Esercitazioni Scratch

BETA VERSION 0.2 (DRAFT 21/05/2014)

Capitolo 1

Introduzione

Scratch è un nuovo linguaggio di programmazione visuale che ti permette di creare storie interattive, applicazioni educative, animazioni, giochi, etc.

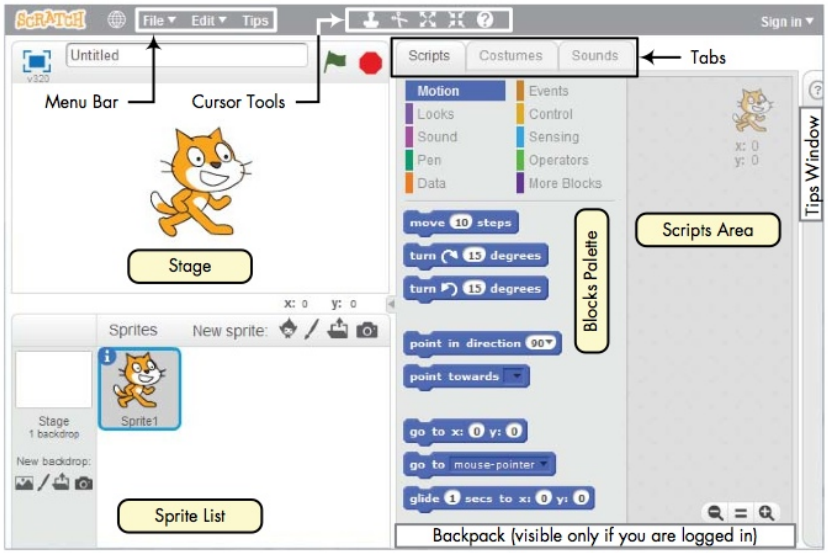
Scratch offre un ambiente di programmazione user-friendly che nella versione 2.0 è sia completamente on-line che in versione desktop (cioè completamente off-line).

L'ambiente Scratch

1. Un linguaggio di programmazione visuale permette di scrivere programmi collegando tra di loro dei blocchi grafici
2. La metafora utilizzata da *Scratch* è quella della scrittura di un copione (*script*) per uno o più attori (*sprite*) che agiscono in una o più scene (*scene*) di una rappresentazione teatrale
3. Quindi, uno *script* fornisce le istruzioni (organizzate in forma di programma-algoritmo) per uno *sprite*
4. Un programma *Scratch* è costituito da più *script* che decidono il comportamento dei singoli *sprite* che, eventualmente, interagiscono tra loro (e.g. utilizzando messaggi o eventi)

L'ambiente di Scratch è costituito da diverse sezioni o aree di lavoro:

- Stage,
- Sprite List
- Scripts tab (con all'interno Blocks tab e Scripts Area)



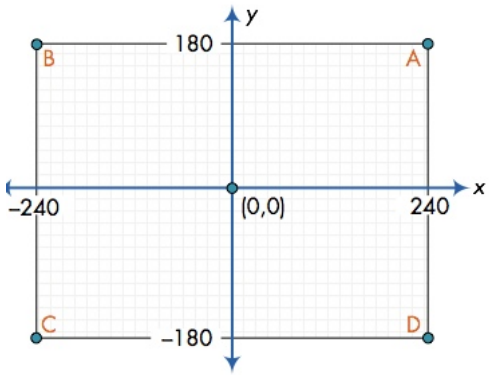
Inoltre, la barra in alto è suddivisa in due parti:

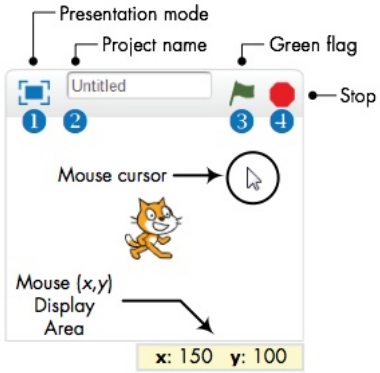
- *Menu Bar* (che permette di generare un nuovo progetto vuoto, caricare un progetto salvato, salvare il progetto corrente, etc.)
- *Cursor Tools* (che offre la possibilità di eseguire operazioni quali, ad esempio, elimina, duplica, etc.)

Lo *Stage* è lo schermo nel quale il risultato del nostro programma (col quale l'utente finale interagirà) sarà visualizzato.

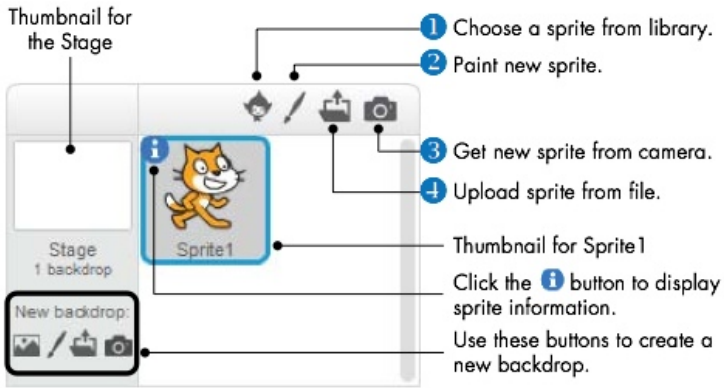
Ai diversi punti dello *Stage* si può accedere grazie alle coordinate cartesiane.

Le dimensioni dello *Stage* sono di 480 passi in larghezza e di 260 in altezza (ricordiamoci che si misura in passi dato che l'obiettivo è animare un attore detto Sprite).

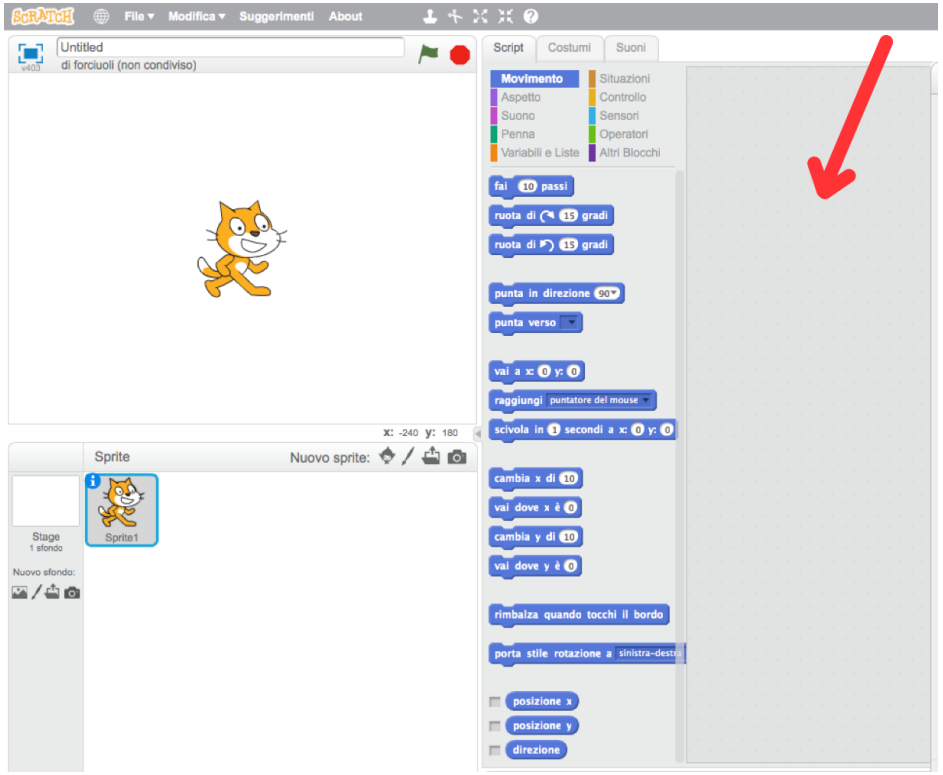




Nella sezione *Sprite List* è possibile gestire uno o più *sprite* inclusi nel nostro programma.



Cliccando sullo *sprite* è possibile visualizzare lo *script* corrispondente nello *Scripts tab* (che, in un nuovo progetto, risulterà vuoto). Nella figura seguente, il pannello per scrivere gli script è indicato on una freccia rossa.



Per scrivere lo *script* è necessario trascinare i blocchi nel pannello indicato dalla freccia rossa.

La *Blocks tab* prevede 10 categorie di blocchi dette "palette":

- *Motion* (movimento)
- *Looks* (aspetto)

- *Sound* (suono)
- *Pen* (penna)
- *Data* (variabili e liste)
- *Events* (situazioni)
- *Control* (controllo)
- *Sensing* (sensori)
- *Operators* (operatori)
- *More Blocks* (altri blocchi)

I blocchi **movimento** includono tutte le istruzioni che è possibile impartire ad uno *sprite* per consentirgli di muoversi sulla scena.

I blocchi **aspetto** includono tutte le istruzioni per consentire ad uno *sprite* di parlare e pensare (con il meccanismo dei fumetti, quindi visualizzando del testo), di apparire, scomparire e cambiare costume.

I blocchi **suono** forniscono le istruzioni che è possibile inserire in un programma per riprodurre dei suoni (e.g. voce, musica, effetti, etc.).

I blocchi **penna** includono tutte le istruzioni per disegnare sullo schermo (scena).

I blocchi **variabili e liste** includono tutte le istruzioni per creare nuove variabili/liste e per utilizzarle all'interno dei nostri programmi. Le variabili sono dei contenitori residenti in memoria (del computer) che servono a conservare e manipolare i dati (e.g. numeri, caratteri, parole, frasi, etc.). Le liste permetto di gestire più variabili correlate tre loro.

I blocchi **situazioni** forniscono le istruzioni per avviare uno script, per avviare tutti gli script, per inviare un messaggio (ad un altro script) e per gestire alcuni eventi quali, ad esempio, la rumorosità.

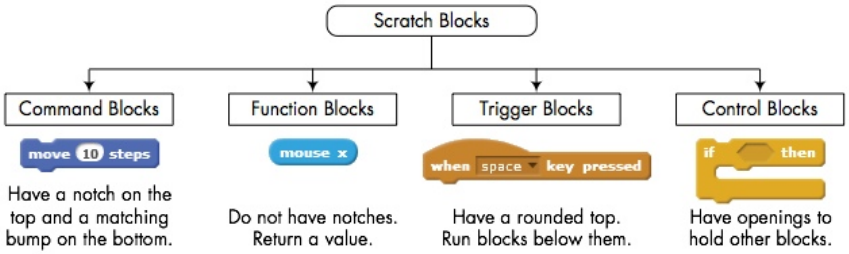
I blocchi **controllo** includono le istruzioni per permettere ad uno *sprite* di ripetere più volte lo stesso comportamento, di clonarsi, di fermarsi (fermare l'esecuzione dello *script*) di attendere degli eventi specifici, etc.

