ARDUINO – GENUINO & CO

Mini corso introduttivo

A cura del
Ferrara Linux User Group
Paolo Coatti

Con il patrocinio del
Comune di Voghiera

12-19 novembre 2015

Comune di Voghiera

Provincia di Ferrara

Cosa è Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects.

definizione tratta dal sito ufficiale http://arduino.cc

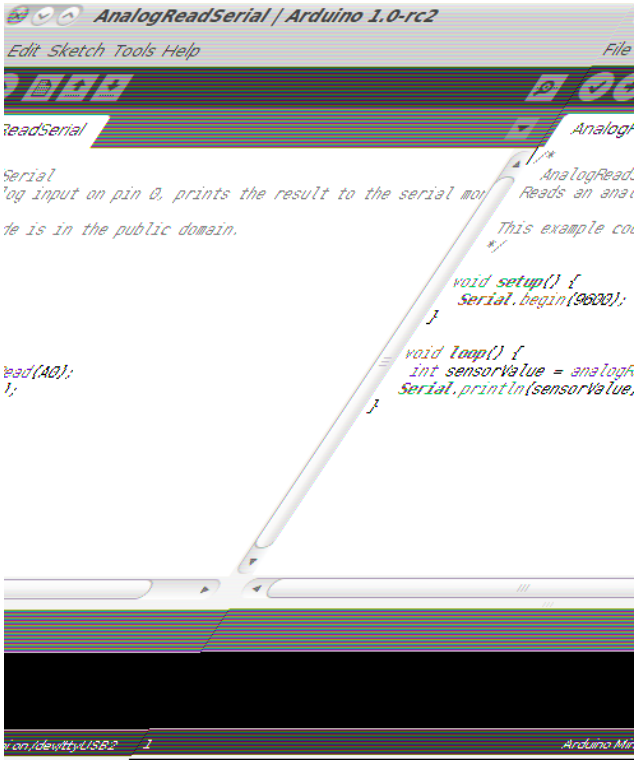
In pratica

È una piattaforma Hardware e Software opensource per la proto-tipizzazione. Gli ideatori di tutto questo, Arduino Team, hanno creato un' ambiente aperto e semplice. Questa *apertura* ha permesso ad una comunità di nascere fornendo un grande patrimonio hardware con schede ufficiali, varie compatibili e schede opzionali(shield) il tutto corredato di librerie software per l'uso delle stesse e di documentazione. Tutti noi possiamo usare nei nostri progetti quanto reso disposizione sia per i nostri progetti sia a casa che per lavoro, rispettando la licenza. Anche noi possiamo contribuire a tutto questo con le nostre idee, il nostro codice, hardware da mettere a disposizione degli altri come altri hanno fatto con noi.

Cosa mi serve per iniziare

- Una Scheda (Board) Arduino
- Un computer
- Un cavetto USB
- L'IDE di Arduino scaricabile gratuitamente
- Tanta voglia di impare e sperimentare

Cos'è Arduino



- É una scheda a microcontrollore basata sui microcontrollori della serie megaAVR a 8 bit di Atmel
- Sono uscite versioni a 32 bit con un Atmel SAM3U (Cortex-M3 ARM)
- La piattaforma si basa su un elettronica semplice e utilizzabile subito senza particolari problemi
- Non serve avere un programmatore specifico per programmare il micro. Si programma tramite la porta USB
- Sia la stesura del programma, compilazione e programmazione della board avviene tramite un software gratuito, *IDE Arduino environment*

Boards e Shields

Boards ci sono le originale e le compatibili, quelle con marchio Arduino e il nuovo marchio Genuino, una vera giungla dove ci si può perdere facilmente.

Il mio consiglio … Come inizio prendetene una facile e originale, così almeno non dovrete lottare con problemi di stabilità e compatibilità, quando sarete esperti potrete usare di tutto.

Shield sono schede impilabili sulle board di base che forniscono funzionalità aggiuntive a volte costano di più della stessa board, ma fanno delle cose veramente incredibili.

Descrizioni comuni delle schede

Descrizioni comuni

Tutte le board si basano sui **microcontrollori ATMEL.**
Hanno in comune diverse parti.

- Microcontrollore Atmel ATmega328 per UNO, NANO, MINI, PRO, LYLYPAD
  - AVR 8bit 16MHz
  - Flash Memory da 32 KB (0.5 KB usati dal bootloader)
  - SDRAM 2 KB – EEPROM 1 KB
  - 14 I/O digitali – ADC a 10 bit con 6 o 8 ingressi Analogici
  - 1 seriale
  - 2 interrupt

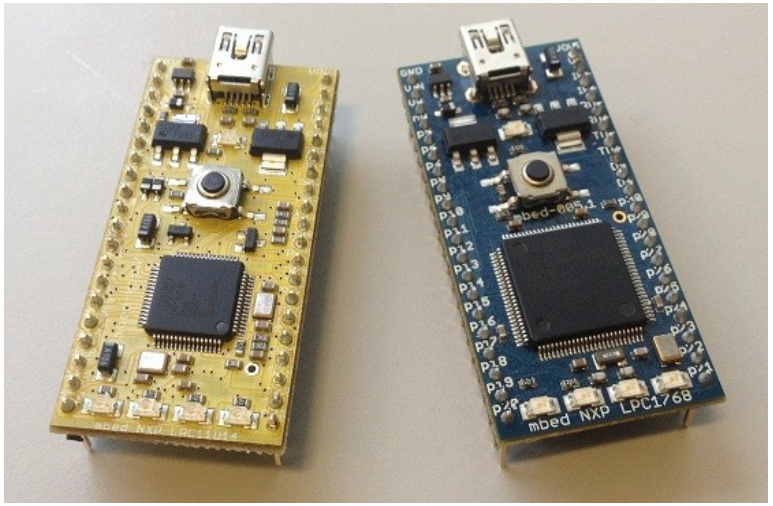
- Microcontrollore Atmel ATmega2560 per MEGA2560 e ADK
  - AVR 8bit 16MHz
  - Flash Memory da 256 KB (8 KB usati dal bootloader)
  - SDRAM 8 KB – EEPROM 4 KB
  - 54 I/O digitali – ADC a 10 bit con 16 ingressi Analogici
  - 4 seriale
  - 6 interrupt

I “*concorrenti*” o compagni di viaggio

Arduino è un progetto pioniere nel suo campo con la sua filosofia **open** non solo nel

software ma anche nel **hardware**.
Tuttavia nel mondo dell'elettronica non è solo...

I “*concorrenti*” o compagni di viaggio



MBED è una comunità sulle stesse filosofia di Arduino.
In questo caso abbiamo un hardware più potente basate su un **microcontrollore ARM a 32 bit** della NXP.

Interessante SDK dove tutto gira su una “nuvola” su http://mbed.org/. Si ha accesso ad una vastità di librerie software pronte. È possibile condividere i propri progetti e lavorare in gruppo. Si programma in C++

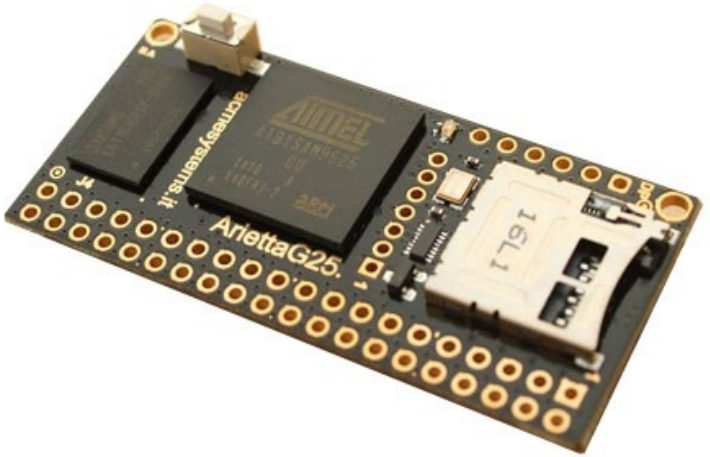
NON è compatibile pin layout di Arduino, ma nessuno vieta di adattare le shields nate per Arduino con questa.

Attualmente ci sono due versioni di board dove le differenze sono la velocità di clock (48Mhz e 92Mhz), I/O, bus, ethernet, ADC...
Prezzi di circa € 52,00 e € 62,00 iva compresa

Da prendere in considerazione

Info http://mbed.org/

I “*concorrenti*” o compagni di viaggio



Questo è un progetto **opensource ITALIANO molto interessante e merita attenzione.**
La **ARIETTA G25** scheda è un sistema embedded basato su un processore CPU Atmel AT91SAM9G25 SoC (ARM9 @ 400Mhz) a 32 bit. La scheda dispone di diversi GPIO, bus, ADC. Ci installate LINUX, e ci programmate come programmate in LINUX (Python, C++, PHP, Java...) Libreria permettono l'accesso alle GPIO, bus, ADC, ecc. Ci sono due versioni una da 128 MB di ram l'altra da 256 MB, i prezzi circa 26 e 32 euro. Esiste inoltre una minischeda wifi dedicata e dal basso prezzo 12 Euro

Per info e comunità http://www.acmesystems.it/

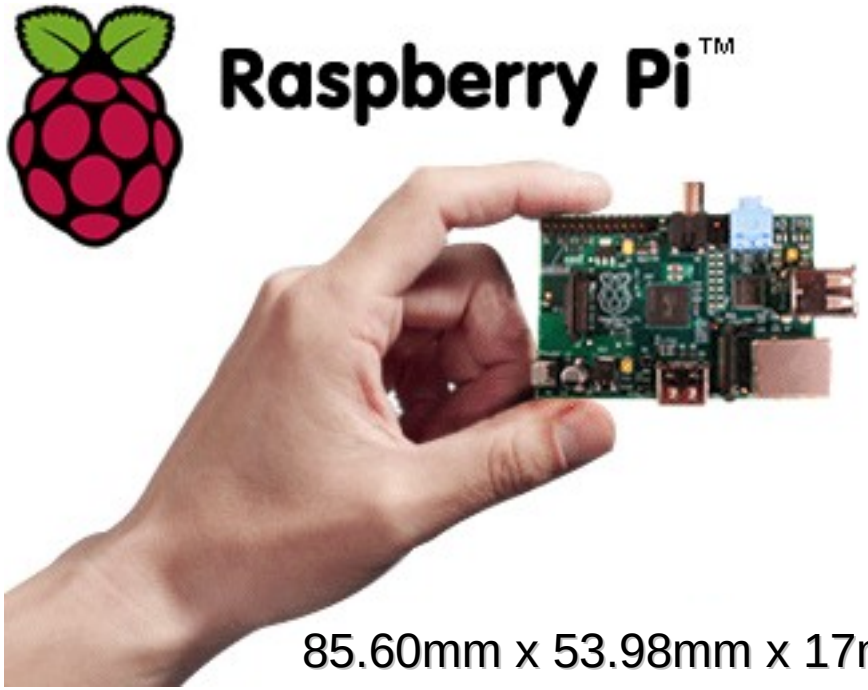
I “*concorrenti*” o compagni di viaggio

È un progetto opensource sponsorizzato da TI e usa i loro processori ARM
Sono sistemi embedded e con Linux, e ci programmate in Python, C++, PHP, Java...
Le schede variano per tipo di processore ARM, a ci trovate GPIO, ADC, ethernet, LCD, USB...
La **BeagleBone Black** dispone di una uscita HDMI,

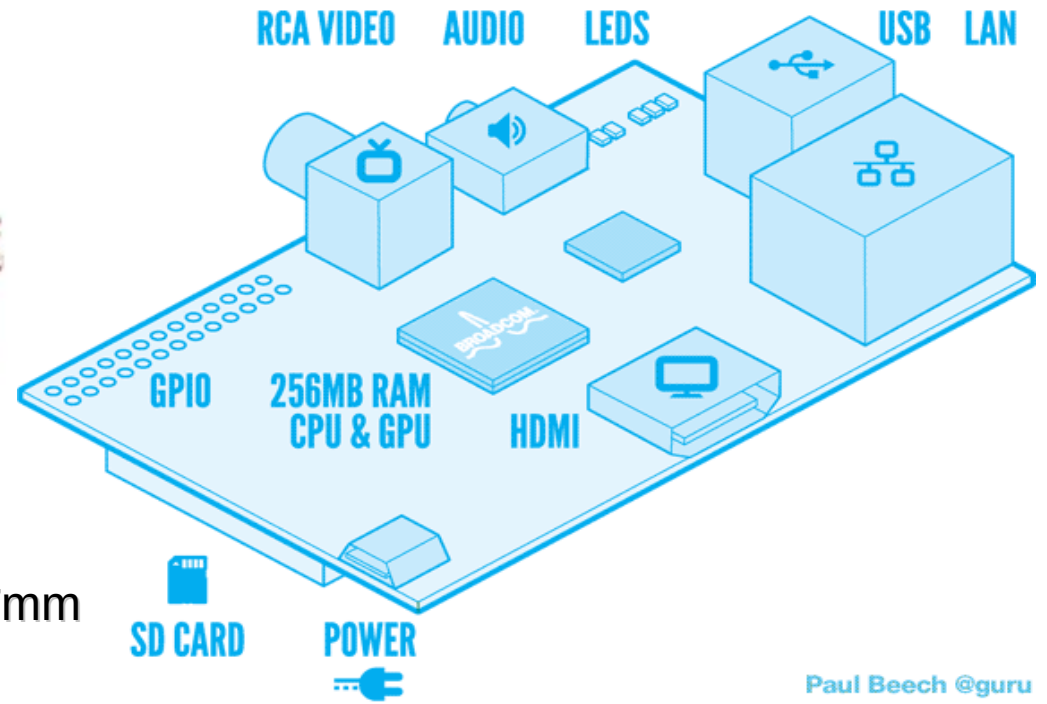
I prezzi sono interessati, e poi ci sono i Cape (scheda di espansione progettate appositamente).

