorate che ne indicano il valore espresso in Ohm (Ω), la loro tolleranza e, qualora si tratt di resistenze di precisione, il coefficiente di temperatura espresso in parti per milione per grado Kelvin (ppm/K).

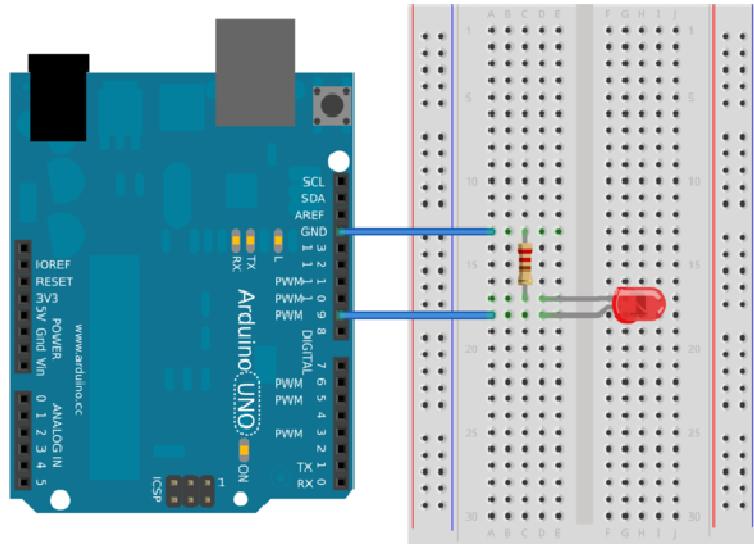
Tabella dei colori delle resistenze con 3 o 4 anelli.

colore	1° anello	2° anello	3° anello	4° anello
-----	1° cifra	2° cifra	moltiplicatore	tolleranza
nero	.	0	x1	-
marrone	1	1	x10	-
rosso	2	2	x100	-
arancione	3	3	x1000 (1K Ω)	-
giallo	4	4	x10000 (10K Ω)	-
verde	5	5	x100000 (100K Ω)	-
blu	6	6	x1000000 (1M Ω)	-
viola	7	7	x10000000 (10M Ω)	-
grigio	8	8	x100000000	-
bianco	9	9	-	5%
oro	-	-	:10	10%
argento	-	-	:100	20%

Tabella dei colori delle resistenze con 5 o 6 anelli.

colore	1° anello	2° anello	3° anello	4° anello	5° anello	6° anello
- - - - -	1° cifra	2° cifra	3° cifra	moltiplicatore	tolleranza	coef. temp.
nero	0	0	0	x1	-	200 ppm/K
marrone	1	1	1	x10	±1%	100 ppm/K
rosso	2	2	2	x100	±2%	50 ppm/K
arancione	3	3	3	x1000 (1KΩ)	-	15 ppm/K
giallo	4	4	4	x10000 (10KΩ)	-	25 ppm/K
verde	5	5	5	x100000 (100KΩ)	±0,5%	-
blu	6	6	6	x1000000 (1MΩ)	±0,25%	10 ppm/K
viola	7	7	7	x10000000(10MΩ)	±0,1%	5 ppm/K
grigio	8	8	8	x1000000000	±0,025%	-
bianco	9	9	9	x10000000000	-	-
oro	-	-	-	:10	±5%	-
argento	-	-	-	:100	±10%	-

Schema elettrico



La resistenza, per una tensione di 5V, è da 220 Ohm, essendo:

$$(5 - 1.5)/0.016 = 218,75$$

Sketch

Lo sketch è uguale a quello dell'applicazione precedente, ma si può utilizzare uno qualunque dei pin digitali (nello schema elettrico è utilizzato il pin 9).

Acquisire un input digitale: pulsante e led

Scopo	accendere un LED con la pressione di un pulsante
Componenti	LED, resistori da 100Ω , 220Ω e $10k\Omega$, pulsante NO