I blocchi **sensori** includono le istruzioni per percepire gli eventi esterni (clic del mouse, pressione su tastiera, movimento nella web cam) e interni relativi all'esecuzione di un pro-

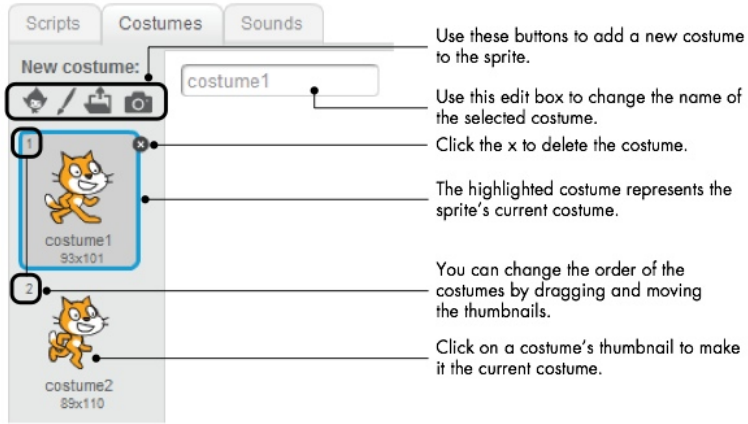
gramma e consentire a uno *sprite* di reagire in maniera specifica a un singolo evento.

I blocchi **operatori** includono le istruzioni per eseguire operazioni aritmetiche e logiche, di valutare condizioni, di manipolare testi, etc.

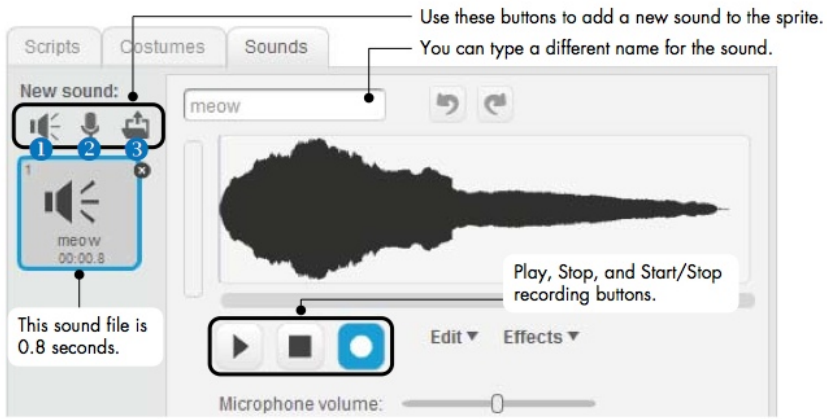
Gli **altri blocchi** consentono al programmatore di definire nuovi blocchi con nuove istruzioni.



Ritornando alla descrizione generale dell'ambiente *Scratch*, il *Costumes Tab* permette di associare a ogni *sprite* un aspetto (costume) differente al fine di gestire nuovi stati in cui si trova lo *sprite* così da poter creare, ad esempio, piccole animazioni.



Inoltre, nel *Sounds Tab* è possibile organizzare, registrare e riprodurre i suoni da associare al comportamento degli *sprite*.



I blocchi si incastrano uno nell'altro e uno sull'altro.

Per

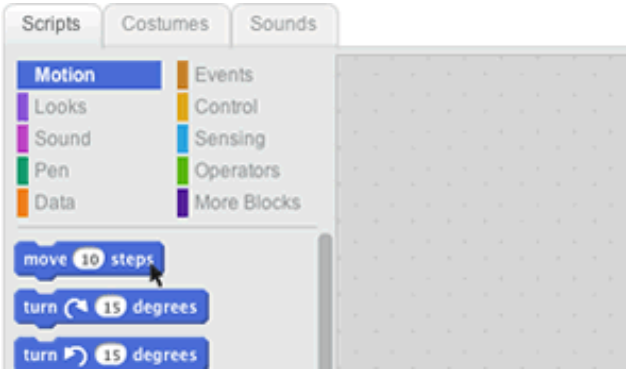
SEZIONE 2

Primi passi con Scratch

1. Inizia a muoverti
2. Aggiungi un suono
3. Inizia a ballare
4. Ancora e ancora
5. Dire qualcosa
6. Bandiera verde
7. Cambiare colore
8. Pressione di un tasto
9. Aggiungi uno sfondo
10. Aggiungere uno *sprite*
11. Esplora

★ Inizia a muoverti

Trascina il blocco FAI PASSI nell'area degli *script*.



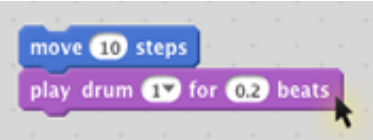
Quindi clicca sul blocco per far muovere il gatto:

★ Aggiungi un suono

Trascina il blocco SUONA TAMBURO (*play drums*) nell'area degli *script* e aggancialo al blocco del movimento.

IMPORTANTE: E' interessante notare che i blocchi vengono sistemati uno di seguito all'altro per riprodurre la sequenza di un algoritmo così come si è appreso sul funzionamento della macchina di Von Neumann.

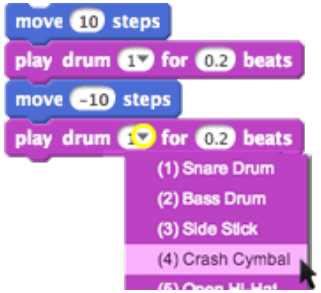
Clicca e ascolta.



Inizia a ballare

Aggiungi un altro blocco FAI PASSI. Clicca all'interno del blocco e inserisci un segno "meno".

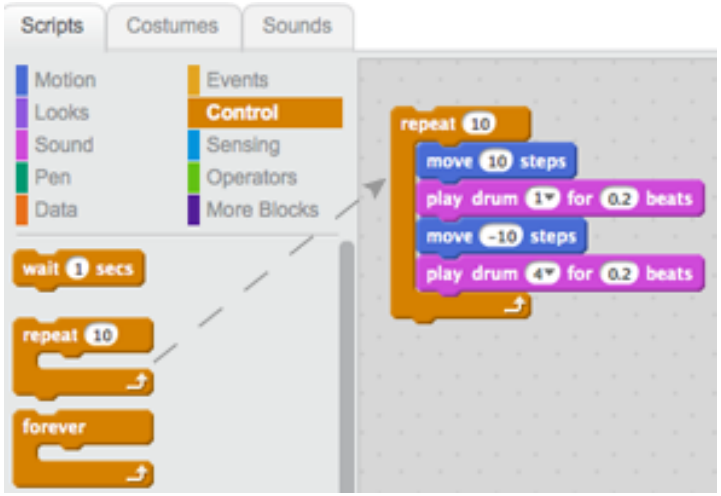
Aggiungi ora un altro blocco SUONA TAMBURO, poi scegli con il menu il tipo di percussione che preferisci.



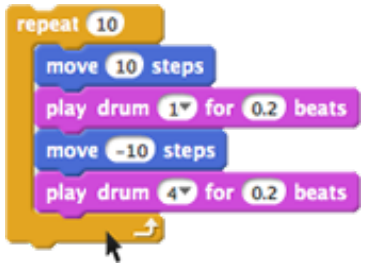
Clicca su un blocco qualunque per eseguire tutti i blocchi della pila:

Ancora e ancora

Trascina un blocco RIPETI e posizionalo in cima agli altri blocchi (Devi far avvolgere gli altri blocchi dalla "bocca" del blocco RIPETI).

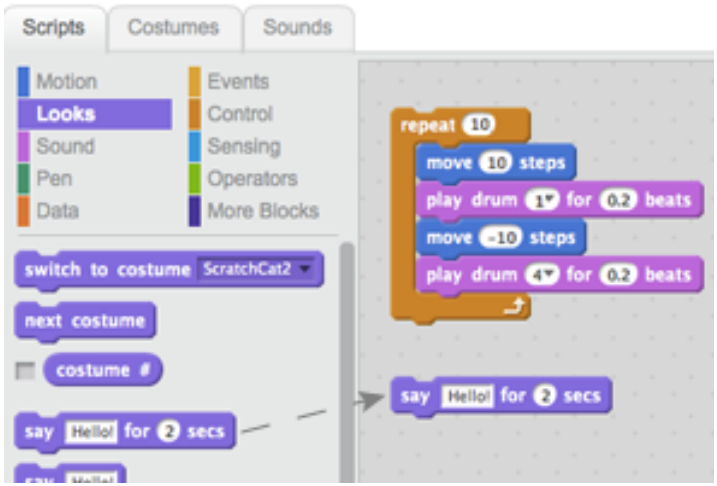


Clicca la pila di blocchi per eseguirla:

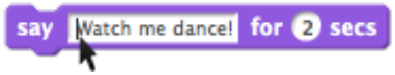


Dire qualcosa

Clicca ASPETTO e trascina un blocco DIRE.



Scrivi all'interno del blocco per modificare le parole:



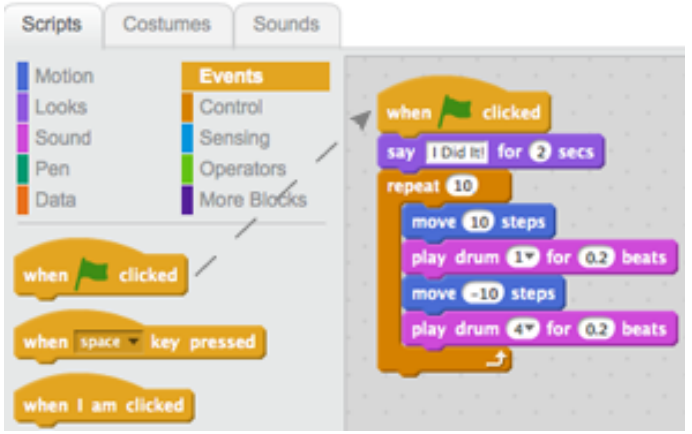
Quindi attaccalo in cima agli altri blocchi già posizionati.

Clicca sulla sequenza di blocchi per eseguirla.

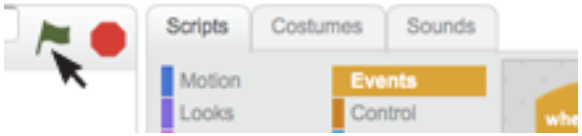
★ Bandiera verde

Trascina un blocco 

e aggancialo in cima alla pila già presente.



