ARDUINO partendo da ZERO

Probabilmente qualcuno ha già sentito il nome ARDUINO, e molti lo associano ad una scheda elettronica con cui realizzare dei progetti. In realtà non è esattamente così e per spiegare cosa rappresenta questo nome, dobbiamo fare un piccolo passo indietro nel tempo.

Nel 2003 ad Ivrea era operativa la **Interaction Design Institute,** una scuola post-laurea, nata dall'iniziativa di Olivetti e Telecom Italia. Nel corso della sua attività sono stati portati avanti diversi progetti software ed hardware, tra cui la nascita del progetto Arduino.

Nel 2003 dei membri di questa scuola, cominciano a pensare ad una piattaforma hardware e software con cui realizzare veloci prototipi per scopi hobbystici, ma anche professionali.

Il team composto da; Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Melli realizzano così nel 2005 la piattaforma hardware e software ARDUINO. Il nome nasce da un bar ritrovo frequentato dagli ideatori del progetto, nome a sua volta derivante dall'omonimo Re d'Italia di Ivrea del 1002. Non possiamo perciò non considerare ARDUINO una creatura Italiana.

La definizione esatta associabile a questo nome perciò è quella menzionato sopra, cioè **una piattaforma hardware e software.** In maniera concreta esso la piattaforma ARDUINO composto da una scheda elettronica programmabile a microcontrollore e da un software per realizzare i vari progetti.

La scheda elettronica, è composta dalle seguenti parti:

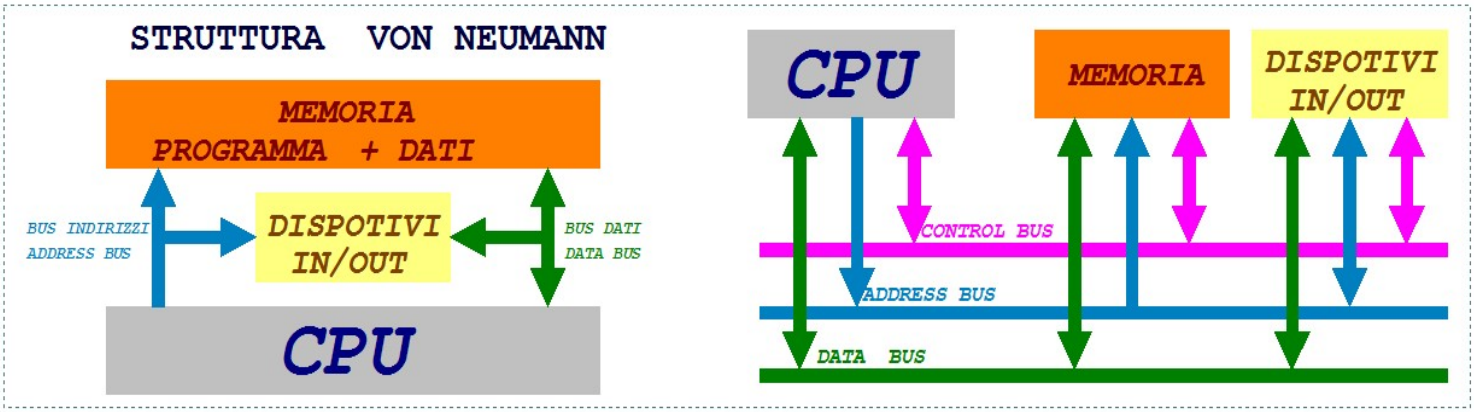
- Un microcontrollore ATMEL, cioè un un'unità di elaborazione e calcolo contenente una CPU, una RAM ed una FLASH (cioè una memoria non volatile)
- un circuito di alimentazione della scheda composto da un regolatore di tensione, che rende la scheda alimentabile da apposito alimentatore o da porta USB,
- un circuito convertitore USB-RS232,
- dei PIN (piedini di collegamento) connessi alle porte GPIO (**G**eneral **P**urpose **I**nput ed **O**utput) del microcontrollore
- dei PIN connessi agli ingressi analogici del microcontrollore.

Non tutti conoscono il significato dei termini menzionati sopra, pertanto è doveroso un piccolo approfondimento.

IL MICROCONTROLLORE

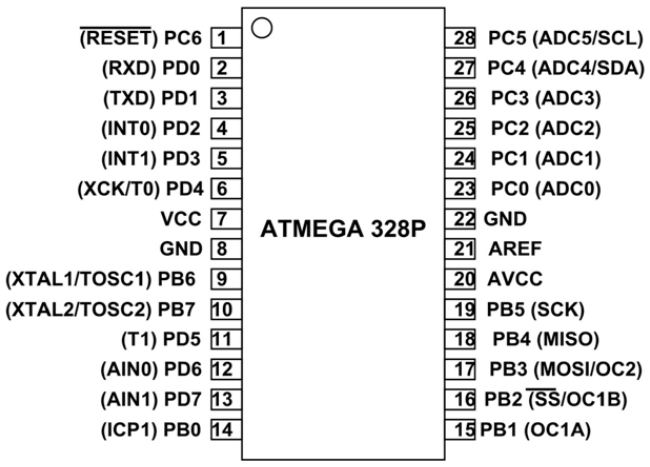
Sicuramente quasi tutti sanno che all'interno di un PC c'è un processore, o meglio un **microprocessore**, definito anche **CPU** (**C**entral **P**rocessing **U**nit). Questo è il componente che decodifica ed esegue le istruzioni impartite dal programmatore e scritte in un programma. Il microprocessore del PC utilizza altri dispositivi esterni ad esso, come ad esempio una memoria RAM (una memoria in grado di contenere dati, che può essere letta e scritta, ma che allo spegnimento perde tutto il suo contenuto, cioè **una memoria volatile**) ed una **memoria non volatile** dove salvare i nostri programmi ed i nostri dati, che nel PC è costituita dall'hard disk. Il nostro PC ha inoltre dei dispositivi di **input ed output**, come ad esempio la tastiera, il mouse, in monitor e cosi via, cioè tutti quei dispositivi che ci permettono di interagire con la macchina e con il microprocessore.

