Università degli Studi di Padova

Anno accademico 2010/2011

Tesina:

Evoluzione dei linguaggi di programmazione educativi: Scratch e B.Y.O.B.

Laureando:
Almir Avdic, matricola 522918-IF

Relatore:
Moro Michele

Data:
29 Marzo 2011

Indice

1 - INTRODUZIONE

In informatica, un **linguaggio di programmazione** è un linguaggio formale, dotato (al pari di un qualsiasi linguaggio naturale) di un lessico, di una sintassi e di una semantica ben definiti, utilizzabile per il controllo del comportamento di una macchina formale o di una implementazione di essa (tipicamente, un computer).

Il primo linguaggio di programmazione della storia, se si esclude il linguaggio meccanico adoperato da Ada Lovelace per la programmazione della macchina di Charles Babbage, è a rigor di termini il Plankalkül di Konrad Zuse, sviluppato da lui nella Svizzera neutrale durante la seconda guerra mondiale e pubblicato nel 1946. Plankalkül non venne mai realmente usato per programmare. La programmazione dei primi elaboratori veniva fatta invece in short code[1], da cui poi si è evoluto l'assembly, che costituisce una rappresentazione simbolica del linguaggio macchina. La sola forma di controllo di flusso è l'istruzione di salto condizionato, che porta a scrivere programmi molto difficili da seguire logicamente per via dei continui salti da un punto all'altro del codice. La maggior parte dei linguaggi di programmazione successivi cercarono di astrarsi da tale livello basilare, dando la possibilità di rappresentare strutture dati e strutture di controllo più generali e più vicine alla maniera (umana) di rappresentare i termini dei problemi per i quali ci si prefigge di scrivere programmi. Tra i primi linguaggi ad alto livello a raggiungere una certa popolarità ci fu il Fortran, creato nel 1957 da John Backus, da cui derivò successivamente il BASIC (1964): oltre al salto condizionato, reso con l'istruzione IF, questa nuova generazione di linguaggi introduce nuove strutture di controllo di flusso come i cicli WHILE e FOR e le istruzioni CASE e SWITCH: in questo modo diminuisce molto il ricorso alle istruzioni di salto (GOTO), cosa che rende il codice più chiaro ed elegante, e quindi di più facile manutenzione. Dopo la comparsa del Fortran nacquero una serie di altri linguaggi di programmazione storici, che implementarono una serie di idee e paradigmi innovativi: i più importanti sono l'ALGOL (1960) e il Lisp (1959). Tutti i linguaggi di programmazione oggi esistenti possono essere considerati discendenti da uno o più di questi primi linguaggi, di cui mutuano molti concetti di base; l'ultimo grande progenitore dei linguaggi moderni fu il Simula (1967), che introdusse per primo il concetto (allora appena abbozzato) di *oggetto* software. Nel 1970 Niklaus Wirth pubblica il Pascal, il primo linguaggio strutturato, a scopo didattico; nel 1972 dal BCPL nascono prima il B (rapidamente dimenticato) e poi il C, che invece fu fin dall'inizio un grande successo. Nello stesso anno compare anche il Prolog, finora il principale esempio di linguaggio logico, che pur non essendo di norma utilizzato per lo sviluppo industriale del software (a causa della sua inefficienza) rappresenta una possibilità teorica estremamente affascinante. Con i primi mini e microcomputer e le ricerche a Palo Alto, nel 1983 vede la luce Smalltalk, il primo linguaggio realmente e completamente ad oggetti, che si ispira al Simula e al Lisp: oltre a essere in uso tutt'oggi in determinati settori, Smalltalk viene ricordato per l'influenza enorme che ha esercitato sulla storia dei linguaggi di programmazione, introducendo il paradigma object-oriented nella sua prima incarnazione *matura*. Esempi di linguaggi object-oriented odierni sono Eiffel (1986), C++ (che esce nello stesso anno di Eiffel) e successivamente Java, classe 1995.

Non ha senso, in generale, parlare di linguaggi migliori o peggiori, o di linguaggi migliori in assoluto: ogni linguaggio nasce per affrontare una classe di problemi più o meno ampia, in un certo modo e in un certo ambito. Però, dovendo dire se un dato linguaggio sia adatto o no per un certo uso, è necessario valutare le caratteristiche dei vari linguaggi. Distinguiamo

tra due tipi di caratteristiche principali: le caratteristiche intrinseche, e quelle esterne. Tra le caratteristiche intrinseche: espressività, leggibilità, generalità, robustezza, modularità, flessibilità, efficienza e coerenza, quella che interessa di più a noi e' la **didattica.** Questa caratteristica indica la semplicità del linguaggio e la rapidità con cui lo si può imparare. Il BASIC, per esempio, è un linguaggio facile da imparare: poche regole, una sintassi molto chiara e limiti ben definiti fra quello che è permesso e quello che non lo è. Il Pascal non solo ha i pregi del BASIC ma educa anche il neo-programmatore ad adottare uno stile corretto che evita molti errori e porta a scrivere codice migliore. Al contrario, il C non è un linguaggio didattico perché pur avendo poche regole ha una semantica molto complessa, a volte oscura, che lo rende molto efficiente ed espressivo ma richiede tempo per essere padroneggiata.

Negli ultimi anni, i dipartimenti di informatica presso diverse università stanno compiendo sforzi per attirare un maggiore numero di studenti. Alcuni paesi, ad esempio gli Stati Uniti d'America, hanno dichiarato che risolvere la carenza di studenti in campo informatico è una priorità nazionale. Uno dei metodi principali di questo sforzo è l'abbandono di un approccio eccessivamente improntato sulle tecniche di programmazione e la ricerca di un linguaggio che eviti la complessità sintattica.

Il Logo è un linguaggio di programmazione per computer creato da Seymour Papert nel 1967 all'interno del MIT (Massachusetts Institute of Tecnology). Esso è stato pensato come strumento per agevolare e migliorare l'apprendimento. Le sue caratteristiche possono essere riassunte nelle espressioni: modularità, estensibilità, interattività e flessibilità. La programmazione con il Logo infatti non è mai fine a se stessa, ma sempre pensata in relazione ad un progetto relativo, alle discipline più svariate: alla matematica, alla lingua, alla musica, alla realizzazione di un videogioco o di un robot... Fra gli ambienti di apprendimento che il Logo offre, quello più conosciuto e sicuramente più usato nelle scuole, anche in Italia, è la Geometria della Tartaruga. Originariamente la Tartaruga era un robot che si muoveva su una superficie tramite comandi impartiti attraverso un computer. In seguito divenne uno strumento grafico, fu trasferita sul monitor del computer ed usata per disegnare e creare immagini. Nel corso degli anni '70 il Logo ha cominciato a diffondersi e sono state sviluppate diverse esperienze soprattutto negli Stati Uniti; nel 1980 è nata la LCSI (Logo Computer Systems Inc.), che crea, implementa e produce le varie versioni di Logo che conosciamo anche in Italia. Nello stesso anno Seymour Papert pubblica Mindstorms , tradotto in Italia con il sottotitolo: "Ali per la mente", un testo in cui non soltanto l'autore ha spiegato le caratteristiche e le potenzialità del Logo, ma ha anche delineato le idee portanti di un tipo di pedagogia che assieme ad altri modelli teorici è compresa sotto il nome di costruttivismo. Nel corso degli anni '80 l'uso del Logo è cresciuto considerevolmente e sono state realizzate molte esperienze anche in Italia. Negli Stati Uniti intanto, alla LCSI viene introdotto il Logo Writer, che oltre al linguaggio di programmazione include un word processor ed un'interfaccia semplificata e più intuitiva. Un'altra innovazione di quegli anni è costituita da LEGOLogo. E' stato cioè ideato un sistema che usa il Logo come interfaccia per motori, luci e sensori incorporati nelle macchine costruite dalla LEGO. Questo esperimento è diventato negli Stati Uniti un grande successo commerciale che ha richiamato l'attenzione

di molti insegnanti e studenti.

Ora ci concentriamo su due linguaggi che sono stati influenzati da LOGO e che ne hanno migliorato l'aspetto grafico, rendendolo ancora piu' accessibile ai ragazzi di qualsiasi età, compresi i più piccoli: Scratrch e B.Y.O.B.In particolare ci concentriamo su due linguaggi di programmazione educativi, pensati per essere comprensibili ai ragazzi di qualsiasi età, compresi i più piccoli: Scratch e B.Y.O.B.

2 - SCRATCH

Scratch e' un linguaggio di programmazione educativo creato da Michael Resnick e dal suo team al MIT (Massachussets Institute of Technology). L'atto di programmazione è stato pensato per i bambini, completo di immagini ed icone, e la tecnologia drag'n'drop. La sua semplicità sta nel fatto di poter sviluppare programmi, semplicemente trascinando e combinando tra di loro gli oggetti presenti nel menù. Questa proprietà lo rende molto accessibile e facilmente comprensibile a ragazzi di qualsiasi età senza la necessità di fare loro corsi di programmazione informatica. E' stato pensato principalmente per ragazzi dagli 8 anni in su, ma i creatori hanno promosso il suo utilizzo anche con bambini di età inferiore a 8 anni, e studenti di college come strumento introduttivo.