Il tuo script inizierà ad essere eseguito ogni volta che clicchi la bandiera verde : 

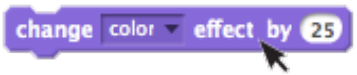


Per fermalo, clicca il pulsante Ferma Tutto:

★ Cambia colore

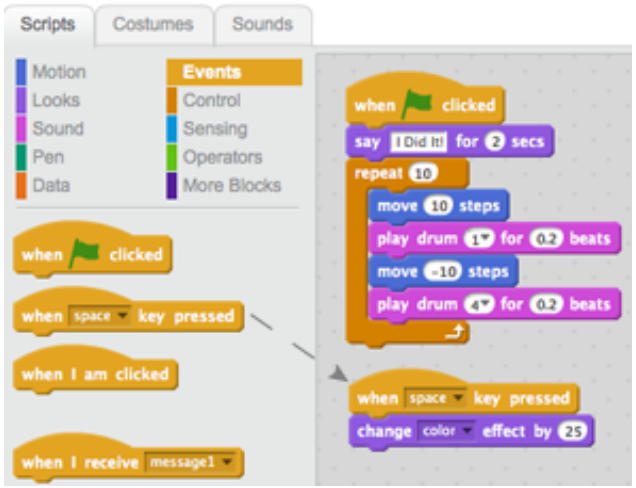
Prendi un blocco CAMBIA EFFETTO.

Cliccalo per vedere cosa fa.



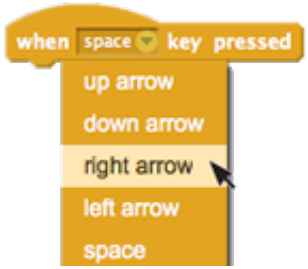
★ Pressione di un tasto

Attacca un blocco 



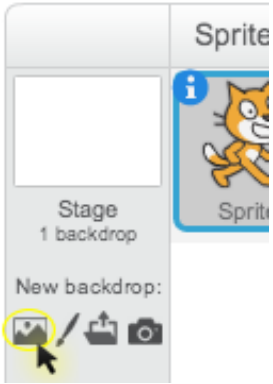
Ora premi la barra spaziatrice della tua tastiera e guarda cosa accade.

Puoi scegliere un tasto diverso nel menu:

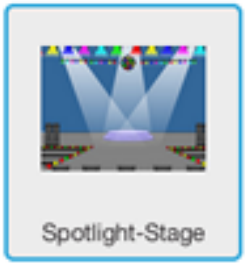


★ Aggiungi uno sfondo

Puoi aggiungere nuovi sfondi alla scena.



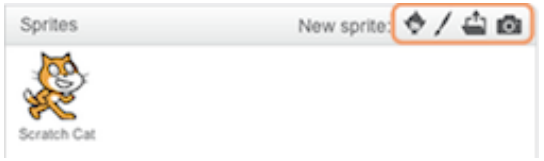
Clicca per selezionare un nuovo sfondo dalla Libreria (ad esempio "Spotlight-Stage"):



Clicca su OK.

★ Aggiungi uno sprite

Come già detto, Ogni oggetto di Scratch è chiamato *sprite*. Per aggiungere un nuovo *sprite* clicca uno di questi pulsanti:



PULSANTI PER LA CREAZIONE DI UN NUOVO SPRITE:

- Scegli uno *sprite* dalla libreria
- Disegna il tuo *sprite*
- Carica una tua immagine o un tuo *sprite*
- Scatta una foto (dalla webcam)

Per aggiungere questo *sprite*, clicca poi seleziona "Cassy Dancing":

Trascina il personaggio dove ti serve sulla scena.

★ Esplora

Puoi aggiunge suoni e animazioni al tuo nuovo sprite.

AGGIUNGERE UN SUONO

Clicca sulla scheda Sounds Suoni . Puoi scegliere un suono , registrare un tuo suono o importare un file audio .

Poi clicca Scripts e trascina nell'area degli script il seguente blocco:

Scegli il suono che preferisci nel menu e clicca il blocco per riprodurlo.

ANIMAZIONI

Clicca Costumes per vedere i costume di uno *sprite*.

Puoi animarlo passando da un costume all'altro.

Clicca Scripts e crea uno script che faccia passare da un costume all'altro:

Approfondimenti sul movimento e altro ancora

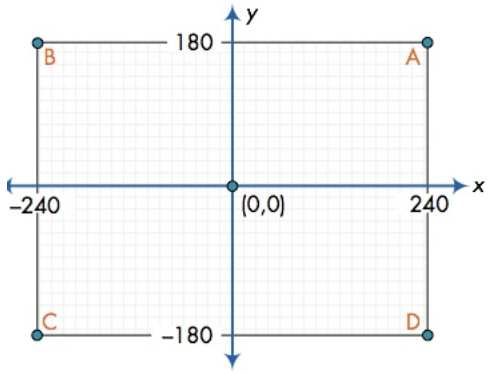
Lo *Stage* è lo schermo nel quale il risultato del nostro programma (col quale l'utente finale interagirà) sarà visualizzato.

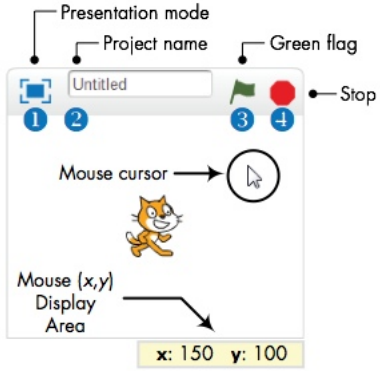
SEZIONE 1

Spostamenti all'interno dello Stage

Dimensioni dello Stage

Ai diversi punti dello *Stage* si può accedere grazie alle note coordinate cartesiane e le sue dimensioni sono di **480** passi in larghezza e di **260** in altezza (ricordiamoci che si misura in passi dato che l'obiettivo è animare un attore detto Sprite).





Spostamenti "assoluti"

Per spostare uno *sprite* in una posizione specifica dello schermo è possibile utilizzare il blocco:

per considerare entrambe le dimensioni. Oppure i due blocchi:

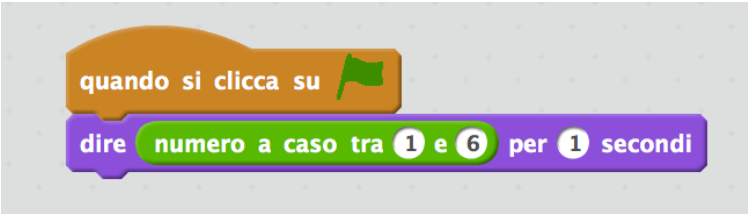
per considerare una dimensione alla volta.

★ Generazione di numeri casuali

Molto spesso, in un programma (es. simulare il comportamento di un dado), c'è bisogno di far generare dei numeri in maniera casuale. Scratch mette a disposizione dei blocchi per eseguire questo compito.

pick random 0 to 1	{0, 1}
pick random 0 to 10	{0, 1, 2, 3, ... , 10}
pick random -2 to 2	{–2, –1, 0, 1, 2}
10 * pick random 0 to 10	{0, 10, 20, 30, ... , 100}
pick random 0 to 1.0	{0, 0.1, 0.15, 0.267, 0.3894, ... , 1.0}
pick random 0 to 100 / 100	{0, 0.01, 0.12, 0.34, 0.58, ... , 1.0}

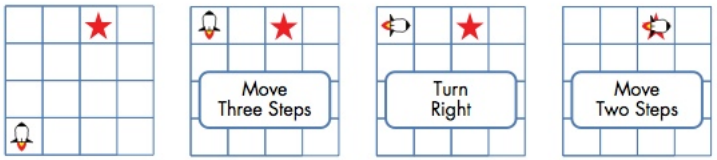
Il seguente programma simula, appunto, il comportamento di un dado a sei facce:



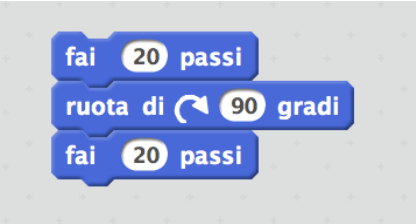
IMPORTANTE: E' interessante vedere come il blocco verde si incastra nel blocco viola. In pratica il risultato del blocco verde (un numero a caso tra 1 e 6) viene consegnato al blocco DIRE per scrivere a video un risultato. Questo è un esempio di dato di input (numero da scrivere a video) che viene generato come dato di output (numero generato casualmente) da un altro blocco.

★ Spostamenti "relativi"

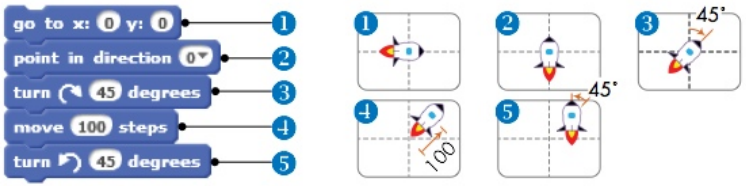
In Scratch è possibile far compiere agli *sprite* anche degli spostamenti "relativi", ovvero considerando la loro posizione corrente e la direzione in cui essi puntano (data dalla loro rotazione). La figura seguente mostra uno spostamento relativo realizzato in tre passi.



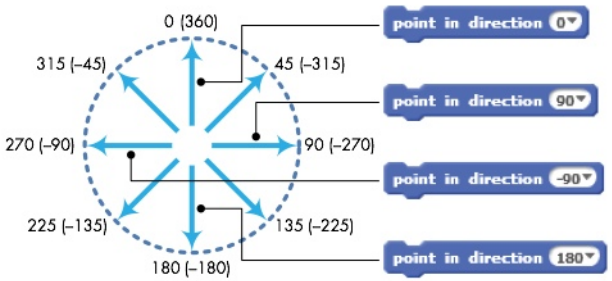
Ad esempio, è possibile riprodurre il comportamento della figura precedente con le istruzioni seguenti:



Nella figura seguente viene proposto un esempio più complesso.



Per quanto riguarda, invece, il blocco PUNTA IN DIREZIONE, la figura seguente mostra il modo di indicare in maniera assoluta i parametri di rotazione dello *sprite*.



Disegno

Per scrivere e disegnare sullo *Stage* è possibile utilizzare il blocchi PENNA.

