# *Appunti*

# sulla PROGRAMMAZIONE

# del MICROCONTROLLORE

# ARDUINO

Testi di riferimento
Massimo Banzi, **Getting starter width Arduino**
Brian W. Evans, **Arduino Programming Notebook**,

# Indice

**Prefazione**

Questi appunti, facili da usare, vogliono essere un utile riferimento per programmare e per conoscere la struttura e la sintassi di base dei comandi del microcontrollore Arduino. Sono stati esclusi argomenti avanzati che fanno di questi appunti un utilizzo adatto a principianti per cui si rimanda ad altri fonti: siti web, libri, workshop.

La struttura di base del linguaggio di programmazione da cui Arduino deriva è il C. Questi appunti vogliono descrivere la sintassi degli elementi più comuni del linguaggio di programmazione e illustra il loro utilizzo, con esempi e frammenti di codice. Questo include molte funzioni della libreria di base seguita da un'appendice con schemi campione e programmi iniziali. I complimenti del progetto generale vanno a O'Sullivan e a Igoe per il testo *Physical Computing* dove possibile.

Per un'introduzione alla progettazione interattiva di Arduino, consultate il libro di Massimo Banzi *Getting Started with Arduino*, *Arduino. La guida ufficiale* o altri libri su Arduino.

Per chi fosse interessato a conoscere più approfonditamente il linguaggio di programmazione in C, si suggerisce Kernighan e Ritchie, *The C Programming Language*, seconda edizione, così come Prinz e C Crawford *Nutshell*; essi forniscono tante altre spiegazioni sulla sintassi originale di programmazione.

Soprattutto, questi appunti non sarebbero stati possibile senza la grande comunità e al materiale originale che si trova sul sito Arduino, http://www.arduino.cc.

## STRUTTURA DEL PROGRAMMA

### Structure - struttura

La struttura di base del linguaggio di programmazione Arduino (sketch[1]) è abbastanza semplice e viene eseguito in almeno due parti. Queste due parti necessarie (dette anche funzioni) sono racchiuse nei seguenti blocchi di istruzioni.

```
void setup()
{
  istruzioni che devono essere eseguire una sola volta all'inizio del programma;
}

void loop()
{
  ciclo in cui viene racchiuso il programma vero e proprio che deve essere eseguito ripetutamente finché la scheda non viene spenta;
}
```

Dove setup () è la preparazione, loop () è l'esecuzione.
Entrambe le funzioni sono necessarie per l'esecuzione del programma.
Il contenuto scritto dentro le parentesi graffe, e quindi dopo la funzione setup, contiene ed esegue la dichiarazione di tutte le variabili. La funzione setup è posta sempre all'inizio del programma. Essa viene eseguita solo una volta, ed è usata per impostare i pin del microcontrollore Arduino, il cosidetto pinMode, o inizializzare la comunicazione seriale.
Segue la funzione loop. Essa include il codice da eseguire in modo continuo: lettura ingressi, attivazione uscite, ecc. Questa funzione è il cuore di tutti i programmi di Arduino.

### setup ()

Il setup () viene eseguito una volta sola appena si avvia il programma. Viene utilizzato per inizializzare i pin del microcontrollore Arduino e quindi stabilisce i pin di ingresso e di uscita, e/o inizializza la comunicazione seriale. Deve essere incluso in un programma, anche se non ci sono istruzioni da eseguire.

```
void setup ()
{
  pinMode (pin, OUTPUT); / / imposta la variabile 'pin' come uscita
}
```

### loop ()

La funzione loop () fa girare consecutivamente il programma contenuto all'interno delle parentesi graffe, permettendo al programma di scambiare, rispondere e controllare la scheda Arduino.

```
void loop ()
```

[1] I programmi in Arduino sono chiamati skecth. Un programma è una serie di istruzioni che vengono lette dall'alto verso il basso e convertite in eseguibile e poi trasferite sulla scheda dall'IDE Arduino.

```
{
  digitalWrite (pin, HIGH);       / / il 'pin' è su
  delay (1000);                   / / un secondo di pausa
  digitalWrite (pin, LOW);        / / il 'pin' è giù
  delay (1000);                   / / un secondo di pausa
}
```

**Functions - Funzione**

Una funzione è un blocco di codice che ha un nome ben definito, quindi è un blocco di istruzioni che vengono eseguiti quando la funzione viene chiamata. La funzione setup () e void loop () sono già stati già descritti. Le altre funzioni saranno descritte più avanti.
Le funzioni sono utilizzate per eseguire operazioni ripetitive in modo da ridurre il codice programma ed evitare quindi confusione nel programma stesso.
Le funzioni sono dichiarate all'inizio del programma e specificate dal tipo di funzione.

La struttura della funzione è la seguente:
<Tipo del valore restituito> <nome funzione> ( <elenco dei parametri> )
Dopo il tipo, occorre dichiarare il nome dato alla funzione e tra parentesi i parametri che vengono passati alla funzione.

Esempio:
int delayVal()
Il tipo di valore che viene restituito dalla funzione 'int' per la funzione delayVal è di tipo intero. Se nessun valore deve essere restituito il tipo di funzione è nullo.
I tipi di dati (int, byte, long, insigne long) verranno spiegati più avanti.

