

Corso Base Arduino

Cos'è Arduino? E' una piattaforma hardware e software “open-source” a basso costo sviluppata nel 2005 a Ivrea da un gruppetto di quattro persone, formata da una scheda elettronica programmabile con un linguaggio di alto livello, alla quale possiamo connettere una serie di altri moduli elettronici già pronti, o sviluppare noi stessi dei circuiti, utilizzabili in molte applicazioni soprattutto in ambito di robotica e automazioni. Esempi possono essere: sensori di temperatura, umidità, pressione, luce, gas, livello, vento, ultrasuoni ma anche accelerometri, altimetri, giroscopi. Questi sono solo alcuni esempi di come possiamo acquisire informazioni e, in risposta a queste, possiamo comandare motori, pompe, luci, suoni e tanto altro....

Per poter sfruttare le possibilità offerte da queste schede, bisogna però unire la conoscenza elettronica e quella informatica.

Infatti dobbiamo essere in grado prima di assemblare un circuito elettrico e poi di programmare il nostro Arduino per farlo lavorare con il circuito creato.

Lo scopo di queste 8 lezioni, è quello di gettare le fondamenta di queste conoscenze, insegnandovi i concetti base dell'elettronica e i primi rudimenti di programmazione, per fare questo useremo il materiale presente in uno dei tanti kit di sviluppo che si possono trovare su internet a prezzi ragionevoli.

Una volta completato il corso, chi si sarà appassionato a questo mondo, potrà trovare migliaia di pagine (soprattutto in lingua inglese, ma non solo) di progetti ed esempi su internet, sia sul sito ufficiale (www.arduino.cc) che sui motori di ricerca e youtube, il tutto GRATIS, SENZA COSTI!!!.

Gli unici costi da sostenere sono quelli per l'acquisto dei componenti elettronici e di nuove schede Arduino.

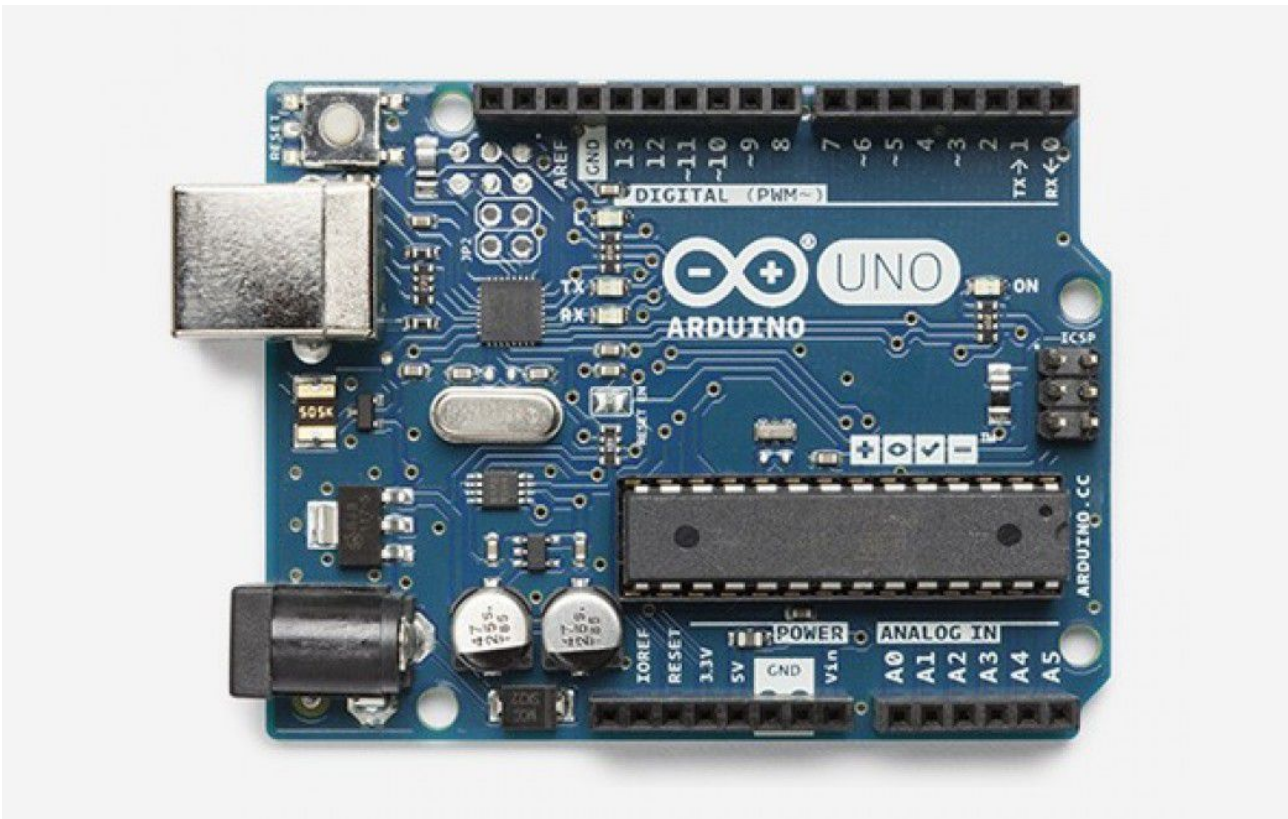
Un altro punto molto importante è che Arduino è un marchio registrato ed è il nome dell'azienda che produce (in Italia) le schede Arduino, queste schede però sono prodotte con licenza “Open-Source”. Cosa vuol dire questo? Vuol dire che chiunque può andare sul sito ufficiale e scaricarsi gli schemi, il progetto della schedina e i files necessari per costruirselo in autonomia. Questo vale ovviamente anche per le altre industrie ed infatti vi sono centinaia di cloni delle schede Arduino provenienti per lo più dai paesi asiatici. Queste però non possono ovviamente chiamarsi Arduino. Alcuni cloni sono molto interessanti perché di buona fattura e con prezzo dimezzato. Inoltre tutti i progetti realizzati con l'utilizzo di Arduino, diventano anch'essi “Open-Source”, cioè non è possibile ottenere dei brevetti o restrizioni sull'utilizzo del prodotto finito. Le schede Arduino, come altre più o meno evolute dello stesso genere, sono definite “schede prototipali”, infatti è molto frequente che in un progetto vengano utilizzate meno del 50% delle capacità effettive della scheda e quindi sarebbe impensabile sviluppare applicazioni a livello commerciale che possano essere concorrenziali come prezzo. Quindi dopo aver realizzato il prototipo se lo si ritiene conveniente si può passare alla realizzazione del prodotto finito mediante l'utilizzo di altri componenti più specifici e dal costo contenuto.