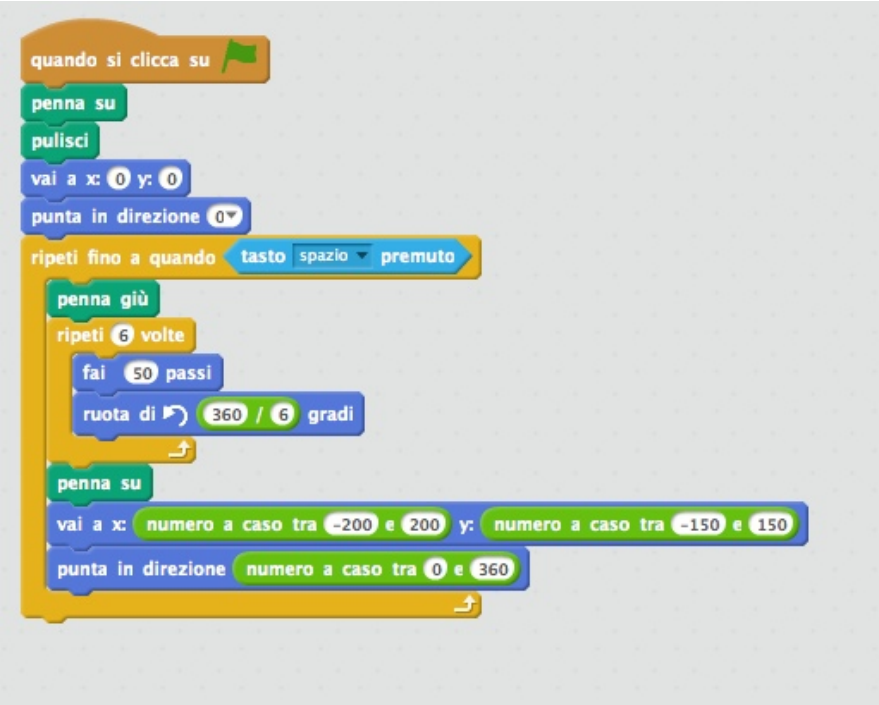
Basta utilizzare PENNA GIU' e far muovere lo *sprite* per rappresentare con un disegno il suo spostamento. Per smettere di scrivere è necessario utilizzare il blocco PENNA SU. Per cancellare il disegno bisogna utilizzare il blocco PULISCI.

Esempi

Di seguito, il programma per disegnare un esagono.

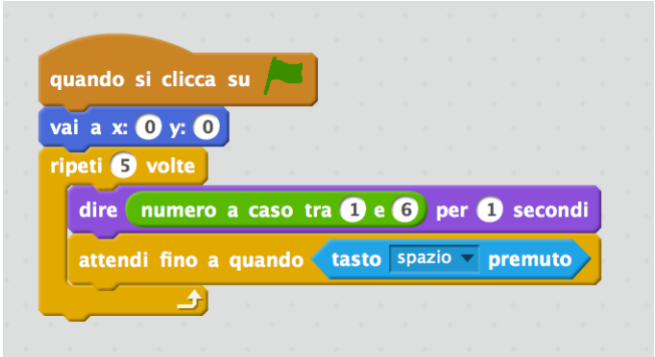
Provate a commentare l'esempio precedente, blocco-per-blocco, e a modificarlo per disegnare un quadrato.

Nell'esempio successivo si disegnano, uno dopo l'altro, degli esagoni distanziandoli in maniera casuale. Il programma termina quando viene premuto il tasto "spazio".



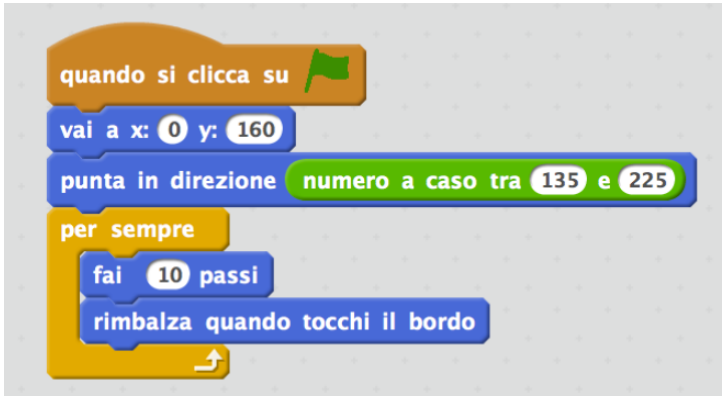
Provate a commentare l'esempio precedente, blocco-per-blocco.

Nell'esempio seguente, lo *sprite* lancia un dado e aspetta che venga premuto il tasto "spazio" per rilanciarlo. In totale, i lanci sono cinque.



IMPORTANTE: nell'esempio precedente c'è l'istruzione ATTENDI che riceve in input un valore (vero o falso) che corrisponde al verificarsi o meno di una condizione (i.e. che è stato premuto un tasto e che questo corrisponda a "spazio). L'istruzione TASTO PREMUTO intercetta una pressione sulla tastiera e verifica che effettivamente sia stato premuto il tasto "spazio". A questo punto, comunica all'istruzione ATTENDI che può smettere di attendere. L'istruzione ATTENDI, invece, fa restare lo *sprite* in attesa fino a quando non viene premuto il tasto "spazio".

L'ultimo esempio è un semplice gioco. Lo *sprite* rimbalza sulle pareti dello *Stage*.


quando si clicca su
vai a x: 0 y: 160
punta in direzione numero a caso tra 135 e 225
per sempre
fai 10 passi
rimbalza quando tocchi il bordo

Capitolo 3

Impariamo a programmare

La programmazione è l'arte di far fare ad un computer quello che noi vogliamo.

Semplici sequenze

★ Ciao Mondo!

La più semplice sequenza di istruzioni che possiamo scrivere è quella costituita da un solo comando.

Vediamo come si fa a visualizzare un testo a video. In questo caso faremo DIRE, al nostro *sprite*, la frase **Ciao Mondo!**

Per fare questo bisogna selezionare e trascinare nel pannello dello Script il seguente blocco:

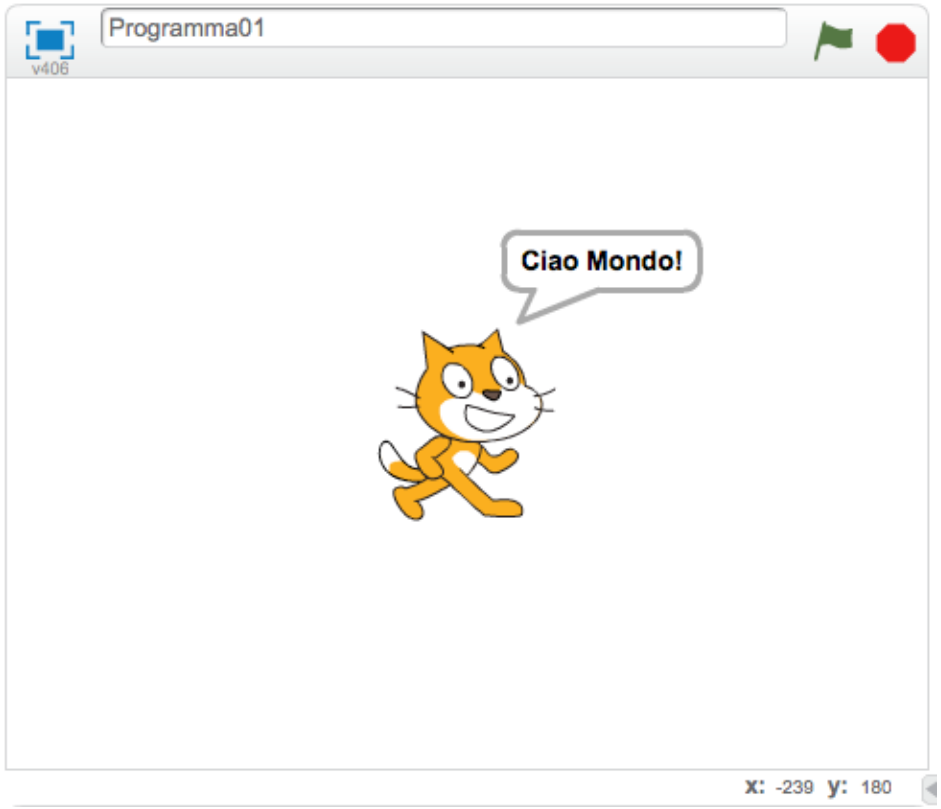
che si trova tra i blocchi di tipo ASPETTO.

Una volta trascinato, il blocco deve essere modificato sostituendo il testo **Hello!** con il testo **Ciao Mondo!**. Questa operazione può essere realizzata facilmente facendo clic su **Hello!** e, successivamente, scrivendo con la tastiera il testo **Ciao Mondo!**

Il risultato dei passi precedenti è il seguente:

A questo punto per eseguire il nostro programma basterà cliccare sulla parte colorata del blocco.

Il risultato dell'esecuzione del nostro programma potrà essere osservata nello Stage:



E' possibile far DIRE tutto ciò che vogliamo allo *sprite*, sostituendo il testo **Ciao Mondo!** con altri caratteri, parole o frasi.

★ Aritmetica

Con Scratch si possono visualizzare altre cose, oltre ai testi. Infatti, se "incastriamo" un blocco ADDIZIONE con il blocco DIRE possiamo visualizzare, ad esempio, il risultato di **6 + 5** utilizzando il blocco  che ha due caselle vuote nelle quali è possibile cliccare e inserire i numeri (gli operandi) da sommare.

In questo caso il risultato sarà:

Attenzione, se non usiamo il blocco ADDIZIONE, ma soltanto il blocco DIRE:

Il risultato sarà:

Questo avviene perchè, non utilizzando il blocco ADDIZIONE di tipo OPERATORI, stiamo dicendo a Scratch di visualizzare la frase **6+5** e non la **somma di 6+5**. Non dimentichiamo mai che il computer fa soltanto quello che gli stiamo dicendo e non interpreta la nostra volontà. E' il programmatore che deve fornire delle indicazioni precise.

Scratch mette a disposizione diversi operatori per eseguire operazioni aritmetiche: addizione (+), sottrazione (-), moltiplicazione (*) e divisione (/).

Tali operatori possono essere combinati per eseguire delle espressioni complesse:

Il risultato è:

Fai attenzione, a comporre le operazioni utilizzando il principio che usi per le parentesi nelle espressioni. Questo serve ad assegnare le giuste priorità agli operatori.

Notare che l'istruzione:

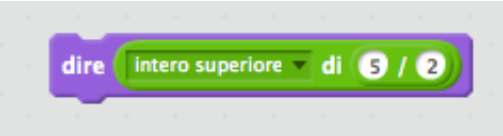
ha come risultato:

Per avere un numero “intero” come risultato bisogna utilizzare, in combinazione con l’ADDIZIONE, un blocco ARROTONDA (che restituisce il numero intero più vicino al risultato della divisione) o un blocco FUNZIONE (intero inferiore o intero superiore).

Per ARROTONDA abbiamo:

con risultato pari a **3**. Mentre per il blocco FUNZIONE abbiamo:

con risultato pari a **2** e:



con risultato pari a **3**.