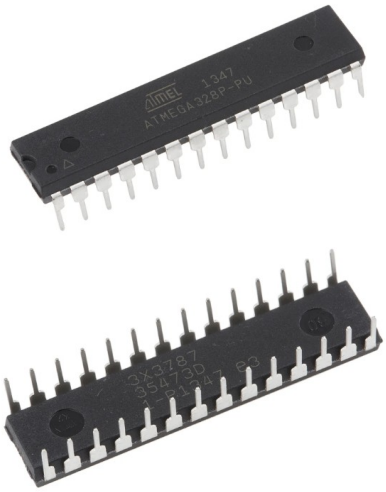
L'architettura di questo sistema potrebbe essere sintetizzata con la seguente immagine che rappresenta una struttura chiamata Von Neumann, in nome del matematico che l'ha ideata.



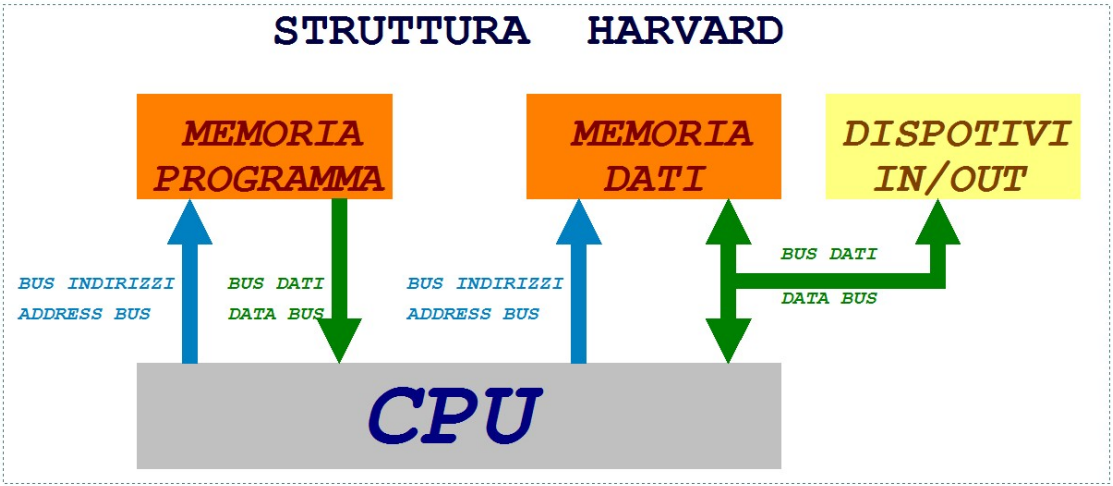
Un microcontrollore invece è un componente che al suo interno possiede i dispositivi prima descritti esterni al microprocessore e cioè; una CPU, una RAM ed una FLASH (una memoria non volatile che, come nel caso dell'hard disk, conterrà il programma da eseguire).

Inoltre un microcontrollore ha dei piedini digitali (cioè dei terminali che possono assumere due soli valori logici 0 ed 1 corrispondenti a due tensioni elettriche definite livello basso e livello alto) e dei piedini analogici, cioè dei terminali a cui possiamo collegare dei valori di tensione che possono assumere tutti i valori compresi tra il livello basso ed il livello alto. Insomma possiamo in estrema sintesi definire il microcontrollore come se fosse un piccolo PC integrato in un unico componente.