Esempio: la funzione int per la variabile delayVal() è di tipo intero e viene usato per impostare un valore di ritardo in un programma leggendo il valore di un potenziometro. Prima occorre dichiarare una variabile locale v; viene letto il valore del potenziometro che dà un numero compreso tra 0-1023, poi si divide tale valore per 4, così si ha un valore finale compreso tra 0-255, e infine viene restituito il valore al programma principale.

Esempio:
```
int delayVal()
{
  int v;                    // crea temporaneamente una variabile di nome 'v'
  v = analogRead(pot);      // legge il valore del potenziometro
  v /= 4;                   // converte il valore 0-1023 a 0-255
  return v;                 // restituisce il valore finale
}
```

## { } Curly braces - Le parentesi graffe

Le parentesi graffe (note anche semplicemente come "parentesi" o "parentesi graffe") definiscono l'inizio e la fine dei blocchi di codice e dei blocchi di istruzioni come il void loop () e le istruzioni condizionate if.

```
Type function ()
{
  statements;
}
```

Una parentesi graffa aperta { deve essere sempre seguita da una parentesi graffa di chiusura }.
Il numero delle parentesi graffe devono essere uguali. Una parentesi aperta e non chiusa può portare a errori nascosti che il compilatore del programma non vede e che a volte può essere difficile da rintracciare in un programma di grandi dimensioni.
L'ambiente Arduino include una comoda funzione per controllare le parentesi graffe. Basta selezionare un check, o anche fare clic sul punto di inserimento subito dopo una parentesi, e la sua corrispondente parentesi di chiusura sarà evidenziata.

## ; Semicolon - Punto e virgola

Il punto e virgola è utilizzato per chiudere ogni istruzione o dichiarazione e serve a separare gli elementi del programma. Il punto e virgola permette di scrivere due o più istruzioni su una stessa riga, ma questo rende il codice più difficile da leggere.
Un punto e virgola è anche usato per separare gli elementi in un ciclo for.
int x = 13; // assegna alla variabile 'x' il numero intero 13

Nota: Dimenticare di terminare una riga con un punto e virgola si tradurrà in un errore di compilazione. L'errore può anche non essere evidente. Se il programma genera un errore di compilazione, una delle prime cose da controllare sono i punto e virgola mancanti, nei pressi della linea dove il compilatore si trova.

## / * ... * / Block comments - Blocco commenti

All'inizio di ogni programma è conveniente scrivere un commento al programma stesso. Una serie di righe di commenti costituiscono un blocco. Il blocco del commento è un'area di testo ignorato dal programma e viene utilizzato per aiutare a capire le parti del programma.

Iniziano con / * e finiscono con * / e possono estendersi su più righe.

```
/ *     Questo è un blocco di apertura del commento.
        Non dimenticare il blocco di commento di chiusura.
        Essi devono essere sempre bilanciati!
* /
```

Poiché i commenti sono ignorati dal processore di Arduino non occupano spazio di memoria, quindi possono essere usati con generosità e possono anche essere usati per "commentare" blocchi di codice per il debug dello skecth.
Nota: Mentre è possibile completare ogni singola riga del programma con un blocco di commento, non è permesso inserire un secondo blocco di commenti al suo interno.

## // Line comments - Singola linea di commento

I commenti di singola linea iniziano con // e terminano con la prossima linea di codice. Come i commenti a blocchi, essi sono ignorati dal programma e non occupano spazio di memoria.

```
// Questo è un commento singola linea
```

I commenti di singola linea sono spesso utilizzati dopo una dichiarazione valida per fornire maggiori informazioni su ciò che la dichiarazione compie o per fornire un promemoria.

## VARIABILI

### Variables - Le variabili

Qualsiasi dato presente in uno skecth deve essere associato a un tipo di dato. La variabile, proprio per la sua caratteristica, contiene un valore numerico che può cambiare durante lo svolgimento del programma stesso.
Il valore assegnato alle costanti, che vedremo più avanti, vengono definite anch'esse all'inizio del programma, ma il suo valore non può essere modificato o sostituito.

La variabile deve essere dichiarata e, opzionalmente, le può essere assegnato un valore che resta in memoria per tutto il tempo dell'esecuzione del programma.

Il codice seguente dichiara una variabile denominata inputVariable e poi le viene passato il valore ottenuto sul pin di ingresso analogico 2:

```
int inputVariable = 0;              // dichiara una variabile e assegna il valore 0
inputVariable = analogRead (2); // il valore acquisito sul pin analogico 2 viene
                                    // assegnato alla variabile inputVariable
```

La prima riga dichiara che conterrà un valore int, (abbreviazione di numero intero). La seconda riga assegna (=) alla variabile inputVariable il valore analogico prelevato sul pin 2 del microncontrollore Arduino. Questo rende il valore del pin accessibile in altre parti del programma.
Una volta che una variabile è stata assegnata, o ri-assegnata, è possibile verificare il suo valore per vedere se soddisfa determinate condizioni, oppure è possibile utilizzare direttamente il suo valore.
Di seguito sono illustrate tre operazioni utili con le variabili.
L'esempio seguente verifica il contenuto della variabile inputVariable.
Se è inferiore a 100, inputVariable assume il valore 100, e quindi viene impostato un ritardo in base a inputVariable che ora è di 100:

```
if (inputVariable <100) // test variabile se è meno di 100
{
inputVariable = 100; // se il valore è uguale a 100
}
ritardo (inputVariable); // utilizza il valore contenuto nella variabile per impostare il ritardo
```

Nota: Le variabili dovrebbe avere un nome descrittivo, per rendere il programma più leggibile. I nomi delle variabili come tiltSensor o pulsante aiutano il programmatore alla lettura del codice e a capire che cosa rappresenta la variabile. I nomi delle variabili come var o di valore, d'altro canto, fanno ben poco per rendere il codice leggibile e vengono utilizzati solo qui come esempi. Una variabile può essere chiamata con qualsiasi altra parola che non sia contenuta nel set dei comandi di Arduino.