La comunità http://beagleboard.org/

I “*concorrenti*” o compagni di viaggio



85.60mm x 53.98mm x 17mm



Una buona potenza di calcolo e un costo allettante, una vasta comunità.
Arduino + Raspberry Pi è un abbinamento molto interessate, potente, flessibile con costi bassi.
Info:
http://www.element14.com/community/groups/raspberry-pi
http://www.raspberrypi.org/

La scheda Arduino

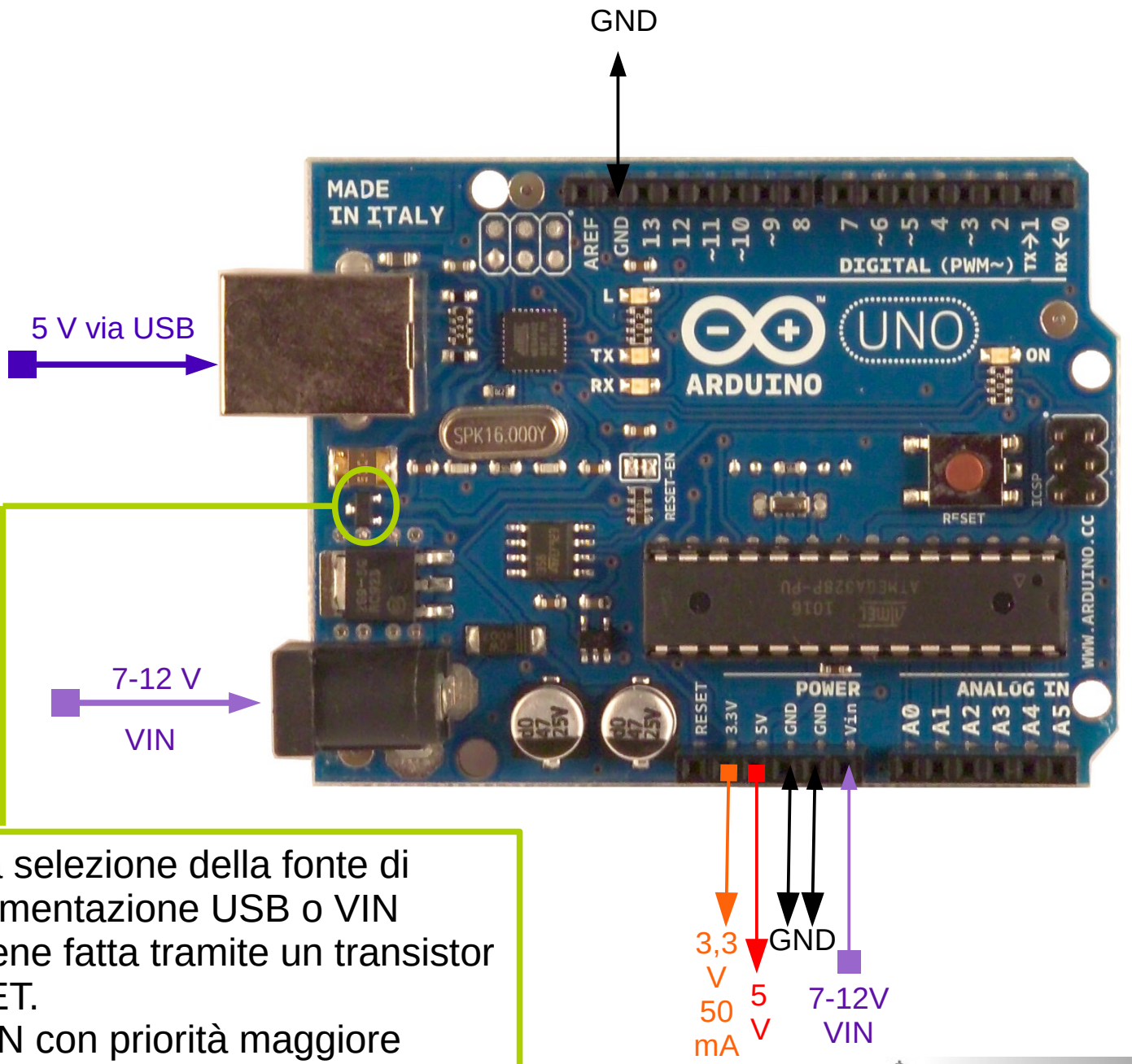
Il Bootloader

Il **bootloader** è una piccolo programma residente nel Flash Memory del microcontrollore.

Il suo compito è quello di:

- Permettere la programmazione del micro tramite la porta Seriale/USB. Quando noi inviamo tramite IDE il nuovo programma compilato alla board, è questo piccolo programma che si occupa di memorizzarlo nella flash memory, evitando l'uso di uno programmatore specifico.
- Nel caso della board UNO occupa solo 0,5KB.
- Esso viene eseguito al momento dell'accensione della Board o dopo un reset.
- Su alcune board di *vecchio design* per caricare il programma va premuto il tasto di reset e poi inviato il nuovo programma. Su quelle recenti 2009/UNO/MEGA tramite DTR della seriale viene eseguito un autoreset
- Per riprogrammare il bootloader del micro si usa la porta ICSP usando IDE oppure altro software tipo AVR Studio .
- Si può usare anche un'altra board Arduino per la programamzione.
- Potrebbe essere necessario aggiornarlo in seguito ad una nuova versione dello stesso oppure perché abbiamo acquistato dei microcontrollori “vergini”.
- Per caricare il booloader http://www.labarduino.eu/blog/caricareilbootloaderarduino

Piedinatura dell'alimentazione



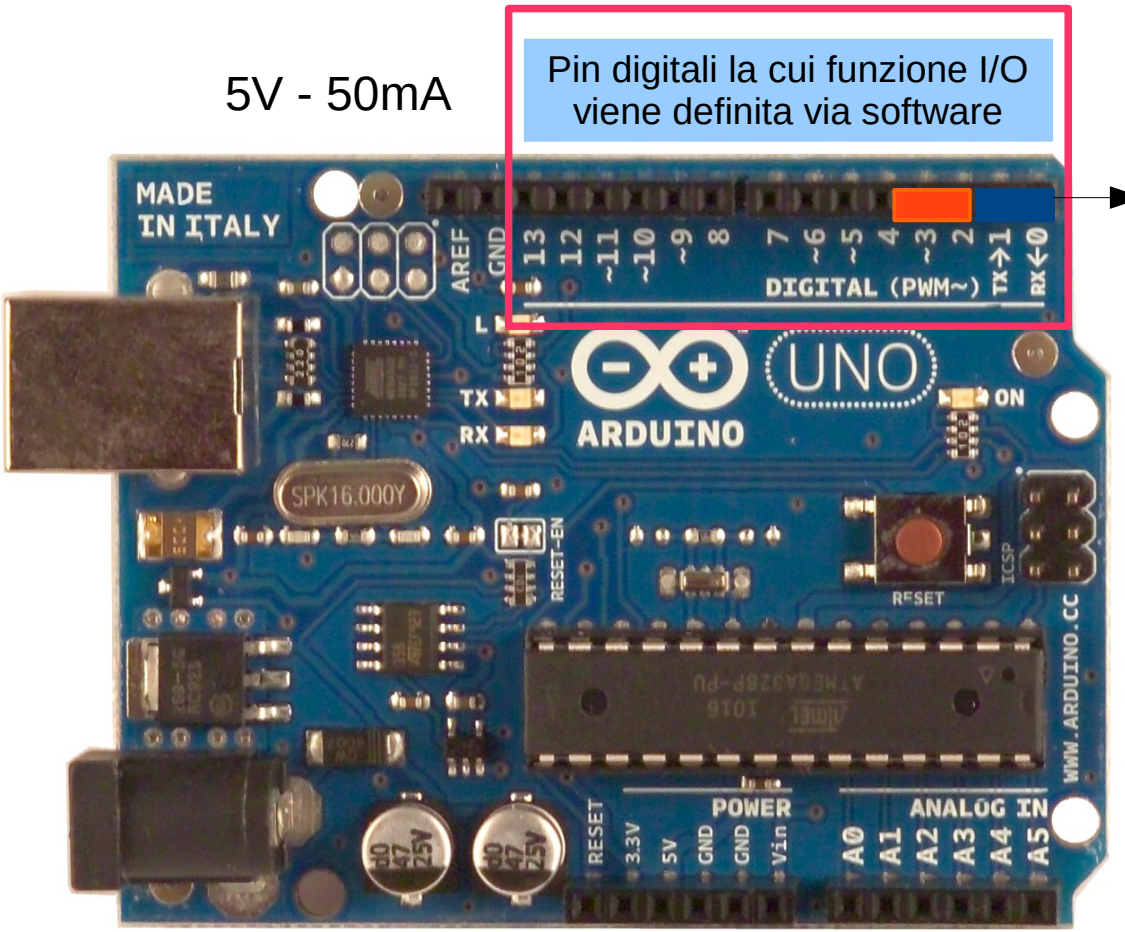
Può essere alimentata tramite i 5v della USB o tramite altra fonte da circa 7-12V consigliati

La selezione della fonte di alimentazione USB o VIN viene fatta tramite un transistor FET.
VIN con priorità maggiore

Piedinatura I/O digitali

5V - 50mA

Pin digitali la cui funzione I/O viene definita via software



Usati anche dalla seriale

Usati anche come Interrupt esterni

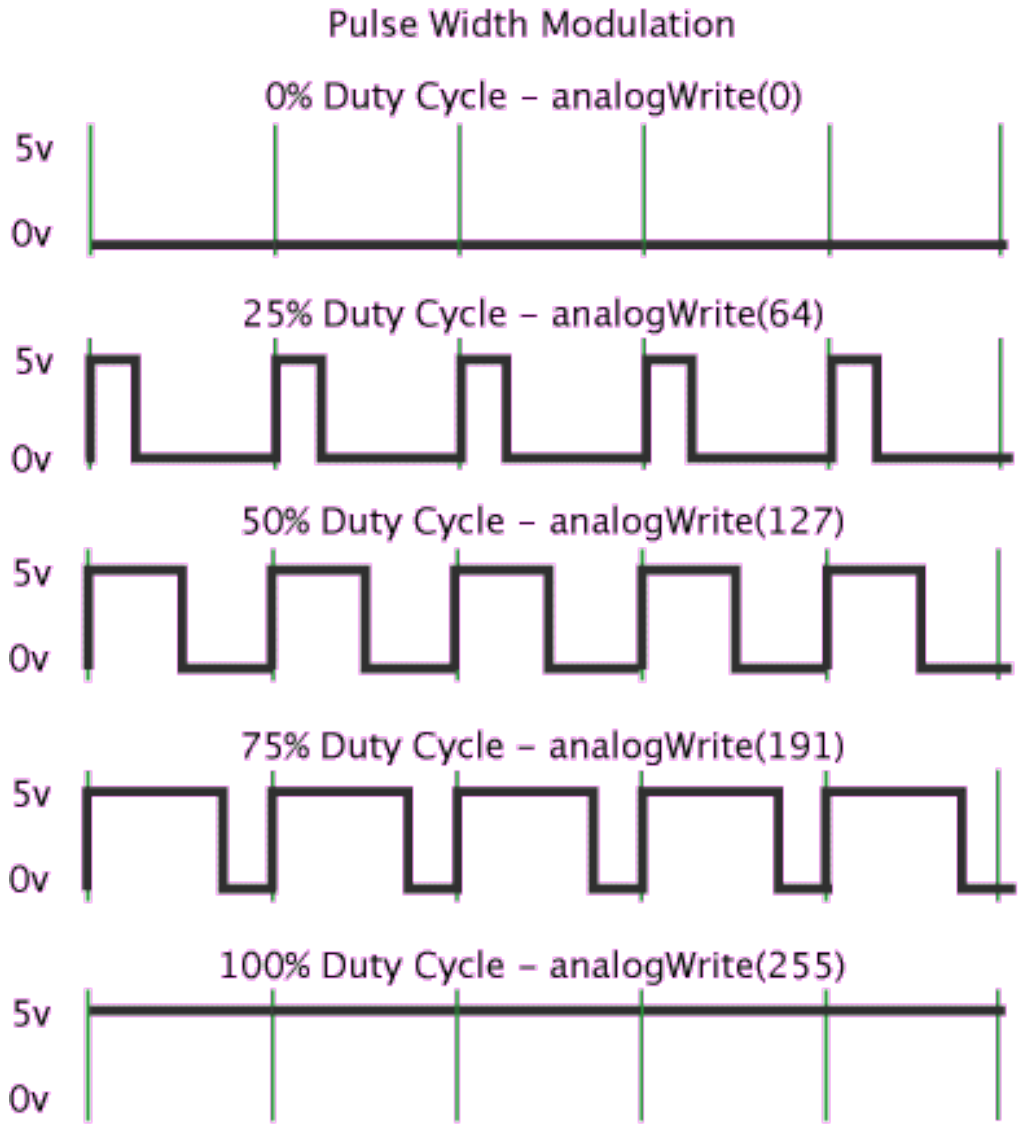
```
pinMode(n, OUTPUT); //  → OUT
```

```
pinMode(n, INPUT); //  ← IN
```