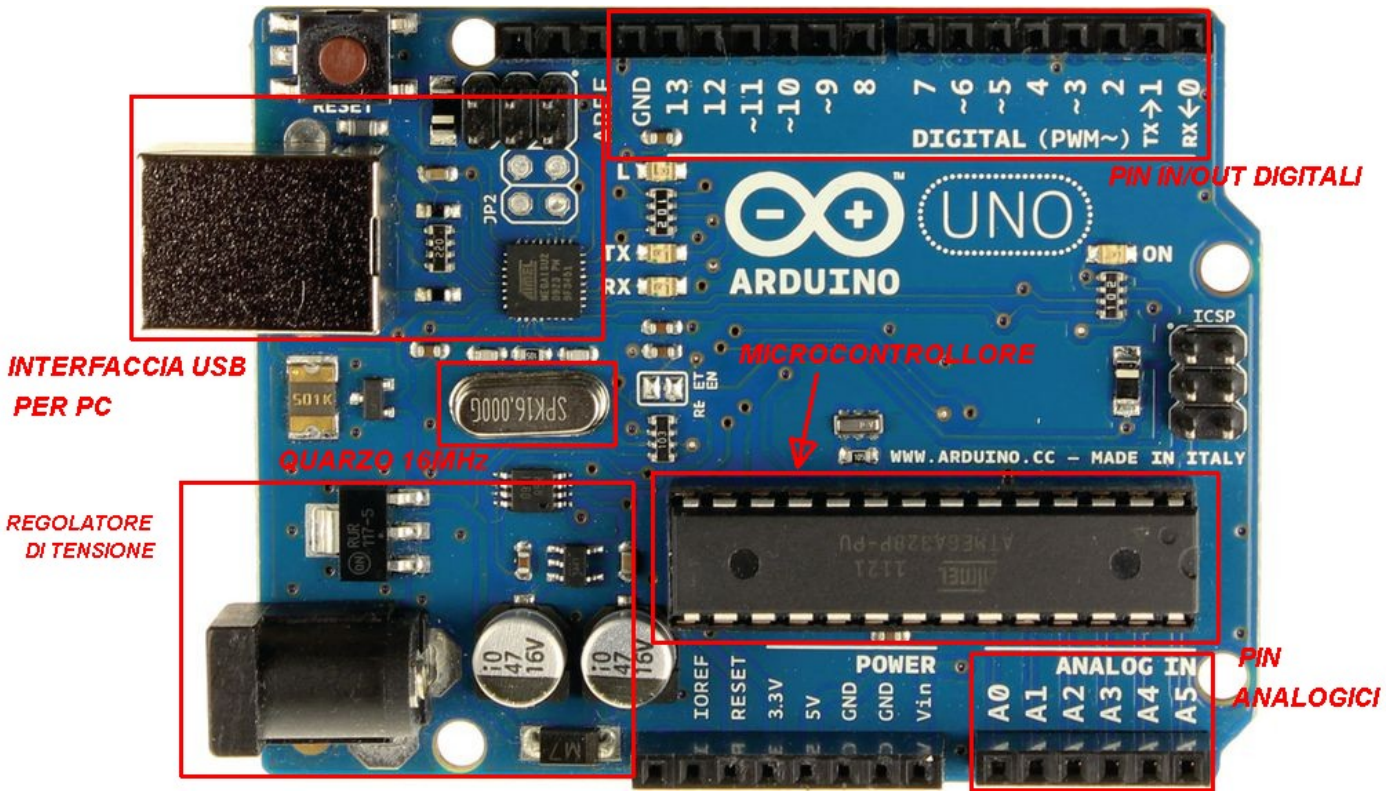


Il microcontrollore ha una struttura diversa dal microprocessore, esso utilizza una struttura Harvard che prevede due memorie interne differenti.



All'interno del microcontrollore non c'è un'unica memoria condivisa come nel caso del microprocessore, ma una memoria per il programma (memoria non volatile FLASH memory) ed una memoria per i dati (memoria volatire, RAM e registri interni).

La piattaforma ARDUINO, è composta da diverse schede, la più conosciuta si chiama ARDUINO UNO, ed è composta da un microcontrollore della ditta ATMEL, modello ATMEGA328, funzionante con un clock da 16MHz, una RAM da 2kB, una Flash da 32kB ed una EEPROM da 1kB (la EEPROM è anch'essa una memoria non volatile, utilzzata però solo per i dati.



Come si può notare dai valori detti sopra (frequenza del clock e dimensione della memoria) non possiamo pensare di installare su una scheda ARDUINO UNO, un sistema operativo, le sue risorse rispetto ai classici PC sono sicuramente inferiori anzi irrisorie. Questo tipo di scheda non deve comportarsi come un generico PC, ma eseguire delle operazioni ben definite e soprattutto deve eseguire un solo pogramma alla volta.
Siamo oggi abituati a pensare a frequenze di clock dell'ordine dei 3GHz (3 miliardi di Hertz) e dimensioni della RAM di 16GB (16 miliardi di byte) perciò leggere che la scheda ARDUINO UNO lavora ad una frequenza di 16MHz (16 milioni di Hertz) con una RAM da 1kB (1000 byte) ci fa sembrare questa scheda quasi un giocattolo, ma in realtà non è così.
Un normale PC deve offrire la possibilità di far girare diversi programmi contemporaneamente, basti pensare al solo S.O. (**S**istema **O**perativo) Windows, Linux o MacOS, che al semplice avvio, manda in esecuzione diverse decine di programmi in simultanea. Un normale PC deve inoltre gestire un monitor con una risoluzione di milioni di pixel, è ovvio che le risorse necessarie devono essere superiori.
Ma una scheda come ARDUINO UNO, ha una finalità diversa, che è quella di far funzionare una sola applicazione, anche complessa, ma comunque limitata nell'uso delle risorse, per questo motivo la velocità o la memoria sopra descritta è più che sufficiente per gran parte delle applicazioni che vengono sviluppate su questa piattaforma.

Come abbiamo detto, nel mondo ARDUINO esistono altre schede, e con esse si possono realizzare svariati progetti. Basta girare un po' in rete o anche su youtube, e vedere come i tanti sviluppatori e "maker", hanno dato sfogo alla propria fantasia, realizzando a volte dei veri piccoli capolavori della tecnologia.
Sul sito ufficiale di ARDUINO al seguente link https://www.arduino.cc/en/Main/Products troviamo le varie schede disponibili.

Una parte importante e fondamentale della piattaforma ARDUINO è proprio il suo sito https://www.arduino.cc/ .
Qui possiamo trovare tutte le informazioni tecniche ed una completa guida contenente tutte le informazioni hardware e software. Inoltre dal sito possiamo scaricare liberamente la parte software della piattaforma e cioè l'ambiente di sviluppo che andremo più avanti a descrivere https://www.arduino.cc/en/Main/Software .

Tutto l'ambiente è basato sul concetto di open source, troviamo perciò tutte le informazioni dettagliate sul software e sull'hardware, comprese anche degli schemi elettrici.
Perciò se decidiamo di acquistare un kit di ARDUINO ed entrare in questo mondo, il primo passa sarà quello di scaricare ed installare il suo ambiente di sviluppo software definito I.D.E. (**I**ntegrated **D**evelopment **E**nvironment, ambiente di svilupp integrato).

PIATTAFORMA SOFTWARE DI ARDUINO

Nel mondo dell'elettronica e dell'informatica, come un po' in ogni settore, è indispensabile conoscere l'inglese. Anche nel mondo di ARDUINO vige questa regola, in pratica è tutto in inglese, il sito, la guida di riferimento e tutte le informazioni che possiamo trovare, sono rigorosamente in inglese.
Questo particolare non dovrebbe spaventarci più di tanto, perché oltre alla documentazione esistono diversi esempi che ci consentono di capire come realizzare un programma o come utilizzare determinate **librerie**.
Anche in questo caso è doverosa una piccola spegazione per i non addetti ai lavori.