Ovviamente in ambito domestico o di sviluppo interno aziendale nessuno vieta di impiegare questi moduli per produrre apparecchiature anche sofisticate di controllo, senza dover pagare royalty a chicchessia. In questo caso, è buona pratica usare schede Arduino originali, certificate e garantite dal marchio.

Vediamo ora come è fatto Arduino UNO, questo è il nome della scheda che andremo ad utilizzare, in particolare utilizzeremo la Rev. 3 (la terza e più recente, revisione della scheda) si tratta di uno dei

modelli definiti: “entry level”, cioè a basso costo e adatto per chi vuole cominciare.

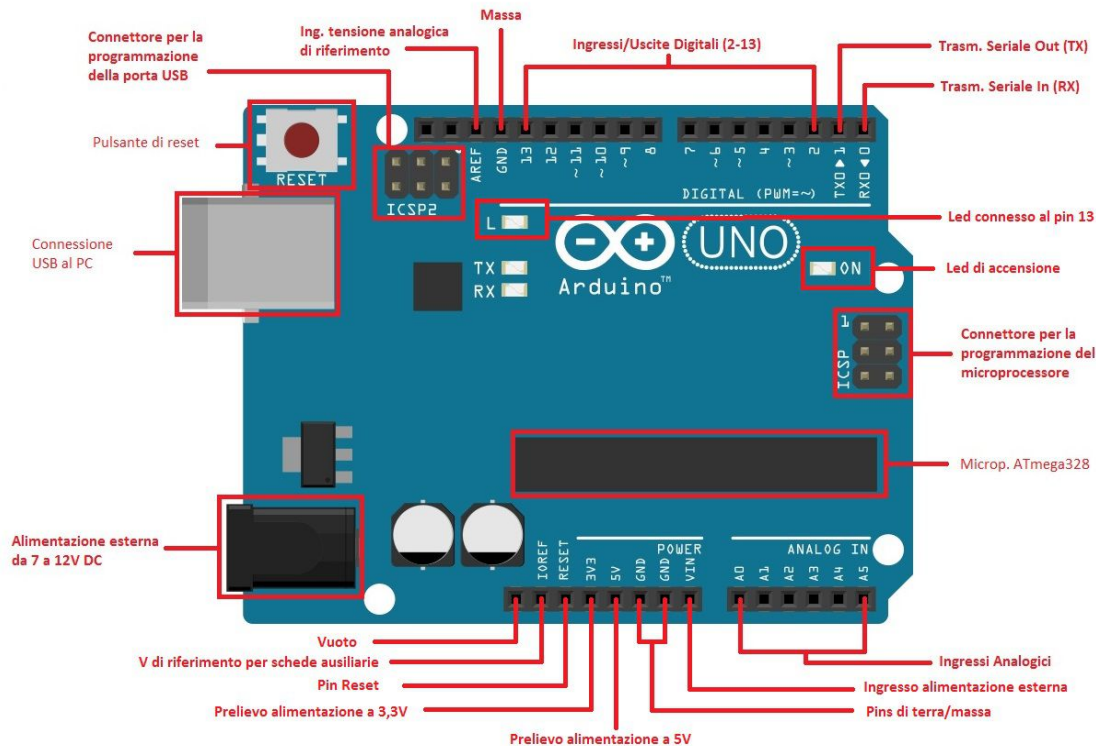
Questa è la scheda



E queste sono le sue specifiche

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Ora vediamo in dettaglio:



in senso orario dall'angolo superiore sinistro troviamo:

- Pulsante di reset: premendolo il funzionamento viene interrotto e tutte le operazioni ricominciano da capo, è come togliere corrente e ridarla. Non cancella il programma memorizzato!
- Connettore programmazione controllore porta USB: serve per poter variare o aggiornare il programma di interfaccia con la porta USB
- Ingresso tensione analogica di riferimento: normalmente viene usata la tensione di funzionamento (5V) come riferimento, ma componenti particolari possono richiedere tensioni inferiori che devono essere fornite tramite questo pin
- Pin di terra o massa
- 14 contatti digitali: di cui i primi 2 (pin0 e pin1) vengono usati per la comunicazione seriale con il pc o altre schede. Questi pin possono essere usati come ingressi (*INPUT*), per ricevere dei segnali dall'esterno, o come uscite (*OUTPUT*), per inviare segnali. L'informazione che viaggia attraverso questi pin è appunto digitale, cioè può assumere solamente 2 valori: alto e basso cioè alto quando c'è presenza di tensione (5V) e basso quando non c'è tensione (0V). Noterete che accanto ai pin 3, 5, 6, 9, 10, 11 è presente un tilde (~), questa indica che questi contatti possono gestire un uscita PWM (*Pulse Width Modulation*), più avanti vedremo a cosa serve.
- Led collegato all'uscita 13, al contatto 13 è stato collegato un led con relativa resistenza limitatrice per i primi test
- Led di accensione: si accende quando la scheda è alimentata