Approfondimento sui componenti: il pulsante

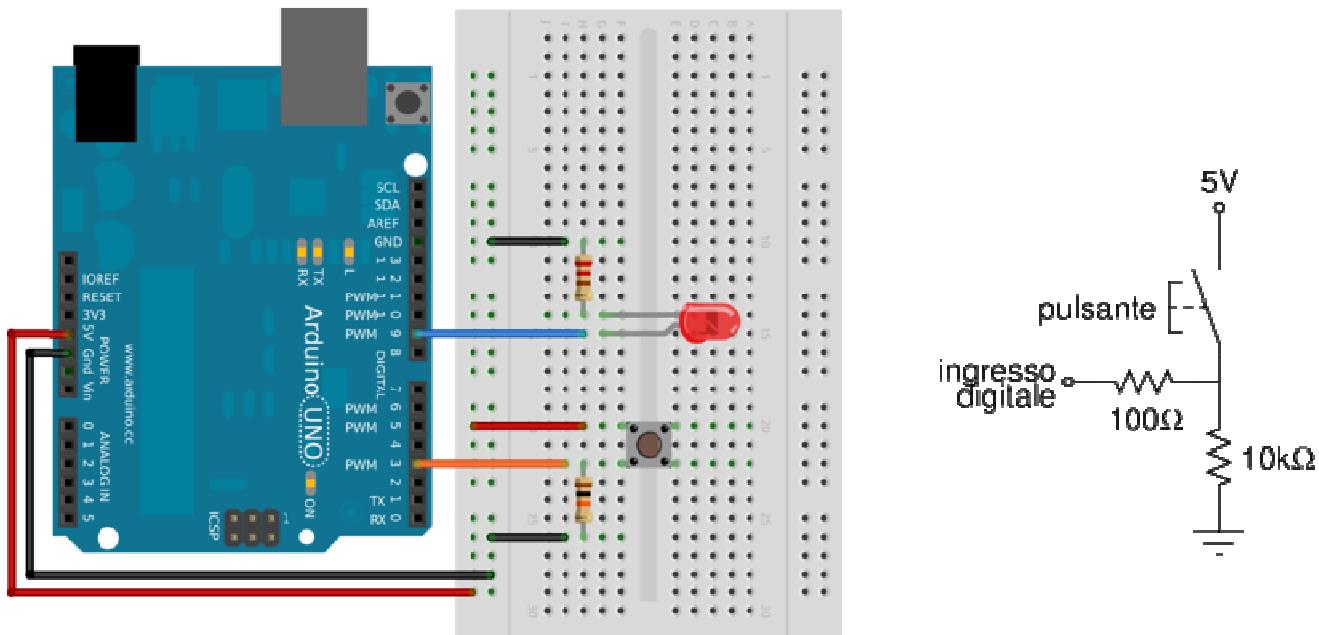
Il pulsante è in grado di aprire e chiudere un circuito come un interruttore, ma presenta una sola posizione stabile:



- il pulsante *normalmente aperto (NO)* chiude il circuito quando viene premuto ma lo riapre immediatamente quando viene rilasciato;
- il pulsante *normalmente chiuso (NC)* apre il circuito quando viene premuto e lo chiude immediatamente quando viene rilasciato.

L'apertura o chiusura automatica avviene grazie a un meccanismo a molla che riporta i contatti nella posizione predefinita. Questa soluzione meccanica può causare dei problemi perché la molla tende a far rimbalzare i contatti e il passaggio da aperto a chiuso e viceversa può non essere *netto*.

Schema elettrico



Note:

- Nello schema, il filo rosso è usato per i collegamenti con l'alimentazione (5V), quello nero è usato per la massa (GND).
- Al posto del filo arancione è bene mettere una resistenza da 100Ω per evitare la rottura del pin, se lo si imposta per errore come output (v. circuito a destra).
- La resistenza da $10k\Omega$ è detta **resistenza di pull-down** e serve a evitare che l'ingresso del pin digitale assuma un valore non valido quando il pulsante non è premuto; quando il pulsante è premuto il pin è collegato ai 5 Volt (HIGH) dell'alimentazione, quando non è premuto è collegato a massa attraverso la resistenza (LOW) come mostrato in figura

Lettura dai pin: digitalRead

In questo caso è necessario acquisire il valore sul pin di input determinato dalla pressione (HIGH) o non pressione (LOW) del pulsante.

La funzione a disposizione è **digitalRead(pin)** che prevede in ingresso solo un parametro:

<i>pin</i>	il numero del pin di cui leggere il valore.
------------	---

Sketch

```
// Accende un LED con la pressione di un pulsante

// pin utilizzati per accendere il LED e leggere lo stato del pulsante
int led = 9;
int pulsante = 3;

// impostiamo i due pin digitali come output e input
void setup() {
  pinMode(led, OUTPUT);
  pinMode(pulsante, INPUT);
}

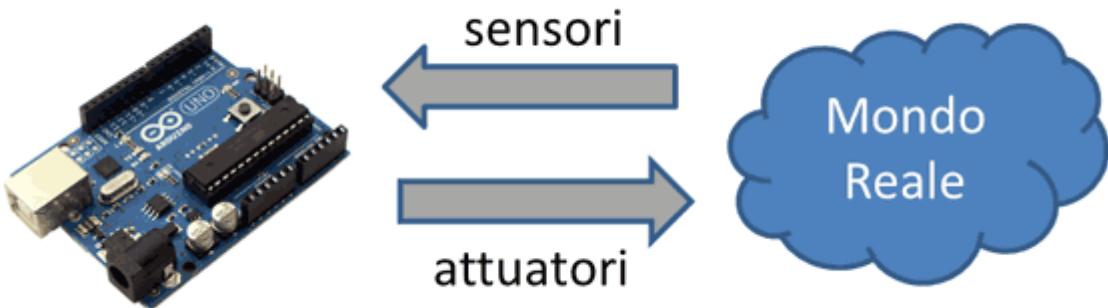
void loop() {
  if(digitalRead(pulsante) == HIGH) { // se è premuto il pulsante
    digitalWrite(led, HIGH);        // accende il LED
  }
  else {
    digitalWrite(led, LOW);        // altrimenti spegne il LED
  }
}
```

