Corso di programmazione per le scuole con Arduino – PARTE 1

Quello che segue, e che proseguirà nelle prossime puntate, è un corso per assoluti principianti. È un corso per chi non ha mai scritto una linea di codice, ma vuole imparare a programmare. E come base per imparare i fondamenti della programmazione, abbiamo scelto C++ e Arduino. Perché è lo strumento più semplice da programmare e, soprattutto, concreto. Chi si avvicina alla programmazione ha bisogno di riscontri immediati, deve vedere immediatamente quale possa essere l'applicazione pratica di un concetto, altrimenti finirà per annoiarsi e rinunciare a studiare. Arduino è la soluzione perfetta per cominciare perché con un paio di righe di codice si possono ottenere risultati tangibili, e stimolare la curiosità per i capitoli successivi del corso di programmazione.

Per i docenti delle scuole (secondarie di primo e secondo grado): sentitevi pure liberi di utilizzare questa serie di articoli come "libro di testo".

Table of Contents

Una introduzione per gli insegnanti

Nelle scuole italiane l'insegnamento dell'informatica è spesso trascurato, soprattutto per quanto riguarda la programmazione, che dovrebbe invece essere fondamentale per permettere agli alunni lo sviluppo della logica e della capacità di risoluzione dei problemi.

Sicuramente, una parte di questo atteggiamento deriva dalla tendenza tutta italiana a considerare i computer soltanto come macchine da scrivere molto costose, invece che come strumenti davvero interattivi. è per questo motivo che dal ministero dell'istruzione, nonostante le riforme si rincorrano ogni 5 anni circa, non sono mai arrivate linee guida che suggeriscano come utilizzare i computer per l'insegnamento. E tutto viene lasciato nelle mani dei docenti, che per quanta buona volontà possano avere spesso non dispongono delle basi di informatica perché non hanno mai fatto parte della loro formazione.

Negli altri paesi non è così: la formazione digitale è considerata decisamente importante, e uno dei requisiti per diventare insegnante. Il rapporto del 2017 (http://ec.europa.eu/newsroom/document.cfm?doc_id=44390), a pagina 3, ci presenta tra le ultime posizioni sia in termini di competenze avanzate che come competenze basilari. A pagina 9 è persino possibile notare che, se consideriamo solo i lavoratori, le competenze digitali sono ancora minori, e ci fanno scendere ulteriormente in classifica, fino al terz'ultimo posto. E non è che nel nostro paese i computer siano davvero poco diffusi: tutte le scuole ne hanno a disposizione. Il problema è il modo in cui è pensata l'informatica a livello ministeriale, con i programmi che sono sempre troppo datati e troppo nozionistici, con poca attenzione all'aspetto pratico. Per cui è inevitabile che anche la formazione dei docenti stessi ne risenta, visto che

la direzione verso cui si punta è sbagliata.

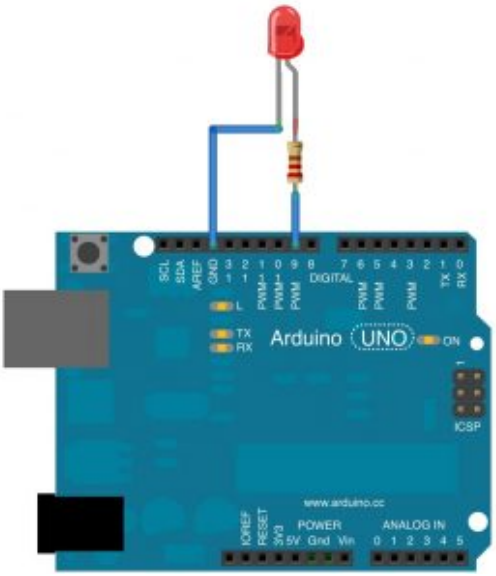
Nei rari istituti in cui si insegna la programmazione, solitamente scuole secondarie di secondo grado, lo si fa con strumenti e metodi obsoleti. Si utilizzano ancora Pascal e Fortran, linguaggi con cui uno studente oggi può fare ben poco di pratico e ai quali quindi non si affezionerà. Gli studenti vogliono qualcosa da poter esibire agli amici, e un programma a riga di comando che somma un paio di numeri non è un gran trofeo. E non è nemmeno una cosa davvero utile, perché per la maggioranza delle operazioni che vengono descritte nei corsi di programmazione delle scuole esistono già altri programmi che risolvono il problema in modo molto più semplice e veloce. Siccome di solito le lezioni di programmazione sono tenute dai docenti di matematica, vengono descritte principalmente operazioni matematiche. Ma in realtà non ha senso: per imparare a fare la media di un elenco di numeri basta usare un foglio di calcolo, sarebbe una soluzione molto più adatta all'uso che nel mondo reale si fa dei computer. Se si vuole insegnare la programmazione bisogna prima di tutto spiegare il senso stesso della programmazione, cioè il motivo per cui valga la pena imparare come scrivere programmi. È un problema comune a molte discipline: gli studenti si annoiano sempre a realizzare riassunti nei compiti di italiano, e questo perché nessuno spiega loro qual è il senso stesso del riassunto (cioè imparare a estrarre informazioni da un testo e rielaborarle in modo proprio). Nessuno, studenti adolescenti in particolare, sarà mai ben disposto a fare qualcosa di cui non capisce l'utilità.