- Connettore per la programmazione ICSP (*in-circuit serial programming*): abbiamo la possibilità di modificare le istruzioni base di funzionamento del microcontrollore, senza doverlo rimuovere dalla scheda
- Microcontrollore, in questo caso Atmega328P della Atmel
- 6 ingressi analogici: questi pin al contrario di quelli digitali possono ricevere in ingresso un valore di tensione che può variare da 0 (zero) a 5 volt (o inferiore se fornita tramite il pin AREF), in maniera continua. Questo valore viene trasformato in un numero che varia da 0 (zero) a 1023 quindi la minima variazione rilevata sarà di $5 / 1024 =$ circa 49 millivolt
- contatto per alimentazione esterna
- 2 contatti di terra o massa
- contatto prelievo 5V per alimentare piccoli circuiti
- contatto prelievo 3,3V per alimentare piccoli circuiti
- contatto reset: portare a 5V questo contatto equivale a premere il pulsante reset
- contatto IOREF tensione di riferimento 5V: per le schede “Shield” che si possono installare su Arduino
- Presa jack per alimentazione esterna: da collegare ad un alimentatore che fornisca da 7 a 12 Volt CC (corrente continua) con polo positivo centrale
- Presa USB per la connessione al computer. Se connesso al pc tramite questa presa, non necessita di altre connessioni per alimentazione scheda

La massima corrente erogabile dalla scheda non può superare i 200mA

La massima corrente erogabile per contatto è di 40mA ma è raccomandato non superare i 20mA

La massima corrente erogabile da ciascun gruppo di contatti non deve essere superiore a 100mA.

Per quanto riguarda la parte “Hardware” per il momento questo è tutto ciò che ci interessa, vediamo ora la parte “Software”.

Il software per programmare Arduino, anch'esso gratuito è scaricabile a questi indirizzi:

https://www.arduino.cc/download_handler.php?f=/arduino-1.8.3-windows.exe

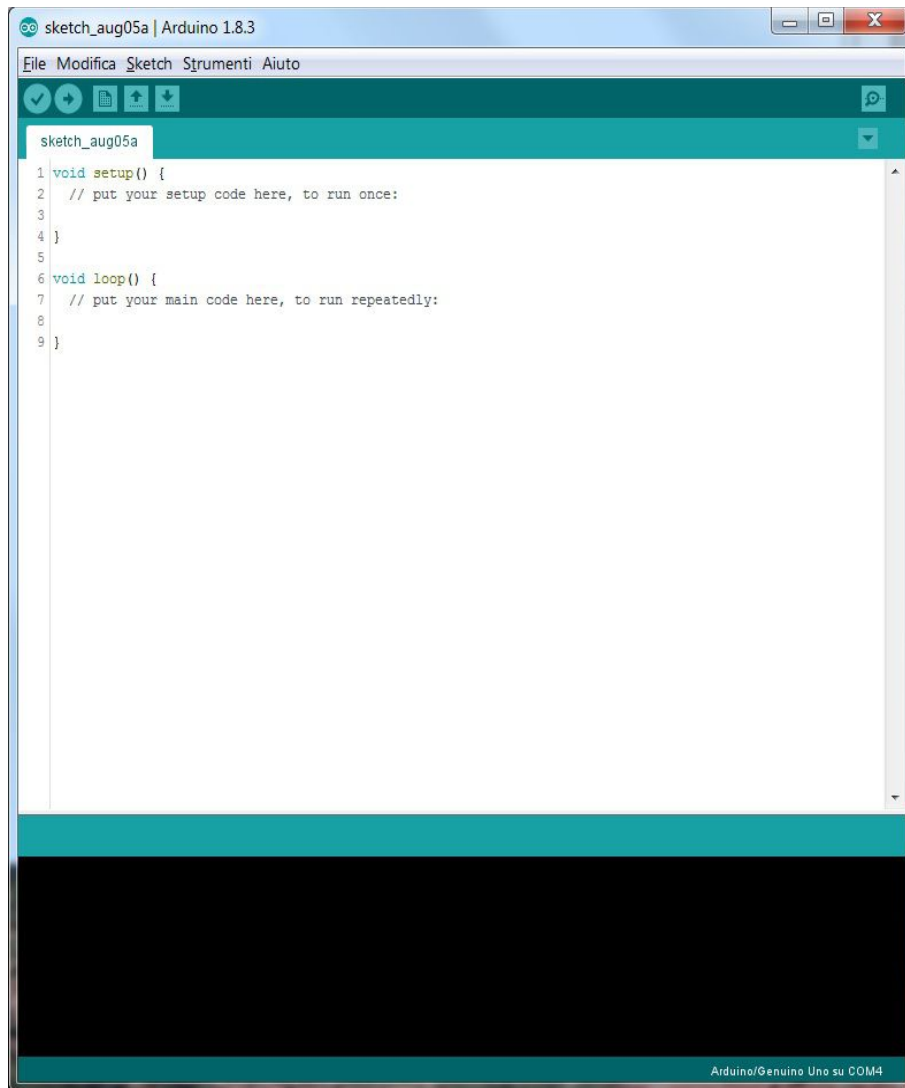
questa è l'installazione in automatico, metodo che consiglio in quanto provvederà ad aggiungere e installare anche i driver necessari al funzionamento.

https://www.arduino.cc/download_handler.php?f=/arduino-1.8.3-windows.zip

questo è l'archivio compresso, adatto a chi sa districarsi tra configurazioni e driver.