Se si usa i pin 0 e 1 nel progetto, quando si programma la board si consiglia di lasciare liberi tali pin per evitare interferenze. Questi sono usati per la programmazione tramite il chip convertitore seriale/USB

Le OUTPUT PWM

http://it.wikipedia.org/wiki/Pulse-width_modulation



La modulazione di larghezza di impulso, dall'inglese pulse-width modulation o PWM, è un tipo di modulazione digitale che permette di ottenere una tensione media variabile dipendente dal rapporto tra la durata dell' impulso positivo e di quello negativo, allo stesso modo è utilizzato per protocolli di comunicazione in cui l'informazione è codificata sotto forma di durata nel tempo di ciascun impulso. Grazie ai moderni microcontrollori è possibile attivare o inattivare un interruttore ad alta frequenza e allo stesso modo rilevare lo stato e il periodo di un impulso

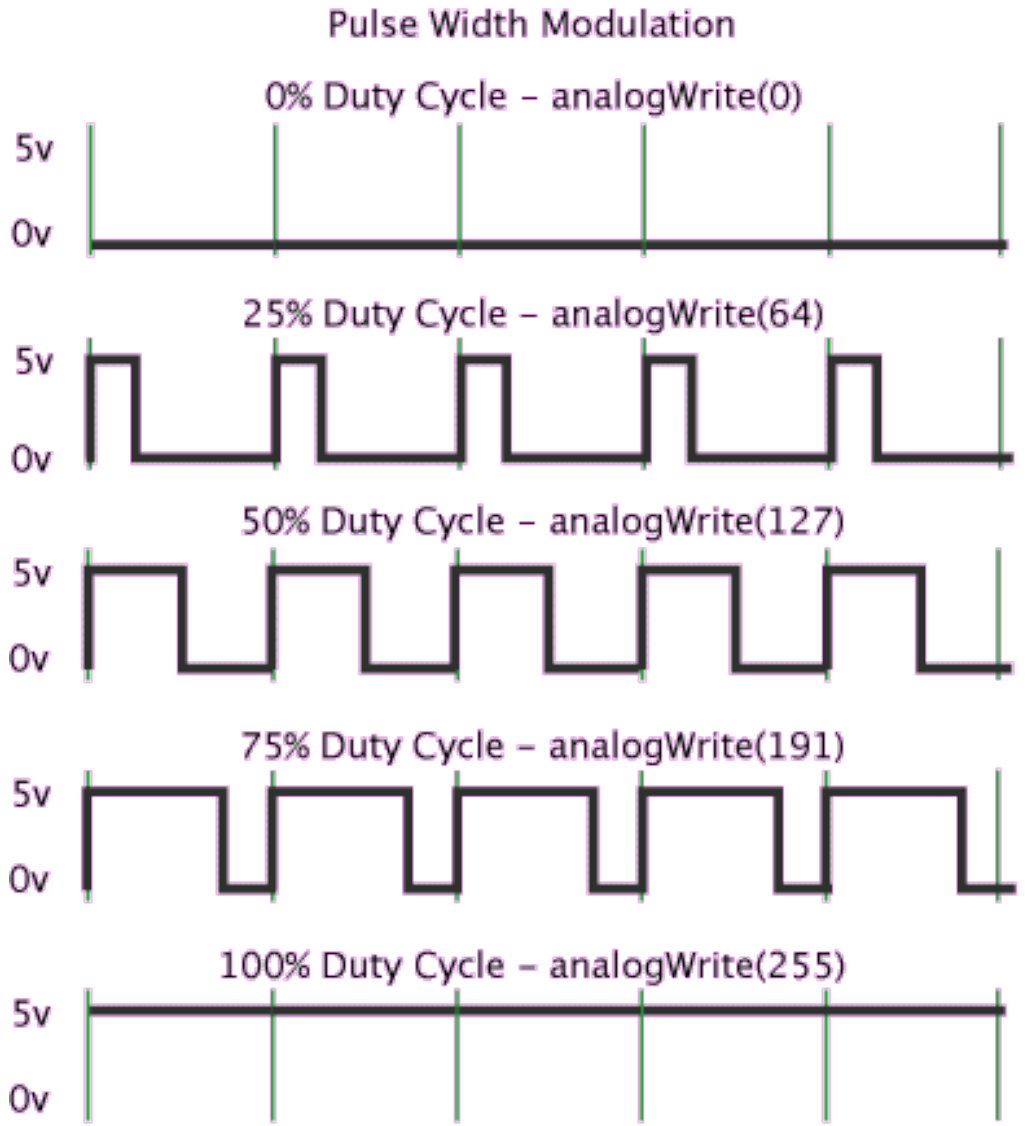
Descrizione

La durata di ciascun impulso può essere espressa in rapporto al periodo tra due impulsi successivi, implicando il concetto di duty cycle. Un duty cycle pari a 0% indica un impulso di durata nulla, in pratica assenza di segnale, mentre un valore del 100% indica che l'impulso termina nel momento in cui inizia il successivo.

http://arduino.cc/en/Tutorial/PWM

Le OUTPUT PWM

http://it.wikipedia.org/wiki/Pulse-width_modulation



Applicazioni

La modulazione a larghezza di impulso è largamente utilizzata anche per regolare la potenza elettrica inviata ad un carico, per esempio negli inverter, per regolare la velocità dei motori in corrente continua e per variare la luminosità delle lampadine.

Come si può intuire, con un duty cycle pari a zero la potenza trasferita è nulla, mentre al 100% la potenza corrisponde al valore massimo trasferito nel caso non sia presente il circuito di modulazione. Ogni valore intermedio determina una corrispondente fornitura di potenza.

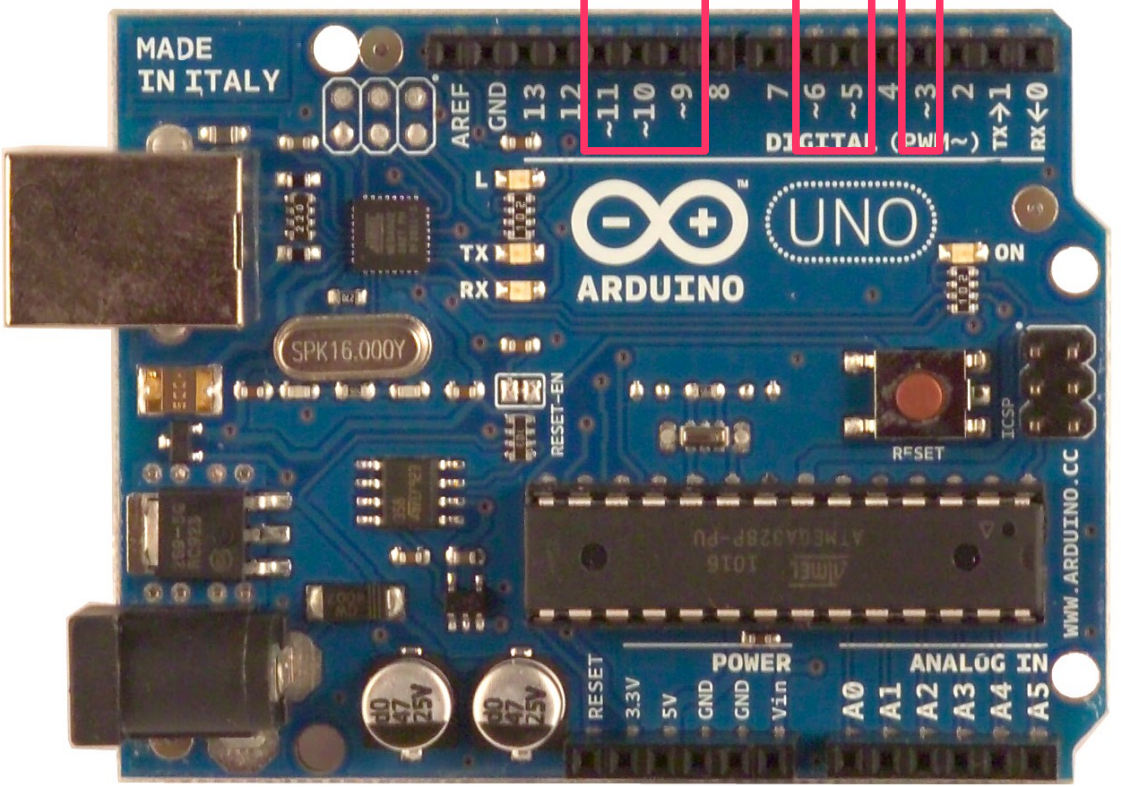
Il vantaggio di questa tecnica è di ridurre drasticamente la potenza dissipata dal circuito limitatore rispetto all'impiego di transistor controllati analogicamente. In un semiconduttore la potenza dissipata è determinata dalla corrente che lo attraversa per la differenza di potenziale presente ai suoi capi. In un circuito PWM il transistor in un istante conduce completamente, riducendo al minimo la caduta ai suoi capi, oppure non conduce, annullando la corrente, ed in entrambi i casi la potenza dissipata è minima.

http://arduino.cc/en/Tutorial/PWM

Piedinatura OUTPUT PWM

- Pulse Wildh Modulation -

5V - 50mA



Pin 3 usato anche come Interrupt

```
analogWrite(pin, value)
```

http://arduino.cc/en/Tutorial/PWM

Input Analogici

- Arduino dispone di alcuni pin analogici e di un relativo convertitore analogico/digitale (ADC).
- **Un ADC converte dei valori di tensione in bit**, quindi la tensione ***diventa*** **digitale per essere gestiti al microcontrollore come l'Atmel.**
- La precisione della conversione viene definita in *bit di risoluzione.* Maggiori sono i bit maggiore è la risoluzione.
- Questi sono un sequenza in uscita digitale di 0 e 1 la cui combinazione è la tensione letta in ingresso
- Gli ADC hanno una tensione di riferimento che determina il range in cui operano la lettura.
- Per calcolare quanti possibili combinazioni posso esserci si usa la seguente formula

Tensione_di_riferimento / 2^{bit}

// 10 bit = 2^{10} = 1024 combinazioni

- Per determinare a che tensione corrisponde un valore digitale
  - Con **8 bit** di risoluzione (256 combinazioni) e una tensione di riferimento di 12V (0-12V)

    12V / 256 = **0,047V** = 47mV per ogni valore
  - Con **10 bit** di risoluzione (1024 combinazioni) e una tensione di riferimento di 12V (0-12V)

    12V / 1024 = **0,012V** = 12mV per ogni valore
  - Con **16 bit** di risoluzione (65536 combinazioni) e una tensione di riferimento di 12V (0-12V)

    12V / 65536 = **0.00018V** = 0,18mV = 180nV per ogni valore

Input Analogici

- Il rapporto segnale/rumore è un fattore importante soprattutto con tensioni basse e/o con l'aumentare del bit di risoluzione.
- Infatti maggiore è la precisione in bit e minore è la variazione di tensione tra un valore e l'altro di lettura.
- Con il termine *Sampling rate* si definisce la frequenza di campionamento del ADC. Infatti ADC non fa una lettura immediata ma campiona in un strettissimo tempo una serie di valori analogici in modo da avere una certa stabilità e poi converte la media. In questo modo si migliora la precisione. Viene espressa in kSPS
- Gli ADC posso essere integrati nei microcontrollori come nel caso di Arduino oppure in un chip specifico che poi vengo collegati al micro tramite un bus (linee di comunicazione di periferiche).
- ADC integrato in Arduino dispone di una risoluzione di 10bit.
- Se ci serve maggior precisione possiamo collegare un ADC esterno ad Arduino se proprio non dovesse bastare.
- 10 bit per la maggior parte dei casi sono sufficienti.

Applicazioni
Gli ingressi analogici sono usati in qui casi in cui bisogna leggere una tensione analogica e gestirla in digitale, per esempio una tensione si alimentazione, di un partitore di tensione, sensori che forniscono in uscita tensione analogica come nel caso di alcuni di temperatura...

Per approfondimenti sui ADC http://it.wikipedia.org/wiki/Convertitore_analogico-digitale

Input Analogici

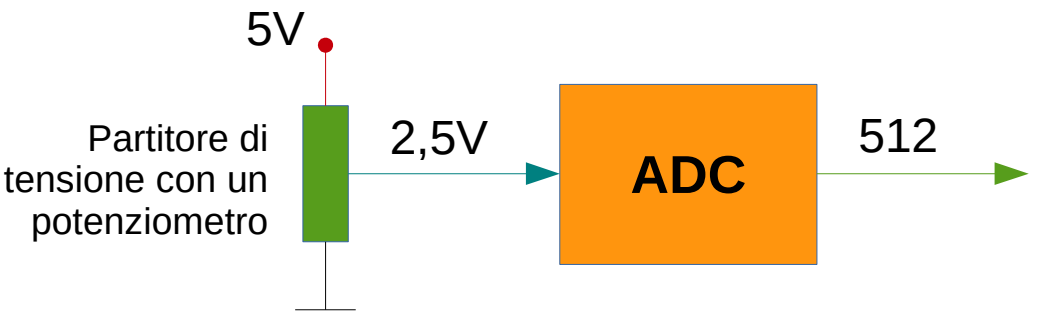
- Arduino grazie al ATMEGA328 dell'Atmel dispone al suo interno di un ADC da **10 bit,** 1024 combinazioni che vanno da 0-1023

con 5V di riferimento e **10 bit** di risoluzione (1024 combinazioni)

5V / 1024 = 0.0049V = 4.9mV per ogni valore

- Per esempio con una lettura in uscita dal ADC abbiamo 512, la tensione analogica in ingresso è:

512 * 4,9mV = 2,5V circa di tensione in ingresso



Un valore di 0 in digitale corrispondono a 0V in ingresso
Un valore di 1023 corrisponde un valore di 5V in ingresso se riferimento di default, ma può essere definita.