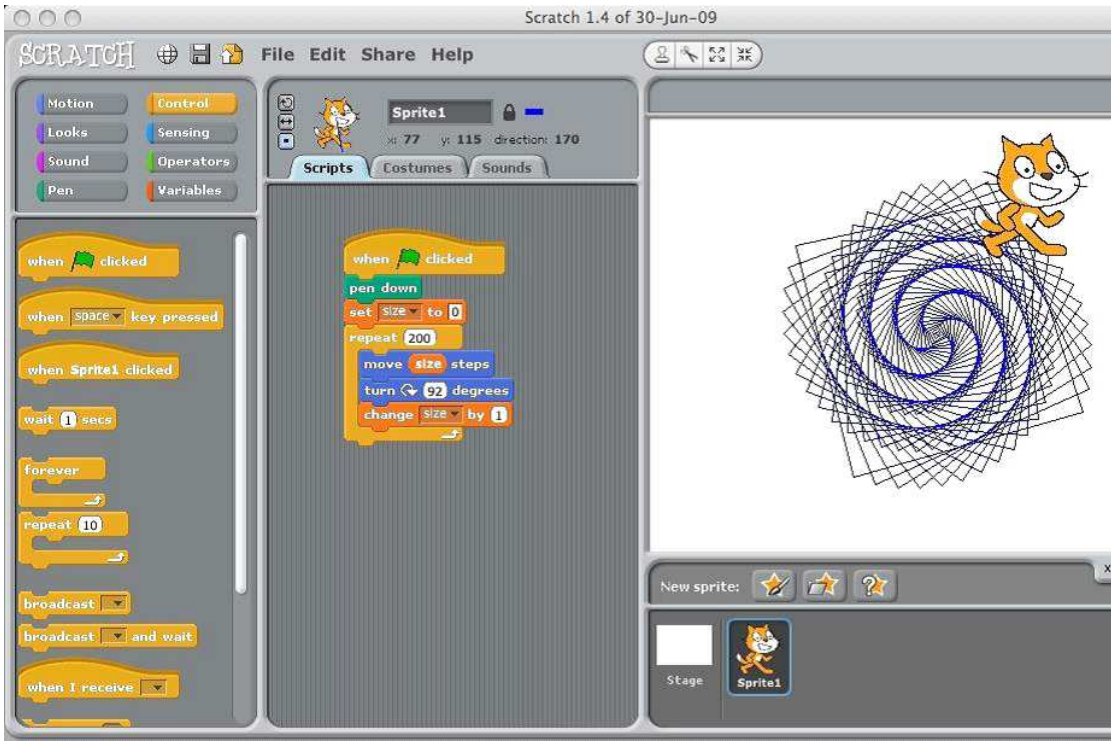


Figure 1. The Scratch menus, scripting area, and stage

Scratch illustra la programmazione orientata agli oggetti in forma di molteplici "sprite" animati presi o dalla propria library, o importati da un qualsiasi file immagine. Ogni "sprite" ha una propria area di script e le sue variabili di stato locali. Un "broadcast" primitivo fornisce una forma rudimentale di message passing, ma con le limitazioni che i messaggi non possono essere indirizzati ad un singolo sprite, e gli script non possono avere degli argomenti o dei valori di ritorno.

Contribuendo a formare la capacità di pensare in modo creativo, comunicare e analizzare, utilizzare le tecnologie, collaborare e progettare, Scratch ha un codice sorgente basato su

Squeak, a sua volta basato su Smalltalk, linguaggio nato negli anni settanta che ha pesantemente influenzato alcuni linguaggi come Objective C, C#, C++, Actor, Java e Ruby. E' gratuito e funziona su Windows e Mac OS X (la versione per Linux è in fase di costruzione). C'è una comunità online di Scratch, al sito **http://scratch.mit.edu/**. Cui si possono aggiungere i propri progetti , e scaricare i progetti già presenti, così come lasciare commenti.

Ma Scratch ha anche dei punti deboli. Il fatto di essere così semplice ed intuitivo, ha fatto si che non fosse molto solido per poter contenere strutture di dati complesse. Un'altra carenza di Scratch, forse la più importante, è la mancanza della "ricorsione", concetto fondamentale dell'Informatica. Per fare in modo di rendere Scratch accessibile a tutti, e' stato proposto di dividerne lo sviluppo in due versioni del linguaggio, una per bambini e ragazzi ed una per utenti avanzati.

L'evoluzione di Scratch per utenti avanzati si chiama B.Y.O.B. (Build Your Own Block).

3 - BYOB

L'ultima versione di BYOB (3.1) è uscita il 23 febbraio 2011 e permette, come dice il nome stesso, la creazione di blocchi propri di programmazione. Inoltre consente l'utilizzo e l'implementazione di liste, procedure e sprite di prima classe.

- **- Costruzione di un blocco**

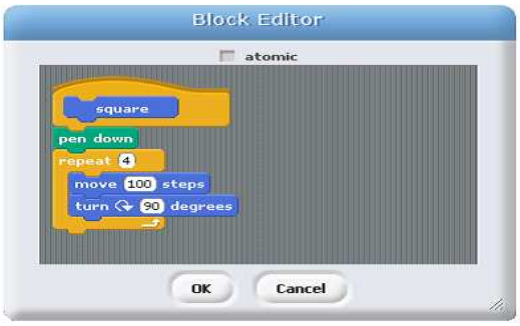
    - **Semplici blocchi**

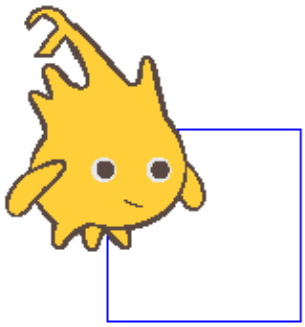
Nella palette delle variabili, in basso, c'è il tasto "Make a Block". Premendolo si apre una tavolozza che permette di selezionare tra otto categorie principali di blocco. Le categorie sono: movimento, sguardo, suono, recinto, controllo, rilevamento, operatore e variabile, con l'aggiunta di lista e altro. Ogni categoria ha anche un colore che serve per identificarla più facilmente.

Ci sono tre forme principali di blocchi, a seguito di una convenzione che dovrebbe essere familiare agli utenti di Scratch:

- i blocchi a forma di puzzle sono i comandi e non riportano valori;
- i blocchi ovali sono i Reporters (gli output);
- i blocchi esagonali sono i predicati (controlli delle condizioni);

Supponiamo di voler creare un blocco denominato "piazza" che semplicemente disegna un quadrato. Si crea un blocco iniziale detto "cappello" del blocco, sotto il quale verranno posizionati i blocchi che faranno in modo di disegnare un quadrato. Otterremmo un blocco di questo tipo ed il risultato:

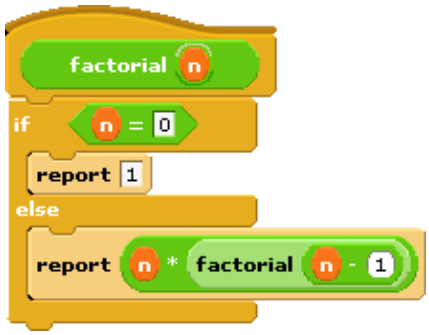




Da questo blocco iniziale possiamo creare quadrati di dimensioni diverse, semplicemente aggiungendo una variabile al blocco iniziale “square”. Sulla finestra del Block Editor, in alto c’è una checkbox denominata “atomic”che se selezionata, fa eseguire lo script intero del blocco in un unico ciclo (come in Scratch). Questo crea un vantaggio in quanto lo script viene eseguito più velocemente, ma allo stesso tempo si ha lo svantaggio di vedere solamente il risultato finale del blocco, e non i passi intermedi.

- **Blocchi ricorsivi**

Dal momento che il nuovo blocco personalizzato viene visualizzato nella sua tavolozza non appena si inizia a modificarlo, per creare blocchi ricorsivi, trascinando il blocco nella sua stessa definizione. Un esempio di blocco ricorsivo e’ i calcolo del fattoriale:



Da notare l’utilizzo del blocco “*report*”. Questo blocco è molto simile al blocco “*stop script*”, con la differenza che quest’ultimo blocca non solo l’invocazione del blocco corrente, permettendo ai blocchi al suo interno di girare, ma l’intero script.