### Variable declaration - Dichiarazione delle variabili

Tutte le variabili devono essere dichiarate prima di poter essere utilizzati. La dichiarazione di una variabile significa definire il suo tipo di valore, come int, long, float, ecc, la definizione di un nome specifico, e facoltativamente l'assegnazione di un valore iniziale. Questo deve solo essere fatto una volta in un programma ma il valore può essere modificato in qualsiasi momento usando gli operatori aritmetici e la ri-assegnazione del valore.

L'esempio seguente dichiara che inputVariable è un int, cioè di tipo intero, e che il suo valore iniziale è uguale a zero. Questa operazione è chiamata "dichiarazione semplice".

```
int  inputVariable = 0;
```

Una variabile può essere dichiarata in qualunque punto del programma e svolge il suo compito nella parte del programma in cui viene utilizzata.

**Variable scope - Portata della variabile**

Una variabile può essere dichiarata all'inizio del programma prima di void setup (), oppure localmente cioè all'interno di funzioni, e talvolta la variabile può essere dichiarata all'interno di un blocco di istruzioni come in un ciclo for.

Nel punto di programma in cui la variabile viene dichiarata, determina la portata della variabile stessa, o la capacità in alcune parti di un programma di utilizzare la variabile stessa.

Una variabile è globale se è dichiarata all'inizio del programma, prima della funzione setup () e può essere utilizzata da ogni funzione e istruzione in tutto il programma.

Una variabile è locale quando è definita all'interno di una funzione o come parte di un ciclo for. La variabile locale è visibile e quindi può essere utilizzata solo all'interno della funzione in cui è stata dichiarata. E' quindi possibile avere due o più variabili con lo stesso nome in diverse parti dello stesso programma che contengono valori diversi. Garantendo che solo una funzione ha accesso alle sue variabili semplifica il programma e riduce il rischio di errori di programmazione.

Il seguente esempio mostra come dichiarare alcuni tipi diversi di variabili e dimostra la visibilità di ogni variabile:

```
int value;     // 'value' è visibile in tutto il programma e in ogni funzione

void setup()
{
                // non è necessaria alcuna dichiarazione di variabile
}
void loop()
{
for (int i=0; i<20;) // 'i' è visibile solo all'interno del ciclo for
{
i++;
}
float f;                // 'f' è visibile dentro la funzione loop
}
```

**TIPI DI DATI**

| Tipi di dichiarazione | Rappresentazione | N. di byte | Intervallo |
|---|---|---|---|
| Bolean | | | True – False / On –Off / High - Low |
| Char | Carattere | 1 (8 bit) | - 127 +127 |
| Byte | Carattere | 1 (8 bit) | 0 +255 |
| Int | Numero intero | 2 (16 bit) | -32.768 + 32.767 |
| Unsigned int | Numero intero | 2 (16 bit) | 0 + 65.535 |
| Short | Numero intero "corto" | 2 (16 bit) | |
| Long | Numero intero "lungo" | 4 (32 bit) | -2.147.483.648 +2.147.483.647 |
| Float | Numero reale | 4 (32 bit) | -3.4028235E+38 a + 3.4028235E+38 |
| Double | Numero reale "lungo" | 8 (64 bit) | $1.7976931348623157 \times 10^{308}$ |

**Boolean - boleano**
La variabile di tipo boleano può assumere un solo valore: vero o falso.

**Char**
Il tipo **char** contiene uno ed un solo carattere definito secondo lo standard ASCII, quindi qualsiasi lettera (maiuscola o minuscola), cifra (da 0 a 9) e simbolo previsto dalla codifica. Arduino lo conserva in forma di numero, anche se quello che si vede è un testo.
Per dichiarare una variabile char, ad esempio inizializzandola con la lettera 'r', basta scrivere: char a = 'r'.
Può contenere valori compresi tra -128 a 127 e occupa un byte di memoria.

**Byte**
il byte occupa 8 bit, può contenere un valore numerico senza decimali e può assumere un valore compreso tra 0 e 255. Esempio:
byte someVariable = 180; // dichiara 'someVariable' come un tipo byte

**Int**
La variabile di tipo intero è un tipo di dato, molto utilizzato in Arduino, per memorizzare numeri senza decimali. Può assumere valori negativi. Occupa 2 byte, quindi 16 bit di memoria. Il valore può essere compreso tra -32.768 e 32.767 .
Esempio:
int someVariable = 1500; // dichiara 'someVariable' come un tipo intero

Nota: Le variabili intere se raggiungono il valore massimo (o valori minimi) in una operazione aritmetica o di confronto, per esempio, se x = 32767 e ad una successiva istruzione si aggiunge 1 a x (si toglie 1 a x), il valore di x ricomincia da -32.768 (il valore diventa + 32767).

**Unsigned int**
È uguale a int, occupa 2 byte di memoria e non può assumere valori negativi. Il suo intervallo va da 0 a 65.535.