Input Analogici

- La tensione di riferimento del ADC di default è quella di alimentazione, ma possiamo definire un altro valore anche esterno.
- Il il pin indicato come **AREF** *(Reference voltage for the analog inputs)* definisce un riferimento esterno per ADC e può avere un valore da 0 a 5V.
- Questa tensione diventa il valore di riferimento della ADC con un range che va da 0 a AREF
- Quindi con un valore 0 in digitale abbiamo una tensione di 0V ; con **1023 abbiamo AREF**
- Ma possiamo anche definirla a livello di solo software senza usare AREF.
- Per usare tale riferimento oltre alla parte hardware con una tensione fissa sul pin AREF nel caso di un riferimento esterno, dobbiamo indicare a livello di codice che voliamo usare un riferimento specifico.
- Si definisce usando l'istruzione **`analogReference(type)`**

Type può assumere

DEFAULT: the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)
INTERNAL: an built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328 and 2.56 volts on the ATmega8 (not available on the Arduino Mega)
INTERNAL1V1: a built-in 1.1V reference (Arduino Mega only)
INTERNAL2V56: a built-in 2.56V reference (Arduino Mega only)
EXTERNAL: the voltage applied to the AREF pin (0 to 5V only) is used as the reference.

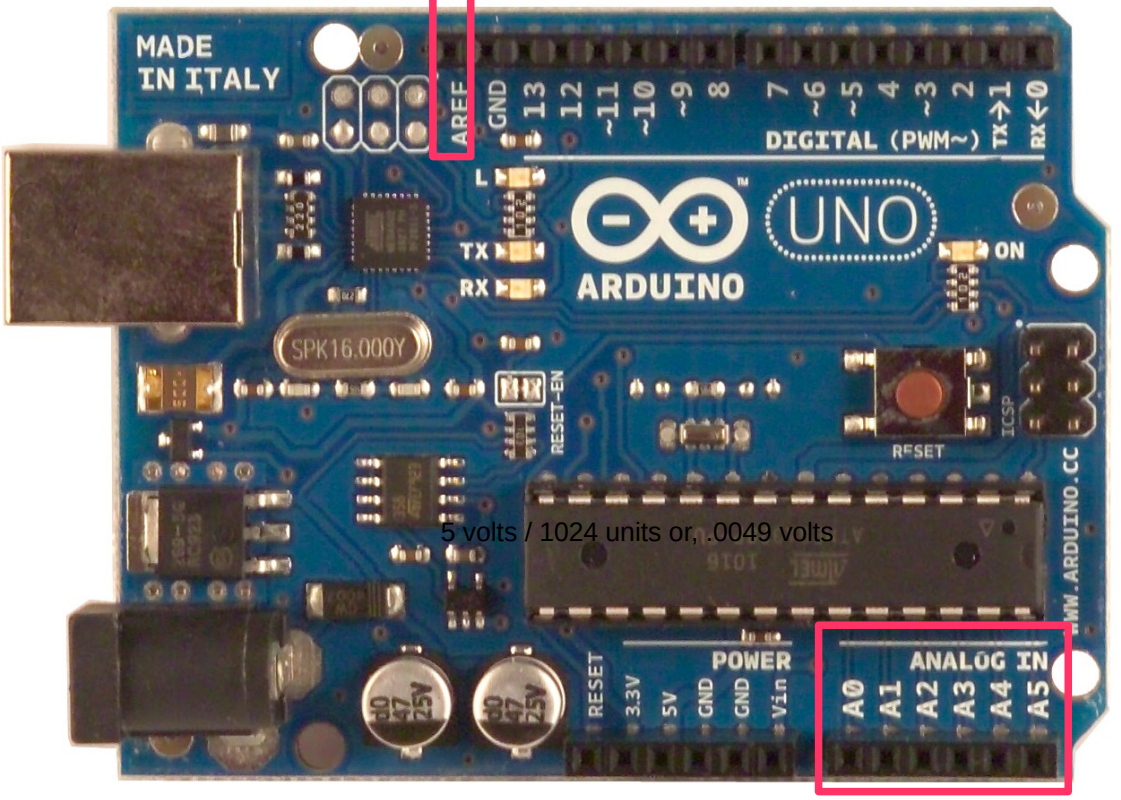
In colore nero valori per Arduino MEGA2560/ADK

Se non viene dichiarato `analogReference` si presume che il riferimento è `DEFAULT`

Piedinatura Input Analogici

AREF
Tensione di riferimento 0-5V per gli IN analogici

```
analogReference(EXTERNAL)
```

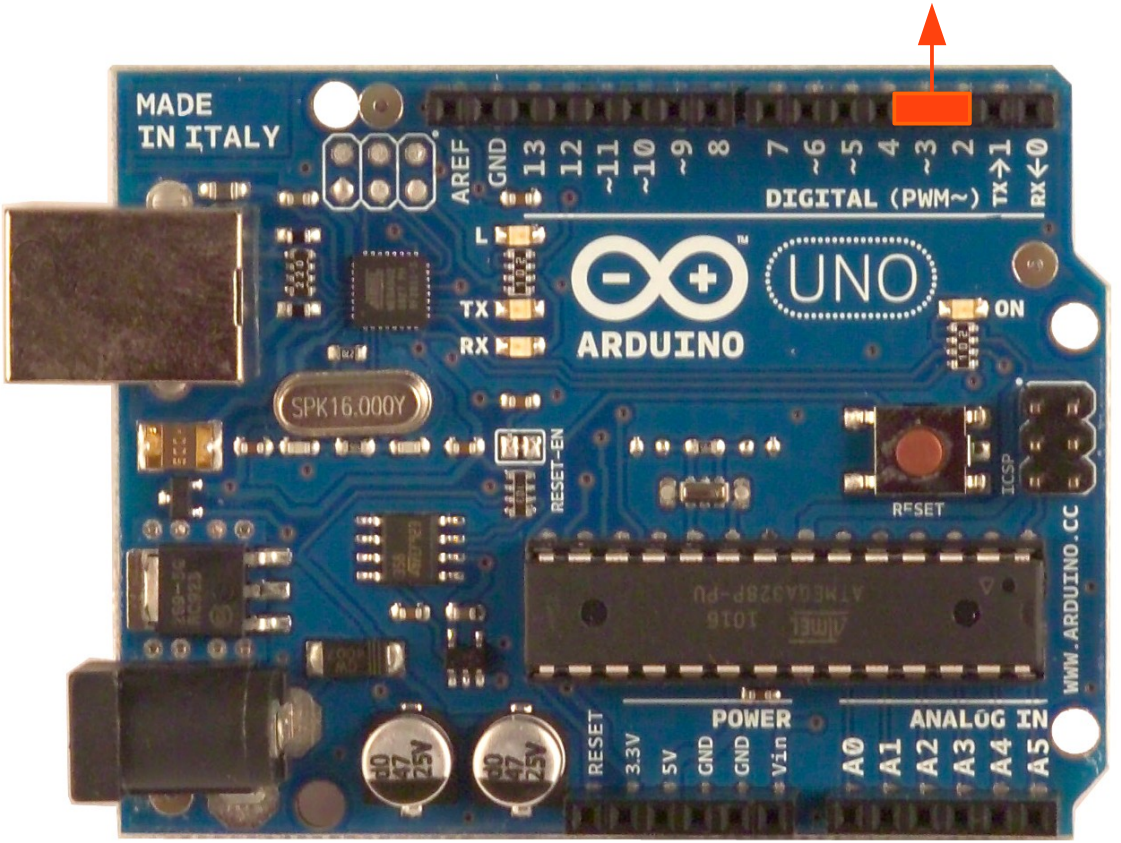


IN Analogici

```
analogRead(pin)
Ritorna un valore
numerico intero 0-1023
```

Piedinatura Interrupt esterni

Usati come Interrupt esterni



Assegnazioni
Pin 2 = int 0
Pin 3 = int 1

Gli Interrup sono utilizzati per interrompere quello che il microcontrollore sta *facendo* per fargli fare qualche altra cosa più *urgente* in seguito ad un evento, per esempio una condizione di allarme.

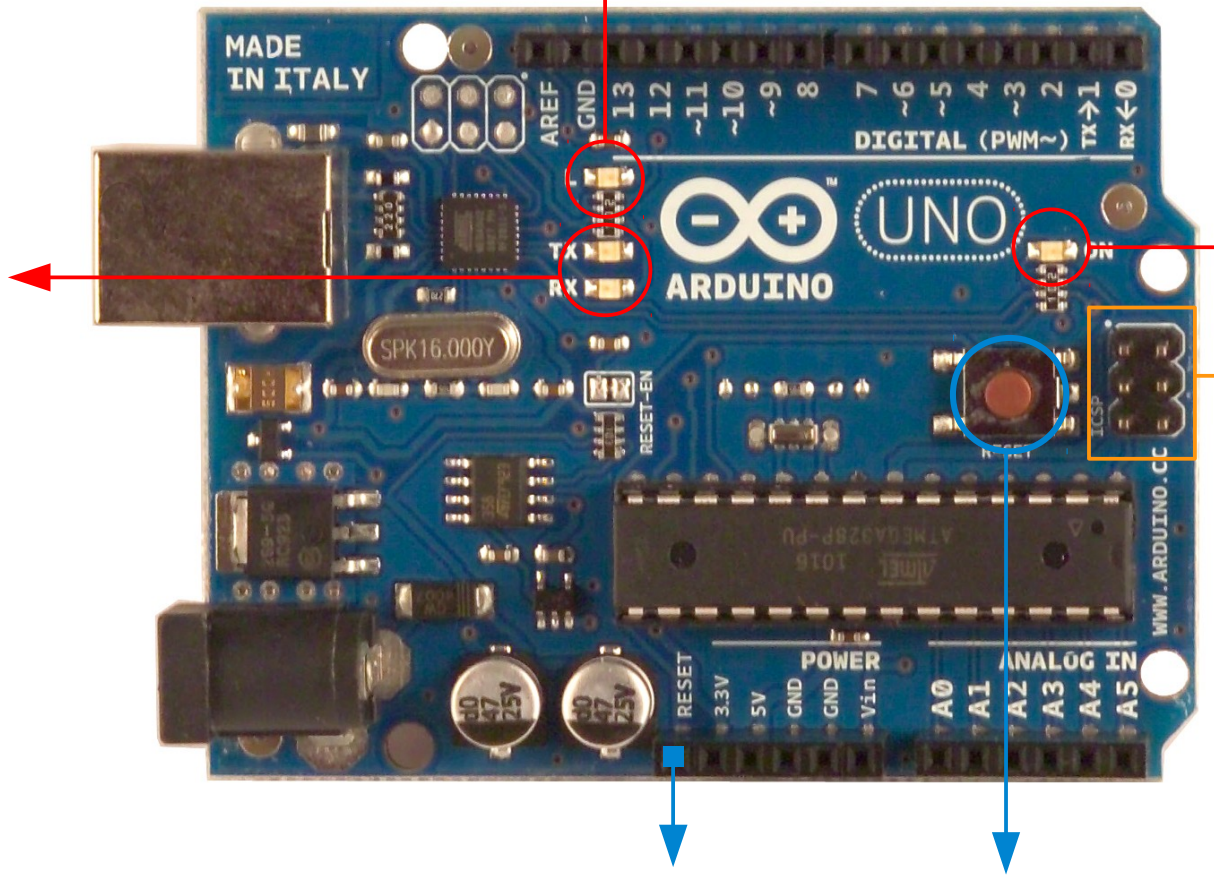
Come si dichiara nel codice

```
AttachInterrupt (int, funzione, modo);
```

Pin Seriale – ICP - LED sulla Board

LED collegato al pin 13

I 2 LED TX e RX della seriale

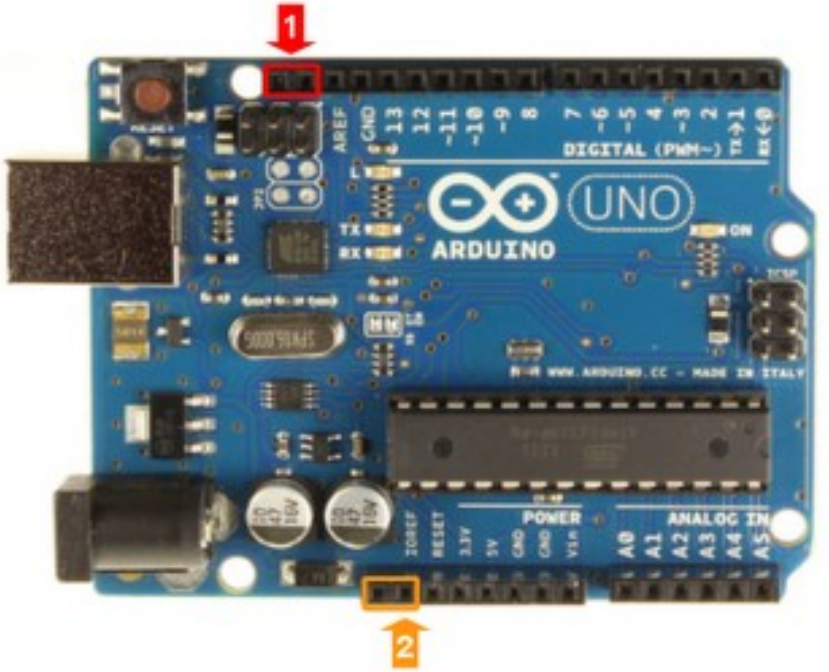


Power (5V)

ICP usata per la riprogrammare il bootloader tramite un programmatore oppure con una seconda board

Linea e pulsante di reset

Arduino UNO rev 3



La board Arduino UNO ha subito un revisione del hardware rispetto alle precedete rev1 e rev2 con introduzione del **pinout 1.0** e altre piccole modifiche, mantenendo la compatibilità hardware e software con le precedenti.
Sono state introdotte:

- Sono stati aggiunti nuovi **due pin SDA e SCL dedicati al bus I2C**, Sono posti vicino al **pin AREF** e questi non sono altro che delle duplicazione del pin **A4 (=SDA) e A5 (=SCL)** usati per il bus I2C. Questo lo potete vedere anche dallo schema.