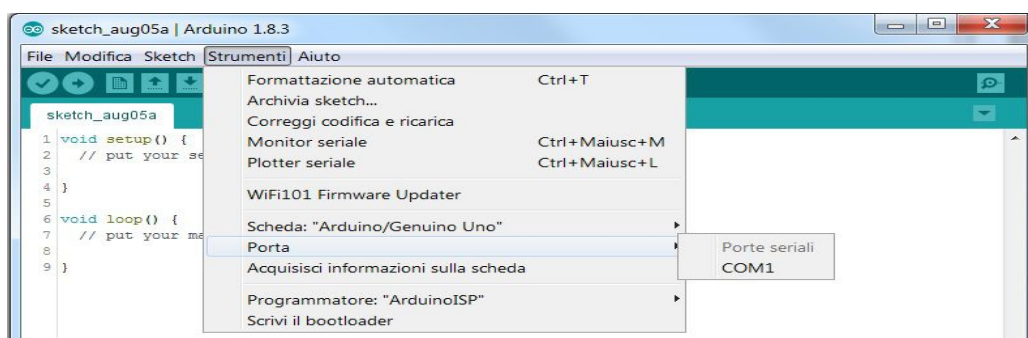
Una volta completata l'installazione sul pc, siamo pronti per iniziare.

L'IDE (integrated development environment) o ambiente di sviluppo integrato di Arduino, si presenta molto semplice e spartano, il programma è realizzato in java ed è multi-piattaforma, il codice usato deriva dal linguaggio processing e wiring, in sostanza una versione adattata/semplificata di C e C++. con un doppio click sull'icona di Arduino si apre una finestra come la seguente:

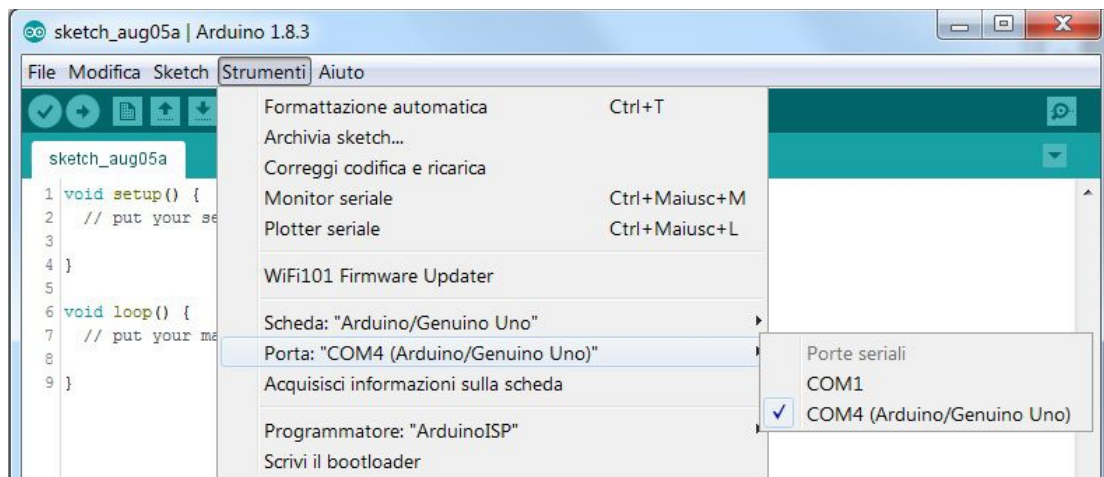


Nella riga superiore è presente un classico menù a tendina e quello che ci interessa ora è il menù strumenti, all'interno del quale troviamo la voce “porta” che una volta selezionata ci indica le porte di comunicazione disponibili sul pc, a questo punto collegando la scheda Arduino al pc con il cavo USB andremo ad utilizzare una di queste porte e vicino al nome della porta usata apparirà un segno di spunta.

Prima:



Dopo:



Se la porta non risulta selezionata, dovremo farlo noi cliccandoci sopra.

Questa è la conferma che tutti i driver necessari sono stati installati con successo.

Vediamo ora brevemente l'ambiente di sviluppo. I programmi che scriviamo in questo ambiente si chiamano "sketch" e come si vede dal nome sulla linguetta, Arduino attribuisce un nome generico al nuovo programma in questo caso sketch_aug5a dove aug5 è la data di creazione del file e la a finale indica che è il primo progetto creato in quel giorno ovviamente possiamo cambiare il nome durante il salvataggio del file.

Un ulteriore accorgimento che secondo me semplifica l'utilizzo è quello di selezionare dal menu *File* la voce *Impostazioni* e spuntare *visualizza numeri di riga* e *consenti raggruppamento codice*. Sarà più semplice trovare gli errori e verificare il programma.

Come si vede dalla figura più sopra, ogni sketch deve avere obbligatoriamente almeno due sezioni con il nome di *setup* e *loop* la prima (*setup*) conterrà le istruzioni che vengono eseguite una volta sola alla partenza del programma e la seconda (*loop*), quelle che vengono ripetute di continuo fino a che non viene tolta la corrente. Queste due funzioni sono di tipo *void* cioè non restituiscono nessun valore quando terminano e come ogni funzione iniziano e finiscono con una coppia di parentesi graffe. In ogni punto dello sketch possono essere inseriti dei commenti che servono a capire cosa sta facendo il programma a chi lo legge, tali commenti se sono su una riga sola cominciano con due barre diagonali (*//*) mentre se si vuole scrivere un testo su più righe si inizia con barra diagonale e asterisco */** e si finisce al contrario con asterisco e barra diagonale **/*.

ed ora scriviamo il primo e più semplice programma per il nostro Arduino.

All'interno della funzione *setup* scriviamo:

```
sketch_aug05a $
1 void setup() {
2   // put your setup code here, to run once:
3   pinMode(13,OUTPUT);
4 }
```

Fate attenzione alla scrittura in quanto il programma è sensibile alle maiuscole e minuscole cioè: le parole chiave che come si può notare assumono colorazioni diverse vengono riconosciute solo se scritte correttamente infatti

```
1 void setup() {
2   // put your setup code here, to run once:
3   pinMode(13,OUTPUT);
4   pinmode(13,output);
5   PinMode(13,OutPut);
6 }
```

in questo esempio si vede che ne la parola *pinmode* ne la parola *output* sono state riconosciute.