Procurarsi un arduino

Arduino è il minicomputer più diffuso tra gli artisti che vogliono rendere interattive le loro creazioni (è il motivo per cui venne creato in primo luogo), tra gli appassionati di modellismo che realizzano droni, ed anche tra i progettisti di

dispositivi intelligenti (in particolare per i dispositivi Internet of Things). Ma è anche molto utilizzato nella scuole, soprattutto nei paesi del Nord Europa, per avvicinare i bambini alla programmazione. Arduino ha infatti il pregio di essere estremamente semplice da programmare, e avere tante applicazioni pratiche capaci di stimolare l'interesse dei neofiti. In Italia non è ancora molto diffuso a questo scopo, ma abbiamo pensato di proporvi alcuni progetti interessanti con cui avvicinare alla programmazione i vostri figli e tutti gli amici che vorrebbero cominciare a scrivere codice ma si annoiano con i corsi teorici tradizionali. Del resto, una volta molti ragazzi diventavano programmatori perché attratti dalla possibilità di inventare un videogioco: oggi l'internet of things e la possibilità di costruire oggetti intelligenti, unendo la programmazione al bricolage, possono fungere da motivazione per avvicinarsi alla programmazione. Chi vuole procurarsi un Arduino originale può trovarlo su Amazon: per le scuole, la soluzione migliore è rappresentata da Farnell (https://it.farnell.com/arduino/a000066/arduino-uno-evaluation-board/dp/2075382), che permette pagamenti tramite il sistema MEPA contattando il servizio clienti per una quotazione personalizzata. In alternativa, un docente può semplicemente mettere dei fondi di tasca propria (o farli raccogliere dal rappresentante dei genitori) e comprare dei cloni di Arduino Uno su AliExpress (https://it.aliexpress.com/w/wholesale-arduino-uno-r3.html): costano mediamente 2,80 euro l'uno, quindi se si hanno 20 studenti bastano 56 euro per garantire a tutti un Arduino Uno. Tra l'altro, gli esempi che proponiamo possono essere realizzati anche usando un clone di Arduino Nano, che costa appena 1,70 euro. Così se uno studente brucia un paio di schede mandandole in corto circuito non è un grave danno economico. Se poi non si vuole dare una scheda a ciascuno studente, ma dividere la classe in gruppi di lavoro, è possibile permettere la sperimentazione sul sito TinkerCAD: https://www.tinkercad.com/circuits. Si tratta di un simulatore completamente gratuito, che permette di disegnare un circuito