- Vicino al pin di RESET è stato introdotto il **pin IOREF** che serve come riferimento della tensioni di uscita delle I/O per le varie schede opzionali (shields) in modo che si adeguino alla tensione corretta. Nella board che usano AVR come la UNO, MEGA2560 queste sono a 5V, mentre nella **Arduino DUE è a 3,3V**. È stato introdotto un ulteriore pin per usi futuri.

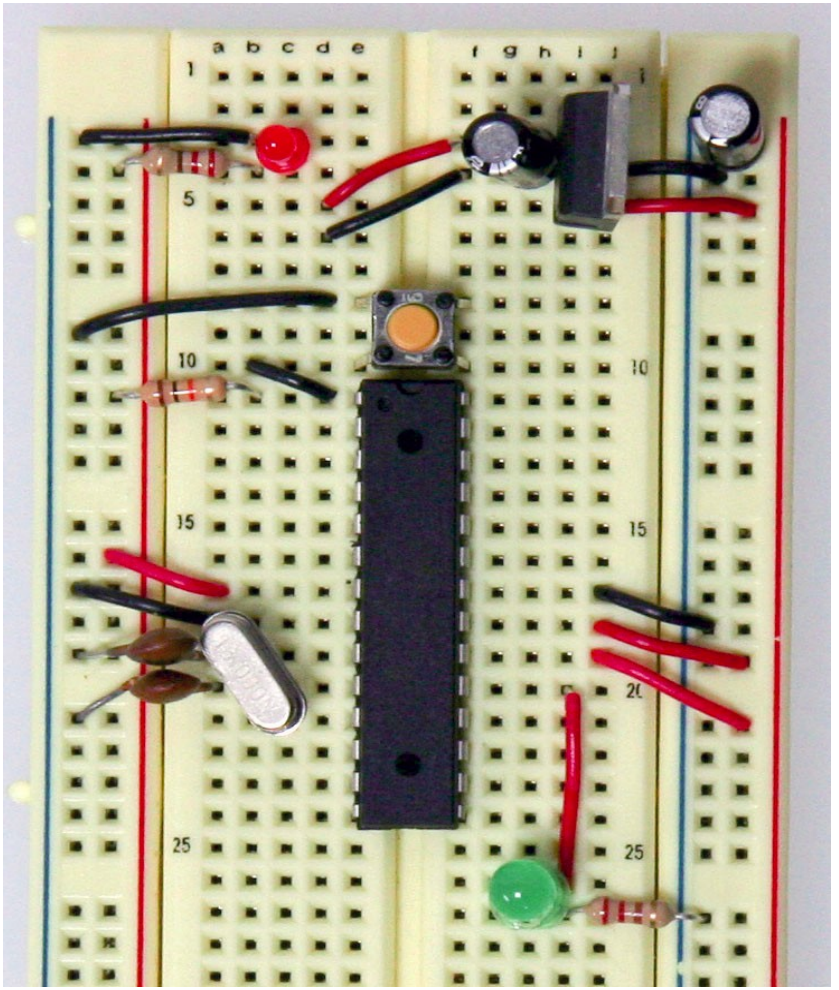
- Il convertitore USB passa dal ATMEGA8u2 (8K flash) **al ATMEGA16u2 (16K flash)**. Questa modifica per il momento non comporta variazioni importanti, ma visto un maggior memoria della flash in futuro potrà essere usata per esempio per gestire periferiche come tastiere e mouse USB.

- Migliorato il circuito di reset. È stato spostato pulsate vicino al connettore USB.

- La parte posteriore della board ora contradistinta dal colore bianco che da un bel effetto nel design visto della board.

- Le modifiche introdotte riguardano anche le altre board Arduino come la MEGA2560.

Usare Arduino senza “board”



Fin'ora abbiamo parlato di boards Arduino ufficiali e cloni, ma potete usare Arduino anche senza una scheda Arduino.

Unica cosa che serve è il micro Atmel con caricato il bootloader Arduino. Il resto del circuito lo disegnate attorno al micro con la componentistica che si serve. Praticamente disegnate la board Arduino del vostro progetto.

I vantaggio sono:

• In generale costi minori

• dimensioni del PCB ottimizzate per il vostro progetto, utilizzando solo quello che vi serve. Per esempio se non usate gli input analogici semplicemente non mettete i connettore o se non usare USB non mettete la parte USB.

L'esempio della foto è come usare il Arduino con una breadboard. Il micro Atmel contiene il bootloader Arduino.
É un tutorial della arduino cc
https://www.arduino.cc/en/Main/Standalone

Parliamo di software

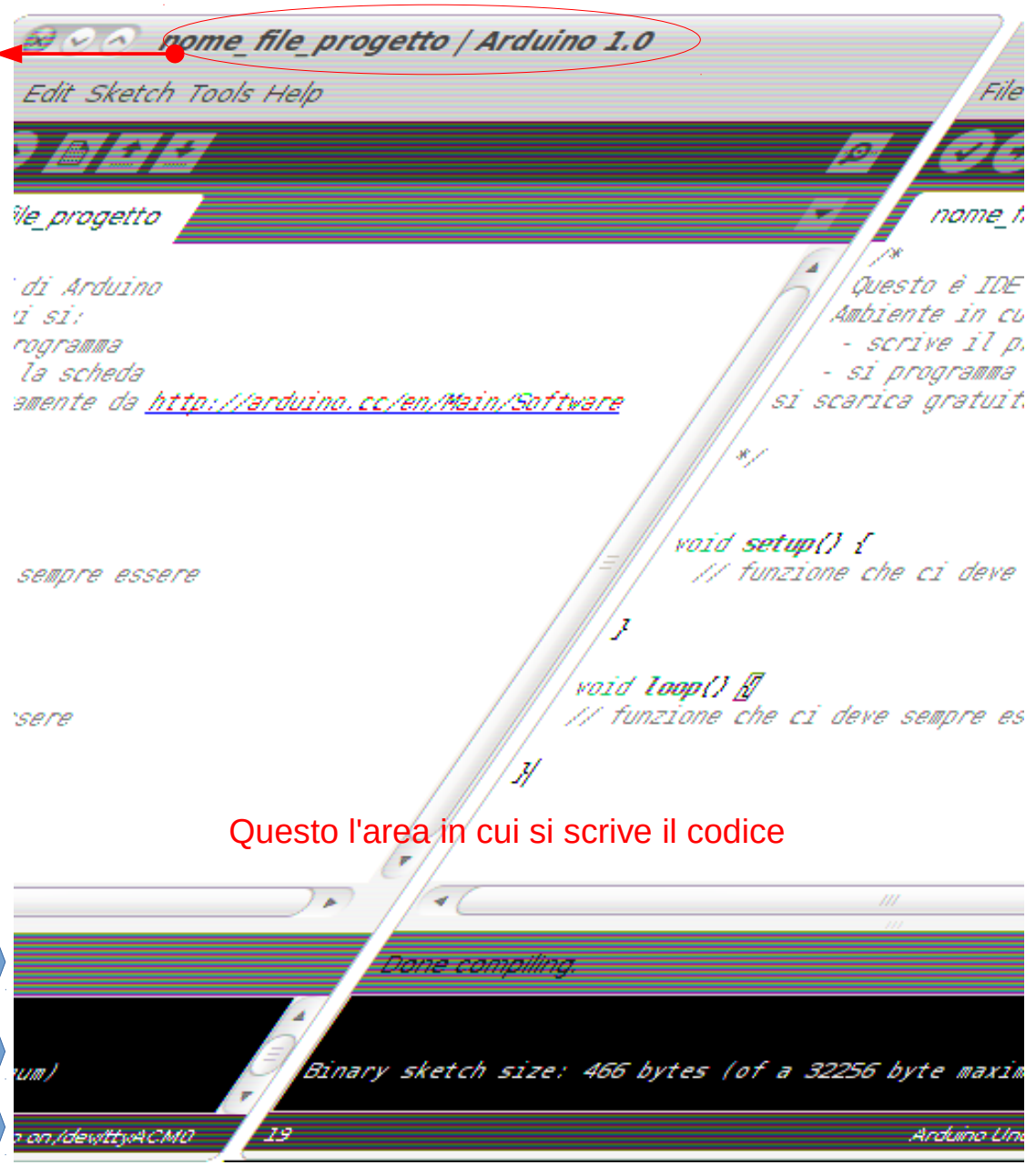
Software Developer Kit di Arduino

Per programmare dobbiamo avere tra le mani qualche cosa che permetta di farlo, questo è Software Developer Kit (SDK) di Arduino.
Nel SDK fanno parte l'ambiente IDE grafico di Arduino, il compilatore per AVR che genera il codice macchina per il micro e delle *parti* di software di permettano di gestire la schede e con delle funzioni base.
Tutto SDK è **Opensource** e **multipiatatforma**.

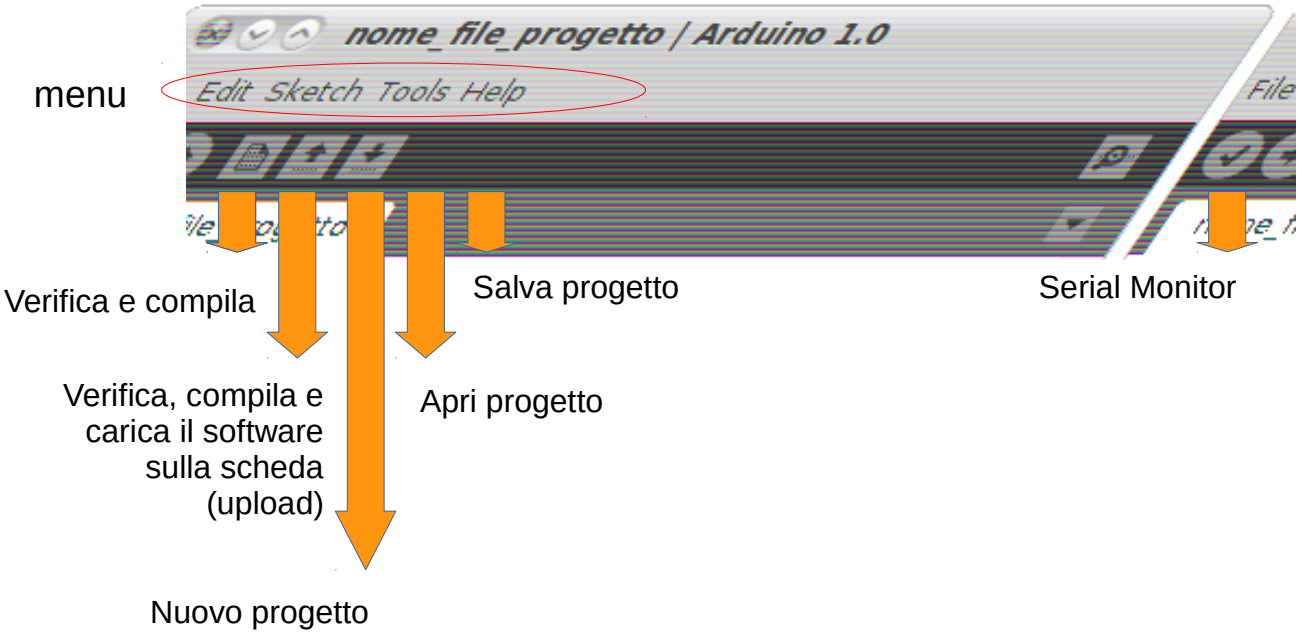
Si scarica gratuitamente da http://arduino.cc/en/Main/Software per il vostro sistema operativo Linux, Mac OS X, Windows.