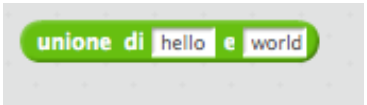
Inoltre, per calcolare quoziente intero e resto di una divisione è possibile utilizzare le istruzioni seguenti. Per quanto riguarda il quoziente interno possiamo utilizzare il blocco FUNZIONE con INTERO INFERIORE come visto precedentemente.

Per quanto riguarda il resto, è possibile utilizzare l’istruzione combinata:

con risultato pari a **1**.

★ Combinare testo e numeri

Ricapitolando, abbiamo visto come visualizzare testi e come visualizzare numeri e risultati delle operazioni sui numeri. In Scratch è anche possibile visualizzare sia numeri che testo, combinandoli insieme. In questo caso utilizzeremo il blocco UNIONE:

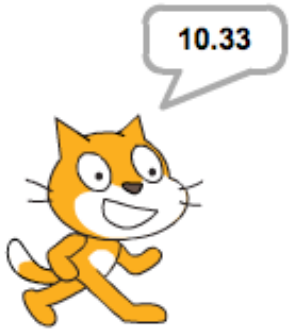


Al posto di **hello** e **world** è possibile inserire nuovi testi o blocchi. Se la nostra intenzione è di visualizzare la seguente espressione: *Il risultato di (8+7) è 15 e il risultato di (8-7) è 1* allora possiamo scrivere la seguente istruzione:

Il trucco è di scomporre la nostra espressione in tante sotto-espressioni composte dall’unione di due elementi.

Per eseguire operazioni con numeri con virgola bisogna scrivere questi numeri utilizzando il '**.**' al posto della '**,**' . Ad esempio, per moltiplicare **3,55** per **2,91**:

Il risultato dell'istruzione precedente è:



SEZIONE 2

I dati e le variabili

1. Introduzione: cosa sono i dati
2. Le variabili
3. Uso delle variabili numeriche
4. Utili esempi
5. Uso delle variabili testo (o stringa)

★ Cosa sono i dati

I dati sono le "cose", i blocchi elementari per costruire le informazioni, che i programmi manipolano. Senza dati un programma non può svolgere alcuna funzione utile. I programmi trattano i dati in molti modi, spesso a seconda del loro *tipo*. Ogni *tipo* di dato ha un insieme di operazioni, azioni che si possono fare con il dato stesso. Ad esempio abbiamo visto che possiamo sommare fra loro i numeri. La somma è una operazione sui dati di *tipo numerico*.

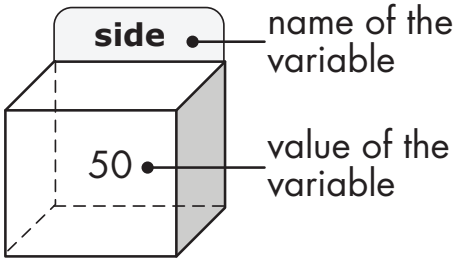
★ Le variabili

Durante l'esecuzione di un programma, i dati sono immagazzinati nella memoria centrale del computer (RAM). Potete immaginarvela come la parete di cassette usate negli uffici postali per smistare la posta. È possibile mettere una lettera in una cassetta, ma questo serve a poco se le cassette non sono etichettate con l'indirizzo di destinazione. Le variabili sono le etichette delle cassette che costituiscono la memoria del computer.

IMPORTANTE: per semplificare la vita del programmatore, le variabili sono configurate come indirizzi simbolici e non numerici.

Una variabile, quindi, è un'area della RAM del computer che può essere "pensata" come una scatola che contiene un singolo dato (e.g. un numero, un carattere, una parola, una frase,

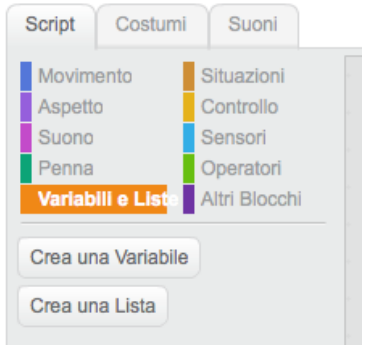
etc.). Nella figura seguente, è rappresentata una variabile di nome *side* che contiene il dato *50*.



Quando creiamo una variabile, Scratch automaticamente costruisce una scatola in memoria e le assegna il nome che noi stessi abbiamo dato alla variabile.

Dopo aver creato la variabile e averle assegnato un nome, il programmatore può utilizzare quel nome per fare riferimento al valore (del dato) che la variabile stessa contiene.

Vediamo, ora, come si creano le variabili in Scratch. Prima di tutto è necessario cliccare sui blocchi di tipo **Variabili e Liste**:



Successivamente, bisogna cliccare sul pulsante **Crea una Variabile**. A questo punto apparirà la seguente *form*:

E' necessario, ora, scrivere il nome da associare alla variabile e poi cliccare sul pulsante **OK**.

★ Le variabili numeriche

Supponiamo di aver scelto il nome **altezza**, Scratch farà apparire nuovi blocchi da poter utilizzare:

Supponiamo di voler far DIRE allo *sprite* che la sua altezza è di 100 cm e di voler usare la variabile **altezza** piuttosto che direttamente il numero **100**.

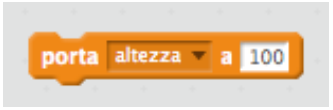
Il programma da scrivere è il seguente:

Se proviamo ad eseguire il programma, il risultato sarà:

Spieghiamo il programma istruzione per istruzione.

L'istruzione 

serve ad eliminare dallo *Stage* la visualizzazione permanente del contenuto della variabile **altezza**, ovvero il box in alto a sinistra della figura seguente:

L'istruzione 

serve ad inserire il dato (in questo caso, il numero) **100** all'interno della variabile **altezza**. Da questo momento in poi ogni volta che vorremo utilizzare **100** useremo la variabile **altezza**.

Infatti, all'interno della istruzione:

utilizziamo il blocco con il nome della variabile **altezza** e non il numero **100**, anche se nel risultato dell'esecuzione al posto di **altezza** viene visualizzato **100**. Quindi non il nome della variabile ma il contenuto.

Facciamo un altro esempio. Supponiamo di voler calcolare la seguente espressione:

$$\frac{(1/5)+(5/7)}{(7/8)-(2/3)}$$

Allora, per quello che abbiamo visto precedentemente, è possibile scrivere la seguente istruzione:

Se volessimo utilizzare le variabili dovremmo, prima di tutto, crearne, ad esempio, due: **num** e **den**. Le due variabili rappresenteranno il numeratore e il denominatore della nostra frazione.

Più nel dettaglio, bisogna trascinare il blocco 

nel pannello dello script, cliccare sulla freccetta che punta verso il basso 

e cliccare su **num**: 

A questo punto è possibile scrivere il dato (o valore) che vogliamo inserire all'interno della variabile **num**. Nel nostro caso il dato è l'espressione **(1/5) + (5/7)**, per cui abbiamo bisogno di utilizzare dei blocchi Operatori:

IMPORTANTE: per eseguire l'istruzione precedente, Scratch calcolerà prima il valore del risultato dell'espressione indicata dai blocchi Operatori (quelli verdi), prenderà soltanto il risultato e lo memorizzerà nella variabile **num**. Quindi, fate attenzione che da questo momento in poi, Scratch avrà memo-

ria solo del risultato dell'espressione indicata nei blocchi verdi e non delle varie operazioni che la compongono.

Bisognerà ripetere lo stesso procedimento per la variabile **den** e ottenere quindi:

Il programma completo sarà quindi:

Proviamo a complicare il programma e far calcolare anche la nuova espressione **((1/5) + (5/7)) / ((6/7) - (3/4))**. Notiamo che la prima parte dell'espressione, ovvero il numeratore della frazione principale è esattamente la stessa dell'espressione precedente. Per cui possiamo utilizzare il valore già calcolato e salvato nella variabile **num**, mentre dovremmo "cambiare" il valore della variabile **den**. Il programma completo è il seguente:

Nel programma precedente abbiamo fatto uso di due nuovi blocchi mai usati prima.

Il primo è 

che serve a modificare il valore (dato) contenuto nella variabile **den**. Dopo aver eseguito questa istruzione, la scatola **den** perderà per sempre il valore dell'espressione **((7/8) - (2/3))** e acquisirà e manterrà il valore dell'espressione **((7/8) - (2/3))** + **((6/7) - (3/4))**. Il CAMBIA a differenza del PORTA non sostituisce il vecchio valore presente nella variabile ma somma a questo il valore calcolato dall'espressione a destra del "di".

Il secondo è 

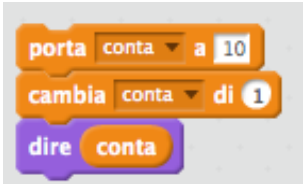
che serve a congelare l'esecuzione del programma per 4 secondi in maniera tale da dare la possibilità a chi assiste al risultato di osservare prima il risultato della prima espressione e poi quello della seconda espressione. Se al posto di questo blocco inseriamo il

semplice blocco DIRE non sarà possibile osservare il risultato della prima espressione perchè sostituito immediatamente (i tempi di esecuzione di queste istruzioni non sono percepibili all'occhio umano) dal risultato della seconda.

IMPORTANTE: C'è da notare, infine, che l'uso della variabile **den** ci ha permesso di riusare un valore già calcolato in precedenza (prima espressione) e di non dover eseguire nuovamente gli stessi calcoli nella seconda espressione.

★ Utili esempi

Per incrementare di una unità il valore contenuto in una variabile è possibile utilizzare le seguenti istruzioni:

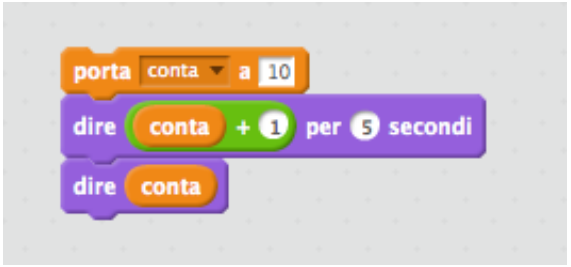


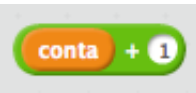
Infatti, il blocco CAMBIA è capace di sommare **1** al contenuto di una variabile (nel nostro caso chiamata **conta**). Per incrementare di **2** il valore di **conta** bisogna sostituire **1** con **2** nel blocco CAMBIA. E così via.

Per decrementare il contenuto di una variabile di **1** è possibile utilizzare la seguente istruzione:

IMPORTANTE: le istruzioni precedenti modificano il contenuto di una variabile.

Per incrementare di **1** il valore di una variabile senza modificare il contenuto originale della variabile stessa non bisogna utilizzare il blocco CAMBIA o il blocco PORTA come mostrato nel seguente programma:



Infatti, se eseguiamo il programma precedente ci accorgiamo che utilizzando la composizione 

all'interno del blocco DIRE, lo *sprite* "dice" esattamente il numero **11** (ovvero il contenuto di **conta** incrementato di un'unità). Ma poi se proviamo a visualizzare il contenuto di conta, lo *sprite* ci darà il risultato sperato ovvero il valore corretto (quello originale) di **conta** che è **10**.