**Long**
Il tipo long è come un int, che utilizza 4 byte, quindi 32 bit per estendere il valore da memorizzare. Infatti il valore che può contenere è compreso tra -2.147.483.648 e 2.147.483.647. Il tipo long non contiene decimali. Esempio:
long someVariable = 90000 // dichiara 'someVariable' di tipo long

**Unsigned long**
Unsigned long è una versione di long che non contiene numeri negativi; infatti può contenere valori compresi tra 0 e 4.294.967.295.

**Float**
Memorizza un numero a virgola mobile compreso tra -3.4028235E+38 a + 3.4028235E+38. Questo tipo di variabile può rappresentare numeri molto piccoli o numeri molto grandi, positivi e negativi con o senza decimali. La precisione dopo il punto decimale è di 7 cifre. I numeri in virgola mobile per la loro maggiore risoluzione rispetto agli interi, 4 byte cioè 32 bit. Esempio:
Float someVariable = 3,14; // dichiara 'someVariable' come un tipo in virgola mobile

Nota: i numeri a virgola mobile non sono esatti, e possono portare a risultati anche strani. I calcoli matematici sono più lenti rispetto ai calcoli con variabili intere. Se possibile le variabili di tipo float sono da evitare.

**Double**
Il tipo double è un numero che ha una precisione a virgola mobile doppia rispetto a Float e può contenere un valore massimo di $1.7976931348623157 \times 10^{308}$.

**String**
Il set dei caratteri ASCII può essere usato per contenere informazioni testuali (un messaggio su un display LCD o un messaggio attraverso la comunicazione seriale). Per la memorizzazione viene utilizzato un byte per ogni carattere, più un carattere null (vuoto) per dire ad Arduino che la stringa di caratteri è finita.
Un esempio:
```
char string1[] = "Arduino";      // 7 caratteri + 1 carattere null
char string2[8] = "Arduino";     // Come sopra
```

**Array**
Un array è un insieme di valori impilati a cui si accede attraverso un indice. Ogni valore nella tabella o matrice può essere richiamato indirizzando il puntatore numerico nell'indice della tabella. Gli array sono indicizzati a partire dal numero zero, infatti il primo valore posto all'inizio della matrice ha come indice numerico proprio il numero 0.
Un array deve essere dichiarato e, opzionalmente, possono essere assegnati anche i valori prima di utilizzarlo.
Int myArray [] = {valore 0, valore 1, valore 2, ...}

Allo stesso modo è possibile dichiarare un array dichiarando il tipo di array e la dimensione e poi assegnare i valori:
```
int myArray [5];    // dichiara un array di interi avente 6 caselle
myArray [3] = 10; // assegna all'indice 4 il valore 10
```
Per recuperare un valore contenuto in un array, occorre dichiarare una variabile e poi assegnarla all'indice numerico dell'array:
```
x = myArray [3]; // ad x  verrà assegnato il valore 10
```

Gli array sono spesso utilizzati nei cicli for, in cui il contatore di incremento viene utilizzato anche come posizione di indice per ogni valore contenuto nella matrice. L'esempio seguente utilizza un array per lo sfarfallio di un LED. Utilizzando un ciclo for, il contatore inizia a scrivere il valore nella posizione 0 dell'indice nell'array sfarfallio [], in questo caso 180, al pin PWM 10, pausa per 200ms, poi si sposta alla posizione di indice successiva.

```
int ledPin = 10;     // inizializzata la variabile ledPin e assegna il valore 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};
                     // Sopra sono riportati 8 valori diversi
void setup()
{
pinMode(ledPin, OUTPUT); // il pin 10 è un uscita del microcontrollore
}
void loop()
{
for(int i=0; i<7; i++)      // esegue un ciclo utilizzando i numeri contenuti nell'array
{
analogWrite(ledPin, flicker[i]); // attiva il Pin specificato dalla variabile LedPin con il
valore contenuto nell'array con la posizione dell'indice "i"
delay(200);                  // pausa di 200ms
}
}
```

## OPERAZIONI ARITMETICHE

### Calcoli aritmetici e formule

Gli operatori aritmetici sono addizione, sottrazione, moltiplicazione e divisione. L'operazione restituisce la somma, differenza, prodotto o quoziente (rispettivamente) di due operandi.

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

L'operazione è condotta utilizzando il tipo di dati degli operandi, così, per esempio, 9/4 il risultato è 2 invece di 2,25 perché 9 e 4 sono due interi e quindi non può fornire un risultato con il punto decimale. Questo significa anche che l'operazione può uscire fuori dalla memoria (overflow) se il risultato è più grande di ciò che può essere memorizzato nel tipo di dati.
Se gli operandi sono di tipo diverso, il tipo più grande è utilizzato per il calcolo. Per esempio, se uno dei numeri (operandi) è di tipo float e l'altro di tipo int, il microcontrollore utilizzerà per il calcolo la virgola mobile.
Quindi occorre dimensionare in modo opportuno le variabili in modo da contenere il risultato dal calcolo in modo appropriato.
Conoscere prima a che punto la variabile andrà in errore è importante; anche quando ricomincia il conteggio ciclico. Per i calcoli matematici che richiedono frazioni, è bene utilizzare le variabili float, ma con la consapevolezza del loro svantaggio: grandi dimensioni e velocità di calcolo lento.