con la classica breadboard virtuale e testare il codice. È quindi perfetto per permettere agli studenti di provare sia il circuito che il codice del programma prima di caricarlo sul un vero Arduino e vedere come funziona. Chiaramente, oltre ad Arduino sono necessari anche alcuni componenti, come LED, sensori, buzzer, eccetera: i vari siti che abbiamo citato (Amazon, Farnell, AliExpress) offrono tutto il materiale di cui si può avere bisogno. Ci si può chiedere perché puntare su Arduino, piuttosto che su PLC Siemens che costano decine di euro ciascuno: il fatto è che Arduino costa molto meno, ha un'ottima documentazione, e se si compra la scheda originale invece dei cloni si ha già una certificazione CE (http://web.archive.org/web/20170320180212/https://www.arduino.cc/en/Main/warranty). Questo significa che gli studenti che imparano a usare Arduino a scuola potranno anche utilizzarlo in seguito nella loro carriera lavorativa. Per quanto i PLC siano al momento più diffusi nei macchinari industriali, infatti, Arduino è certamente il modo migliore per iniziare, e molte aziende stanno cominciando a puntare soprattutto sui modelli più piccoli (Arduino Nano e Micro) per le applicazioni IoT. E se si deve cominciare a imparare la programmazione, è sempre meglio bruciare un piccolo Arduino da 2 euro, piuttosto che un PLC da 200 euro.



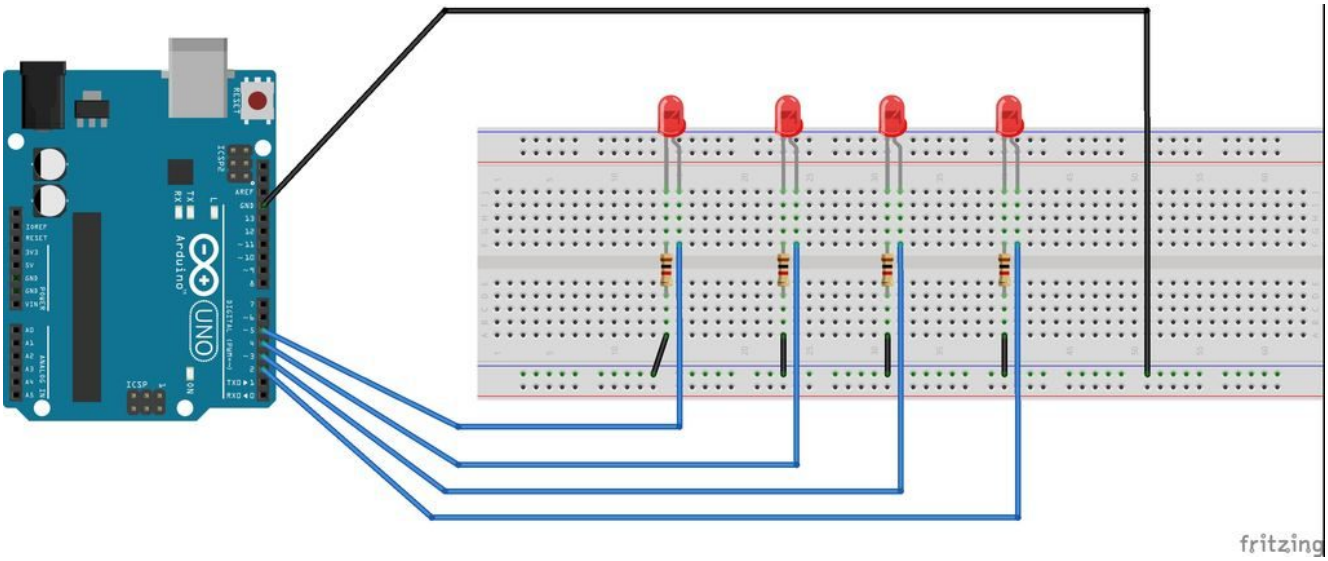
Un semplice Arduino UNO

con un LED che può essere acceso o spento con un programma

Ottenuto un Arduino, lo si può programmare con l'Arduino IDE, che si può recuperare dalla pagina https://www.arduino.cc/en/Main/Software. È importante non confondersi con l'Arduino Web Editor, che è un'altra cosa. L'Arduino IDE è un semplice programma gratuito e open source installabile su tutti i sistemi operativi, che permette di scrivere il codice dei propri programmi e caricarlo su una qualsiasi scheda Arduino (o derivate).