Infine, supponiamo di voler elevare alla terza potenza il valore contenuto nella variabile **conta** e poi sostituire il suo valore originale con il risultato della potenza. Il programma da scrivere è il seguente:

★ Uso delle variabili testo (o stringa)

In Scratch è possibile usare le variabili anche per contenere caratteri, parole e intere frasi.

Vediamo il seguente programma:

e il suo risultato:

Apportiamo qualche modifica al programma precedente, provando ad utilizzare più volte la stessa variabile:

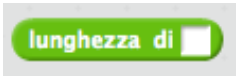
e osserviamo il risultato:

Il blocco 

può essere utilizzato, quindi, per concatenare caratteri, parole e frasi, due alla volta.

Altri blocchi interessanti per manipolare le parole sono 

per estrarre una lettera da una parola o frase, e

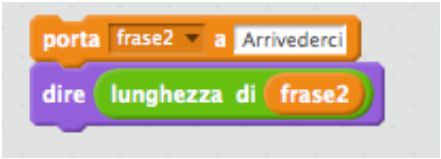


per contare il numero di caratteri che compone una parola o frase.

Vediamo come si usano.

Il programma precedente estrae la terza lettera della parola contenuta nella variabile **frase2** e la visualizza. Il risultato è il seguente:

Invece, il seguente programma:



calcola la lunghezza (il numero dei caratteri) di **frase2** e la visualizza. Il risultato è il seguente:

SEZIONE 3

Input dei dati

1. Leggere l'input dell'utente
2. Usare i dati in input
3. Conservare i dati in input
4. Elaborare i dati in input

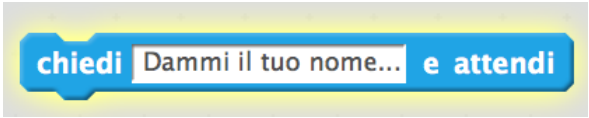
★ Leggere l'input dell'utente

Scratch offre la possibilità di definire dei programmi che interagiscono con l'utente anche in fase di immissione (input) dei dati e non solo di visualizzazione (output) degli stessi.

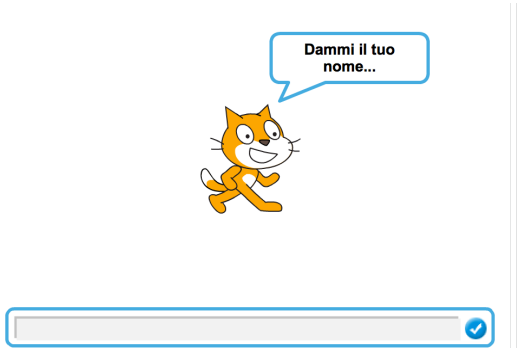
Il blocco necessario per leggere l'input dell'utente è:

che è possibile trovare nel pannello relativo ai blocchi di tipo Sensori.

Vediamo come si usa con un semplice esempio:



Questo semplice programma ci permette di chiedere all'utente il suo nome. L'esecuzione del programma ha il seguente risultato:



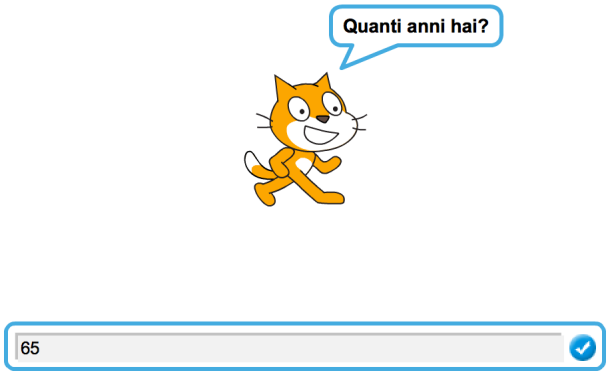
Come è possibile notare dalla figura precedente, Scratch visualizzerà il messaggio che abbiamo scritto all'interno del blocco CHIEDI E ATTENDI e farà comparire un'area dedicata all'inserimento dei dati da parte dell'utente. Scratch attenderà fino a quando non sarà premuto il tasto INVIO da parte dell'utente e solo successivamente continuerà normalmente con l'esecuzione del programma.

★ Usare i dati in input

Proviamo ora a scrivere un programma che chiede l'età all'utente e la fa DIRE allo *sprite*:

Facendo clic sulla bandiera verde possiamo eseguire il programma. L'esecuzione si ferma sul blocco CHIEDI:

A questo punto l'utente può scrivere nella casella preposta all'input dei dati:



Quando l'utente preme il tasto INVIO, Scratch legge il dato in input e lo deposita in una variabile speciale chiamata **risposta** (che troviamo tra i blocchi di tipo Sensori).

Il programma prosegue la sua esecuzione:

Lo *sprite* visualizza la giusta età dell'utente grazie all'esecuzione dell'istruzione:

E' importante notare che per visualizzare la frase **Hai 65 anni** abbiamo bisogno di comporre la parola **Hai** con il contenuto della variabile speciale **risposta** (che contiene il dato di input dell'utente) e la parola **anni**.

★ Conservare i dati in input

Complichiamo un po' il problema. La nostra intenzione è di scrivere un programma che fa due domande all'utente, ricevendo due risposte e, infine, combina le risposte visualizzandole attraverso lo *sprite*.

Osserviamo il seguente programma:

Se alla domanda **Come ti chiami?** l'utente risponde **Francesco** e alla domanda **Quanti anni hai?** l'utente risponde **39** il risultato del programma è il seguente:

Il risultato ottenuto non è quello che ci aspettavamo. Perchè?

Perchè la variabile speciale chiamata **risposta** può contenere un unico dato che corrisponde all'ultimo input inserito dall'utente (eliminando il precedente dato immesso). Nel nostro caso l'ultimo dato inserito dall'utente è **39**. Per cui, provando a visualizzare due volte il contenuto di **risposta**, visualizzeremo due volte il numero **39**.

Per ottenere il risultato sperato, dobbiamo modificare il programma nel seguente modo:

La soluzione è introdurre le due variabili **nome** e **anni** che conterranno rispettivamente la risposta alla domanda **Come ti chiami?** e la risposta alla domanda **Quanti anni hai?**

Di seguito si riporta il risultato del programma corretto.

Più nel dettaglio, nel precedente programma bisogna fare attenzione alle seguenti istruzioni:

che catturano l'input dell'utente (che si troverà nella variabile speciale **risposta**) e lo inseriscono nella variabile **nome**.

La stessa cosa va fatta per la seconda domanda.

Inoltre, quando si usa il blocco DIRE, non si dovrà più utilizzare la variabile **risposta** ma le variabili **nome** e **anni**.

★ Elaborare i dati in input

Proviamo a trasformare Scratch in una calcolatrice che per il momento esegue soltanto le moltiplicazioni tra due numeri.

Per fare questo dobbiamo scrivere un programma che riceve in input due numeri (gli operando) e visualizza il risultato della moltiplicazione (operatore):

Gli operandi dati in input dall'utente vengono depositati rispettivamente nelle variabili **a** e **b**. La novità introdotta in questo programma è che i due dati in input vengono elaborati utilizzando l'istruzione 

, il cui valore risultante viene depositato nella variabile **c**, che conterrà quindi il risultato della moltiplicazione.

La variabile **c** viene riempita con il risultato della moltiplicazione con l'istruzione 

.

IMPORTANTE: quello appena analizzato è un esempio di algoritmo/programma parametrico.

SEZIONE 4

Prendere decisioni

1. Operatori di confronto (relazionali)
2. Decisioni con IF-THEN-ELSE
3. Operatori logici
4. Approfondimento su IF-THEN-ELSE

Nella stragrande maggioranza dei programmi è necessario alterare l'esecuzione sequenziale delle istruzioni. Ad esempio, se abbiamo la necessità di eseguire delle istruzioni per premiare le risposte corrette in un gioco e non premiare in caso di risposte sbagliate, dobbiamo "confrontare" le risposte del giocatore (che nel nostro caso è l'utente) con le risposte effettivamente corrette. Questo procedimento è alla base delle "decisioni" nei linguaggi di programmazione.

In questa sezione vedremo come è possibile far prendere "decisioni" a Scratch in maniera tale da alterare il flusso sequenziale delle istruzioni di un programma.

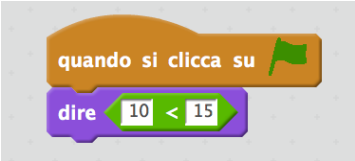
★ Operatori di confronto (relazionali)

Gli operatori di confronto messi a disposizione da Scratch sono i seguenti:

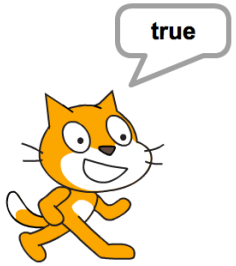
Operator	Meaning	Example
	greater than	 Is price greater than 2,000?
	less than	 Is price less than 2,000?
	equal to	 Is price equal to 2,000?

Proviamo a confrontare dei numeri. Facciamo degli esempi.

Iniziamo con il confrontare due numeri e chiedere a Scratch se il primo è più piccolo del secondo:

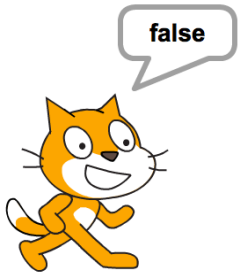


Il risultato del programma è il seguente:



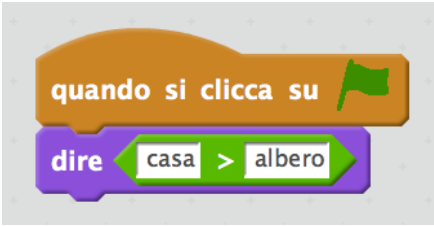
In questo caso lo *sprite* ci dice che il confronto risulta vero (true) cioè che il numero **10** è effettivamente minore di **15**.

Se modifichiamo il programma invertendo le posizioni di **10** e **15** rispetto all'operatore <, abbiamo il seguente risultato:

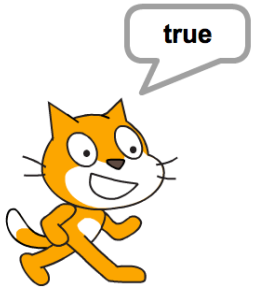


Gli operatori > e = funzionano esattamente allo stesso modo ma verificano rispettivamente che il primo operando è maggiore del secondo e che il primo operando è uguale al secondo.