Si decomprimere il file .zip salvando la cartella (*arduino-versione*) sul harddisk.
Eseguire il file **arduino** dalla directory decompressa... ecco l'ambiente di programmazione Arduino

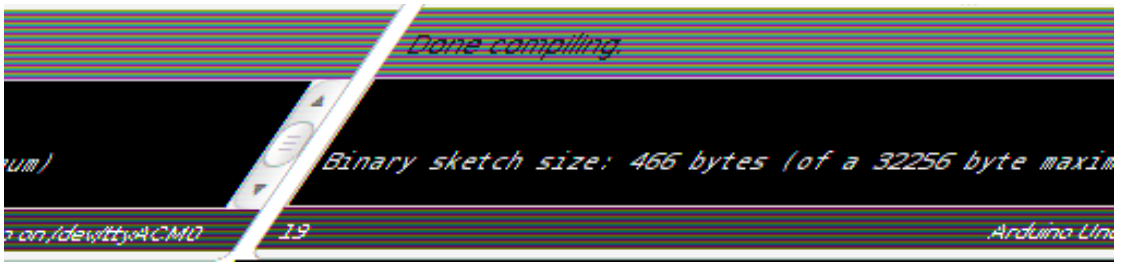
Integrated Development Environment (IDE) di Arduino



Integrated Development Environment (IDE) di Arduino



Questo l'area in cui si scrive il codice



Integrated Development Environment (IDE) di Arduino

I progetti vengono salvati in una sotto directory chiamata **sketchbook**
nella vostra directory personale, per esempio

per linux: /home/nome_utente/sketchbook
per windows: Documenti/sketchbook

A loro volta vengono organizzati con una ulteriore sottodirectory
nome_dato_al_progetto e un file **nome_dato_al_progetto.ino**

per esempio, progetto *corso*

sketchbook
| → *corso*
| → *corso.ino*

.ino è l'estensione della versione 1.0.
Le precedenti versioni (00nn) avevano l'estensione **.pde**
Dal IDE menu **File >>> Preferences** potete cambiare la directory di salvataggio
Le impostazione del vostro ambiente IDE si trovano in un file dal nome
preferences.txt

Integrated Development Environment (IDE) di Arduino

Per eseguire l'ambiente IDE di Arduino è richiesto **Java**.
Pertanto deve essere installato il JRE o JDK per utilizzarlo.
http://www.oracle.com/technetwork/java/index.html

IDE è scritto con il linguaggio opensource **Processing**
http://processing.org

Processing *è un linguaggio di programmazione basato su Java, che consente di sviluppare diverse applicazioni come giochi, animazioni e contenuti interattivi. Da Java eredita completamente la sintassi, i comandi e il paradigma di programmazione orientata agli oggetti, ma in più mette a disposizione numerose funzioni ad alto livello per gestire facilmente l'aspetto grafico e multimediale. È distribuito sotto licenza Open Source, ed è supportato dai sistemi operativi GNU/Linux, Mac OS X e Windows.*

Da http://it.wikipedia.org/wiki/Processing

Invece come come linguaggio di programmazione non è ne Java ne Processing ...

Il linguaggio di programmazione

Per programmare la nostra board dobbiamo usare un linguaggio di programmazione, per esempio AVR? anche. Ma Arduino è una piattaforma “semplice” e per la programmazione si usa il linguaggio **Wiring** http://wiring.org.co/

Wiring è un linguaggio di programmazioni opensource per microcontrollori derivato dal C/C++.
Oltre ad essere un linguaggio di programmazione è anche un IDE basato anche esso su *Processing*. La sintassi è simile al C++. Prima della compilazione il codice scritto viene convertito in un file temporaneo in c++ e poi passato al compilatore per la compilazione.

Il linguaggio di programmazione

Il linguaggio è **case sensitive,** dove a minuscola è **diverso** da A maiuscola

i commenti si scrivono con ***//*** su una sola riga oppure si usa per aprire ***/**** il mio commento ****/*** per chiudere. Questi servono per documentare il codice e renderlo più leggibile

Con la parola riservata **#include *nome_libreria*** si indica che volgiamo usare una libreria software nel programma. Il compilatore caricare il file sorgente della libreria prima della compilazione e compila tutto il codice assieme

Il linguaggio di programmazione

Lo scopo delle librerie software è quello di fornire una collezione di entità di base pronte per l'uso ovvero riuso di codice, evitando al programmatore di dover riscrivere ogni volta le stesse funzioni o strutture dati e facilitando così le operazioni di manutenzione.
Da http://it.wikipedia.org/wiki/Libreria_%28software%29

Ritornando all'IDE... sotto la directory *arduino-versione* trovare alcune sottodirectory, una di queste è **libraries** che **contiene le varie librerie** per gestire hardware e funzionalità software (funzioni).

Chiunque può scrivere le libreria e metterle a disposizione.
Per usa queste librerie basta **copiarle** sotto *libraries*. Nel LAB5 vedremo come installare e utilizzare la libreria Tone.

Dai un occhiata dal IDE Arduino **Menu >>> Sketch >>> Import Library** per le librerie già disponibili. Ci sono già: per gestire bus SPI, Wire; EEPROM per leggere e scrivere la EEPROM; SD (diverse versioni) ; LCD; Ethernet; Servomotori...

Vai su **File >>> Examples** per gli esempi

Il linguaggio di programmazione

L'elenco delle librerie è vasto e vanno dalla gestione della hardware a delle funzioni software per velocizzare la scrittura di funzioni ripetitive o per fare determinate cose. In questo caso dobbiamo essere grati a chi le ha sviluppate e rese disponibile

Potete trovare librerie che fanno la stessa cosa, con caratteristiche simili e prestazioni diverse.
Un esempio sono le librerie per gestire Secure Digital. Qui ci sono quelle che permetto di lavorare sono:
- con un tipo di *file system*, per esempio FAT16 o sia FAT16/32;
- che possono formattare la SD oppure no;
- che posso leggere e scrive un file ma non crearlo (dovete crearlo vuoi il file vuoto a parte sul pc)...

Vi verrà di pensare di usare quella che fa tutto, si ma questa occupa più spazio di memoria nel micro che è già limitata. Quante volte formattereste la SD da Arduino? Allora vi serve sprecare lo spazio in memoria delle funzioni dedicate alla formattazione? Quante volte creerete un file invece di usare solo sempre lo stesso?

Pertanto conviene tenerne conto quando la memoria del micro e le prestazioni sono un fattore importante quale libreria usare quando è possibile scegliere.

Anche il fattore della semplicità di uso della libreria può essere un fattore per la scelta.

Il linguaggio di programmazione

Le funzioni software

In informatica, una funzione è un costrutto che permette di raggruppare una sequenza di istruzioni in un unico blocco di istruzioni che fanno parte di un programma espletando così una determinata operazione sui dati del programma stesso.

Da http://it.wikipedia.org/wiki/Funzione_%28informatica%29

Il linguaggio di programmazione

*Le **funzioni** si dichiarano con con la parola riservata **void**, che è un tipo di dato.*
*Ci posso essere più funzioni all'interno del programma, ma **devono** essere presenti queste due funzioni:*

setup() *viene eseguita una sola volta, all'avvio del programma, che può essere utilizzata per definire delle impostazioni del programma che non verranno più cambiate nel corso della sua esecuzione. Può contenere richiami ad altre funzioni*

loop() *in questa funzione viene eseguito il nostro programma dalla prima all'ultima per poi ricominciare dalla prima fino alle spegnimento. Può contenere al suo interno richiami ad altre funzioni*

Si posso definire a piacimento altre funzioni per rendere il programma più leggibile e più modulare. Uso delle funzioni per suddividere i vari compiti del programma è un tecnica consigliata in moda da tenere loop() più semplice. In questo modo si semplifica anche la scoperta di bug.

Il linguaggio di programmazione

*Le funzioni si **definiscono** in questo modo*

```
void nome_funzione() {
   // questa è un commento alla funzione
   codice_della_funzione;
   ....;

}
```

Il linguaggio di programmazione

Esempio di blink.ino

```
void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}
```

```
void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Il linguaggio di programmazione

Ogni istruzione **deve** terminare con il carattere di **punto e virgola ;** tranne nel caso di una una operazione di condizione o funzione che vengono aperte da **{** e chiusa da **}**

```
prima istruzione;
seconda istruzione;

if ( condizione ) {
    se vera allora esegui questo;
}
... istruzione;
```

Il linguaggio di programmazione

Nel caso delle condizioni le parentesi **{** e **}** casi indicano il blocco da eseguire se quella condizione è vera

```
if ( condizione  ) {
     se vera allora esegui questo;
     poi questo;
}
... istruzione; // poi continua con questo o se la condizione è falsa
```

Ci sono dei casi in cui possono essere omesse se il blocco per esempio nel caso di un **if/else**. In questo caso else chiude/apre il blocco precedete/successivo.

Il linguaggio di programmazione

Operatori aritmetici

= assegna un operatore (per es. variabile = 1)
+ somma
- sottrazione
***** moltiplicazione
/ divisione
% Calcola il resto quando un intero è diviso da un altro.

Operatori di comparazione

== uguale a...
!= non uguale a...
< minore di...
> maggiore di...
<= minore o uguale a...
>= maggiore a...

Il linguaggio di programmazione

Operatori composti

servono per velocizzare e semplificare la scrittura del codice

++ Incremento
-- decremento

Come si usano

dipende da dove si trova la variabile rispetto all'operatore, esempio con ++

```
v=1;
x=v++; // x == 1 e v == 2      : equivale a  x=v; v=v+1;
x=++v; // x == 2 e v == 2      : equivale a  v=v+1; x=v;
```

Altri operatori composti

```
+= somma composta              : equivale a x = x + y;
-= sottrazione composta        : equivale a x = x - y;
*= Moltiplicazione composto    : equivale a x = x * y;
/= Divisione composta          : equivale a x = x / y;
```

Consultate la pagina di cui sotto per un elenco completo
http://arduino.cc/en/Reference/HomePage

Operatori booleani

&& ***and*** *se tutte le condizioni sono vere...*

se la condizione1 è vera ***E*** *la condizione2 è vera allora esegui il { blocco }*

```
if (x==2) && (y<5)  {
        blocco;
 }
```

|| ***or*** *se una delle condizioni è vera...*

se la condizione1 è vera ***O*** *la condizione2 è vera allora esegui il { blocco }*

```
if (x==2) || (y<5)  {
        blocco;
 }
```

! ***not*** *se non è*

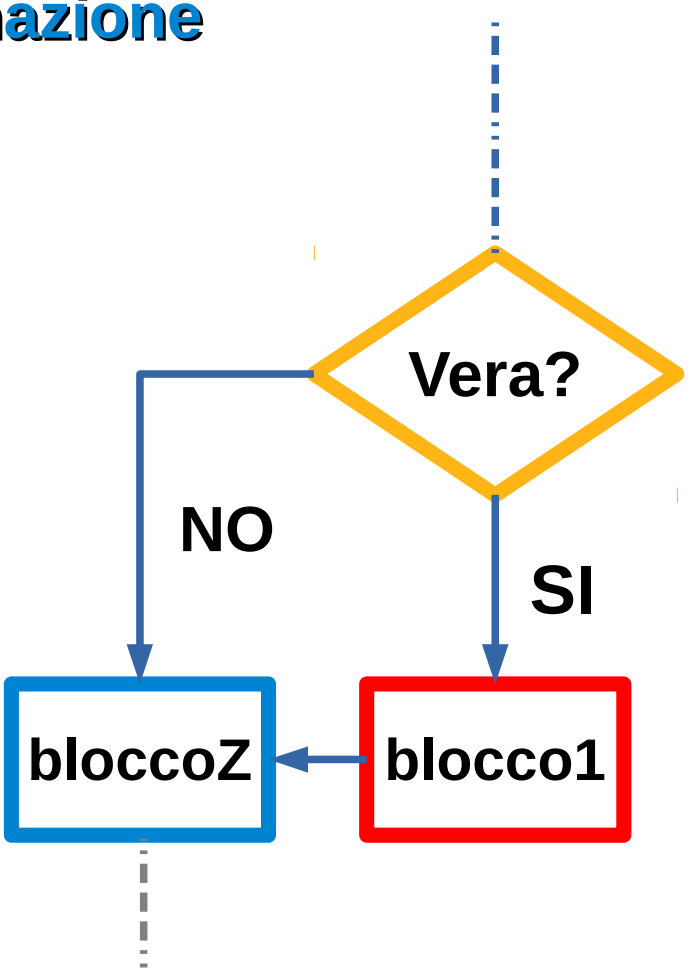
vera se ***l'operando (x) è falso*** *{ blocco }*