Nota: Le variabili possono essere convertite al volo. Per esempio, i = (int) 3,6 imposterà i uguale a 3.

### Compound assignments - Assegnazioni compound

Si tratta di operatori speciali che si usano per rendere più conciso il codice di programma. Esso combina un'operazione aritmetica con un'assegnazione di variabile.
Esempi:
a = a + 1 si può scrivere a++
a = a +2 si può scrivere a += 2
Attenzione! Se scrivo: value++, prima valuta la variabile value e poi la incrementa di 1; se invece scrivo: ++value prima incrementa di 1 e poi lo valuta. Lo stesso vale per –-

Questi operatori speciali si trovano comunemente nei cicli for. Le assegnazioni più comuni includono:

```
x + +       // è uguale a x = x + 1, incrementa la variabile x di +1
x  - -      // è uguale a x = x - 1, decrementa x di -1
x += y      // è uguale a x = x + y, incrementa x di + y
x -= y      // è uguale a x = x - y, decrementa x di -y
x *= y      // è uguale a x = x * y, moltiplica x per y
x / = y     // è uguale a x = x / y, divide x per y
```

Nota: per esempio, x *= 3 da come risultato il triplo del valore di x e poi viene riassegnato alla variabile x.

**Comparison operators - Operatori di confronto**
Alcune volte si ha bisogno di confrontare una variabile o una costante con un'altra. Il confronto si uso nelle istruzioni condizionate if, while e for per verificare se una determinata condizione è vera.
Esempi:

```
x == y      // x è uguale a y
x != y      // x è diverso da y
x <y        // x è minore di y
x > y       // x è maggiore di y
x <= y      // x è minore o uguale a y
x >= y      // x è maggiore o uguale a y
```

**Logical operation - Gli operatori logici o operatori booleani**
Gli operatori logici sono un modo per confrontare due espressioni. Si usano anche quando si vogliono combinare diverse condizioni. Restituiscono una funzione TRUE o FALSE.
Ci sono tre operatori logici AND, OR e NOT, che vengono utilizzati in istruzioni *if*:

AND logico:
if (x > 0 **& &** x < 5) // vera solo se entrambe le espressioni sono vere

OR logico:
if (x > 0 **||** y> 0) // vero se uno delle due espressioni è vera

NOT logico:
if (!x > 0) // vera solo se l'espressione è falsa

## COSTANTI

### Costanti

Il linguaggio Arduino ha una serie di parole chiave predefinite con valori speciali.
**HIGH** e **LOW** si usano, quando si vuole accendere o spegnere un pin di Arduino.
**Input** e **Output** si usano per impostare un determinato pin come ingresso o uscita.
**True/false** indica l fatto che una condizione o un'espressione è vera o falsa.

True/false - vero / falso
Questi sono costanti booleane che definiscono il livello logico. FALSO è facilmente definita come 0 (zero), mentre VERO viene spesso definito come 1, ma può anche essere altro eccetto lo zero.
Quindi in un certo senso booleano, -1, 2, e -200 sono anche definiti come VERO.

```
if (b == TRUE);
{
doSomething;
}
```

HIGH / LOW - alta / bassa
Queste costanti definiscono il livello del pin come alto o basso e vengono utilizzati durante la lettura o la scrittura del pin digitale. HIGH viene definito come livello logico 1, ON o 5 volt mentre LOW livello logico 0, OFF o 0 volt.

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

Input / Output - ingresso / uscita
Costanti usate con il pinMode () per definire la modalità di un pin digitale come ingresso o uscita.

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

## CONTROLLI DI FLUSSO

### If - se

L'istruzioni if verifica se una certa condizione è stata raggiunta, come ad esempio un valore analogico al di sopra di un certo numero. Se l'espressione è vera lo sketch esegue le istruzioni che seguono. Se falso il programma ignora la dichiarazione. Un esempio:

```
if (someVariable == value)
{
esegui etc.;
}
```

L'esempio precedente compara (l'operazione di confronto è eseguita dall'istruzione ==) someVariable al valore contenuto in value, che può essere una variabile o una costante. Se l'operazione di confronto, o la condizione tra parentesi risulta vera, le istruzioni contenute all'interno delle parentesi graffe vengono eseguite. In caso contrario, il programma salta su di loro e continua dopo le parentesi graffe.

Nota: Attenzione all'uso accidentale del simbolo dentro le parentesi dell'istruzione if '=', infatti (x = 10), tecnicamente è un'operazione valida, infatti assegna alla variabile x il valore 10 ed è di conseguenza una condizione sempre vera, per cui lo sketch si comporta in modo diverso da come ci si aspettava.
Occorre usare invece il doppio uguale '==', infatti (x == 10), esegue il confronto tra la variabile e il valore, cioè confronta solo e soltanto se x è uguale al valore 10.
Occorre pensare a '=' come "assegna a" e
'==' a "confronta con".

### if ... else

La struttura if ... else è utilizzata per verificare determinate condizioni e quindi eseguire decisioni. Se l'espressione contenuta all'interno delle parentesi tonde è vera, viene eseguito il codice di programma che segue. Se l'espressione è falsa vengono eseguite le righe di codice subito dopo l'istruzione else.