Tutte le linee di istruzioni devono poi terminare con il punto e virgola.

Con questa istruzione definiamo in che modo verrà utilizzato il pin 13 (*pinMode*) e in particolare definiamo che sarà un'uscita (*OUTPUT*) e non un ingresso (*INPUT*) le parole *input* e *output* definiscono due costanti interne al programma e per questo motivo vengono colorate in ciano.

Quindi riassumendo l'istruzione *pinMode(numero_pin, modalità_di_uso)* possiede due parametri e deve essere utilizzata per definire quale pin vogliamo usare e come vogliamo usarlo, ovviamente dovremo scriverne una per ogni riga, tante quanti sono i pin che useremo nel programma.

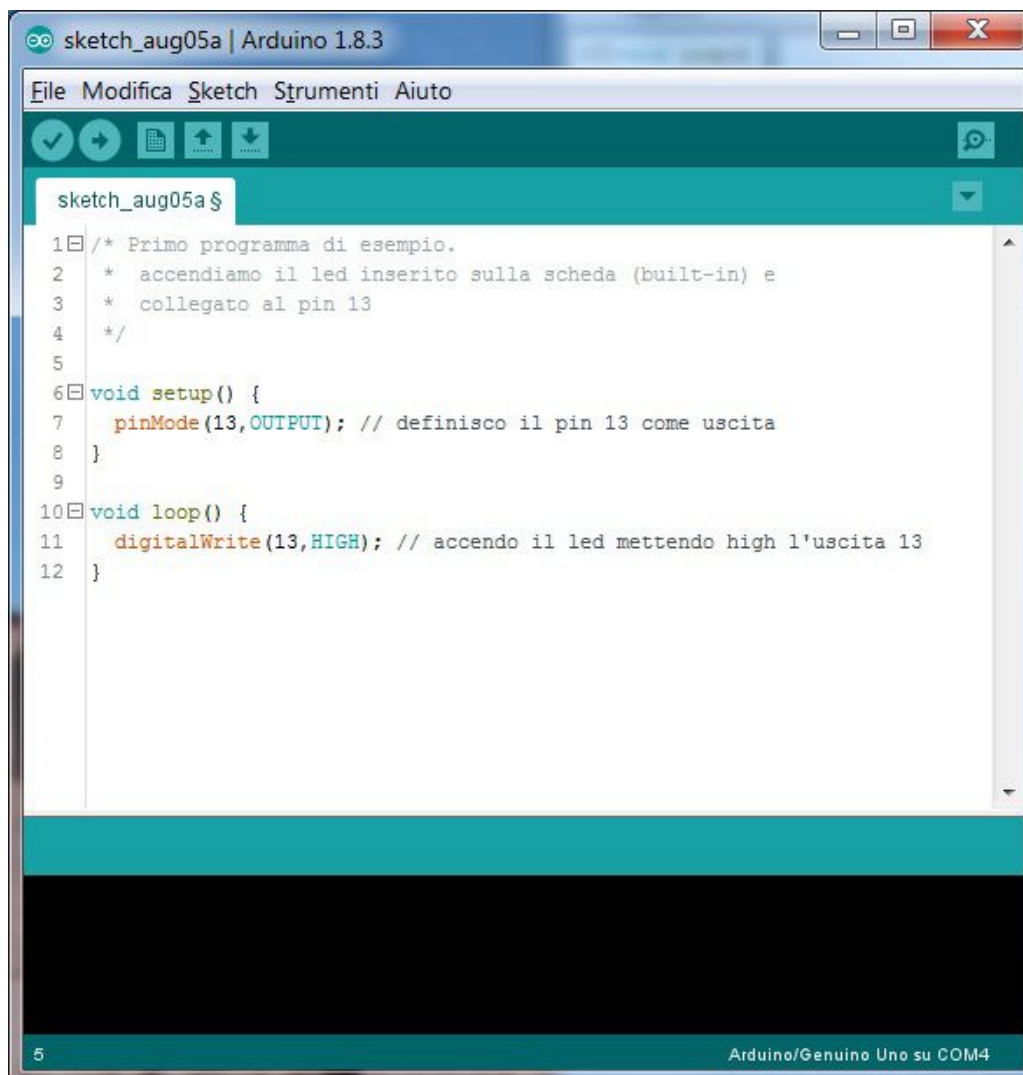
Passiamo ora alla sezione *loop* e scriviamo la seguente riga:

```
6 void loop() {
7   // put your main code here, to run repeatedly:
8   digitalWrite(13,HIGH);
9 }
```

Anche qui una sola istruzione, se ricordate il pin 13 è l'ultimo della fila di pin digitali di I/O (input/output) e più precisamente è il pin collegato al led inserito sulla scheda per facilitare appunto le prime prove.

L'istruzione *digitalWrite(numero_pin, stato_del_pin)* applicabile alle uscite digitali come dice il nome, ci consente di ordinare alla scheda di portare a livello alto (*HIGH*) cioè a +5 Volt o a livello basso (*LOW*) cioè 0 Volt il pin indicato come primo argomento.... a condizione che il pin sia stato indicato precedentemente come uscita (*OUTPUT*).