Arduino, sensori e attuatori: il Physical Computing

Cos'è il Physical Computing

Uno dei campi di maggiore applicazione di Arduino può essere assolutamente considerato il **Physical Computing**, che per definizione rappresenta la possibilità di realizzare dei sistemi, attraverso l'uso di hardware e di software, che possano interagire con il mondo esterno. Tale interazione avviene in entrambi i versi, nel senso che il sistema può acquisire e misurare grandezze reali attraverso l'utilizzo dei **sensori** e può intervenire verso l'esterno mediante l'uso di **attuatori**.

Physical Computing : interazione con il mondo reale



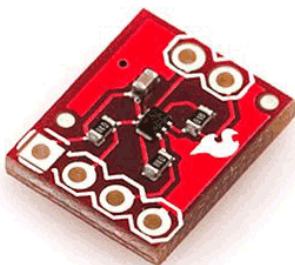
Sensori

Esistono numerosi sensori che possono essere utilizzati per acquisire **grandezze fisiche** come:

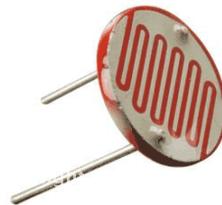
- temperatura,
- umidità,
- pressione,
- luminosità,
- concentrazione di gas nell'aria,
- velocità del vento.

Ad essi possiamo aggiungere i sensori per:

- **ultrasuoni** (Ultrasonic), attraverso i quali è possibile misurare la distanza da un oggetto;
- **infrarossi** (PIR, Passive InfraRed), che permettono di rilevare i raggi infrarossi emessi da un corpo (es. il corpo umano) e quindi utili per realizzare sistemi per la rilevazione del movimento (es. sistema di allarme);
- **forza/pressione**, mediante i quali è possibile rilevare la forza (e quindi la pressione) esercitata da un corpo.



Sensore temperatura digitale TMP102



Sensore di luminosità analogico (fotoresistore)

Un altro sensore molto noto, che ritroviamo a bordo degli smartphone, è l'**accelerometro** attraverso il quale è possibile misurare l'accelerazione dell'oggetto cui esso è collegato ed è molto spesso utilizzato per valutarne l'inclinazione (es. nel caso degli smartphone è usato per

rilevare la rotazione del display in modo che l'applicazione in esecuzione adatti la propria interfaccia grafica).

Attuatori

Grazie agli attuatori, Arduino può agire concretamente sull'ambiente, modificandone eventualmente lo stato. In tal senso, anche far lampeggiare un led può essere considerato come l'attivazione di un attuatore che altera la luminosità esterna.

Un motore è un altro esempio di attuatore, che mette in movimento un oggetto nello spazio (es. una ventola di raffreddamento) così come un **relay**, in genere utilizzato come interruttore laddove ci sia la necessità di passaggio di corrente a elevato voltaggio (es. per accendere una lampadina) da pilotare con le basse tensioni gestite da Arduino.

Acquisire, elaborare e attuare

Tipicamente, il Physical Computing prevede l'utilizzo di entrambe le categorie di oggetti, in quanto un sistema:

- **acquisisce i dati dal mondo reale** (grazie ai sensori),
- **esegue valutazioni** su di essi,
- **agisce di conseguenza** esercitando azioni sull'ambiente (grazie agli attuatori).

Le valutazioni sono caratterizzate da un'elaborazione sui dati acquisiti per prendere opportune decisioni; tale operazione di "processing" può essere:

- eseguita direttamente dal microcontrollore;
- remotizzata (es. nel Cloud).

Nel secondo caso, il sistema embedded trasmette i dati a un server, che li elabora per inviare successivamente un comando sulla base della "decisione" presa; questo tipo di approccio rientra nel mondo dell'**Internet of Things** e della *M2M (Machine To Machine) Communication*.

Sensori analogici e digitali

I sensori si possono distinguere in due categorie: analogici e digitali.

Il **sensore analogico** restituisce un **valore continuo nel tempo**, al contrario di quello **digitale** che fornisce un **valore numerico discreto**.

Tutte le grandezze fisiche sono analogiche, nel senso che hanno una variazione del proprio valore continua nel tempo (e non discreta). I computer invece per cui riescono a gestire solo ed esclusivamente valori numerici in un intervallo discreto e ben definito.