1 Una scala di led che si accendono in sequenza

Per cominciare, realizziamo un progetto semplice: una serie di led (Light Emissing Diode, piccole lampadine per circuiti elettrici a basso consumo) che si accenda a seconda della posizione di un potenziometro. In poche parole, avremo una rotella che si potrà girare per accendere da 1 a 5 lampadine. Un potenziometro è infatti una resistenza variabile, tipicamente una rotella od una leva che possiamo spostare per modificare la resistenza (come quelle con cui si cambia il volume sugli impianti stereo). Arduino è in grado di leggere un valore numerico in funzione della resistenza: il valore è compreso tra 0 e 1023, e noi possiamo scrivere un programma che riconosce questo numero ed in base alla sua posizione nell'intervallo 0-1023 accende 5 led.



Con una breadboard è facile collegare tanti led senza bisogno di saldare nulla

Per esempio, significa che se il potenziometro è a 0 tutti i led saranno spenti, se è a 1023 saranno tutti accesi, e se è a 200 soltanto due led saranno accesi. Per poter leggere il numero, il potenziometro va collegato ad uno dei pin analogici, mentre i led devono essere collegati a dei pin digitali, così potremo tenerli accesi o spenti. Il pin di sinistra del potenziometro va collegato al GND di Arduino, il pin centrale va connesso ad uno dei pin analogici, ed il pin di destra va connesso ai 5V di Arduino. Il pin lungo dei led va collegato ad uno dei pin digitali di Arduino, mentre il pin corto di ogni led va connesso al GND di Arduino. Il codice, che possiamo scrivere direttamente nell'Arduino IDE, è il seguente:

Prima di tutto definiamo alcune variabili: sono delle semplici parole a cui viene associato un valore di qualche tipo. Per esempio, una variabile di tipo **int** contiene un numero intero (mentre i numeri con la virgola sono di tipo **double**). La variabile chiamata **potPin** è quella che indica il pin (analogico) di Arduino cui abbiamo collegato il potenziometro: nell'esempio, si tratta del pin numero 2. Similmente, le varie variabili **ledPin** indicano i pin (digitali) di Arduino cui abbiamo collegato i led: ce ne sono 5 perché ciascuna di esse

indica il pin di uno dei 5 led del nostro progetto. In pratica, abbiamo collegato ad Arduino 5 diversi led, dal pin 3 al 7. Il nome delle variabili è chiaramente a nostra discrezione, avremmo potuto chiamarle “arancia”, “mela”, e “banana”, e si dichiarano sempre nella forma TIPO NOME = VALORE. L’assegnazione del primo valore non è fondamentale, ma è buona norma, e in questo caso si parla di “inizializzazione”. Ora, definiamo la nostra prima funzione:

La funzione setup è una delle due funzioni standard di arduino (l’altra è loop). Tutto il codice contenuto dentro questa funzione verrà eseguito una sola volta, ovvero all’avvio di arduino (appena viene acceso). Una funzione, di fatto, non è altro che un blocco di codice, un po’ come un capitolo all’interno di un libro. Ce ne sono molte già pronte all’uso, il cui codice è già stato scritto da qualcun altro, ma se vogliamo possiamo anche scrivere noi delle funzioni. Quando scriviamo una funzione dobbiamo indicare il suo tipo (void è un tipo molto comune, perché significa che la funzione fa qualcosa ma poi non fornisce un risultato che possa essere registrato dentro una variabile) ed il suo nome, cominciando poi a scrivere il suo codice con la parentesi graffa aperta. I tipi delle funzioni sono gli stessi delle variabili, perché sono tipi di dato generici. Per esempio, una funzione dichiarata come **int** potrà fornire un risultato che è un numero intero. Ma per ora ci bastano le funzioni che svolgono delle operazioni senza però fornire un risultato in termini numerici. Per esempio, nella funzione **setup** del nostro programma cominciamo a scrivere questo codice:

Qui stiamo usando la funzione (che non dobbiamo scrivere noi, è integrata in Arduino) chiamata **pinMode**, ed è anch’essa una funzione **void**, cioè una che non fornisce un risultato numerico ma si limita a fare cose. La funzione ci permette di abilitare dei ping per un segnale elettrico di output. I pin digitali di Arduino possono infatti avere due diverse modalità di

funzionamento: INPUT ed OUTPUT. Siccome i nostri pin sono collegati a dei led, saranno ovviamente pin di output, perché non ci servono per fornire ad Arduino dei dati, ma piuttosto per fornire una informazione da Arduino ai led stessi (cioè, dobbiamo dire ai LED se debbano rimanere accesi o spenti). Quindi utilizzando la funzione **pinMode** possiamo impostare la modalità OUTPUT a tutti i vari pin dei led connessi ad Arduino. Il bello delle funzioni è proprio questo: quando una funzione è stata scritta, poi può essere chiamata ogni volta che vogliamo, eventualmente anche con degli argomenti (cioè informazioni utili alla procedura desiderata, come in questo caso il numero del pin e la modalità). Questo ci risparmia di dover scrivere ogni volta tutto il codice, scrivendo quindi solo una riga. Esistono molte funzioni già presenti in Arduino, quindi non abbiamo bisogno di scriverle di nostro pugno, possiamo direttamente chiamarle.

Il nostro programma è molto semplice, quindi possiamo chiudere qui la funzione setup, utilizzando una parentesi graffa chiusa. Riassumendo, la nostra funzione **setup**, eseguita automaticamente una sola volta all'accensione di Arduino, si limita a chiamare a sua volta la funzione **pinMode** per impostare i pin dei LED come output invece che input.

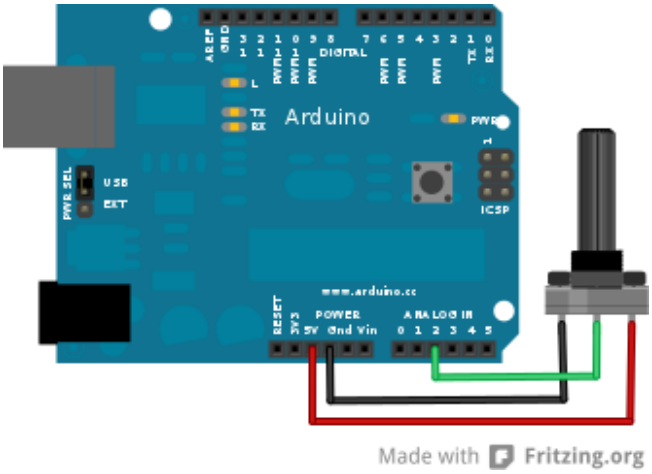
La funzione loop

Ora cominciamo a scrivere l'altra funzione standard di Arduino, chiamata loop.

Il codice contenuto dentro questa funzione verrà ripetuto all'infinito, finché Arduino non verrà spento. Quindi, è il codice che fa effettivamente qualcosa.

Definiamo una nuova variabile chiamata **val**, il cui valore iniziale sia 0 (per convenzione si indica sempre un valore

iniziale pari a 0 per tutte le variabili, ma non è obbligatorio). Poi assegniamole il valore fornito dalla funzione **analogRead**. Questa è la funzione che legge il segnale del pin analogico che indichiamo: nel nostro caso abbiamo indicato il pin analogico cui è connesso il potenziometro, quindi la variabile val conterrà il numero, compreso tra 0 e 1023, relativo alla posizione del potenziometro.



Collegare un potenziometro a Arduino è facile

Per il momento, abbiamo solo letto la posizione del potenziometro, quindi sappiamo come è posizionata la rotella (per esempio, 0 sarà tutto a sinistra, 1023 tutto a destra, 500 circa a metà).

Ora la variabile **val** contiene un numero compreso tra 0 e 1023, ma per noi questo è scomodo: abbiamo 5 led, quindi ci farebbe comodo ricondurre il numero registrato dal potenziometro ad un intervallo compreso tra 0 e 5. Possiamo farlo chiamando la funzione **map**, che si occupa proprio di mappare (cioè tradurre) una variabile da un intervallo ad un altro. Nel caso specifico, chiediamo alla funzione map di mappare la variabile val dall'intervallo che da va 0 a 1023 sull'intervallo che va da 0 a 5. Il risultato di questa traduzione verrà memorizzato nella variabile **mappedval**, che abbiamo appena creato (con la riga int mappedval). Vale a dire che invece di avere un numero

compreso tra 0 e 1023, per indicare la posizione della rotella, ne avremo uno compreso tra 0 e 5. Questa cosa si può fare perché la funzione map (che è sempre integrata in Arduino) non è una **void**, ma una **int**, e quindi ci fornisce un risultato numerico sotto forma di numero intero. Questo risultato può essere assegnato a una variabile semplicemente con il simbolo di uguaglianza.