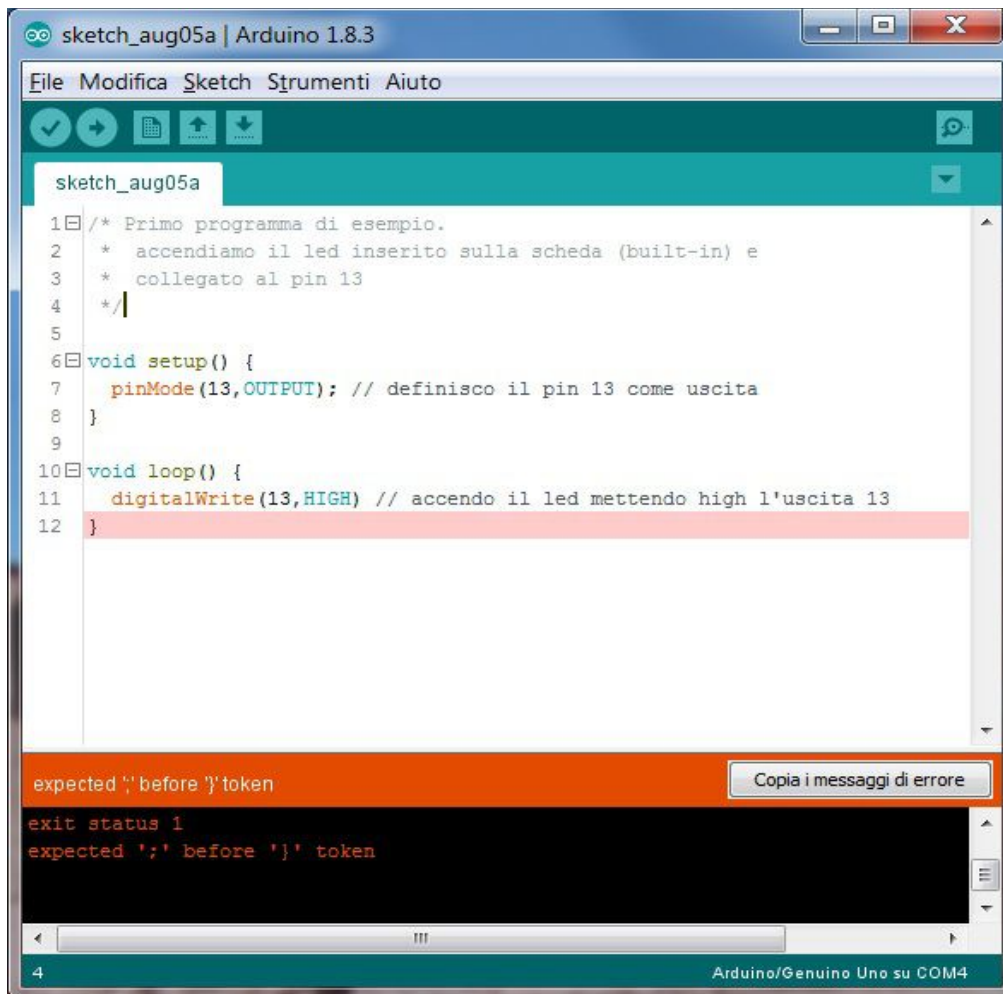
Il primo programma è terminato ed ora ecco come si presenta con alcune righe di commento



```
sketch_aug05a $
1 /* Primo programma di esempio.
2  * accendiamo il led inserito sulla scheda (built-in) e
3  * collegato al pin 13
4  */
5
6 void setup() {
7   pinMode(13,OUTPUT); // definisco il pin 13 come uscita
8 }
9
10 void loop() {
11   digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
12 }
```

5 Arduino/Genuino Uno su COM4

ora, per farlo funzionare, dobbiamo trasferirlo all'interno del microcontrollore, per fare ciò è sufficiente premere la freccia subito sotto al menu Modifica. A questo punto lo sketch verrà verificato per controllare che non vi siano errori, verrà quindi compilato in linguaggio macchina e trasferito nella memoria interna del microcontrollore e immediatamente comincerà a funzionare accendendo il led. Se al contrario troverà un errore la compilazione si bloccherà indicandoci l'errore:



clickando su “copia i messaggi d'errore” e incollandoli in un file di testo questo è il messaggio completo che si ottiene:

```

Arduino:1.8.3 (Windows 7), Scheda:"Arduino/Genuino Uno"
C:\Users\Patrick\Documents\Arduino\sketch_aug05a\sketch_aug05a.ino: In function 'void loop()':
sketch_aug05a:12: error: expected ';' before '}' token
 }
 ^
exit status 1
expected ';' before '}' token

```

Questo report potrebbe essere più ricco di informazioni abilitando l'opzione "Mostra un output dettagliato durante la compilazione" in "File -> Impostazioni"

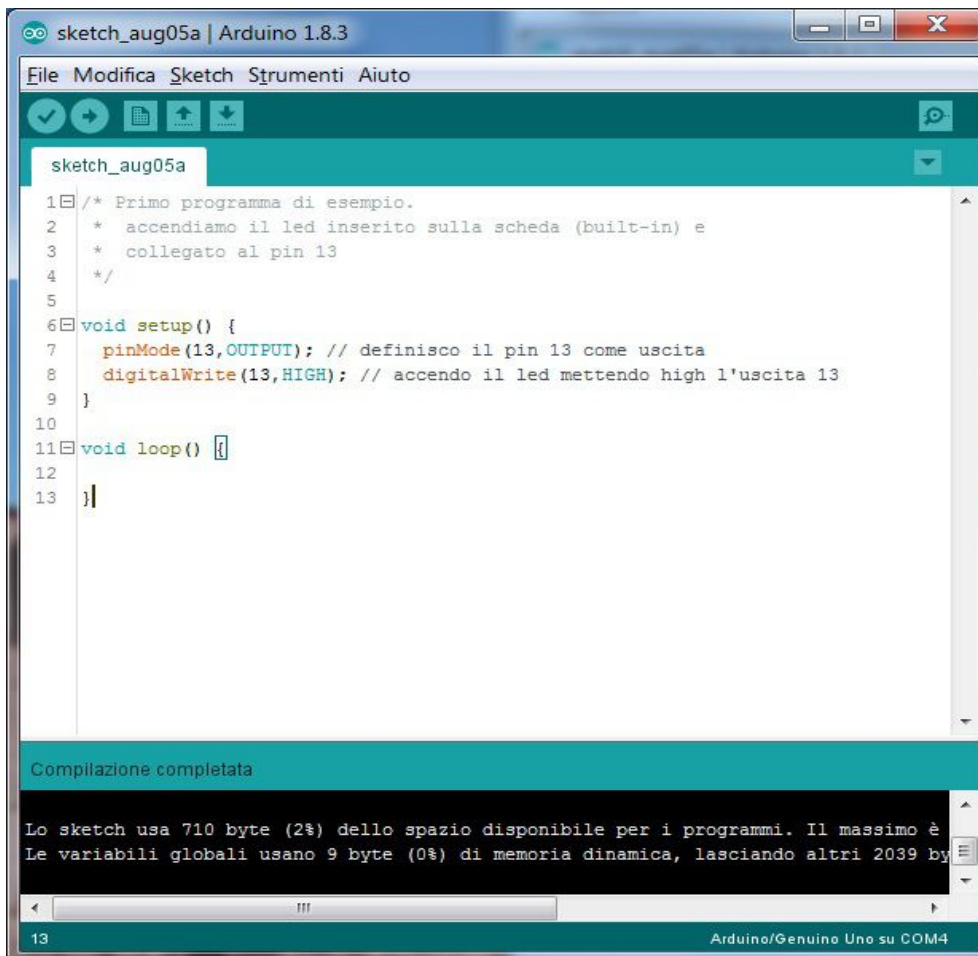
ci sta dicendo che manca il punto e virgola in fondo alla riga **digitalWrite...** prima dell'ultima parentesi graffa a questo punto eliminiamo l'errore e ripetiamo la compilazione che andrà a buon fine dandoci il