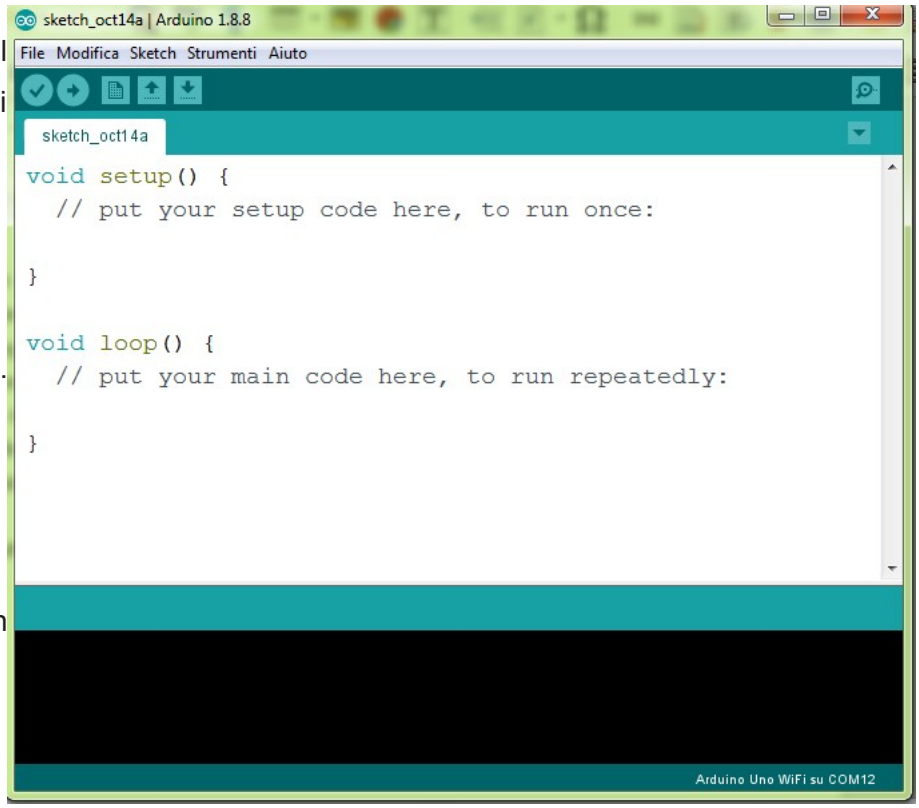
Con il termine librerie, si intedono un insieme di funzioni, cioè delle parti di programma che svolgono determinati compiti.
Ad esempio possiamo trovare la libreria che ci permette di gestire un display a cristalli liquidi, troveremo perciò le funzioni già pronte per eseguire operazioni come; cancellare il contenuto del display, far lampeggiare il cursore in una determinata posizione, scrivere un testo o un singolo carattere e così via.
Queste funzioni vengono richiamate scrivendo il nome ed utilizzando la corretta sintassi, descritta in maniera molto chiara nella guida e nei vari esempi disponibili.

Standard Libraries

- EEPROM - reading and writing to "permanent" storage
- Ethernet - for connecting to the internet using the Arduino Ethernet Shield
- Firmata - for communicating with applications on the computer using a standard serial protocol.
- GSM - for connecting to a GSM/GRPS network with the GSM shield.
- LiquidCrystal - for controlling liquid crystal displays (LCDs)
- SD - for reading and writing SD cards
- Servo - for controlling servo motors
- SPI - for communicating with devices using the Serial Peripheral Interface (SPI) Bus
- SoftwareSerial - for serial communication on any digital pins. Version 1.0 and later of Arduino incorporate Mikal Hart's NewSoftSerial library as SoftwareSerial.
- Stepper - for controlling stepper motors
- TFT - for drawing text , images, and shapes on the Arduino TFT screen
- WiFi - for connecting to the internet using the Arduino WiFi shield
- Wire - Two Wire Interface (TWI/I2C) for sending and receiving data over a net of devices or sensors.

Cominciamo con il dire che la programmazione con la piattaforma ARDUINO, parte dai linguaggi C/C++, che sono ala base dell'ambiente di sviluppo fornito.
Nessuno ci vieta di programmare la scheda ARDUINO UNO con altri linguaggi, ma ovviamente dovremo utilizzare altri ambienti ed altri programmatori.
Utilizzando l'ambiente di sviluppo fornito, rimane tutto più semplice e veloce, perché come già detto vengono fornite delle librerie già pronte per essere utilizzate sulla scheda, ed si può programmare la scheda senza alcun dispositivo esterno.

Una volta installato, possiamo avviare il software e ci troveremo la videata qui di lato.



Saltano subito all'occhio due parti del programma definite con **setup** e **loop**.

Entrambe sono seguite dalla scritta **void**.
Chi conosce un po' il linguaggio C, conosce il significato di questa dicitura.
Chi non lo conosce può per ora ignorarla e ricopiarla così com'è.
Ci basta sapere che la parola **void** in inglese significa, vuoto nullo.
Nel caso specifico questo termine ci indica che la parte di programma che viene descritta successivamente tra le parentesi { e }, non restituisce nessun risultato, viene eseguita e basta.

La parte di programma chiamata **setup,** dovrà essere scritta tra le prime due parentesi graffe e, come recita la scritta in inglese, verrà eseguita una volta sola al'avvio.