Un esempio:

```
#define Acceso 1 // definisce Acceso = 1
#define Spento 0 // definisce Spento = 0

if (pulsante == ON)

{
digitalWrite (rele, acceso);
}
Else
{
digitalWrite(rele, spento);
}
```

Altro esempio.
Se si vuole testare un ingresso digitale, se è HIGH o LOW, si può scrivere:

```
if (inputPin == HIGH)
{
doThingA;
}
```

```
else
{
doThingB;
}
```

Si possono scrivere condizioni if annidate una dentro l'altra o condizioni multiple. Attenzione: solo una serie di dichiarazioni verrà eseguito a seconda della condizione posta nello sketch.

L'istruzione else if permette di fare la scelta tra più condizioni.

```
If (inputPin <500)
{
esegui A;
}
else if (inputPin> = 1000)
{
esegui B;
}
else
{
esegui C;
}
```

Ricapitolando.
Il flusso del programma può eseguire una parte di codice oppure no (nel caso del solo if), di fare una scelta tra due parti di codice (nel caso di If – else) o di fare una scelta tra più parti di codice (nel caso di if – else if – else).

**For**
Il for viene utilizzato per ripetere un blocco di istruzioni racchiuso tra parentesi graffe un determinato numero di volte. Viene utilizzato un contatore per incrementare e terminare il ciclo. Esso è composto da tre parti, separate da punto e virgola (;):

```
for (inizializzazione; condizione; espressione)
{
esegui A;
}
```

Esempio:

```
for (int A = 0; A < 10; A++)
{
esegui le istruzioni;
}
```

Al comando for segue una parentesi tonda. Il contenuto all'interno della parentesi definisce quante volte deve essere ripetuto il blocco di programma contenuto dentro le parentesi graffe.
Int A = 0;
definisce la variabile di tipo intero e viene impostata con un valore iniziale uguale a zero. L'inizializzazione di una variabile locale, o contatore di incremento, avviene all'inizio e viene definita una volta sola.

A < 10;
Specifica che finché la variabile A è minore di 10 il ciclo for viene ripetuto perché la condizione è vera, pertanto vengono eseguite le istruzioni contenute all'interno delle parentesi graffe.

A++
Incrementa la variabile A di una unità. Il significato è uguale a scrivere A=A+1.

Quando la variabile assume il valore di 10, quindi la condizione diventa falsa, il ciclo termina.

Nell'esempio seguente viene inizializzata la variabile i definita come int e posta uguale a 0; viene avviato il ciclo per vedere se i è ancora inferiore a 20 e finché non risulta vera, la variabile i viene incrementata di 1 e vengono eseguite le istruzioni racchiuse tra parentesi graffe:

```
for (int i = 0; i <20; i ++)      // dichiara i, controlla se i è meno
{                                 // di 20, incrementa i di 1
digitalWrite (13, HIGH);          // abilita il pin 13 come uscita alta
delay (250);                      // pausa per 1/4 di secondo
digitalWrite (13, LOW);           // abilita il pin 13 come uscita bassa
delay (250);                      // pausa per 1/4 secondo
}
```

Nota: Nel linguaggio C il ciclo for è molto più flessibile rispetto ai cicli che si trovano in altri linguaggi di programmazione, incluso il BASIC.
Qualsiasi o tutti e tre gli elementi di intestazione possono essere omessi, anche se i punti e virgola sono obbligatori.
Le dichiarazioni per l'inizializzazione, la condizione e l'espressione possono essere sostituiti da valide istruzione di linguaggio C utilizzando delle variabili indipendenti. Questo uso è piuttosto insolito ma può essere utilizzato per delle rare soluzioni ad alcuni problemi di programmazione.

**While**
È un comando simile a If.
Il ciclo while esegue all'infinito le istruzioni racchiuse tra le parentesi graffe fino a quando la condizione racchiusa dentro le parentesi tonde diventa falsa.
Qualcosa deve far cambiare la variabile in esame, o il ciclo while non potrà mai uscire. Questo potrebbe essere scritto nel codice, come ad esempio una variabile che viene incrementata, o una condizione esterna, come il dato proveniente da un sensore.

```
while (someVariable? value)
{
doSomething;
}
```

L'esempio seguente controlla se la varabile 'someVariable' è inferiore a 200 e se è vera esegue le istruzioni all'interno delle parentesi e continuerà all'infinito finché la variabile 'someVariable' non diventa maggiore o uguale a 200.

```
while (someVariable <200) // verifica se meno di 200
{
doSomething;              // esegue le istruzioni racchiuse tra le parentesi graffe
```

```
someVariable++;          // la variabile viene incrementata di una unità
}
```

Do … while
Il ciclo do … while si comporta come l'istruzione while ma con una importante differenza: esegue almeno una volta l'istruzione all'interno del ciclo do-while. Quindi viene usato quando si vuole che il codice dentro le parentesi graffe venga eseguito almeno una volta prima di verificare la condizione.

```
Do
{
doSomething;
} While (someVariable ?? value);
```

L'esempio seguente assegna il valore di readSensors () alla variabile 'x', si ferma per 50 millesimi di secondo, poi inizia il ciclo fino a quando 'x' è meno di 100:

```
do
{
x = readSensors();       // assegna il valore di
                         // readSensors() a x
delay (50);              // pausa di 50 milliseconds
} while (x < 100);       // inizia il ciclo finché x è minore di 100
```

Altro esempio

```
do
{
digitalWrite(13,High);
delay(100);
digitalwrite(13,High)
delay(100);
sensorvalue= analogRead(1);
} while (sensorvalue < 512);
```

## INGRESSI E USCITE DIGITALI

### pinMode (pin, mode)

Utilizzato in void setup (), serve per configurare un determinato pin e stabilire se deve essere un ingresso o un'uscita.

pinMode (pin, OUTPUT); // imposta il 'pin' come uscita

I pin digitali di Arduino di default sono pin di ingresso se non sono esplicitamente dichiarati come ingressi con l'istruzione pinMode (). I pin configurati come ingressi, si dice, hanno un'alta impedenza.