seguente messaggio

Lo sketch usa 724 byte (2%) dello spazio disponibile per i programmi. Il massimo è 32256 byte. Le variabili globali usano 9 byte (0%) di memoria dinamica, lasciando altri 2039 byte liberi per le variabili locali. Il massimo è 2048 byte.

In cui viene indicato lo spazio di memoria usato dal programma e dalle variabili.

Passiamo ora all'analisi concettuale del programma. Alla partenza il programma entra nella procedura di **setup** e imposta il pin 13 come uscita dopodiché entra nella procedura **loop** e accende il led poi accende il led poi accende il led.... migliaia di volte al secondo, infatti le istruzioni contenute in loop sono ripetute continuamente. Quindi in questo particolare caso, è uno spreco scrivere l'istruzione digitalWrite qui dentro in quanto il led resta sempre acceso, infatti possiamo benissimo spostare questa istruzione nel setup e lasciare vuoto il loop in questo modo:



```
sketch_aug05a | Arduino 1.8.3
File Modifica Sketch Strumenti Aiuto
sketch_aug05a
1 /* Primo programma di esempio.
2  * accendiamo il led inserito sulla scheda (built-in) e
3  * collegato al pin 13
4  */
5
6 void setup() {
7   pinMode(13,OUTPUT); // definisco il pin 13 come uscita
8   digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
9 }
10
11 void loop() {
12
13 }
```

Compilazione completata

Lo sketch usa 710 byte (2%) dello spazio disponibile per i programmi. Il massimo è 32256 byte. Le variabili globali usano 9 byte (0%) di memoria dinamica, lasciando altri 2039 byte liberi per le variabili locali. Il massimo è 2048 byte.

13 Arduino/Genuino Uno su COM4

Attenzione!!

La sezione **loop** anche se vuota, non va cancellata!

Ovviamente un programma di questo tipo non ha molto senso. Vediamo quindi come rendere più interessante la cosa facendo lampeggiare il nostro led. La sezione setup non cambia, lasciamo la definizione del pin 13 come uscita mentre, nella sezione loop aggiungiamo le seguenti istruzioni

```
1 /* Primo programma di esempio.
2  * accendiamo il led inserito sulla scheda (built-in) e
3  * collegato al pin 13
4  */
5
6 void setup() {
7   pinMode(13,OUTPUT); // definisco il pin 13 come uscita
8 }
9
10 void loop() {
11   digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
12   delay(1000);           //pausa 1 secondo = 1000 millisec.
13   digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
14   delay(1000);           //pausa 1 secondo = 1000 millisec.
15 }
```

Abbiamo aggiunto la riga con l'istruzione ***digitalWrite(13,LOW)***, per spegnere il led e due righe con l'istruzione ***delay(millisecondi_di_attesa)***, per fermare l'esecuzione del programma per il numero di millisecondi indicato tra parentesi, in questo caso 1 secondo. Ora nel ***loop*** il programma prima accende il led, attende 1 secondo, spegne il led e attende 1 secondo quindi ricomincia da capo. Provate diverse combinazioni di tempo per variare l'effetto ottenuto.

Consideriamo ora questo codice:

```

1  ▢ /* Sos codice morse
2  * -----
3  * S= punto punto punto
4  * O= linea linea linea
5  * S= punto punto punto
6  * -----
7  * usiamo il led inserito sulla scheda (built-in) e
8  * collegato al pin 13
9  */
10
11 ▢ void setup() {
12     pinMode(13,OUTPUT); // definisco il pin 13 come uscita
13 }
14
15 ▢ void loop() {
16     //lettera S
17     digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
18     delay(200);           //pausa 200 millisecc.
19     digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
20     delay(300);           //pausa 300 millisecc.
21     digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
22     delay(200);           //pausa 200 millisecc.
23     digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
24     delay(300);           //pausa 1 secondo = 1000 millisecc.
25     digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
26     delay(200);           //pausa 200 millisecc.
27     digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
28     delay(600);           //pausa 600 millisecc.
29     //lettera O
30     digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
31     delay(600);           //pausa 600 millisecc.
32     digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
33     delay(300);           //pausa 300 millisecc.
34     digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
35     delay(600);           //pausa 600 millisecc.
36     digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
37     delay(300);           //pausa 300 millisecc.
38     digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
39     delay(600);           //pausa 600 millisecc.
40     digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
41     delay(600);           //pausa 600 millisecc.
42     //lettera S
43     digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
44     delay(200);           //pausa 300 millisecc.
45     digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
46     delay(300);           //pausa 300 millisecc.
47     digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
48     delay(200);           //pausa 300 millisecc.
49     digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
50     delay(300);           //pausa 300 millisecc.
51     digitalWrite(13,HIGH); // accendo il led mettendo high l'uscita 13
52     delay(200);           //pausa 300 millisecc.
53     digitalWrite(13,LOW); // spengo il led mettendo low l'uscita 13
54     delay(600);           //pausa 600 millisecc.
55 }

```

Come vedete è un programma piuttosto lungo anche se ancora molto semplice.