"put your setup code here, to run once"

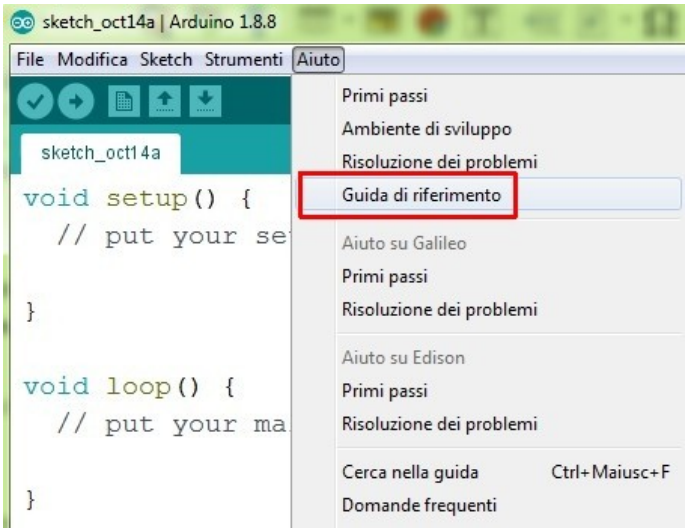
Ci viene perciò consigliato di inserire tra le due parentesi tutto ciò che vogliamo far **eseguire una sola volta all'avvio del programma**, come ad esempio i settaggi o le impostazioni iniziali.

La parte di programma chiamata **loop**, viene invece eseguita ripetutamente e ciclicamente dopo l'avvio.

"put your main code here, to run repeatedly"

Quello che inseriremo tra le due parentesi indicate dal ciclo loop, **verrà invece eseguito ciclicamente**.

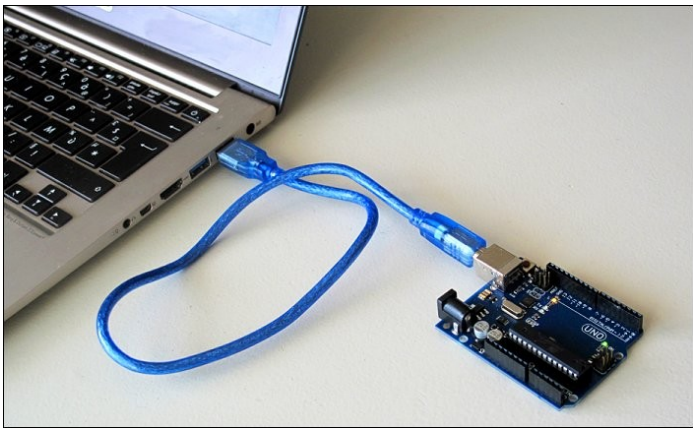
Possiamo a questo punto accedere alla guida online messa a disposizione e studiare i vari comandi e le funzioni. Per fare questo basta accedere alla guida dal menu help.



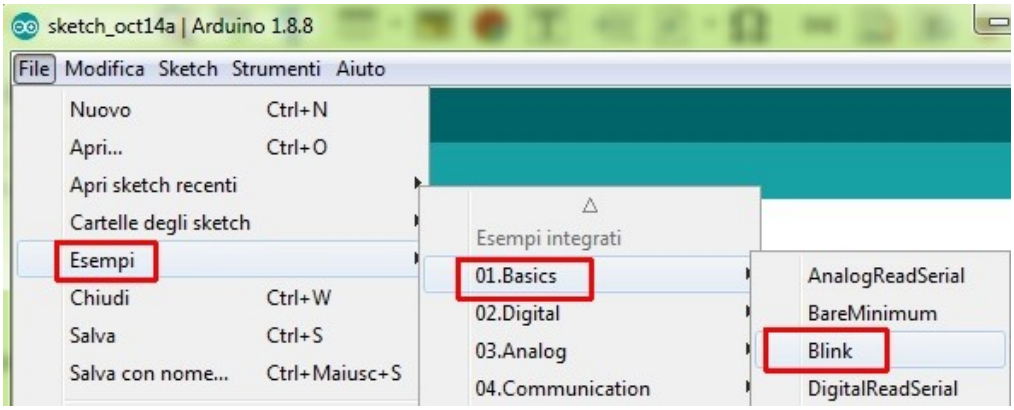
Ma prima di consultare la preziosa guida, consiglio di provare a lanciare il più semplice programma di esempio mandandolo in esecuzione sula scheda.

Le operazioni da fare sono perciò le seguenti:

1. Dopo aver installato il software, collegare la scheda ARDUINO UNO al PC tramite il cavetto USB.



2. Lanciare il software di ARDUINO ed aprire l'esempio **Blink**, sotto alla sezione **Basic**.



A questo punto verrà aperta una nuova finestra contenente il programma **Blink**.

Nel linguaggio C/C++ tutto ciò che è compreso tra i caratteri /* e */ rappresenta un commento, come anche tutto ciò che in una riga viene preceduto dai caratteri //.

All'inizio del programma troveremo perciò una premessa per descrivere il programma compresa tra i caratteri /* e */ . Successivamente troveremo altri commenti preceduti dai caratteri //, tutti i commenti avranno un colore grigio per indicare che non sono istruzioni del programma.

La parte essenziale del programma, cioè ciò che verrà eseguito dal microcontrollore, sarà la seguente:

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Per comprendere il funzionamento consiglio di cancellare tutti i commenti e di sostituire la scritta LED_BUILTIN con il numero 13.
Otterremo perciò il codice qui a fianco.

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

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

Proviamo ora a cercare di capire il significato di questo codice, aiutandoci con la guida di riferimento.