```
if (!x)  { // con ! è come se 0 (false) divetta 1 (true) e 1 diventa 0
        blocco;
 }
```

Operatori di condizioni

if: se la condizione è vera allora fai questo

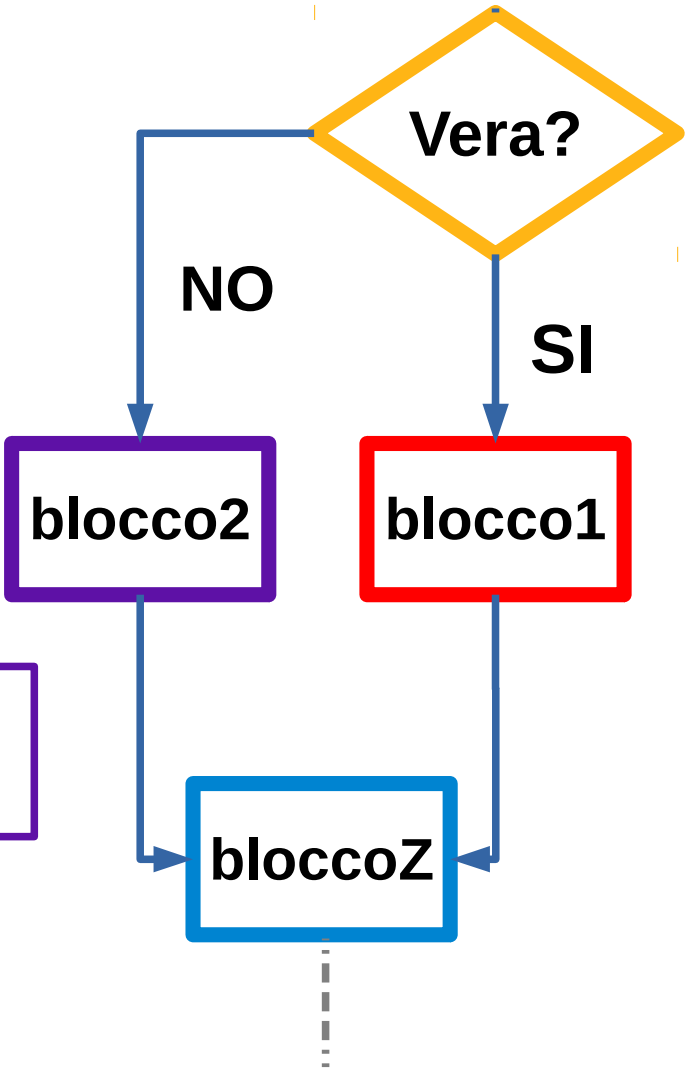
```
if ( condizione  ) {
    // se vera allora esegui questo blocco
    blocco1;
}
// prosegui con le altre istruzioni
bloccoZ;
```



Operatori di condizioni

if se la condizione è vera allora fai questo
else allora è falsa fai quest'altro

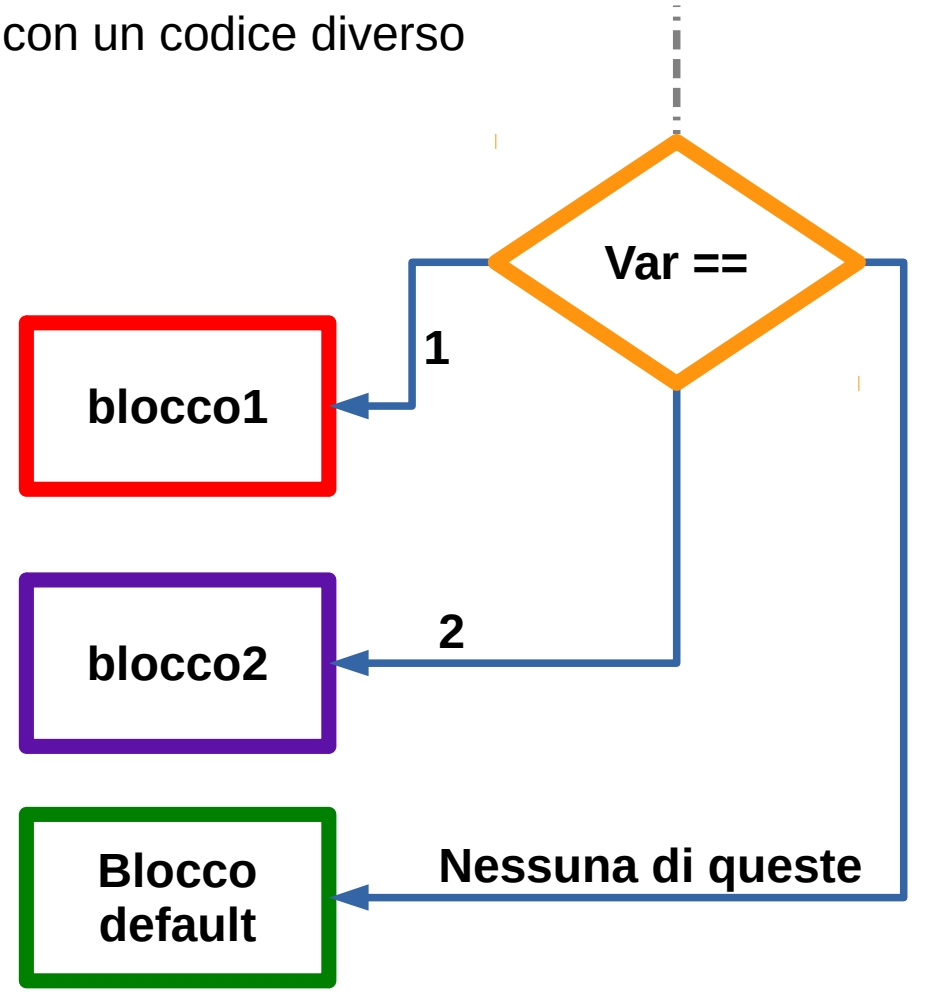
```
if ( condizione ) {
    // se VERA allora esegui questo blocco
    blocco1;
}
else {
    // se FALSA allora esegui questo blocco
    blocco2;
}
// prosegui con le altre istruzioni
bloccoZ;
```



Operatori di condizioni

switch case permette di gestire **più condizioni** con un codice diverso se la condizione è vera. È come un interruttore

```
switch (var) {
  case 1:
    //se var è uguale a 1 esegui blocco1
    blocco1;
    break; // esce da switch e passa al bloccoZ
  case 2:
    //se var è uguale a 2 esegui blocco2
    blocco2;
    break; // esce da switch e passa al bloccoZ
  default:
    //per tutti gli altri valori esegui il blocco_default
    blocco_default;
}
```



/* All'uscita dal blocco il controllo ripassa a **switch case**, si può usare **break** per forzare l'uscita e far eseguire il codice che segue switch. */

bloccoZ

Il linguaggio di programmazione

Qui sotto vede due blocchi di codice che che fanno **esattamente** la stessa cosa.
Il blocco **blu** usa **switch case** mentre il **rosso** usa **if/else.**

```
switch (var) {
    case 1:
      //se var è uguale a 1 esegui blocco1
      blocco1;
      break;
    case 2:
      //se var è uguale a 2 esegui blocco2
      blocco2;
      break;
    default:
      //per tutti le altre condizione
      // il blocco_default
      blocco_default;
      break;
   }

...
```

```
if (var==1) {
    //se var è uguale a 1 esegui blocco1
    blocco1;
  }

else if (var==2) {
    //se var è uguale a 2 esegui blocco2
    blocco2;
  }

else  {
    //per tutti le altre condizione
    // il blocco_default
    blocco_default;
  }
...
```

Il codice **switch case** è da preferire per una migliore leggibilità del codice.

Il linguaggio di programmazione

Scrive un programma si può seguire vari vie che portano allo stesso risultato come abbiamo visto nella slide precedente. Scrivere codice è creatività e arte.

Cercare di scrivere il codice sempre nel migliore del modi comporta:

Vantaggio nella leggibilità dello stesso anche con l'aiuto dei commenti. Questo comporta effettuare aggiornamenti a che a lungo termine e la ricerca di bug sono più facile.

Cercare di usare e scrivere nuove funzioni durante la scrittura del codice invece di mettere tutto dentro una, per esempio loop(). Cercare di rendere le funzioni piccole e specifiche. Usare anche le ***class*** nella programmazione ad oggetti.

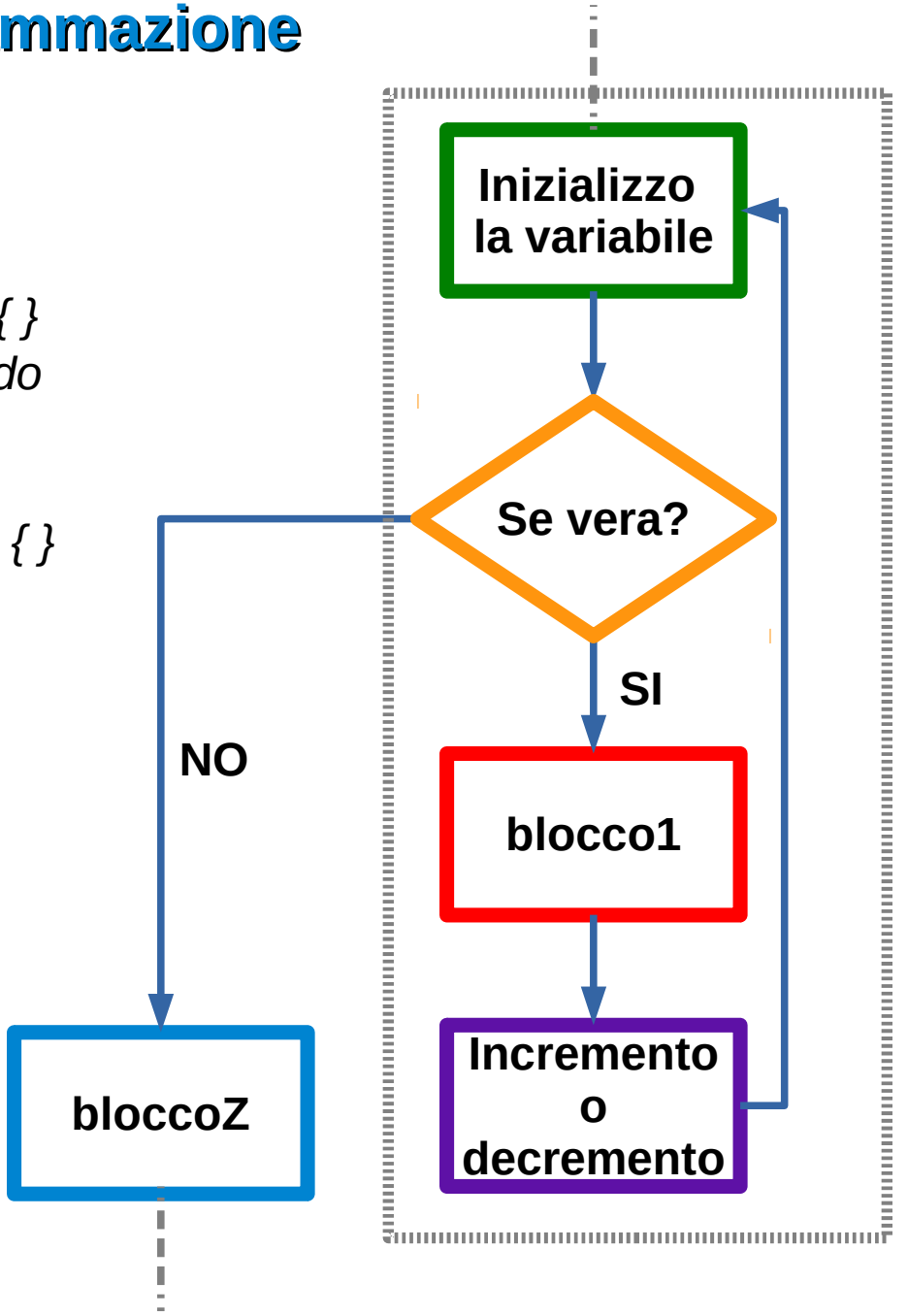
Spesso scrivere buon codice porta anche alla sua ottimizzazione finale sia come dimensioni che come prestazioni generali. A volte va valuta facendo delle prove quali soluzioni da migliori risultai. Questo dipende anche dal tipo di compilatore usato.

Operatori di condizioni

Il ciclo **for** serve per ripetere ciclicamente un blocco{ }
Imposta valore numerico di riferimento e fino a quando rispetta la condizione() viene eseguito il blocco{ }

for (inizializza variabile; condizione; ++ o -- variabile) { }

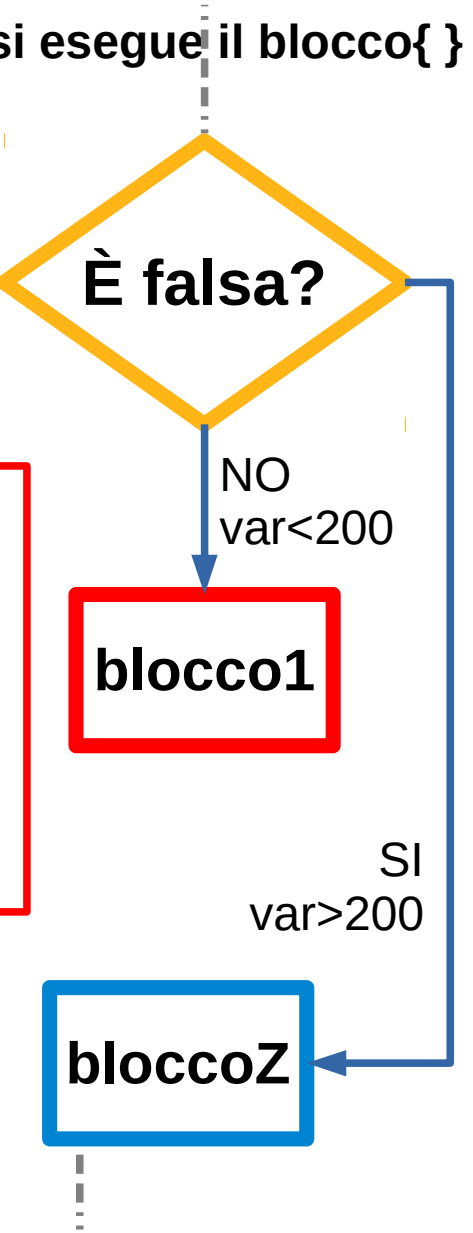
```
for ( int x = 0; x < 10; x++ ) {

    blocco1; // dentro for

}

bloccoZ // fuori da for
```



Operatori di condizioni

while come prima cosa viene **verificata** la **condizione; se vera si esegue il blocco{ } in loop** fino quando al condizione **non diventa falsa.**
Si può uscire anche con **break**

```
var = 0;

while (var < 200)  {

  /*
    blocco1
    var è minore di 100
    la somma incrementale di cui sotto viene eseguito per 200 volte
    poi var diventa maggiore di 200 e il ciclo while termina
  */
    var++; // ++ equivale a scrivere var=var+1