Ora dobbiamo cominciare ad accendere i vari led: ovviamente, la loro accensione dipenderà dal valore appena ottenuto, che è memorizzato nella variabile **mappedval**. Secondo la logica del nostro progetto, il primo led si accenderà se il valore è superiore a 0, il secondo si accenderà se il valore è superiore ad 1, e così via (ricordiamo che lavoriamo con numeri interi, quindi non esistono valori come 0,3 o 1,5, vengono automaticamente arrotondati al numero intero più vicino). Possiamo ottenere questo risultato con un blocco if: i blocchi sono piccole porzioni di codice che hanno una qualche forma di controllo per la loro esecuzione. Per esempio i blocchi if (che in inglese significa il condizionale “se”) sono porzioni di codice che vengono eseguite solo se una certa condizione è valida, mentre in caso contrario vengono completamente ignorate. Scrivendo **if (mappedval>0)**, tutto il codice che indichiamo dopo la parentesi graffa viene eseguito soltanto se la variabile **mappedval** ha un valore maggiore di 0.

Il codice che vogliamo eseguire in tale situazione è ovviamente molto semplice: vogliamo solo accendere il primo led. Per accendere un led con Arduino basta dare un valore alto (HIGH, ovvero 5 Volt) al pin cui il led è collegato, in questo caso ledPin1. Il valore può essere assegnato al pin chiamando la funzione **digitalWrite**.

Analogico o digitale?
La differenza fondamentale tra i due tipi di pin presenti su Arduino è che il primo può gestire segnali analogici, ed il secondo segnali digitali. Ma cosa significa questo, in breve? In pratica, un segnale analogico può avere tutta una serie di valori che, solitamente, spaziano dallo zero all'uno (ad esempio 0,4). Un segnale digitale, invece, può essere solamente 0 oppure 1. Questo non significa che uno sia meglio dell'altro: dipende tutto da ciò che dobbiamo fare. Se vogliamo semplicemente avere (o dare) una informazione del tipo si/no un pin digitale è perfetto. Se, invece, vogliamo utilizzare una diversa scala di valori, ci conviene utilizzare un segnale analogico. Quest'ultimo è, dunque, il migliore per l'uso di sensori ambientali: esistono, comunque, dei sensori digitali, anche se sono più rari. Per quanto riguarda Arduino, i pin digitali sono 12, mentre quelli analogici sono 6. Inoltre è necessario inizializzare solo i pin digitali (scrivendo nel programma la riga pinMode(10, OUTPUT); per specificare che il pin 10 deve essere usato in output).
Chiudiamo il blocco if e continuiamo con il codice:

Uno dei lati interessanti di un blocco if è che quando lo terminiamo, con la parentesi graffa, possiamo indicare un codice alternativo con la parola **else** (che significa "altrimenti"). Quindi, se la la condizione specificata tra parentesi tonde (cioè mappedval>0) è vera, viene eseguito il codice compreso tra le prime parentesi graffe. Altrimenti, se tale condizione non è vera, viene eseguito il codice presente tra le parentesi graffe immediatamente dopo la parola **else**.

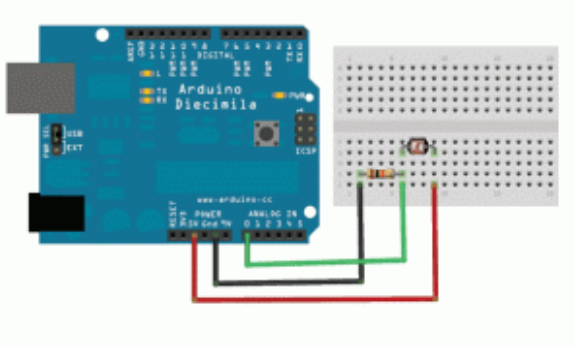
Naturalmente, noi vogliamo che se la condizione del ciclo if non è vera il led sia spento. Per farlo basta scrivere il valore basso (LOW, pari a 0 Volt) sul pin del led. Poi possiamo chiudere anche la parentesi graffa di else.