Come abbiamo già detto nella parte setup, viene messo ciò che vogliamo venga eseguito all'avvio.
Nel nostro caso troviamo la seguente istruzione:

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

Cercando **pinMode** sulla guida di riferimento, troviamo quanto segue:

Description

Configures the specified pin to behave either as an input or an output. See the description of digital pins for details on the functionality of the pins.

Syntax

pinMode(pin, mode) **sintassi corretta per eseguire la funzione pinMode**

Parameters

pin: the number of the pin whose mode you wish to set **(numero del piedino che vogliamo impostare)**

mode: INPUT, OUTPUT, or INPUT_PULLUP. (see the digital pins page for a more complete description of the functionality.) **(tipo di impostazione: INPUT, OUTPUT, INPUT_PULLUP)**

Returns

None **(risultato restituito, nessuno)**

Possiamo facilmente intuire che in questo modo abbiamo impostato il piedino numero 13 come un'uscita.
Perciò d'ora in avanti potremo fare in modo, con le apposite istruzioni, che sul terminale 13 della scheda ci possa essere un livello di tensione basso oppure alto.

Se invece avessimo scritto, `pinMode(13, INPUT);`
avremmo impostato il terminale 13 della scheda come un ingresso e pertanto avremmo potuto, con le apposite istruzioni, leggere il livello di tensione presente sul terminale, 0 oppure 5Volt, cioè 0 oppure 1.

Torniamo ora al nostro programma, ed analizziamo ciò che è compreso nella parte **loop**.
Ci sono due istruzioni uguali con parametri diversi e cioè; `digitalWrite(13, HIGH);`

`digitalWrite(13, LOW);`

Anche in questo caso apriamo la guida e leggiamo il significato della funzione **digitalWrite**.

Description

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

Syntax

digitalWrite(pin, value)

Parameters

pin: the pin number

value: HIGH or LOW

Returns

none

La descrizione in inglese è molto chiara, in pratica **digitalWrite**, consente di mettere a livello alto o basso un terminale definito precedentemente con **pinMode,** come uscita. Il livello alto corrisponde ad una tensione di 5V o 3,3V a seconda del tipo di scheda, il livello basso corrisponde invece a 0 Volt.

La sintassi prevede che all'interno delle parentesi venga indicato il numero del terminale (nel nostro caso 13) e dopo la virgola il suo stato HIGH o LOW (alto o basso).

In pratica le due istruzioni che verranno eseguite ciclicamente, servono per mettere a livello alto ed a livello basso il terminale 13 della scheda.
Sulla scheda ARDUINO UNO, troviamo un led definito **LED_BUILTIN**, collegato direttamente al terminale 13, pertanto le due istruzioni accenderanno e spegneranno il led.

Un'altra funzione contenuta nel programma è il **delay**. `delay(1000);`

Description

Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.) **Ferma il programma per il tempo indicato tra le parentesi della funzione. Espresso in millisecondi.**

Syntax

delay(ms)

Parameters

ms: the number of milliseconds to pause (*unsigned long*)

Returns

nothing **La funzione viene eseguita e non vene restituito alcun risultato.**

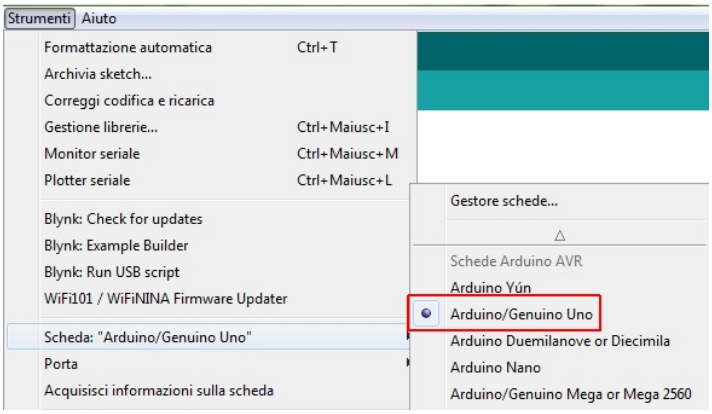
Perciò dopo l'impostazione del terminale 13 come uscita, effettuata nel **setup**, nel ciclo **loop** il terminale 13 ed il LED ad esso collegato, verrà acceso e spento ciclicamente con una pausa di 1 secondo tra l'accensione e lo spegnimento. In sintesi otterremo un semplice lampeggio del LED presente sulla scheda.

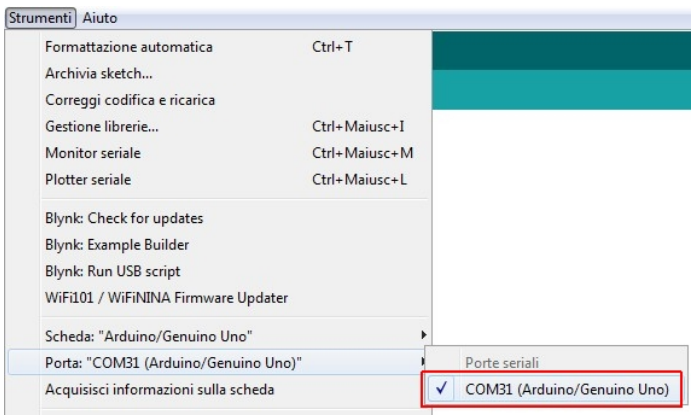
CARICAMENTO DEL PROGRAMMA SUL MICROCONTROLLORE

Prima di caricare il programma dobbiamo indicare sull'IDE di ARDUINO, la presenza della scheda ARDUINO UNO, e la porta seriale utiilizzata.

Il PC comunicherà con la scheda, tramite la porta USB collegata al microcontrollore con un convertitore USB-Seriale RS232. Una volta collegata la scheda, il PC vedrà perciò una porta Seriale RS232 indicata con la scritta COM seguita da un numero.