All'interno dell'integrato Atmega sono già presenti per ogni pin le resistenze di pull-up da 20KΩ che abilitati via software permettono di settare i pin come ingresso. Le istruzioni che permettono ciò sono le seguenti:
pinMode (pin, INPUT); // imposta il 'pin' come ingresso
digitalWrite (pin, HIGH); // attiva il pin a livello alto con la resistenza di pull-up

Le resistenze di pull-up sono normalmente utilizzate per collegare gli ingressi come interruttori. Si noti che nell'esempio di cui sopra non viene configurato un pin come uscita, è semplicemente un modo per attivare il pull-up interno.

I Pin configurati come OUTPUT sono detti a bassa impedenza e sono in grado di fornire 40 mA (milliampere) di corrente ad altri dispositivi o circuiti. Questo è una corrente sufficiente per accendere un LED (non dimenticare di collegare una resistenza in serie), ma non è una corrente sufficiente per comandare la maggior parte di relè, bobine, o motori.
I cortocircuiti sui piedini di Arduino o l'eccessiva corrente possono danneggiare o distruggere il pin di uscita, o danneggiare l'intero chip Atmega. È spesso una buona prassi collegare il pin di uscita ad un dispositivo esterno collegando in serie una resistenza da 470Ω o 1KΩ.

### digitalRead (pin)

L'istruzione permette di leggere lo stato di un pin di input e restituisce un valore HIGH se al pin è applicato un tensione o un valore LOW se non è applicato nessun segnale. Il pin può essere specificato come una variabile o costante (0-13).

Value = digitalRead (Pin); // assegna a 'value' il valore prelevato dal pin

### digitalWrite (pin, valore)

Attiva o disattiva un pin digitale, quindi l'istruzione pone il pin di uscita a livello logico HIGH o LOW. Il pin può essere specificato come una variabile o una costante (0-13).

digitalWrite (pin, HIGH); // imposta il 'pin' a livello alto

Il seguente esempio legge un tasto collegato a un ingresso digitale e gira su un LED connesso a un'uscita digitale quando il pulsante è stato premuto:

```
int led = 13;       // il led è collegato al pin 13
int pin = 7;        // il pulsante è collegato al pin 7
int value = 0;      // viene definita una variabile per memorizzare il valore letto
```

```
void setup ()
{
pinMode (led, OUTPUT);  // imposta il pin 13 come uscita
pinMode (pin, INPUT);     // imposta il pin 7 come input
}
void loop ()
{
value = digitalRead (pin); // imposta 'value' pari al segnale prelevato dal pin
digitalWrite (led, value);  // imposta il led al valore della variabile value
}
```

## INGRESSI E USCITE ANALOGICHE

### analogRead (pin)

Legge il valore di tensione applicato ad un pin di input analogico con una risoluzione pari a 10 bit. Questa funzione restituisce un numero intero compreso tra 0 e 1023.

```
Value = analogRead (pin); // imposta value uguale a 'pin'
```

Nota: il pin analogico a differenza di quelli digitali, non hanno bisogno di essere prima dichiarati come INPUT o OUTPUT.

### analogWrite (pin, value)

Cambia la percentuale della modulazione di larghezza di impulso (Pulse Width Modulation - PWM) su uno dei pin contrassegnati dalla sigla PWM. Sulle nuove schede Arduino con il chip ATmega168, questa funzione è abilitata sui pin 3, 5, 6, 9, 10 e 11. Le schede Arduino più vecchie con un ATmega8 supportano solo i pin 9, 10 e 11. Il valore può essere specificato da una variabile o una costante con un valore compreso tra 0 e255.

```
analogWrite (pin, value); // abilita il pin di uscita al valore analogico
                          // della variabile value
```

Un valore 0 genera una uscita pari a 0 volt al pin specificato; un valore di 255 genera un segnale di 5 volt al pin specificato.
Per valori tra 0 e 255, il valore di uscita sarà compreso tra 0 e 5 volt. Più alto è il valore, più spesso il pin è attivo alto (5 volt). Ad esempio, un valore 64 sarà 0 volt tre quarti del tempo, e 5 volt un quarto del tempo; mentre un valore di 128 sarà a 0 volt la metà del tempo e 5 volt la restante metà del tempo; un valore di 192 fornirà un valore di 0 volt un quarto del tempo e 5 volt tre quarti del tempo.
Poiché questa è una funzione hardware, il pin genererà un'onda quadra a impulsi costante dopo una chiamata all'istruzione analogWrite in background fino alla successiva chiamata analogWrite (o una chiamata a digitalRead o digitalWrite sul pin stesso).

Nota: il pin analogico a differenza di quelli digitali, non hanno bisogno di essere prima dichiarati come INPUT o OUTPUT.