Allo stesso modo è possibile confrontare caratteri e parole:



Il risultato in questo caso è:



Scratch ci dice che **casa** è maggiore di **albero**. Ma cosa significa?

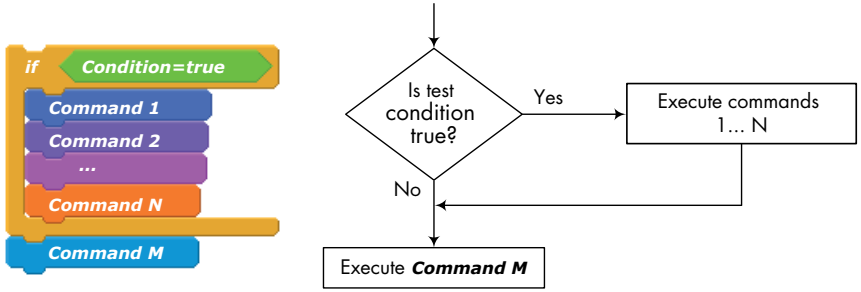
I confronti tra caratteri e parole avvengono seguendo l'ordine lessicografico: la lettera **a** è più piccola della **b** che è più piccola della **c** e così via. Se proviamo ad eseguire il confronto:

la risposta sarà **true**, perchè dato che **cosa** e **casa** iniziano entrambe per **c** si vanno a confrontare il secondo carattere della prima parola con il secondo carattere della seconda parola. Nel nostro caso **o** è maggiore di **a** per cui risulta che **cosa** è più grande di **casa** (sempre considerando l'ordine lessicografico) nel confronto effettuato da Scratch.

★ Decisioni con IF-THEN-ELSE

Attraverso l'uso del blocco IF-THEN è possibile attivare o disattivare l'esecuzione di un intero blocco di istruzioni:

Di seguito mostriamo lo schema generale del funzionamento del blocco IF-THEN:



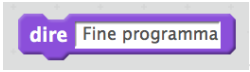
In breve, quando l'esecuzione del programma raggiunge un blocco IF-THEN, il confronto indicato col blocco verde viene eseguito, se il confronto restituisce **true** come risposta (la condizione è vera) allora vengono eseguiti i blocchi chiamati Command 1, Command 2, fino ad arrivare a Command N e poi viene eseguito il blocco chiamato Command M. Se, invece, il confronto restituisce **false** come risposta (la condizione è falsa) allora Scratch esegue direttamente il blocco Command N senza eseguire Command 1, Command 2, ..., Command N.

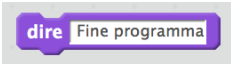
Nella figura precedente si riporta il diagramma di flusso relativo ad un blocco IF-THEN.

Vediamo il seguente programma.

Il programma chiede in input tre dati e li deposita nelle variabili **a**, **b** e **c**. Le prime due rappresentano gli operandi di un'operazione, mentre la terza è il simbolo corrispondente al-

l'operazione aritmetica da eseguire. Il blocco IF-THEN verifica che la variabile **c** contenga il simbolo + (ovvero che l'utente abbia richiesto di eseguire un'addizione). Se così è, Scratch lo verifica eseguendo il confronto 

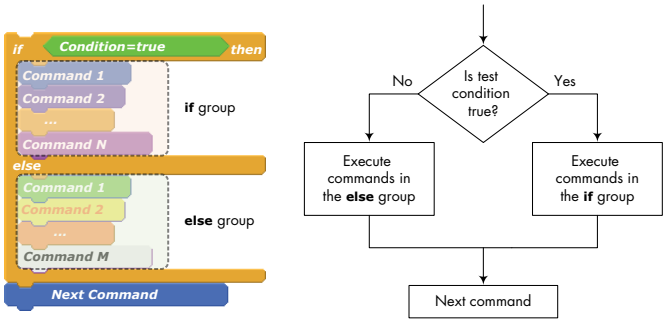
. In questo caso, Scratch calcola il risultato dell'addizione sommando il contenuto di **a** con il contenuto di **b** e visualizza tale risultato aspetta 5 secondi e poi esegue 

. Se, invece, il contenuto di **c** è diverso da + (ovvero il risultato del confronto è **falso**) allora si esegue direttamente il blocco 

.

Il blocco IF-THEN-ELSE fornisce una possibilità in più, ovvero decidere cosa fare quando la condizione dell'IF risulta **falso**.

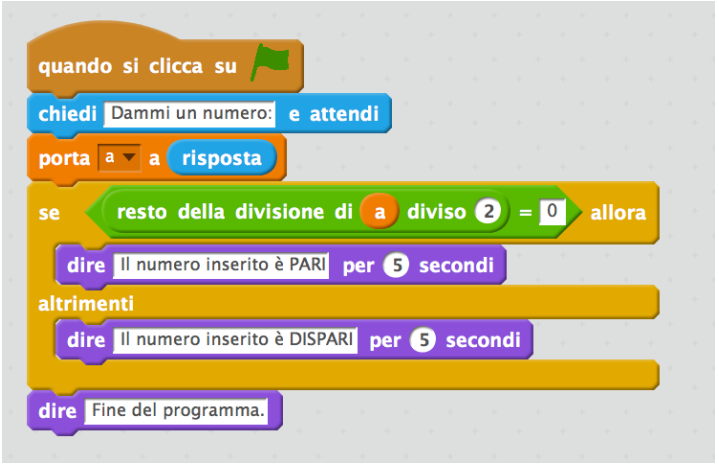
Di seguito mostriamo lo schema generale del funzionamento del blocco IF-THEN-ELSE:



In breve, quando l'esecuzione del programma raggiunge il blocco IF-THEN-ELSE, viene valutata la condizione espressa nel blocco verde. Se il confronto risulta vero allora vengono eseguiti i blocchi Command 1, Command 2,..., Command N (dell'IF GROUP) e poi Next Command. Se il confronto risulta falso allora vengono eseguiti i blocchi Command 1, Command 2, ..., Command M (dell'ELSE GROUP) e poi Next Command.

Anche in questo caso viene riportato il diagramma di flusso relativo all'esecuzione del blocco IF-THEN-ELSE.

Vediamo un esempio:



Il programma precedente sa "capire" se un numero inserito in input è pari oppure dispari.

La parte più importante del programma è il confronto effettuato nell'ambito del blocco IF-THEN-ELSE:

Il blocco precedente è stato ottenuto con il blocco che calcola il resto di una divisione e il blocco confronto =.

Il primo blocco considerato è:

mentre il secondo è:

L'idea che sfruttiamo è che i numeri pari sono divisibili per due, questo equivale a dire che dividendo questi numeri per due abbiamo resto nullo (ovvero uguale a **0**). Ciò non accade per i numeri dispari. Utilizzando, quindi, il calcolo del resto della divisione per due e il confronto di uguaglianza con lo **0** possiamo completare la condizione del blocco IF-THEN-ELSE.

Concludendo, quando il confronto restituisce valore **true**, ovvero il resto ottenuto è **0**, Scratch può "decidere" che il numero in input è pari e scriverlo con il blocco DIRE nel raggruppamento SE (IF GROUP). Alternativamente (il risultato del confronto è **false**), Scratch può "decidere" che il numero in input è dispari e scriverlo con il blocco DIRE presente nel raggruppamento ALTRIMENTI (ELSE GROUP).

★ Operatori logici

In tutti i linguaggi di programmazione è possibile definire dei confronti (o condizioni) composti. Questo è possibile se si utilizzano i cosiddetti operatori logici:

Operator	Meaning
	The result is true only if the two expressions are true.
	The result is true if either of the two expressions is true.
	The result is true if the expression is false.

Per calcolare il risultato degli operatori logici abbiamo bisogno delle rispettive tabelle di verità:

Tabella per AND (e):

X	Y	
true	true	true
true	false	false
false	true	false
false	false	false

Tabella per OR (o):

X	Y	
true	true	true
true	false	true
false	true	true
false	false	false

Tabella per NOT (non):

X	not X
true	false
false	true

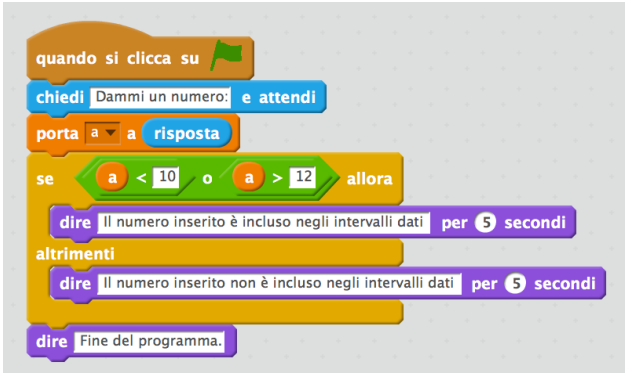
Le tabelle precedenti vanno lette in questo modo. Se abbiamo due confronti legati tra loro da un blocco AND (e) e questi sono entrambi veri allora il risultato del confronto composto è **true**. In tutti gli altri casi il confronto composto ha come risultato **false**. Se, invece, abbiamo due confronti legati tra loro da un blocco OR (o) e questi sono entrambi falsi allora il risultato del confronto composto è **false**. In tutti gli altri casi il risultato del confronto composto è **true**. Infine, l'operatore NOT (non), che è unario, non fa altro che negare il valore di verità. Se il confronto ha risultato **false** allora applicando NOT(non) otteniamo **true**, e viceversa.

Vediamo qualche esempio.

Scriviamo un programma che verifica se il numero inserito dall'utente è compreso nell'intervallo (10, 15) con 10 e 15 esclusi:

In questo caso abbiamo utilizzato l'operatore logico AND (e):

Scriviamo un programma che verifica se un numero inserito in input è minore di 10 oppure è maggiore di 12:



In questo caso abbiamo utilizzato l'operatore logico OR (o):

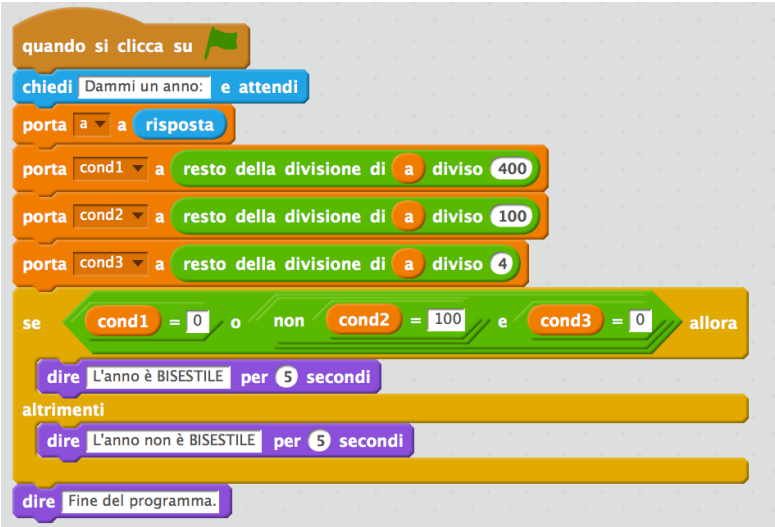
Scriviamo un programma che verifica se un numero inserito in input è diverso da 10:

In questo caso abbiamo utilizzato l'operatore logico NOT (non):

Infine, proviamo a scrivere un programma più complesso per verificare se un anno è bisestile.