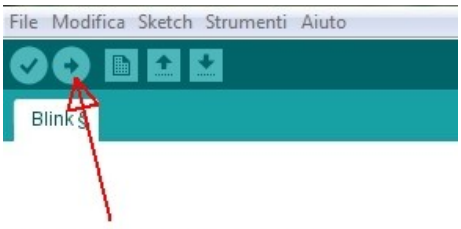
Dobbiamo perciò innanzitutto verificare che nel menù strumenti sia selezionata la scheda giusta e la COM indicata con ARDUINO UNO.





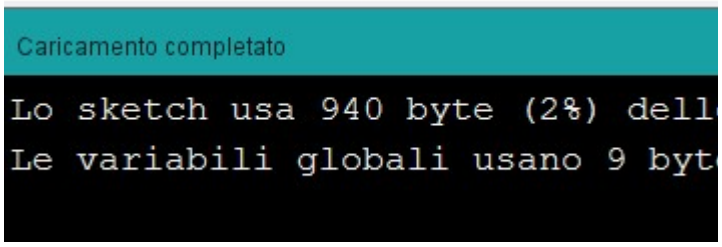
Una volta eseguite queste impostazioni, possiamo avviare la compilazione *(traduzione del programma in linguaggio macchina cioè codice binario)* e la programmazione della scheda con l'apposito pulsante.

Se non ci saranno errori di sintassi, o di altro genere, nella finestra in basso vedremo un riepilogo della memoria usata, e non verrà indicato nessun errore.




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

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



A questo punto vedremo il led lampeggiare sulla scheda.

Dopo questa semplice descrizione andiamo a vedere qualche informazione in più sul linguaggio C/C++ per poter modificare il programma scritto, o per scriverne di nuovi.

LINGUAGGIO C/C++

Ovviamente non si possono trattare i due linguaggi C e C++ in maniera sintetica, ma possiamo fornire qualche essenziale informazione, per poter effettuare qualche modifica dei programmi di esempio, o crearne di nuovi.

Innanzitutto ogni riga di programma deve terminare con il carattere punto e virgola, così come anche le dichiarazioni di variabili che in genere vengono effettuate all'inizio del programma. Apportiamo le seguenti modifiche al programma scritto prima:

```
int conteggio;  //dichiaro una variabile intera

void setup() {
  pinMode(13, OUTPUT); //impostazione pin 13 come uscita
  pinMode(12, OUTPUT); //impostazione pin 12 come uscita
}

void loop() {
  conteggio=5;       //inizializzo il valore della variabile
  while(conteggio>0){ //controllo se eseguire il ciclo while
     digitalWrite(13, HIGH);   //accendo il LED1
     delay(500);               //attendo un secondo
     digitalWrite(13, LOW);    //spengo il LED1
     delay(500);               //attendo un secondo
     conteggio--;     //decremento la variabile conteggio
   }//while
  conteggio=3;       //inizializzo il valore della variabile
  while(conteggio>0){  //controllo se eseguire il ciclo while
     digitalWrite(12, HIGH);   //accendo il LED2
     delay(500);               //attendo un secondo
     digitalWrite(12, LOW);    //spengo il LED2
     delay(500);               //attendo un secondo
     conteggio--;    //decremento la variabile conteggio
   }//while
}
```

Come possiamo notare, ogni riga di programma (fatta eccezione l'apertura e la chiusura della parentesi graffa, che identifica un ciclo o una parte funzione) **termina con il punto e virgola.**

All'inizio del programma viene dichiarata una variabile, cioè uno spazio dove salvare un valore in questo caso intero (**int**) a 16 bit. Un valore a **16 bit con segno** potrà assumere $2^{16}=65536$ combinazioni , ma considerando che l'ultimo bit indicherà il segno, avremo un valore che può andare da **-32768 a +32767.**

Per assegnare un valore alla variabile basta utilizzare **l'operatore =.** All'interno del programma infatti, viene assegnato il valore 5 ed il valore 3 alla variabile conteggio.

conteggio=5 oppure **conteggio=3**

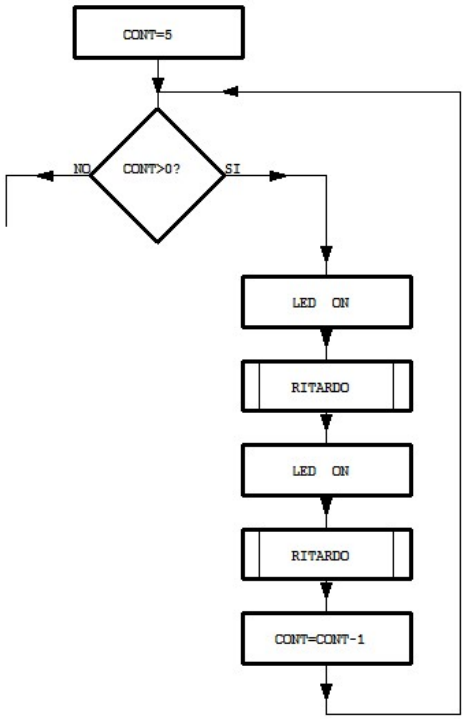
Inoltre nel programma possiamo vedere l'istruzione **conteggio--** che nel linguaggio C indica un decremento del valore della variabile.

Bisogna fare molto attenzione a come scriviamo le istruzioni ed i nomi delle variabili, soprattutto alle minuscole e maiuscole.

Inoltre nel programma troviamo un ciclo while.