- **– First Class Lists**

Un tipo di dati è di “prima classe” in un linguaggio di programmazione, se i dati possono essere:

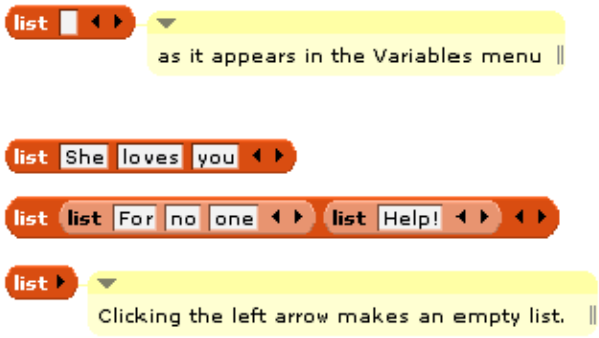
- il valore di una variabile
- l’ input di una procedura
- il valore restituito da una procedura
- un elemento di un insieme di dati
- un anonimo

In Scratch 1.4, numeri e stringhe di testo sono dati di prima classe. Mentre invece non lo

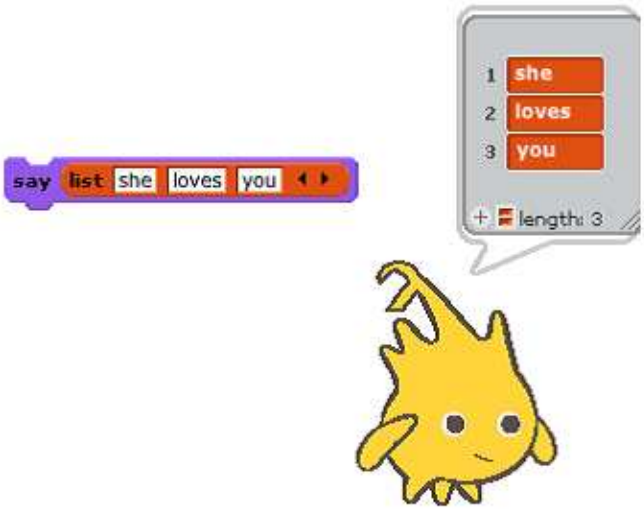
sono le liste. Un principio fondamentale nella progettazione di BYOB è che tutti i dati dovrebbero essere di prima classe.

- **Il blocco lista**

Il blocco che si occupa della creazione di liste di prima classe è il blocco reporter *"lista"*.

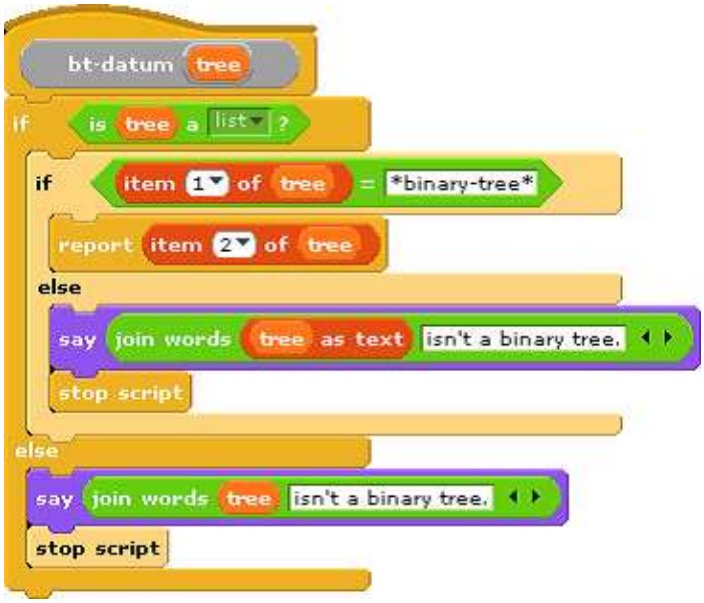


A destra del blocco ci sono due frecce, che servono per aggiungere o diminuire il numero di elementi presenti in una lista. Si può utilizzare questo blocco come input per molti altri blocchi:



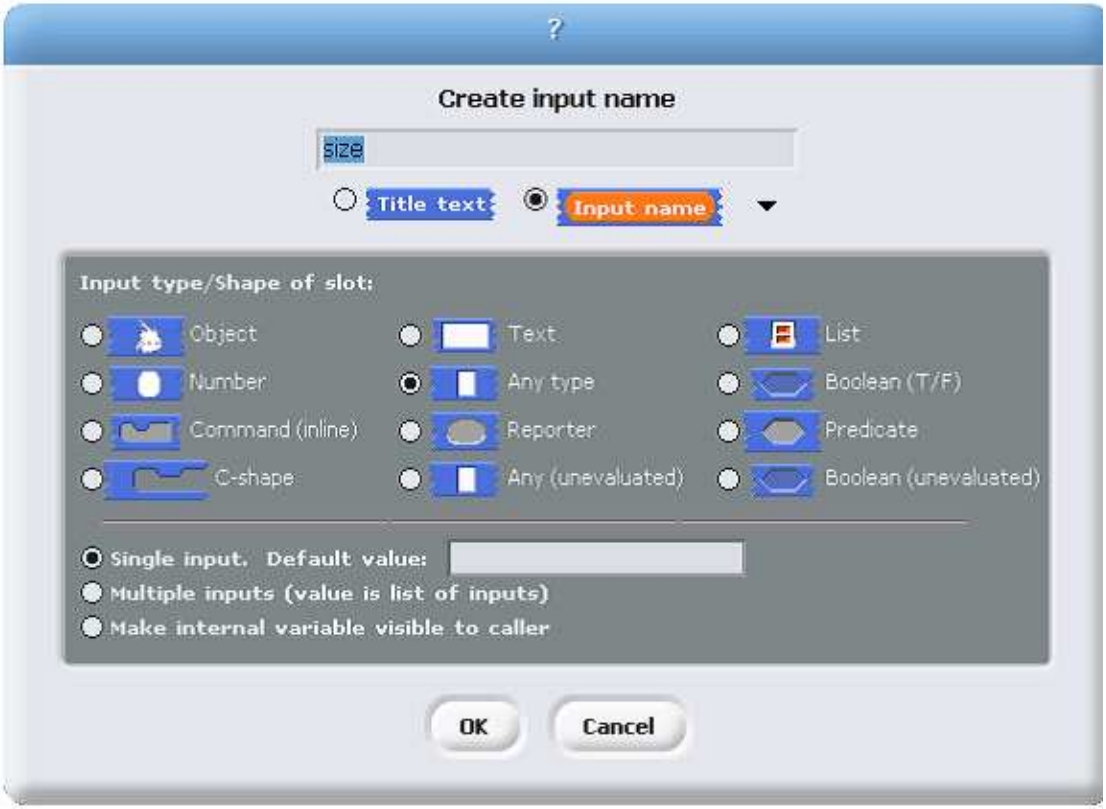
- **Liste di liste**

Le liste possono essere facilmente inserite in altre liste di dimensioni maggiori, creando così una classica struttura di dati come un albero binario:



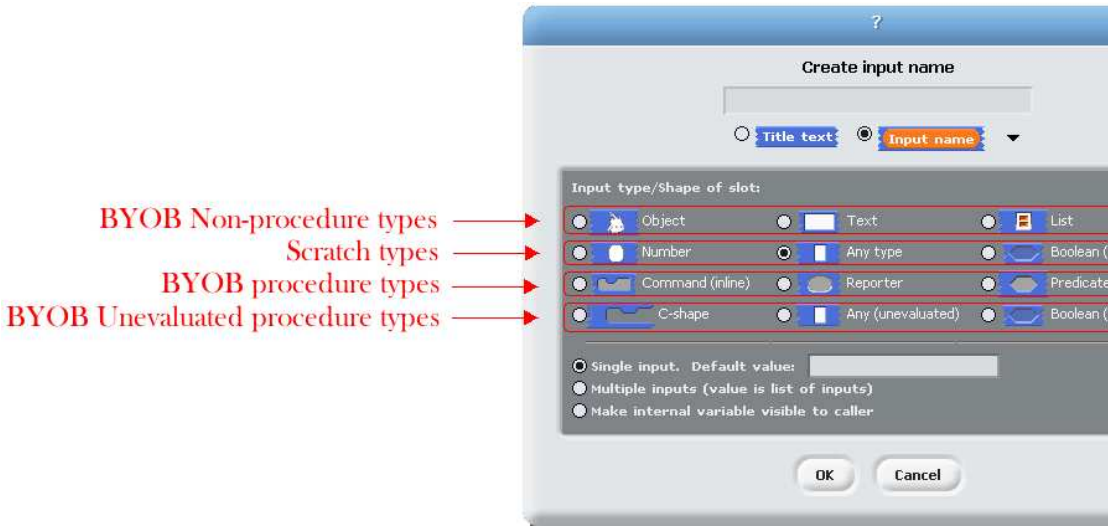
- **- Input**

In Scratch gli ingressi di un blocco sono di due tipi: testo o numero.
In BYOB, e' prevista una procedura di creazione input che permette di gestire procedure, liste e oggetti:

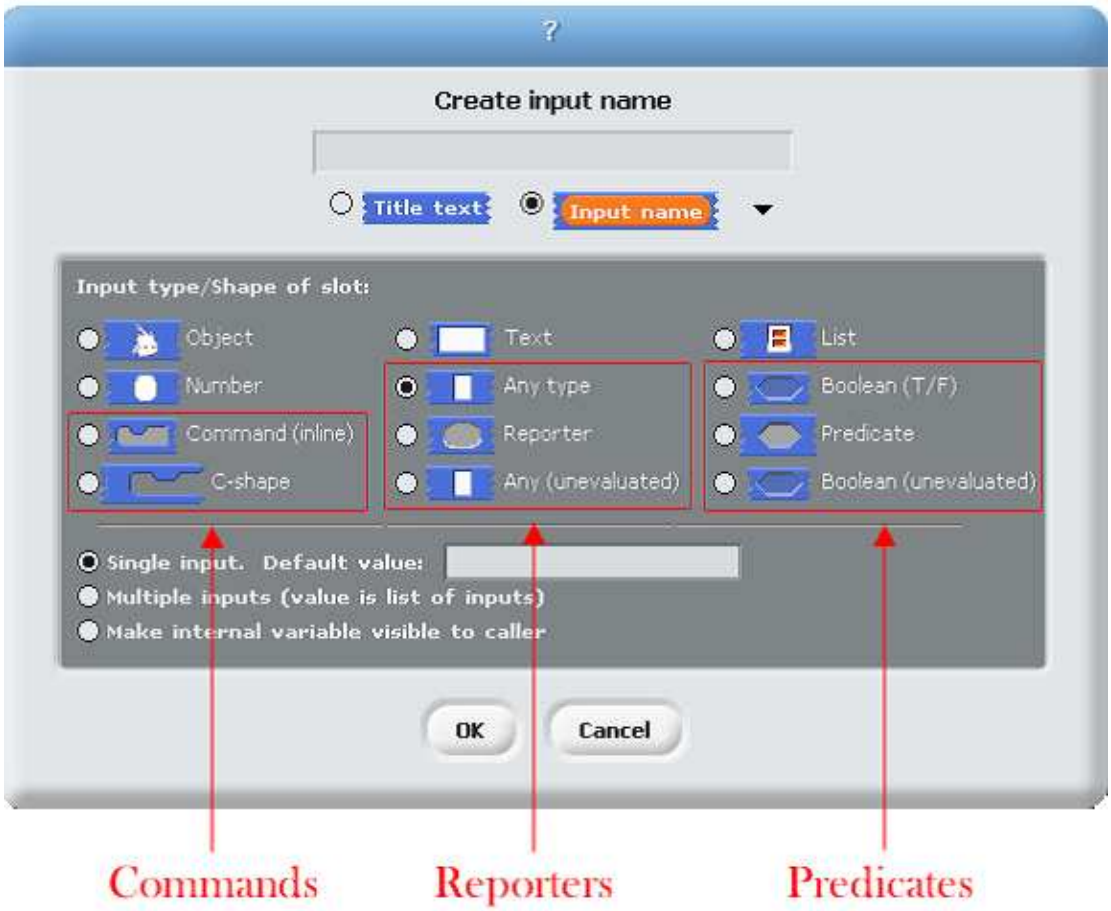


Ci sono ben dodici tipi di ingressi, più tre categorie esclusive. Il posizionamento delle

categorie non e' causale, infatti si suddividono per righe e per colonne, come mostrano le due immagini successive:



La prima riga contiene i nuovi tipi inseriti in BYOB, ad esclusione delle procedure, la seconda riga contiene i tipi di ingressi presenti anche in Scratch, la terza e la quarta riga contengono i tipi di ingressi relativi alle procedure.
Per comprendere la suddivisione in base alle colonne, basta osservare l'immagine seguente:



Scratch ha tre tipi di forme per i blocchi: puzzle per i blocchi di comando, ovali per i blocchi reporter ed esagonali per i blocchi condizionali.
Per quanto riguarda le tre categorie esclusive, invece:

- single input: tutti gli ingressi di Scratch sono di questo tipo. Si può specificare inoltre un valore di default, che verrà mostrato nella tavolozza di programmazione. Ad esempio il blocco "square" con valore di default = 10:

- multiple inputs: il blocco lista introdotto precedentemente accetta qualsiasi numero di elementi per il nuovo ingresso. Per consentire ciò, BYOB introduce la notazione delle frecce per espandere e contrarre il blocco (◂ ▸).

- make internal variable visible: questo tipo di blocco non è realmente un ingresso, ma più una sorta di uscita dal blocco per l'utente. Una freccia che punta verso l'alto indica questo tipo di input:

La variabile "i" può essere trascinata dal blocco for ed utilizzata per i blocchi di tipo puzzle. Cliccando sulla variabile, possiamo cambiarle nome, anche se questo non cambia nel blocco di input. Questo tipo di variabile viene chiamato "upvar", perché passato verso l'alto dal blocco personalizzato per lo script che lo utilizza.

- **- Procedure di dati**

  - **Procedure input types**

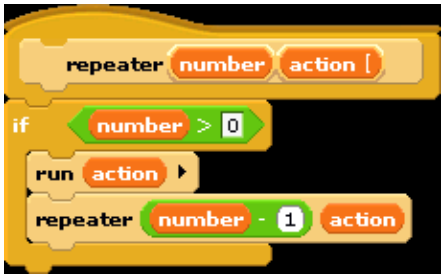
Nel blocco *for* dell'esempio precedente, l'azione di input è stata dichiarata come blocco di tipo Command, per cui a forma di C. Ma come fa il blocco ad ordinare a BYOB di eseguire le operazioni contenute nel blocco C? Nell'immagine che segue vi è una semplice versione dello script del blocco:

E' semplificato perché assume che il valore iniziale sia minore del valore finale, altrimenti lo

script potrebbe o non funzionare, o poter decrementare il valore di i, oltre che incrementarla. La parte importante di questo script è il blocco "**r***un*". Questo è in blocco di BYOB che prende un blocco di comando in input e ne esegue le istruzioni. C'è un blocco "*call*" simile per invocare un blocco di tipo report. I blocchi call e run sono il cuore delle funzionalità delle procedure di prima classe di BYOB, infatti consentono di poter utilizzare degli script o dei blocchi come input. I blocchi call e run hanno due frecce (destra e sinistra) per aggiungere e rimuovere input. Se il numero di input in ingresso ad un blocco call è lo stesso del numero di slots di input, allora, i primi verrano automaticamente riempiti dai secondi con lo stesso ordine (primo input = primo slot, secondo input = secondo slot, ecc…). Nel caso in cui ci sia solo uno slot di input, allora tutti gli input della funzione call verranno valorizzati con lo stesso valore.

- **Scrivere procedure di livello maggiore**

Una procedura di livello superiore è detta tale se prende come input un'altra procedura. Ma perché vogliamo avere procedure con in input altre procedure? I blocchi condizionali e ciclici di Scratch ci permettono di scrivere uno script come input: possiamo scrivere il blocco "*repeat*" come un blocco custom in questo modo:



La parentesi quadra vicino ad action indica che questo è un blocco di tipo Command e che lo script racchiuso nella forma C è l'azione di input dello script.
Per dichiarare una procedura di input, aprire la dialog di creazione input e cliccare sulla freccia per espandere la finestra e visualizzare tutte le opzioni a disposizione, e quindi scegliere il tipo di procedura:

Gli elementi della terza riga hanno lo sfondo grigio per i vari tipi di blocchi (comandi, reporter, condizioni). Nel caso dei comandi viene più spesso utilizzato il blocco C-shape, perché più familiare agli utenti di Scratch. Tecnicamente, il blocco di tipo C, è un tipo di procedura non valorizzata. I due input di tipo comando sono legati dal fatto che se una variabile, un elemento di un blocco lista, o un blocco personalizzato reporter, è inserito in un blocco C, si trasforma in un blocco online, come nella chiamata ricorsiva al blocco ripetitore dell'esempio precedente.

Ora che abbiamo le procedure come input per i blocchi come le usiamo? Usiamo i blocchi "run" per i comandi ed i blocchi "call" per i report. I blocchi run e call hanno delle frecce alla loro destra, che permettono di inserire gli slot per gli input. Come fa BYOB a determinare dove utilizzare questi input? Se la procedura chiamata ha slot di ingresso liberi, BYOB "*fa la cosa giusta*":

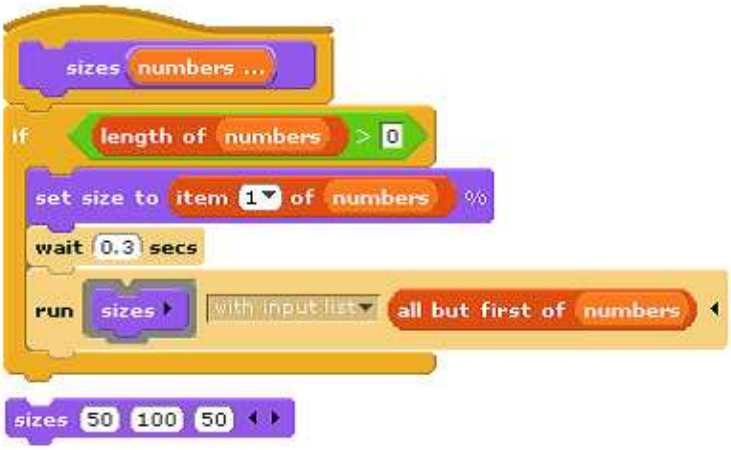
- se il numero di slots vuoti è uguale al numero di ingressi, BYOB riempie gli slot vuoti da sinistra verso destra;

- se c'e' solo un ingresso, BYOB valorizza tutti gli slots di input con quell'ingresso;

- altrimenti BYOB non valorizza nessuno slot, poiché le intenzioni del programmatore non sono chiare.

Se l'utente desidera modificare queste regole, la soluzione è utilizzare nomi espliciti.