L'esempio seguente legge un valore analogico da un pin di ingresso analogico, converte il valore dividendolo per 4, e fornisce un segnale PWM su un pin PWM:

```
int led = 10;         // è collegato un LED con resistenza da 220Ω al pin 10
int pin = 0;          // un potenziometro o un pin analogico viene assegnato il valore 0
int value;            // la variabile value sarà utilizzata per la lettura
void setup () {}      // non è necessaria alcuna configurazione
void loop ()
{
value = analogRead (pin); // assegna a value il valore letto sul 'pin'
valore / = 4              // converte il rapporto 0-1023 nel rapporto 0-255
analogWrite (led, value); // il valore PWM viene assegnato al led
}
```

## OROLOGIO INTERNO

### delay (ms)
Mette in pausa un programma per la quantità di tempo specificato in millisecondi. Il valore 1000 è pari a 1 secondo.
Esempio: delay (1000); // attende un secondo

### DelayMicroseconds(us)
Mette in pausa il programma per la quantità specificata di microsecondi.
Esempio: delayMicroseconds (1000); // attende un millesimo di secondo

### Millis ()
Restituisce il numero di millisecondi da quando la scheda Arduino ha iniziato l'esecuzione del programma corrente. Il tipo di dato è un unsigned long.
value = Millis ();  // imposta la variabile 'value' al numero di millisecondi Millis ()

Nota: Questo numero va in overflow (supera i limiti della memoria per cui ricomincia da zero), dopo circa 9 ore.

Duration = millis() – lastTime; //conta il tempo trascorso a partire da 'lastTime'

**OPERAZIONI MATEMATICHE**

**Min (x, y)**
Calcola il minimo di due numeri di qualsiasi tipo di dati e restituisce il numero più piccolo.

```
value = min(value, 100);        // assegna a 'value' il valore più piccolo di
                                // 'Value' o 100, assicurando che
                                // il risultato non superi 100.
```

**Max (x, y)**
Calcola il massimo di due numeri di qualsiasi tipo di dati e restituisce il numero più grande.

```
value =max (value, 100);        // assegna a 'value' il valore più grande di
                                // 'Value' o 100, assicurando che
                                // il risultato sia almeno 100.
```

**Abs(x)**
Restituisce il valore assoluto di x, che trasforma in positivi i numeri negativi. Se x è 10 restituirà 10, ma se x è –10 restituirà 10.

```
Value = abs(-10); // alla variabile value viene assegnato il valore 10
```

## GENERAZIONE NUMERI CASUALI (RANDOM)

### RandomSeed (seed)
Imposta un valore o un punto di partenza per generare un numero casuale (funzione random (). Esempio:

```
randomSeed (value);          // assegna a 'value' un valore casuale
```

Poiché il microcontrollore Arduino è in grado di creare un numero veramente casuale, la funzione randomSeed permette di inserire una variabile, una costante, o altre funzioni casuali, per generare numeri "casuali" ancora più casuali. Ci sono una varietà di possibilità o funzioni, che possono essere utilizzati in questa funzione; può essere utilizzato il comando millis () o anche analogRead () per leggere il rumore elettrico tramite un pin analogico.

### random (max)
### random (min, max)
La funzione casuale consente di avere numeri pseudo-casuali in un intervallo specificato di valori minimi e massimi.

```
value = random (100, 200);   // assegna a 'value' un numero casuale
                             // compreso tra 100 e 200
```

Nota: utilizzare questo comando dopo aver usato la funzione randomSeed ().

L'esempio seguente crea un valore casuale tra 0 e 255 e fornisce un segnale PWM su un pin PWM pari al valore casuale:

```
int randNumber;              // variabile per memorizzare il valore casuale
int Led = 10;                // un LED con una resistenza da 220Ω è presente
                             // sul pin 10
void setup () {}             // nessuna configurazione è necessaria
void loop ()
{
randomSeed (Millis ());      // imposta millis () come base per generare un
randNumber = random (255);   // numero casuale da 0 a 255
analogWrite (led, randNumber); // uscita segnale PWM
delay (500);                 // pausa per mezzo secondo
}
```

### Serial.begin (rate)
Apre la porta seriale e imposta la velocità di trasmissione seriale per la trasmissione dei dati. La velocità di trasmissione tipica per comunicare con il computer è 9600, anche se sono supportati altre velocità.

```
void setup ()
{
Serial.begin (9600); // apre la porta seriale
}                    // Imposta velocità di trasmissione a 9600 bps
```

Nota: quando si utilizza la comunicazione seriale, i pin digitali 0 (RX) e 1 (TX) non possono essere utilizzati contemporaneamente.

### Serial.println (data)
Stampa i dati alla porta seriale, seguita da un ritorno a capo automatico e avanzamento riga. Questo comando ha la stessa forma Serial.print (), ma è più facile per la lettura dei dati sul monitor seriale.

```
Serial.println (analogValue);      // invia il valore di
                                   // 'AnalogValue'
```

Nota: per ulteriori informazioni sulle varie commutazioni della funzione Serial.println () e Serial.print () fate riferimento al sito web di Arduino.

L'esempio che segue acquisisce un valore dal pin analogico 0 e invia i dati al computer ogni secondo.

```
void setup ()
{
Serial.begin (9600); // imposta la comunicazione seriale a 9600bps
}
void loop ()
{
Serial.println (analogRead (0)); // invia il valore analogico
delay (1000);                    // pausa per un secondo
}
```