}

bloccoZ; // var è maggiore di 200
```

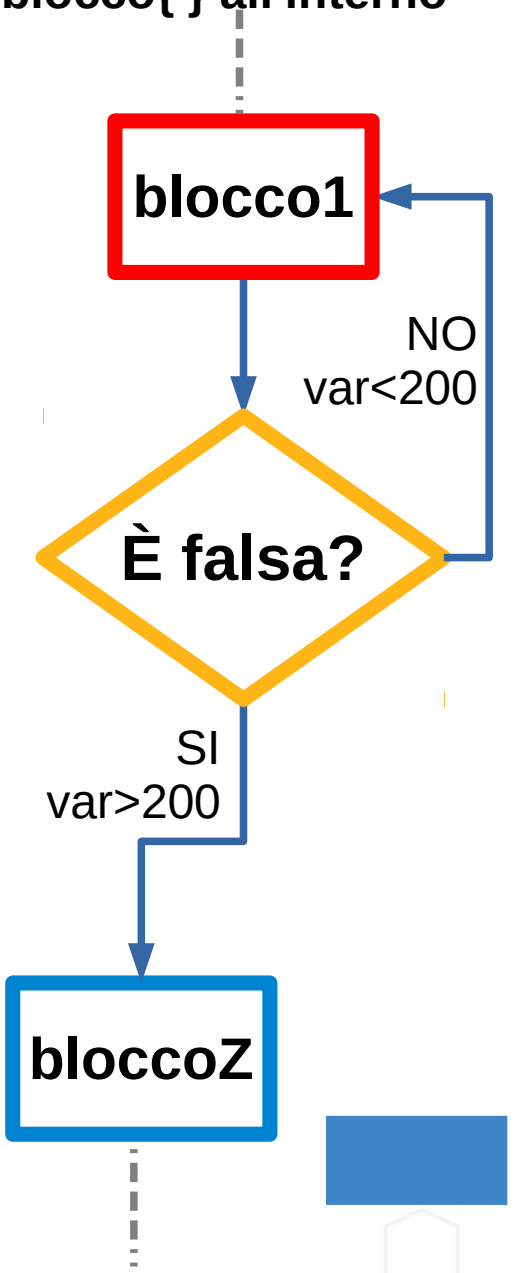


Operatori di condizioni

do..while la differenza rispetto al solo **while** è che in questo caso il **blocco{ } all'interno della condizione viene eseguito almeno una volta**

var = 0;

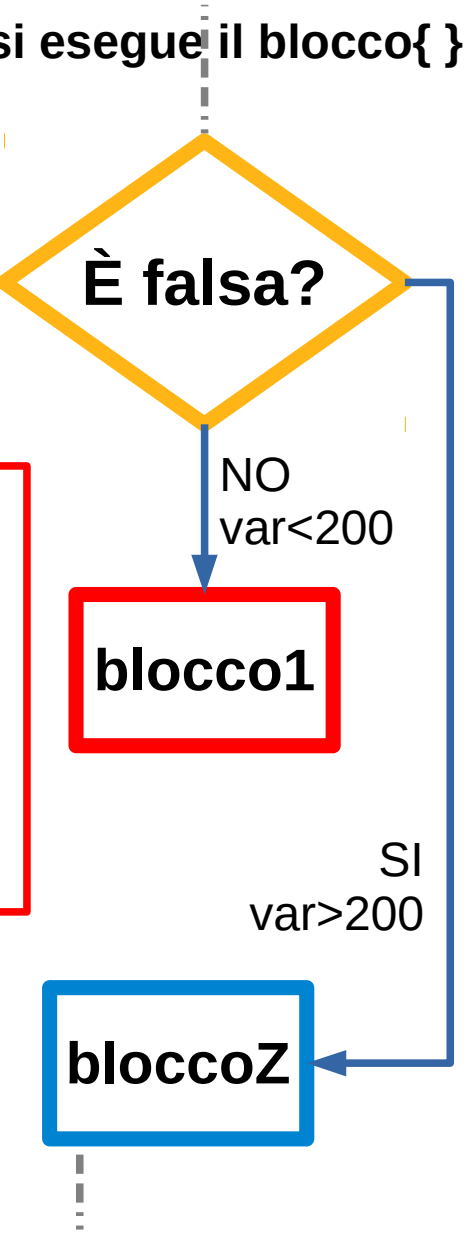
```
do {
  /*
    blocco1
    var è minore di 100
    la somma incrementale di cui sotto viene eseguito per 200 volte
    poi var diventa maggiore di 200 e il ciclo while termina
  */
    var++; // ++ equivale a scrivere var=var+1
} while (var < 200);

bloccoZ; // var è maggiore di 200
```



Operatori di condizioni

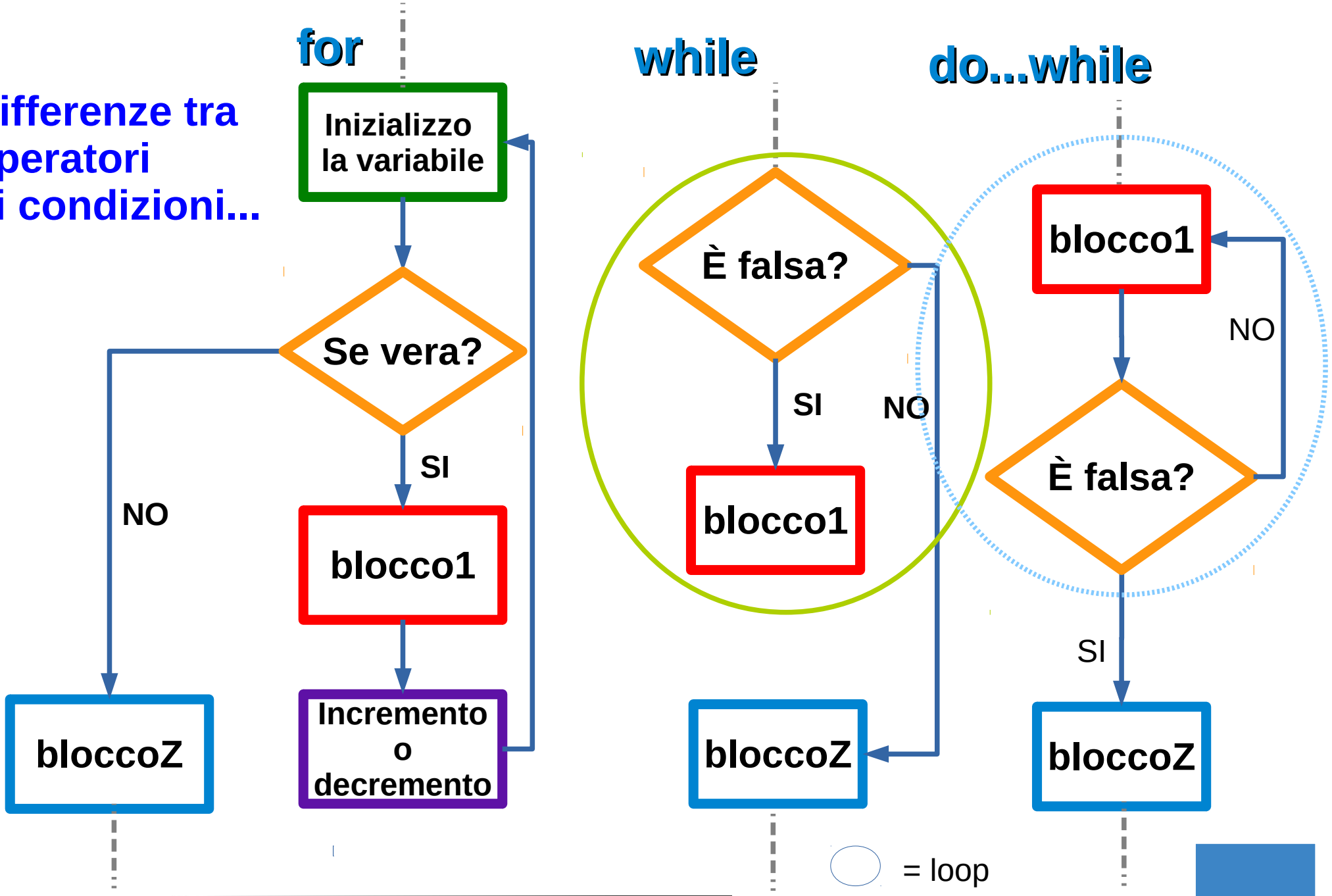
while come prima cosa viene **verificata** la **condizione; se vera si esegue il blocco{ } in loop** fino quando al condizione **non diventa falsa.**
Si può uscire anche con **break**

```
var = 0;

while (var < 200)  {

  /*
    blocco1
    var è minore di 100
    la somma incrementale di cui sotto viene eseguito per 200 volte
    poi var diventa maggiore di 200 e il ciclo while termina
  */
    var++; // ++ equivale a scrivere var=var+1

}

bloccoZ; // var è maggiore di 200
```



Il linguaggio di programmazione

Differenze tra operatori di condizioni...



Altri Operatori

break viene usato per uscire da un do, for, e while, interrompendo la condizione normale ciclo. Viene anche usato per uscire da un'istruzione switch case

continue salta il resto della iterazione corrente di un loop (do, for e while) Si prosegue verificando l'espressione del ciclo, e procedere con qualsiasi iterazioni successive.

return terminata la funzione ritorna un valore (per esempio un risultato) alla funzione chiamante, se richiesto.

goto trasferisce flusso del programma ad un punto etichettato nel programma.
// ndr che brutta istruzione goto :-)

riferimenti http://arduino.cc/en/Reference/HomePage

Variabili, I tipi di dati

I **tipi di dati** sono delle proprietà delle variabili o oggetto che identificano cosa contengono. Vanno dichiarati prima di usarli.

void: dichiarazione di una funzione
boolean: due soli valori **true** e **false**
char: lungo un solo byte memorizza valori numerici della tabella ASCII
byte: lunga un byte e memorizza valori numerici da 0 a 255
unsigned char: lungo un byte senza segno e come il tipo **byte**
int: lungo due byte e contiene valori numerici da -32768 a +32768
unsigned int: come int ma senza segno, da 0 a 65535
word: come il tipo **unsigned int**
long: lungo 4 Byte e contiene valori numerici da -2,147,483,648 a 2,147,483,647.
unsigned long: come **long** ma senza segno 4,294,967,295 (2^32 - 1).
float: occupa 4 byte e memorizza numeri in virgola mobile.
da 3.4028235E+38 a 3.4028235E+38
double: si comporta come tipo **float**. NON è un doppia precisione come in altri linguaggi per esempio il C/C++
array: memorizzano insiemi di valori dello stesso tipo
string - char array: memorizza insieme di valori di tipo char
String - object: Lavora sulle stringhe. È utilizzato per manipolare, concatenare, ricercare e sostituire stringhe

Variabili, le costanti

Le costanti sono variabili predefinite nel linguaggio Arduino. Sono utilizzati per rendere i programmi più facili da leggere. Si classificano in gruppi costanti.

HIGH indica uno stato a **1**. Funziona in due modi:
se il pin è **OUTPUT** allora avremo su quel pin 5V;
se il pin è **INPUT** con una tensione superiore a 3V allora leggiamo uno stato a 1
LOW indica uno stato a **0**. Funziona in due modi:
se il pin è **OUTPUT** allora avremo su quel pin 0V. Può assorbire corrente se fa da GND per un circuito, per esempio il catodo di un LED è collegato su questo pin e anodo a un punto a 1 (vcc);
se il pin è **INPUT** con una tensione inferiore ai 2V allora leggiamo uno stato a 0

INPUT | OUTPUT modo di utilizzo dei pin. Vengono settati con pinMode(). Con INPUT leggiamo lo stato; se OUTPUT impostiamo l'uscita (max 40mA)
true | false condizione che rappresentano la verità (equivale a **1)** e la falsità (**0**) nel linguaggio

integer constants Sono valori interi inseriti direttamente in uno sketch, come 123.
floating point constants sono costanti a virgola mobile

Riferimenti: http://arduino.cc/en/Reference/Constants

Alcune funzioni prestabilite

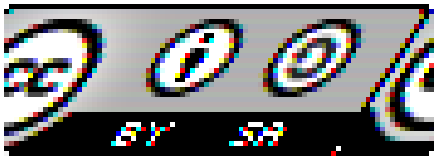
Digital I/O

pinMode() definisce se in pin è:
INPUT allora useremo **digitalRead()** per **leggere** lo stato
OUTPUT allora useremo **digitalWrite()** per **scrivere** lo stato

Analog I/O

analogRead() leggiamo un valore analogico sui pin analog (A0, A1, A2,...)
analogWrite() scriviamo un uscita PWM ~

analogReference()
Tensione di riferimento per la tensione del ADC (AREF) . É la tensione massima dei ingresso analogici, da 0 a analogReference. Può assumere i seguenti valori
DEFAULT: se non specificata è a 5V o 3.3V (per 3.3V Arduino boards)
INTERNAL: riferimento interno a 1.1V per ATmega168 o ATmega328; 2.56V per ATmega8; non presente su Arduino Mega
INTERNAL1V1: riferimento interno a 1.1V per Arduino Mega
INTERNAL2V56: riferimento interno a 2.56V per Arduino Mega
EXTERNAL: la tensione applicata sul pin a AREF (0 to 5V) diventa quella di riferimento.



Grazie a tutti dell'attenzione
E un particolare ringraziamento al
Comune di Voghiera

Comune di Voghiera *- Paolo Coatti -*

Provincia di Ferrara