La scelta di "multiple inputs" dalla finestra di creazione di input type, viene utilizzata solo quando c'è la necessità di effettuare una chiamata ricorsiva ad un blocco che ammette un numero variabile di ingressi. Ad esempio:

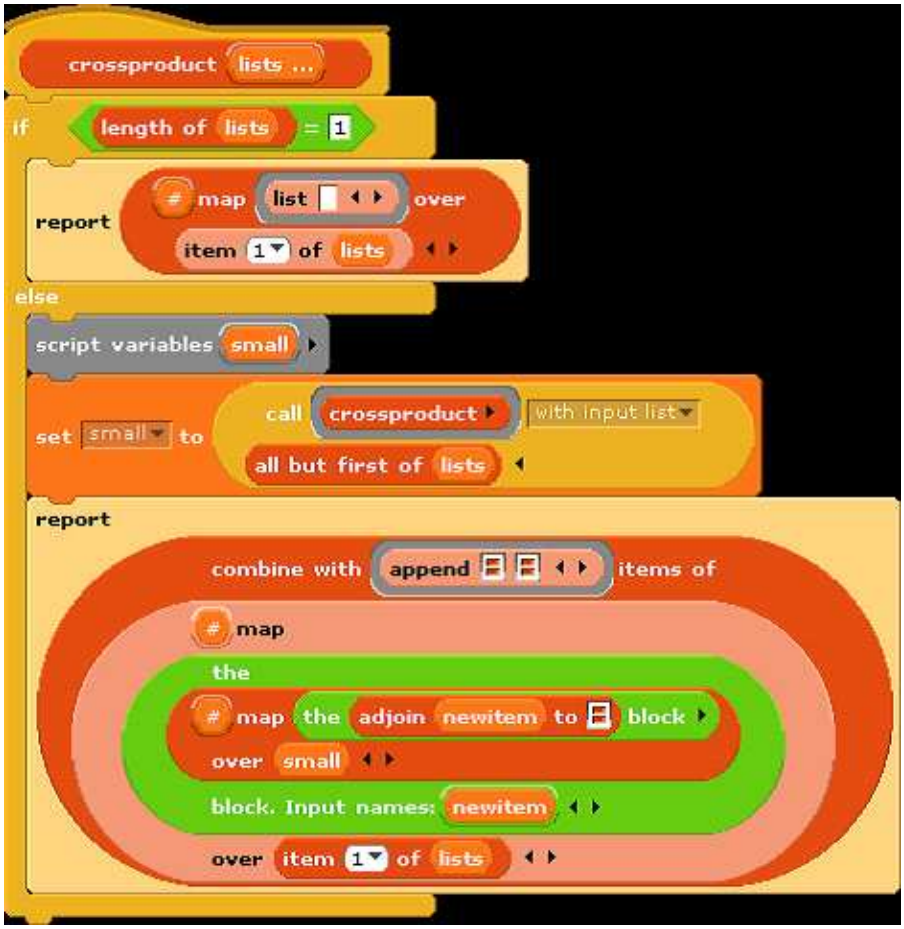


L'utente di questo blocco chiama lo script con una quantità qualsiasi di numeri individuali come input. Ma dentro la definizione del blocco, tutti questi numeri formano una lista, che ha un nome unico: "*sizes*". Ma sizes non prende in input una lista, ma prende dei valori numerici. L'opzione "with input list" sostituisce i valori dello slot di tipo "any-type" con una slot del tipo "list-type".

Gli elementi della lista sono presi singolarmente come input per lo script. Dal momento che "numeri" è una lista di numeri, ogni elemento individuale è a sua volta un numero, quindi ottengo esattamente ciò che volevo.

- **Procedure di dati**

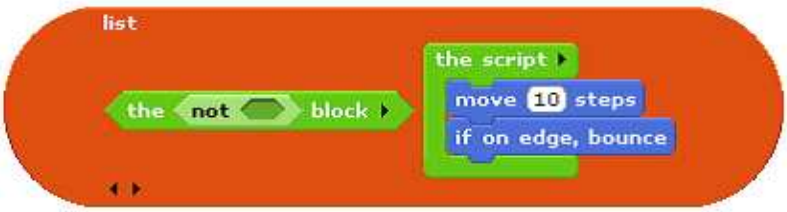
A volte, l'incapsulamento automatico di procedure non è adeguato. Ciò può accadere per due motivi principali: o si vuole avere più controllo sui fattori di input delle procedure, oppure se si vuole utilizzare una procedura come input per uno slot di ingresso che non è dichiarato di tipo procedura.
Primo esempio:

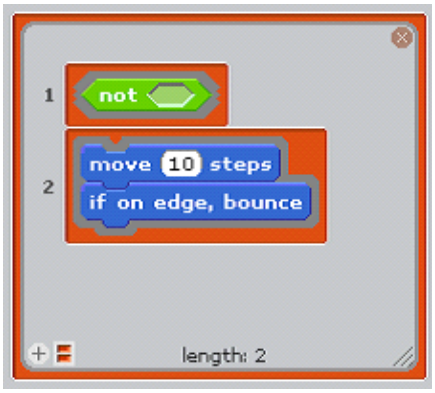


Questa è la definizione di un blocco che prende un numero illimitato di liste, e riporta la lista di tutte le possibili combinazioni di un elemento per ogni lista. Ci sono due chiamate annidate a "*map*", una funzione di ordine superiore che applica una funzione di input ad ogni elemento di una input list. Nel blocco interno, la funzione "mappata" è "*adjoin*" e che prende in ingresso due input. Il secondo, il blocco vuoto "*list-type*" ricaverà il suo valore in ogni chiamata da un elemento della lista di ingresso della mappa interna. Non c'è modo per la mappa esterna di comunicare i valori agli slot vuoti del blocco ad join. Dobbiamo dare un nome esplicito, "*newitem*" al valore che la mappa esterna passa a quella interna, quindi copiare questa variabile nel blocco "*adjoin*".
Proviamo a vedere un esempio più semplice:

Qui vogliamo soltanto metter uno degli ingressi in due slot differenti. Se lasciamo tutti e tre gli slot vuoti, BYOB non ne riempirebbe nessuno, perché il numero di ingressi previsto (2) non coincide con il numero di slot vuoti.

Ecco un altro esempio in cui un procedimento deve essere esplicitamente contrassegnato come dati:



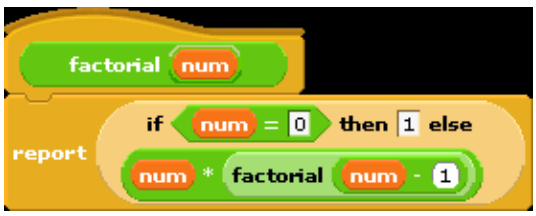


Qui stiamo costruendo una lista di procedure, ma il blocco di lista accetta ingressi di qualsiasi tipo, per cui i suoi slot di ingresso non sono segnati come procedura. Dobbiamo segnalare esplicitamente che vogliamo il blocco stesso come input.
In queste situazioni utilizziamo il "blocco di blocco" o il "blocco di script". Entrambi segnano il loro input come dati. Entrambi hanno la possibilità di dare nomi espliciti per gli slot di input.
Oltre all'esempio di cui sopra, altri blocchi in cui si potrebbe voler inserire delle procedure, siano "*set*" (il valore di una variabile), "*say*" e "*think*" (per visualizzare la procedura all'utente) e "*report*" (per riportare procedure).

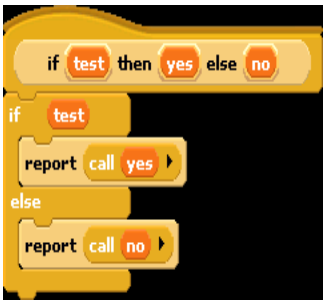
- **- Form speciali**

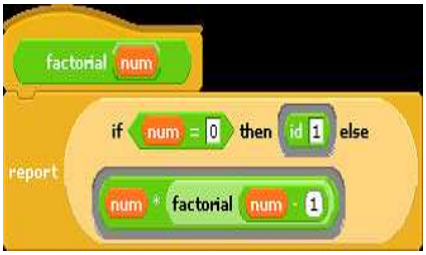
Scratch ha un blocco "*if else*" che ha due slot di tipo command a forma di C, e sceglie uno o l'altro a seconda di un test booleano.

Questo blocco funziona per semplici programmi, ma se lo utilizziamo per una chiamata ricorsiva, va in errore.