Questo ciclo ripete la parte di programma contenuta tra parentesi graffe, fino a quando la condizione descritta tra le parentesi tonde è vera. Nel nostro caso eseguirà il lampeggio del LED collegato al pin 13, fino a quando il valore della variabile conteggio è superiore a zero. E successivamente farà lo stesso con un secondo LED collegato al pin 12. (lo schema al termine)

Il diagramma di flusso di uno dei due cicli while è il seguente:

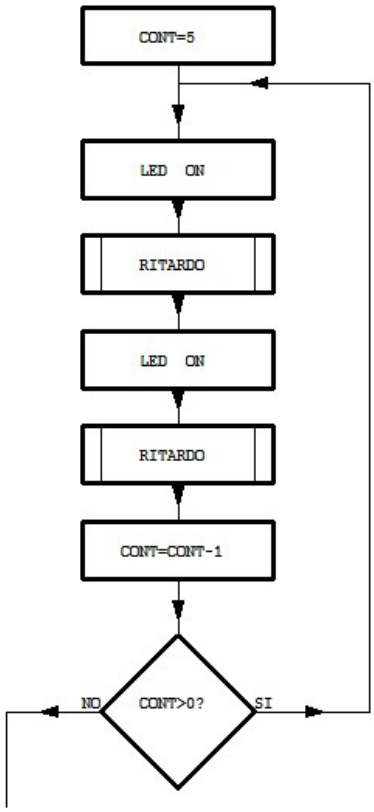


Si può vedere come il controllo viene fatto all'inizio del ciclo.

Cosa diversa avviene con il ciclo **do while()**

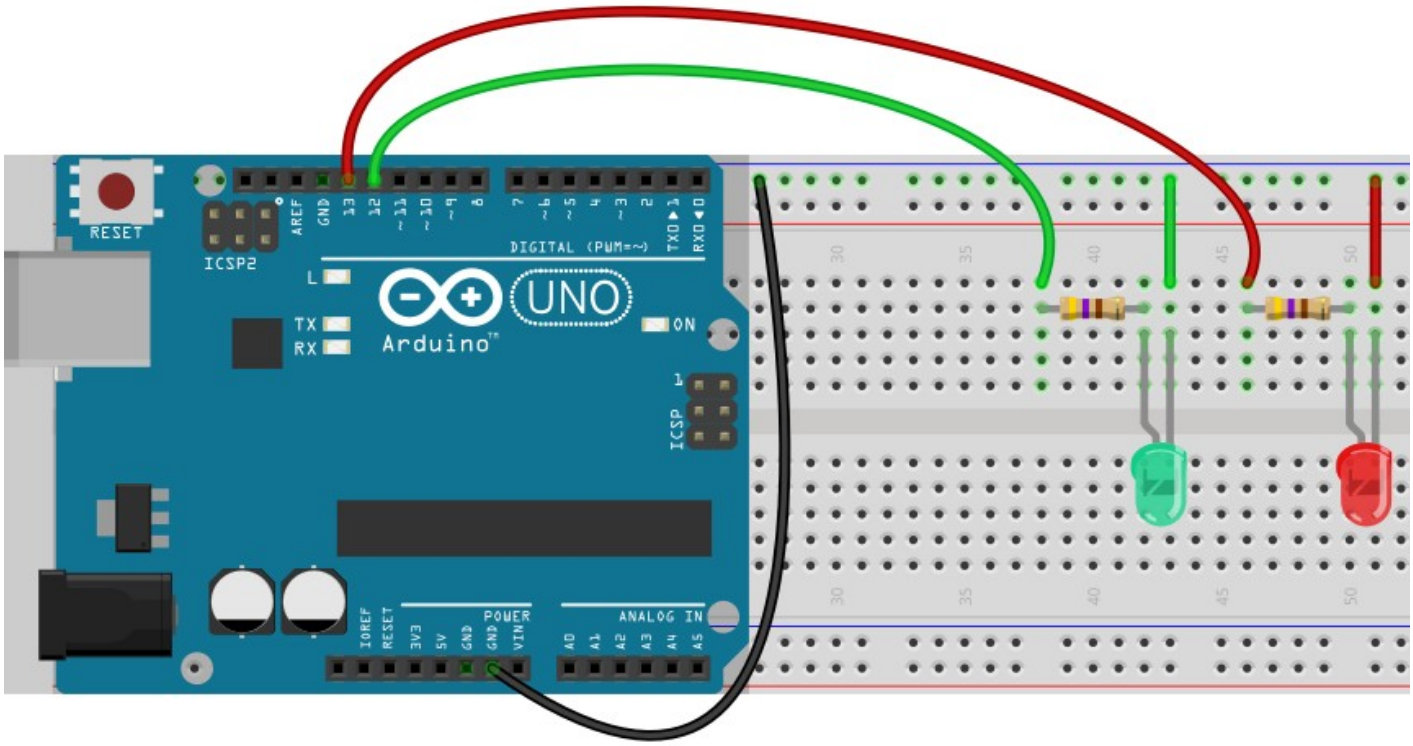
In questo caso il controllo della condizione per vedere se ripetere il ciclo, viene eseguito alla fine.

Da notare che in questo caso dopo la condizione deve essere inserito il punto e virgola.




```
conteggio=5;       //inizializzo il valore della variabile
do {
   digitalWrite(13, HIGH);   //accendo il LED1
   delay(500);              //attendo un secondo
   digitalWrite(13, LOW);    //spengo il LED1
   delay(500);              //attendo un secondo
   conteggio--;     //decremento la variabile conteggio
 }while(conteggio>0); //controllo se eseguire il ciclo while
```

Altri importanti cicli sono il ciclo **for**, il ciclo **switch**, ed il ciclo **if** descritti nella guida**,** servirà però una successiva dispensa dove approfondire gli elementi essenziali del linguaggio C/C++.


RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM=~)
L
TX
RX
UNO
Arduino™
ON
ICSP
POWER
ANALOG IN
IOREF
RESET
3V3
5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5