Perché Arduino, che, come tutti i computer, opera in modalità discreta su sequenze di bit, possa essere collegato a sensori analogici, deve eseguire la conversione da analogico a digitale dei segnali continui prodotti dai sensori, attraverso un campionamento.

Arduino Uno dispone di 6 canali analogici e dei corrispondenti **AD** (Analog to Digital) **converter** che permettono di ottenere un valore numerico compreso tra 0 e 1023, eseguendo il campionamento con 10 bit di risoluzione ($2^{10}=1024$).

Lo svantaggio rispetto a un sensore digitale è che, a valle della conversione, occorre utilizzare una formula che trasformi il valore compreso tra 0 e 1023 in un numero significativo per la grandezza fisica acquisita. Per esempio, se consideriamo un sensore di temperatura, un valore pari a 546 non può essere considerata una temperatura in °C ma necessita di una conversione in un valore "reale" (es. 27 °C).

Arduino: leggere i movimenti da un potenziometro

Un modo per simulare l'input da un sensore analogico e valutare il funzionamento degli AD converter di Arduino è quello di utilizzare un **potenziometro**, che è un componente che consente, attraverso una "manopola", di variare il valore di una resistenza interna e, quindi, la

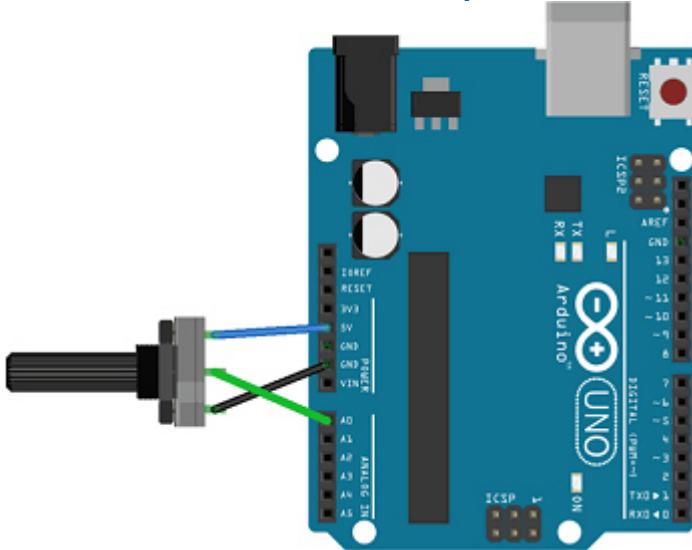
corrispondente tensione ai suoi capi. Esso è caratterizzato da tre pin, di cui due rispettivamente per la massa e la tensione di riferimento (es. 5 V) e il terzo per l'uscita, che va collegato a un ingresso analogico della scheda Arduino.

Supponiamo di voler realizzare un semplice programma attraverso il quale leggere il valore di tensione del potenziometro ed inviarlo in tempo reale al PC di sviluppo attraverso la porta seriale (utilizzando il cavo USB).



Potenziometro

Circuito con “Arduino Uno” e potenziometro



Sketch per leggere il potenziometro

La funzione di libreria che consente di leggere da unpin analogico è
analogRead(pin)

che prevede in ingresso il numero del pin da cui leggere (nell'esempio che segue, il pin 0) e restituisce un valore compreso tra 0 e 1023, in relazione alla tensione di riferimento, che di default è 5V.

Per **cambiare la tensione di riferimento**, possiamo utilizzare la funzione
analogReference(tipo)

il cui parametro rappresenta un enumerativo tra i cui valori c'è anche EXTERNAL con il quale indichiamo di voler utilizzare, come riferimento, la tensione “esterna” applicata al pin AREF. Utilizzando i 5V di default, la risoluzione con cui si riesce ad apprezzare la variazione di tensione del potenziometro è pari a 5/1024 ossia 4.9 mV.

Sviluppando un semplice sketch, possiamo acquisire il valore e inviarlo al PC collegato attraverso la porta seriale (utilizzata come strumento di debug). Per fare questo, possiamo utilizzare la classe **Serial** che fornisce il metodo **begin(velocità)** per l'inizializzazione della porta con il baudrate specificato (va eseguita nella funzione setup() dello sketch) e il metodo **println(valore)** per la scrittura su di essa (da eseguire nell'ambito del loop).

```
int potPin = 0;
int valore = 0;

void setup()
{
    Serial.begin(9600); // inizializzazione porta seriale
}

void loop()
{
    valore = analogRead(potPin); // legge valore potenziometro
    Serial.println(valore); // invia il valore al PC attraverso la seriale, come debugging
    delay(500);
}
```

Nella figura seguente possiamo osservare come i valori ottenuti in seguito alla conversione da analogico a digitale varino da 0 a 1023, al variare della posizione del potenziometro dal minimo (0V) al massimo (5V).

Valori del potenziometro acquisiti da Arduino Uno