Il problema è che in Scratch, quando un blocco viene chiamato, tutti i suoi ingressi sono valutati prima che lo script del blocco venga lanciato. Lo stesso blocco, inoltre, conosce solo i valori dei suoi ingressi, non le espressioni che sono state utilizzate per calcolarli. Questo significa che il fattoriale richiama se stesso ricorsivamente, e causa un loop infinito. Dobbiamo costruire un blocco "*if else*" in grado di selezionare una delle due alternative che devono essere eseguite.
Abbiamo un meccanismo per fare ciò: dichiariamo gli input "then" ed "else", di tipo reporter in questo modo:





In questo modo, la procedura funziona, senza loop infinito, ma abbiamo pagato un prezzo pesante: questo "*reporter-if*" non è più intuitivo come l'if iniziale del comando di Scratch. Bisogna conoscere procedure come dati e funzioni costanti. La funzione id implementa l'identità, ne abbiamo bisogno perché il blocco prende solo "reporter" in ingresso, non numeri. Quello che vorremmo e' un blocco reporter-if che si comporti come l'if normale, ritardando la valutazione dell'input, ma che assomigli alla versione iniziale che è semplice da capire, ma non funziona. Tali blocchi sono effettivamente possibili. Un blocco che prende una semplice espressione come input, ma che ritarda la valutazione di tale input, e se necessario effettua una conversione da dati costanti in funzioni costanti, è chiamato "*form speciale*". Per trasformare il nostro blocco if in un "form speciale", modifichiamo il blocco iniziale, trasformando gli input "*yes*" e "*no*" di tipo Any, invece di Reporter. Lo script del blocco per il resto è uguale allo script della versione seconda.

Gli special forms non sono una nuova invenzione in BYOB. Molti dei blocchi di condizione e ciclici di Scratch sono special forms. Lo slot di input esagonale nel blocco if, è un semplice valore booleano, visto che può essere controllata una volta, prima che il blocco if prenda la sua decisione sul da farsi. Ma i blocchi "*forever if*", "*repeat until*", "*wait until*", non possono essere semplici booleani, devono essere del tipo boolean (unevaluated). Il valore pedagogico dei special forms è provato dal fatto che nessun utente di Scratch si rende conto che c'è qualcosa di strano nel modo in cui vengono valutati gli ingressi esagonali dei blocchi di controllo.

- **- Programmazione Orientata agli oggetti con Sprite**

La programmazione orientata agli oggetti è uno stile basato attorno all' astrazione: una collezione di dati e metodi che interagiscono con essa mediante l'invio di un messaggio, a cui risponde con la realizzazione di un metodo, che può oppure no, segnalare un valore nuovo al richiedente. I motivi per l'utilizzo di OOP variano: alcuni sottolineano l'aspetto del nascondimento dati, altri enfatizzano l'aspetto simulativo. Quest'ultimo è quello che interessa anche agli utenti di BYOB.

Tecnicamente, la programmazione ad oggetti si basa su tre concetti principali:

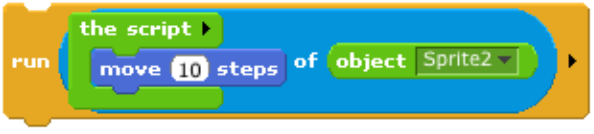
- lo scambio di messaggi: c'è una notazione con cui qualsiasi oggetto può inviare un messaggio ad un altro oggetto;
- stato locale: ogni oggetto può ricordare il passato importante, memorizza la sua storia passata attraverso la conservazione di un set di valori;
- ereditarietà: sarebbe impraticabile se ogni singolo oggetto avesse metodi, molti dei quali superflui, per tutti i messaggi che puo' accettare. Invece abbiamo bisogno di poter dire che questo nuovo oggetto è come quello vecchio, tranne che per pochi differenze.

- **First Class Sprites**

Scratch viene fornito con oggetti naturali: gli sprite. Ogni sprite può possedere le variabili locali, ogni sprite ha il suo script (metodi). Ci sono tre caratteristiche in cui gli sprite di Scratch sono meno versatili degli oggetti di un linguaggio OOP. La prima riguarda il passaggio di messaggi, primitivo sotto tre aspetti: i messaggi possono essere solo trasmessi, non indirizzabili ad un singolo sprite; i messaggi non possono prendere gli input, ed i metodi non possono restituire valori al loro chiamante. La seconda riguarda il meccanismo dell'ereditarietà per gli sprite in Scratch. La terza caratteristica è il fatto che negli OOP, gli oggetti paradigma della programmazione sono dati (valore di una variabile, elemento di una lista, ecc…). In BYOB, gli sprite sono i dati di prima classe. Essi possono essere creati e cancellati da uno script, memorizzati in una variabile o una lista, spedire messaggi individualmente ed ereditare proprietà da un altro sprite. Il blocco principale dei programmi di sprite è l'*oggetto* blocco reporter. Esso ha un menù dropdown che se cliccato elenca tutti gli sprite presenti.

- **Mandare messaggi agli sprite**

I messaggi che uno sprite accetta sono i blocchi nelle sue tavolozze, compresi i blocchi "all-sprites" e "this-sprite-only". Scratch prevede per uno sprite di poter accedere a certi attributi di un altro sprite con il blocco "*the <var> of <sprite>*". BYOB estende questo blocco per poter accedere ad ogni proprietà dello sprite. BYOB include anche un blocco "*launch*", uguale identico al blocco "*run*", eccetto per il fatto che richiama il metodo come uno script separato, quindi lo script chiamante può fare altro.



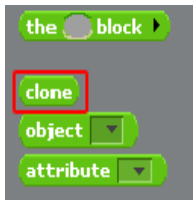
- **Stato locale negli sprite**

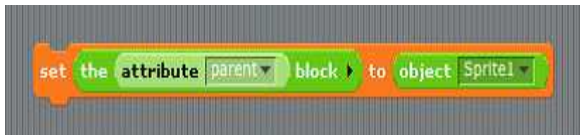
La memoria di uno sprite assume due forme principali. Ha variabili create esplicitamente dall'utente, ma ha anche gli attributi, le caratteristiche che uno sprite ha automaticamente. Ogni variabile può essere esaminata utilizzando la sua forma ovale arancione e modificata

tramite il blocco "*set*". Una direzione di sprite può essere esaminata utilizzando il blocco "*direction*" e modificata con il blocco "*point in direction <dir>*". Allo stesso modo si possono impostare tutte le variabili, tramite i metodi appropriati. In generale, il blocco che esamina una variabile o un attributo, e' chiamato blocco "getter", mentre un blocco che valorizza una variabile o un attributo è chiamato blocco "setter". In BYOB, una variabile o un attributo, sono rappresentati dal loro blocco "getter".

- **Prototipi: Padri e Figli**

La maggior parte dei linguaggi OOP correnti, utilizza un approccio classe/istanza per creare oggetti. Una classe è un particolare tipo di oggetto, ed un'istanza è un oggetto reale di quel tipo. La classe specifica in genere i metodi condivisi da tutti, le istanze contengono dati personalizzati. BYOB utilizza un approccio diverso, chiamato "*prototipi*", in cui non vi è una distinzione tra classi e istanze. Ad esempio si crea un singolo sprite "dog", con i suoi metodi ed i suoi attributi. Si usa questo sprite come prototipo da cui clonare altri sprite. Se poi si vuole modificare un metodo nello sprite "dog" iniziale, la modifica verrà estesa a tutti gli sprite cloni. Ci sono tre metodi principali per creare cloni di uno sprite, cliccare con il tasto destro del mouse e selezionare clone, utilizzare il blocco "*clone*", oppure valorizzare l'attributo "parent".





Nei progetti che creano cloni, il numero di sprite puo' diventare molto grande, perciò nella finestra sprite vengono visualizzati solo gli sprite che non hanno padre.

- **Ereditarietà per Delega**

Un clone eredita le proprietà del padre. Le proprietà includono scripts, blocchi personalizzati, variabili, liste, attributi di sistema, ecc… Ogni singola proprietà può essere condivisa tra genitore e figlio, oppure non condivisa, quindi nel figlio c'e' una proprietà separata. Se il valore di una proprietà condivisa viene cambiato nel padre, il figlio vede anch'esso il nuovo valore per quella proprietà. Se una proprietà condivisa viene cambiata nel figlio, invece, non va propagata al padre. Quindi per rompere il collegamento di condivisione padre-figlio, basta cambiare una proprietà all'interno del figlio e questa non erediterà piu' le modifiche effettuate nel padre. Per cambiare una proprietà non condivisa, il figlio cancella la sua versione privata, utilizzando il comando di cancellazione di un blocco.

Quando uno sprite riceve un messaggio per il quale non ha un blocco corrispondente, il messaggio è delegato al padre. Quando uno sprite ha il blocco corrispondente, il messaggio non viene delegato. Se lo script che implementa un messaggio delegato si riferisce ad "*object(myself)*", si riferisce al figlio a cui il messaggio è stato originariamente inviato e non al padre a cui il messaggio è stato delegato.

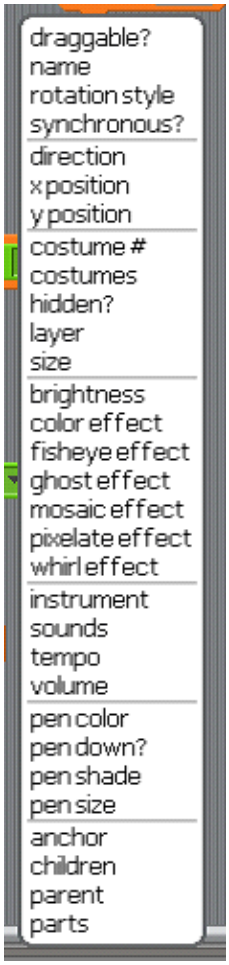
- **Sprite annidati**

A volte è opportuno fare una sorta di "super-sprite", composto di pezzi che si muovono insieme, ma anche separatamente articolato. L'esempio classico è il corpo di una persona, costituito da tronco, arti e testa. BYOB permette ad uno sprite di essere disegnato come forma combinata di altri sprite. Questa è un'ulteriore forma di connessione tra sprite fra gli sprite, divisi da padri e figli. Lo sprite è detto "anchor".