Si ripete per gli altri led

Il codice per il primo led è scritto, occupiamoci ora del secondo:

Similmente, costruiamo un ciclo if-else per il secondo led, identificato dal pin digitale ledPin2, che deve essere acceso (HIGH) solo se la variabile **mappedval** è maggiore di 1, e spento (LOW) in tutte le altre occasioni.

Si ripete lo stesso tipo di ciclo adattandolo agli altri tre led, così anch'essi possono essere accesi e spenti a seconda della posizione della rotella del potenziometro. Intuitivo, vero? Poi si può chiudere la funzione loop, che è quella di cui stavamo scrivendo il codice finora, con una parentesi graffa. Il programma è completo.



Il potenziometro può essere facilmente sostituito con una fotoresistenza

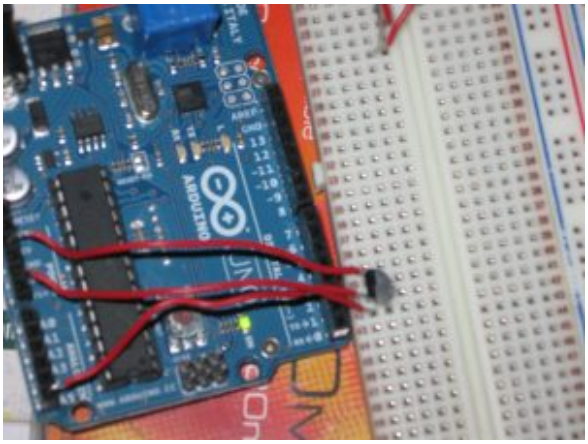
Volendo, si può anche sostituire il potenziometro con una fotoresistenza: sono dei semplici sensori di luminosità. Una fotoresistenza ha due pin perfettamente identici: uno va collegato al pin 5V di Arduino, l'altro al pin analogico 2. Quello collegato al pin analogico 2 va anche collegato, con una semplice resistenza da 10kOhm, al pin GND di Arduino. Similmente, si può sostituire il potenziometro con qualsiasi

altro sensore capace di dare un segnale analogico, per esempio anche un microfono, con il quale si otterrebbe un visualizzatore di volume audio (i led si accenderebbero sempre di più man mano che si parla più forte al microfono). Serve solo un po’ di fantasia.

2 Utilizzare un sensore LM35 per misurare la temperatura

Il potenziometro dell’esempio precedente è un sensore, perché permette di fornire dati ad Arduino. Esistono diversi altri sensori, che misurano informazioni relative all’ambiente: temperatura, pressione ed umidità, per esempio. Uno dei sensori più semplici da utilizzare è chiamato LM35, e misura la temperatura di una stanza. E possiamo utilizzare Arduino per leggerla, inviando il valore esatto al nostro computer attraverso il cavo USB. Cerchiamo innanzitutto di capire come funziona il sensore di temperatura LM35: contrariamente a quanto si potrebbe pensare, il sensore non ci fornisce direttamente la temperatura. Ci fornisce il solito valore compreso tra 0 e 1023, e questo valore è proporzionale alla temperatura. Con una semplice formula matematica siamo in grado di risalire alla temperatura corretta con una precisione di mezzo grado: è sufficiente moltiplicare il numero fornito dall’LM35 per circa 0,5 (esattamente per 500/1023=**0,488**, come indicato dai produttori del sensore) per ottenere la temperatura in gradi Celsius. È poi ovviamente possibile applicarle una serie di trasformazioni per cambiare scala: se per qualche motivo la volessimo in scala assoluta (i cosiddetti gradi Kelvin del sistema internazionale di misure) basterà sommare al valore ottenuto il numero 273,15. Il sensore andrà collegato ad Arduino nel seguente modo: guardando la scritta sul lato piatto, il pin a sinistra va collegato alla alimentazione a 5V, quello centrale va collegato al pin analogico 1 di Arduino, ed infine quello più

a destra va collegato con il pin GND di Arduino. Se preferiamo rendere impermeabile il sensore, in modo da poterlo anche mettere a contatto con liquidi, basta ricoprirlo con della pellicola trasparente per alimenti.