Immaginate ora di costruire un circuito esterno e di volerlo collegare all'uscita 10 della scheda, a questo punto in ogni riga dove c'è scritto 13 dovete mettere un 10 e se un domani voleste cambiare nuovamente l'uscita dovreste ricambiare nuovamente tutti i numeri con notevoli perdite di tempo e possibilità di dimenticarne qualcuno. Introduciamo quindi il concetto di “*costante*” e “*variabile*”. Si definisce costante

un indirizzo di memoria nel quale memorizziamo un valore (numero o testo) che non cambierà e che utilizzeremo poi nel programma mentre si definisce variabile un indirizzo di memoria nel quale è possibile memorizzare un valore (numero o testo) che potrà essere cambiato nel corso del programma. Usiamo 5 costanti: per il pin utilizzato, per la lunghezza del punto, per la linea, per le pause all'interno della stessa lettera e per la pausa tra le lettere.

```

1  /* Sos codice morse
10
11  const int pinled = 13;           // pin collegato al led
12  const int punto = 200;          // tempo led acceso per punto
13  const int linea = 600;          // tempo led acceso per linea
14  const int pausaBreve = 300;     // tempo led spento
15  const int pausaLunga = 800;     // tempo led spento tra 2 lettere
16
17  void setup() {
18      pinMode(pinled,OUTPUT); // definisco il pin 13 come uscita
19  }
20
21  void loop() {
22      //lettera S
23      digitalWrite(pinled,HIGH); // accendo il led mettendo high l'uscita 13
24      delay(punto);              //pausa 200 millisec.
25      digitalWrite(pinled,LOW);  // spengo il led mettendo low l'uscita 13
26      delay(pausaBreve);         //pausa 300 millisec.
27      digitalWrite(pinled,HIGH); // accendo il led mettendo high l'uscita 13
28      delay(punto);              //pausa 200 millisec.
29      digitalWrite(pinled,LOW);  // spengo il led mettendo low l'uscita 13
30      delay(pausaBreve);         //pausa 200 millisec.
31      digitalWrite(pinled,HIGH); // accendo il led mettendo high l'uscita 13
32      delay(punto);              //pausa 200 millisec.
33      digitalWrite(pinled,LOW);  // spengo il led mettendo low l'uscita 13
34      delay(pausaLunga);         //pausa 800 millisec.
35      //lettera O
36      digitalWrite(pinled,HIGH); // accendo il led mettendo high l'uscita 13
37      delay(linea);              //pausa 600 millisec.
38      digitalWrite(pinled,LOW);  // spengo il led mettendo low l'uscita 13
39      delay(pausaBreve);         //pausa 300 millisec.
40      digitalWrite(pinled,HIGH); // accendo il led mettendo high l'uscita 13
41      delay(linea);              //pausa 600 millisec.
42      digitalWrite(pinled,LOW);  // spengo il led mettendo low l'uscita 13
43      delay(pausaBreve);         //pausa 300 millisec.
44      digitalWrite(pinled,HIGH); // accendo il led mettendo high l'uscita 13
45      delay(linea);              //pausa 600 millisec.
46      digitalWrite(pinled,LOW);  // spengo il led mettendo low l'uscita 13
47      delay(pausaLunga);         //pausa 800 millisec.
48      //lettera S
49      digitalWrite(pinled,HIGH); // accendo il led mettendo high l'uscita 13
50      delay(punto);              //pausa 300 millisec.
51      digitalWrite(pinled,LOW);  // spengo il led mettendo low l'uscita 13
52      delay(pausaBreve);         //pausa 300 millisec.
53      digitalWrite(pinled,HIGH); // accendo il led mettendo high l'uscita 13
54      delay(punto);              //pausa 300 millisec.
55      digitalWrite(pinled,LOW);  // spengo il led mettendo low l'uscita 13
56      delay(pausaBreve);         //pausa 300 millisec.
57      digitalWrite(pinled,HIGH); // accendo il led mettendo high l'uscita 13
58      delay(punto);              //pausa 300 millisec.
59      digitalWrite(pinled,LOW);  // spengo il led mettendo low l'uscita 13
60      delay(pausaLunga);         //pausa 800 millisec.
61  }

```

Tutte le variabili (le costanti sono anch'esse un tipo di variabile) normalmente, vengono definite all'inizio del programma e oltre alla parola chiave “*const*” bisogna dichiararne anche il tipo, in questo caso usiamo

dei numeri senza decimali quindi diciamo che sono interi (*int*), fanno parte degli interi i numeri senza virgola memorizzati con 2 byte cioè da -32.768 a +32.767 . Altri tipi di variabili sono ad esempio “*float*” numeri con la virgola memorizzati in 4 byte, “*long*” numeri interi memorizzati con 4 byte ,”*double*” numeri con la virgola memorizzati in 8 byte, “*string*” serie di caratteri e molti altri che vedremo più avanti.

In questo modo non abbiamo accorciato il programma... Anzi l'abbiamo allungato di 5 righe, ma lo abbiamo reso più leggibile e soprattutto se vogliamo operare delle modifiche sui tempi o sui contatti da usare dobbiamo lavorare solamente sulle prime 5 righe.