- **Lista di attributi**

L'immagine che segue rappresenta una lista di attributi di uno sprite. Quattro di essi non sono attributi, bensì liste o cose legate agli attributi:

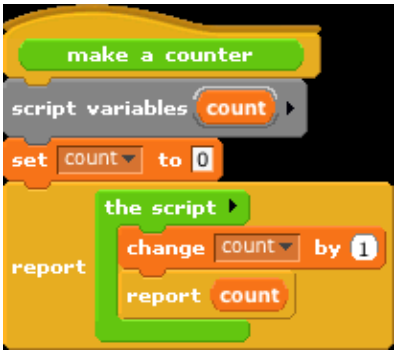
- costumes: lista di nomi dei costumi dello sprite;
- sounds: un elenco dei nomi dei suoni dello sprite;
- children: lista di sprites figli dello sprite;
- parts: lista di sprites, il cui attributo "ancora" è lo sprite corrente.



- **Costruire oggetti esplicitamente**

L'idea di programmazione orientata agli oggetti viene spesso insegnata in un modo che fa sembrare come se fosse necessaria. In realtà, qualsiasi linguaggio con le procedure di prima classe, consente agli oggetti di essere implementati in modo esplicito. L'idea centrale di questa implementazione è che un oggetto è rappresentato come una procedura di spedizione che prende un messaggio in input e riporta il metodo corrispondente.

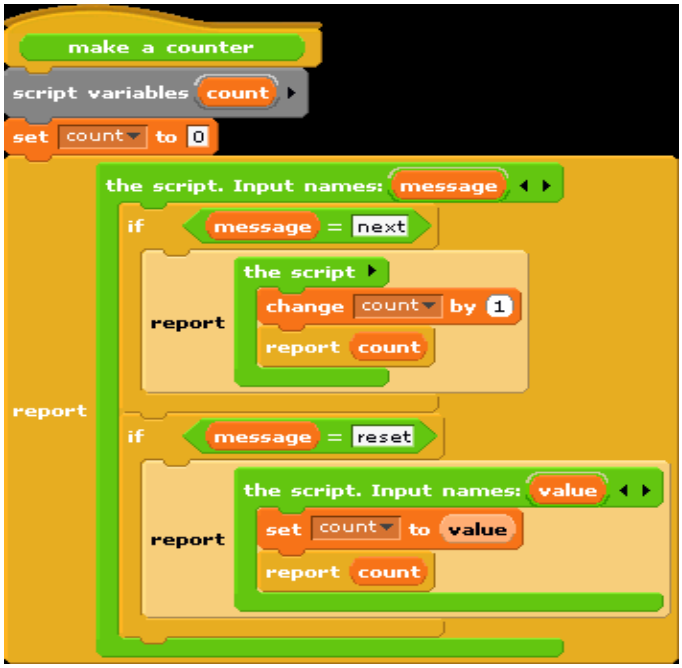
- **Stato locale con Script Variables**



Questo script implementa una classe di oggetti, un tipo di oggetto, ossia la classe contatore. Quando il blocco "*make a counter*" viene chiamato, esso segnala una procedura creata dal blocco di script, all'interno del suo body. Tale procedura consente di implementare uno specifico oggetto "counter", istanza della classe "counter".
In questo approccio all' OOP, stiamo rappresentando sia le classi che le istanze come procedure. Il blocco "*make a counter*" rappresenta la classe, mentre ogni istanza è rappresentata dallo script generato ogni volta che viene chiamata "*make a counter*".

- **Messaggi e procedure di spedizione**

Nella classe semplificata di cui sopra, vi è solo un metodo, e quindi non ci sono messaggi, basta chiamare l'istanza per farle svolgere il suo metodo. Ecco un esempio che utilizza il passaggio di messaggi:

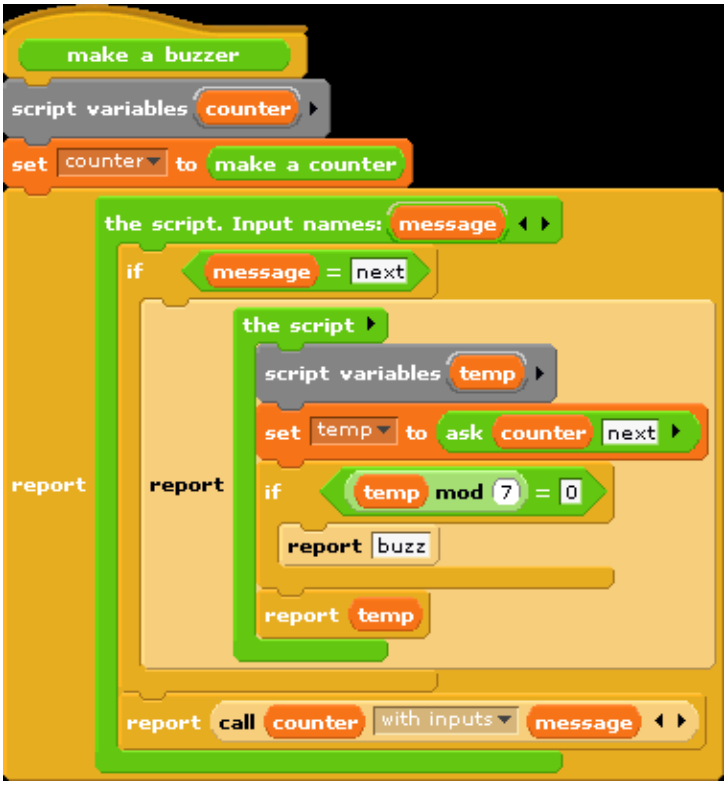


Anche qui il blocco "make a counter" rappresenta la classe e lo script crea una variabile locale "*count*" ogni volta che viene invocato. Il blocco di script esterno rappresenta un'istanza. E' una procedura di invio: che prende un messaggio in input e riporta un metodo. I due blocchi di script più piccoli sono i metodi: "*next*" e "*reset*". Per poter definire un'istanza e chiamare il metodo in un unico blocco, possiamo utlizzare il blocco "*ask*" in questo modo:

Questo blocco richiede due ingressi: object e message, e dispone anche di altri input addizionali come ad esempio args. Ha due blocchi "*call*" annidati: quello più interno chiama l'oggetto, l'altro la procedura di invio.

- **Ereditarietà tramite delega**

Ora i nostri oggetti hanno variabili di stato locale e scambio di messaggi. E per quanto riguarda l'ereditarietà? Siamo in grado di fornirgliela utilizzando la tecnica della delega. Ogni istanza della classe figlio contiene un'istanza della classe padre che semplicemente comunica i messaggi per cui il figlio è delegato.



Questo script implementa la classe "*buzzer*" che è figlia di "*counter*". Invece di avere un count (numero) come variabile locale, ogni buzzer ha un oggetto counter come variabile locale.

- **Altri componenti**

BYOB introduce anche altri blocchi che non sono tra quelli di prima classe o procedure:

Uno per i comandi ed uno per i reporter possono essere inseriti in uno script per farne il debug passo passo, con la possibilità di controllare le variabili.

Questi blocchi contengono il valore costante “true” o “false”.

Questo blocco è utilizzato per verificare il tipo di ogni valore (boolean, lista, comando, reporter, predicato).

Utili nei casi in cui si desideri utilizzare un carattere di indice in un elenco di valori.

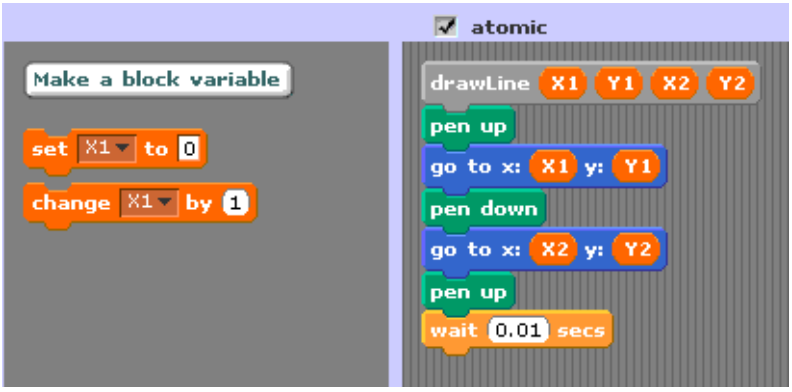
Questo blocco prende una lista e segnala il suo valore come una stringa di testo.

La copia del blocco fa una nuova lista contenente gli stessi elementi.

Crea una o più variabili locali rispetto allo script, che possono essere utilizzate come archivio temporaneo.