Per collegare un sensore di temperatura a Arduino bastano tre cavi e una breadboard

Tenete comunque presente che, anche se il sensore può sopportare temperature che variano dai -55 °C ai 140 °C, potrebbe non essere una buona idea sottoporlo a situazioni critiche. Ad esempio, non immergetelo nella pentola dell'acqua per vedere quando bolle. Potete però appoggiarlo ad un cubetto di ghiaccio per vedere quando fonde. Il codice è relativamente semplice:

Cominciamo anche questo programma dichiarando due variabili: una è un **numero intero**, si chiama **pin**, e rappresenta il pin analogico di Arduino cui abbiamo collegato il sensore LM35, nel nostro caso il pin numero 1. L'altra è di tipo **double** perché si tratta di un **numero con virgola**, si chiama temp e rappresenterà la temperatura rilevata dal sensore (per ora la impostiamo uguale a 0.0, non dimentichiamo che con Arduino si utilizza la notazione internazionale che prevede il punto al posto della virgola per le cifre decimali). Cominciamo subito la funzione setup:

La funzione setup (eseguita una volta sola all'avvio di Arduino) non fa altro che avviare la comunicazione tramite porta seriale, con una velocità (bitrate) di 9600 baud, che è lo standard più comune. La porta seriale ovviamente è la porta USB (Universal Serial Bus), tramite la quale potremo poi leggere i messaggi di Arduino con un apposito terminale dal nostro computer (il Monitor seriale dell'IDE di Arduino), al quale abbiamo collegato Arduino tramite il cavo USB.

La funzione loop, invece, (che è quella eseguita ripetutamente) legge il valore fornito dal sensore chiamando la funzione **analogRead**, che abbiamo già visto, moltiplicando immediatamente questo numero per il valore correttivo, ovvero 500/1023, assegnando il risultato dell'operazione alla variabile temp. In poche parole, abbiamo chiesto ad Arduino di calcolare il risultato di una semplice equazione: il numero fornito dal sensore viene moltiplicato per 500 e diviso per 1023, ed il risultato diventa il valore della variabile temp. Se al posto delle funzione **analogRead** avessimo scritto X ed al posto di temp avessimo scritto Y, l'avreste riconosciuta subito come una banale equazione di primo grado.

Ora possiamo scrivere dei messaggi al computer, il quale li riceverà sulla porta USB. I messaggi sono ovviamente dei testi (che in programmazione sono chiamati **stringhe**, e tipicamente delimitati dalle virgolette "), e possono essere inviati con la funzione **Serial.print**. Ogni messaggio è costituito da una riga, che termina quando chiamiamo la funzione **Serial.println()**. Possiamo quindi scrivere un messaggio composto dalle parole **Stanza1:** seguite dal valore della temperatura memorizzato nella variabile **temp** (che è un numero, ma viene automaticamente tradotto da Arduino in una stringa di testo) ed infine dal simbolo dei gradi Celsius. Il messaggio viene terminato con l'indicazione del termine della riga (**ln** in **println** significa **line new**, ovvero nuova riga).

L’ultima riga di codice della funzione loop dice ad Arduino di attendere 1 secondo (1000 millesimi di secondo, si ragiona sempre in millesimi di secondo) prima di eseguire nuovamente la funzione: questo significa che la temperatura verrà letta una volta al secondo. Possiamo cambiare questo valore se vogliamo che la temperatura sia letta più spesso o più raramente.

I pin del sensore LM35 sono facili da riconoscere: guardando il lato piatto, da sinistra abbiamo il pin 5V (positivo), quello analogico, e il pin GND (negativo)

Dopo avere caricato lo sketch su Arduino, noterete che il piccolo led TX della scheda lampeggia. A questo punto si può cliccare sul menù **Strumenti/Monitor seriale** dell’Arduino IDE: apparirà una nuova finestra in stile prompt dei comandi con una serie di scritte del tipo “Stanza 1: 20.00°C”, una riga per ogni secondo.