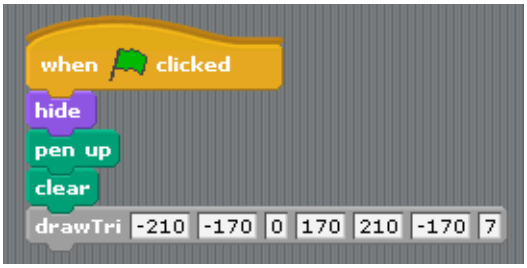
- **ESEMPIO APPLICATIVO**

Mostriamo un esempio di applicazione creata in BYOB. In questo caso cerchiamo di creare una ricorsione per poter disegnare il famoso triangolo di Sierpinski.
Andiamo sulla sezione “variabili” e selezioniamo il comando “make a block”. Innanzitutto creiamo un blocco che disegni una riga. Utilizziamo le coordinate, come in un grafic x ed y. Partiamo da un punto di partenza con coordinate x1,y1 ed un punto di arrivo x2,y2. Per fare cio’, componiamo i blocchi come in figura:

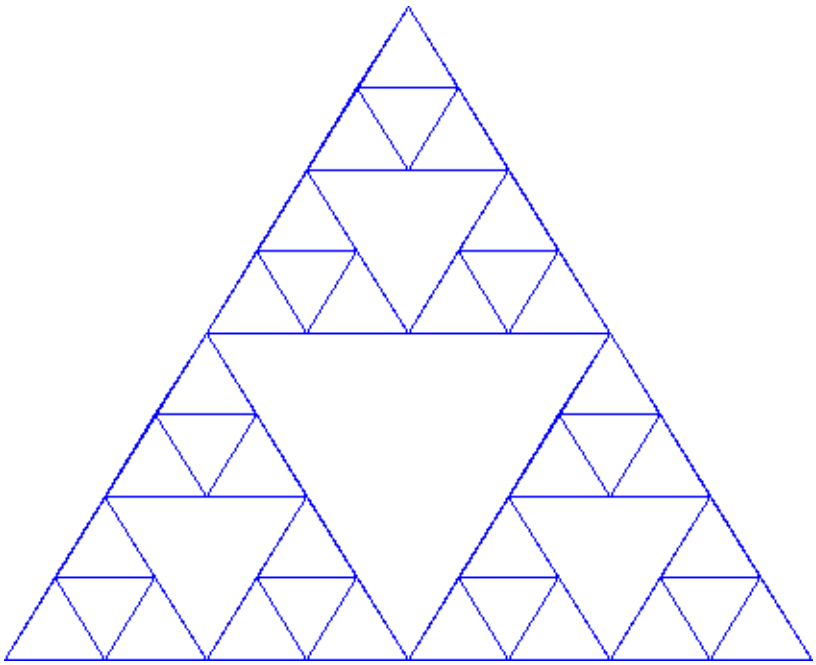


Il blocco drawLine è il blocco “*cappello*”, quindi viene preso lo strumento penna, ci si posiziona sul punto di inizio x1,y1, si appoggia la penna al “foglio” e ci si sposta verso il punto di fine con coordinate x2,y2. Quindi si riprende la penna. Con il blocco “wait”, si riesce a vedere l’operazione mentre viene eseguita, altrimenti si vedrebbe solamente il risultato finale.
Costruiamo ora il metodo ricorsivo “*drawTri*”, per disegnare un triangolo. E’ molto simile al metodo “*drawLine*”, ha semplicemente bisogno di un altro punto con coordinate x3,y3 e di un contatore per il controllo della ricorsione, in questo caso “*c*”.

Come vediamo dalla figura, il metodo inizia con il disegnare il primo triangolo, utilizzando il metodo "*drawLine*". Quindi si calcolano i punti medi dei punti iniziali. Dopo aver effettuato il controllo sulla variabile contatore, si effettuano tre chiamate ricorsive di "*drawTri*" aggiornando di volta in volta il contatore. Ora effettuiamo la chiamata al nostro metodo "*drawTri*" in questo modo:



In questo caso, prima di chiamare il metodo, ci preoccupiamo anche di selezionare la penna e cancellare lo schermo. Nel caso in cui il contatore C sia uguale a 3, otteniamo la seguente figura:



- **CONCLUSIONI**

In Scratch, la mancanza di ricorsione ed altri elementi basilari per la programmazione, viene visto come punto debole. Secondo il mio parere, invece, questo è proprio il suo punto di forza. Infatti, Scratch non riesce ad accontentare uno sviluppatore esperto, ma questo non è il suo scopo. E' stato progettato per attirare un pubblico inesperto per quanto riguarda la programmazione, o l'informatica in generale. Quando utilizziamo Scratch, ci troviamo davanti ad una interfaccia estremamente semplice ed intuitiva. In questo tipo di ambiente possiamo scoprire tutto da soli, senza l'aiuto di esperti. Infatti non ci sono menù nascosti o altre opzioni, ma vediamo immediatamente tutto ciò che c'è da vedere e niente si comporta in modo diverso da come ci si aspetta. In pratica Scratch ci offre un contesto di programmazione concreto. Ci permette di concentrarci sulle idee che vogliamo esprimere con precisione. Inoltre introduce i primi rudimenti del "*problem solving*": insegna in maniera efficace a risolvere un problema suddividendolo in problemi più piccoli, di più facile soluzione. L'essenza di ciò è il fatto che per arrivare ad un risultato finale, dobbiamo utilizzare dei blocchi, *combinandoli* tra di loro, quindi unendo soluzioni di problemi più semplici. Nell'esempio del capitolo precedente, infatti, prima impariamo a disegnare una riga, poi, a disegnarne tre che formano un triangolo.
Un altro punto fondamentale a favore di Scratch lo segna il suo sito, su cui si possono condividere progetti, idee, problemi, soluzioni, ecc. Ciò migliora ulteriormente il processo di apprendimento, dando la possibilità agli utenti di condividere le loro skills e crescere insieme.
Man mano che viene utilizzato, ci si rende conto dei suoi limiti. Infatti, per i progetti più complessi, cominciamo a trovarci di fronte a script di dimensioni notevoli. Ma è un processo graduale. Ce ne rendiamo conto quando Scratch ha fatto per noi ciò che era il suo scopo, ossia introdurci nel mondo della programmazione.
Nel momento in cui cominciamo ad avere degli script di dimensioni notevoli, significa che possiamo passare a BYOB.
Quest'ultimo ci dà la possibilità di creare dei blocchi personalizzati e di riutilizzarli a nostro piacimento in una o più applicazioni. La sua modularità ci permette di scrivere "*codice*" molto più leggibile e comprensivo. Questa proprietà e la possibilità di avere diversi tipi di dati e gestire la ricorsione, lo rendono un ambiente di sviluppo adatto ad utenti che hanno già familiarizzato con i concetti base della programmazione.
Per esperienza personale, affermo che affrontare la schermata di BYOB, senza avere prima utilizzato Scratch risulta più complicato cominciare a "*muoversi*". Mettendo alla prova dei ragazzi delle scuole medie, sottoponendo loro le due schermate di Scratch e BYOB, risulta evidente che il primo riscuote molto più successo. Sia dal punto di vista della comprensione e di conseguenza anche della soddisfazione. Non ce n'e' stato il tempo, ma probabilmente facendo lo stesso esperimento con ragazzi delle scuole superiori e delle università, si otterrebbe il risultato opposto.
In comune Scratch e BYOB hanno l'interfaccia "*drag&drop*" che consente di vedere immediatamente il risultato del proprio lavoro, quindi ad individuare i problemi ed a risolverli. Riassumendo

Scratch:
- e' diretto ad un pubblico inesperto e a ragazzi di un'età che va fino ai 16-17 anni circa
- ha una schermata molto semplice ed intuitiva

- consente di condividere i propri lavori con tutti gli appassionati
- non consente la gestione di tipi di dati complessi e ricorsione

BYOB
- è diretto ad un pubblico con un minimo di esperienza ed un'età che va dai 18-19 anni in su
- ha una schermata semplice ed intuitiva
- granularità del codice, semplificando di molto la lettura e la modifica dei programmi
- consente la gestione di tipi di dati complessi e della ricorsione

Dovendo scegliere, farlo in base a queste caratteristiche, oltre a quelle riguardanti il progetto che si vuole realizzare.

BIBLIOGRAFIA

1. Scratch website: http://scratch.mit.edu
2. Monroy-Hernández, A. and Resnick, M. (2008). Empowering kids to create and share programmable media. ACM interactions 15, 2 (March 2008), 50-53
3. Monroy-Hernández, A. and Hill, B. M. (2010) Cooperation and attribution in an online community of young creators. Poster in ACM Conference on Computer Supported Cooperative Work (CSCW '10)
4. http://www.quantcast.com/scratch.mit.edu
5. http://kids.sapo.pt/scratch
6. http://scratched.media.mit.edu
7. Build Your Own Blocks (BYOB) (Scratch Modification)
8. http://byob.berkeley.edu/BYOBManual.pdf
9. http://www.minecraftforum.net/viewtopic.php?f=1010&t=16482&start=0
10. http://scratch.mit.edu/forums/viewtopic.php?id=10887
11. http://www.programmazione.it/index.php?entity=eitem&idItem=44982
12. http://www.nutt.net/2010/03/06/teaching-recursion-with-scratch-byob/
13. http://www.magic-league.com/phpBB/about2101.html