La regola è la seguente: « Un anno è bisestile se il suo numero è divisibile per 4, con l'eccezione degli anni secolari (quelli divisibili per 100) che non sono divisibili per 400 ».

Dobbiamo trasformare la regola precedente in un programma Scratch:



In breve dobbiamo verificare che l'anno inserito è divisibile per 400 oppure è divisibile per 4 ma non per 100. In questo caso l'anno è bisestile. Altrimenti non lo è.

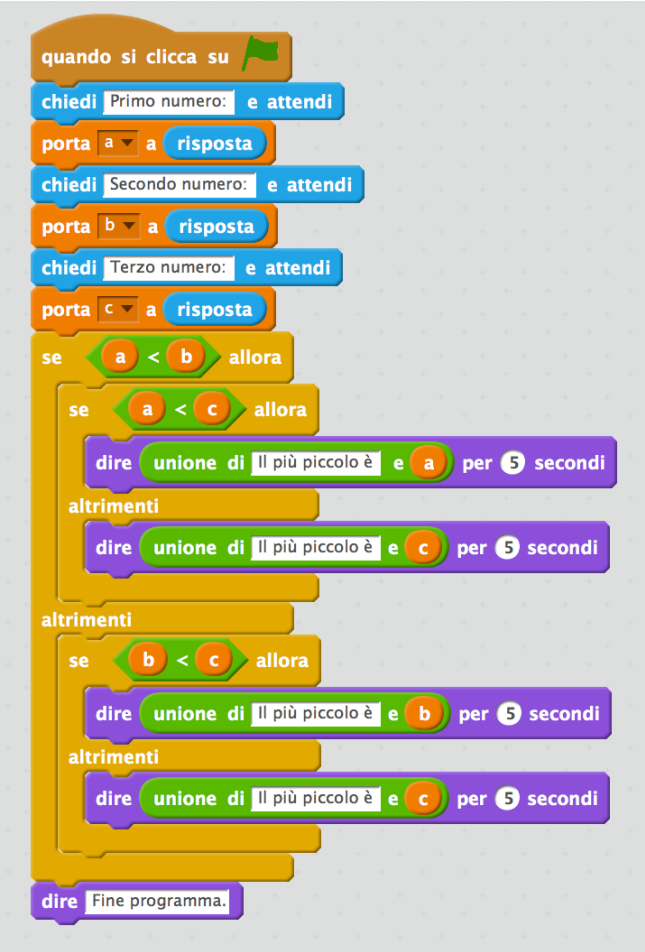
★ Approfondimento su IF-THEN-ELSE

Costruiamo una piccola calcolatrice:

Nel programma precedente sono stati inseriti cinque blocchi IF-THEN a cascata in maniera tale da capire quale operazione è stata richiesta, eseguirla e salvare il risultato nella variabile

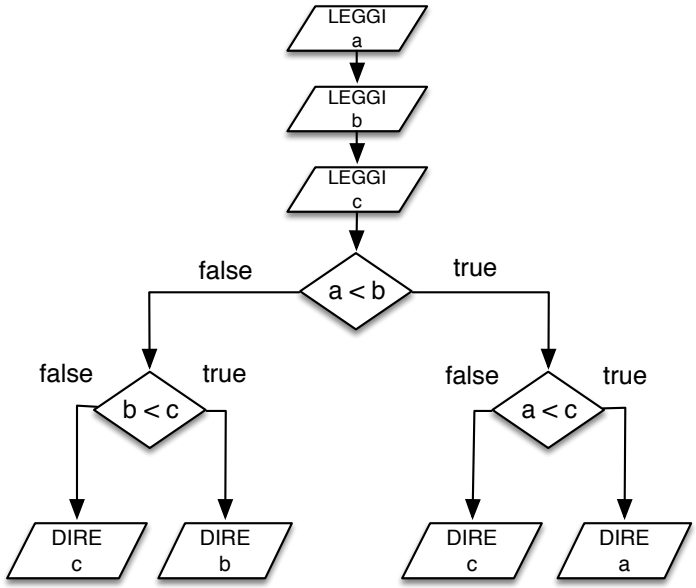
ris. L'ultimo dei blocchi IF-THEN si occupa di controllare che l'operatore inserito in input non corrisponde a nessuno degli operatori gestiti dal programma (i.e. +, -, *, /). Se è così, nella variabile **ris** sarà inserita la parola **ERRORE**.

Infine, i blocchi IF-THEN-ELSE e IF-THEN possono anche essere annidati.



Il programma precedente riconosce il più piccolo tra tre numeri inseriti in input dall'utente. Per fare questo sono stati utilizzati due blocchi IF-THEN-ELSE all'interno di un terzo blocco IF-THEN-ELSE.

Il flow chart corrispondente al programma precedente è:



SEZIONE 5

I cicli (loops)

1. FOREVER (per sempre)
2. REPEAT
3. REPEAT UNTIL
4. CICLI ANNIDATI

Molto spesso capita di dover ripetere più di una volta la stessa sequenza di operazioni. Piuttosto che copiare le stesse istruzioni e metterle in sequenza (soluzione inaffidabile, non efficiente e molto spesso non efficace), Scratch mette a disposizione (come tutti gli altri linguaggi di programmazione) delle istruzioni speciali (di controllo) chiamate CICLI. Esistono tre diversi tipi di CICLI:

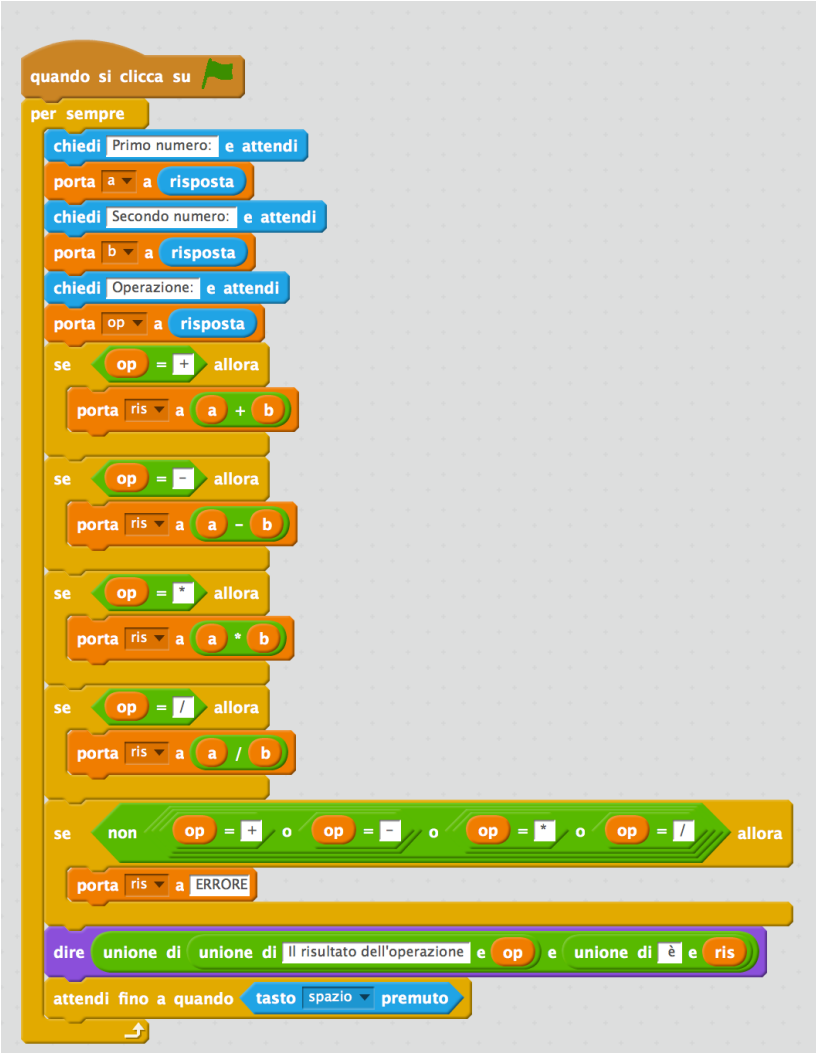
- *infinite loops* (cicli infiniti)
- *counter-controlled loops* (cicli definiti controllati da un contatore)
- *condition-controlled loops* (cicli indefiniti controllati da una condizione)

★ FOREVER (per sempre)

Il blocco FOREVER lo abbiamo già visto nei capitoli precedenti:

Rivediamo l'esempio della calcolatrice della sezione precedente (par. "Approfondimento su IF-THEN-ELSE"). Il programma, una volta avviato, permette di eseguire una sola operazione per poi terminare. Se si vuole eseguire una nuova operazione bisogna avviare nuovamente il programma. E' come se

quando usiamo Word per scrivere una relazione dovessimo avviare (fare doppio clic sull'icona di Word) il programma ad ogni nuova parola che scriviamo. Ci rendiamo tutti conto che per scrivere programmi che si avvicinano a quelli reali dobbiamo superare questa limitazione. Per fare questo possiamo utilizzare i cicli.



Nel programma precedente abbiamo semplicemente inserito il le istruzioni della vecchia calcolatrice all'interno di un blocco FOREVER con la semplice aggiunta dell'istruzione:

che serve a "congelare" temporaneamente l'esecuzione del programma appena visualizzato il risultato in attesa che l'utente prema il tasto "spazio" per richiedere due nuovi numeri e l'operazione da effettuare.

Questo programma non termina o, almeno, non termina fino a quando non si usa il tasto  .

★ REPEAT

Il blocco REPEAT serve ad eseguire lo stesso blocco di istruzioni per un numero definito di volte. Questo numero viene calcolato in base ad un contatore:

Il numero **10** nell'esempio precedente rappresenta il numero di cicli o il numero di volte che vogliamo eseguire le istruzioni "avvolte" nel REPEAT (ripeti-volte). Ovviamente, se, ad esempio, eliminiamo il **10** sostituendolo con **15**, i cicli diventeran-

no appunto quindici. In effetti ci sarebbe un'altra possibilità, utilizzare una variabile al posto del valore fissato nella casella che indica il numero di cicli nel blocco REPEAT (ripeti-volte). Questo ci permette di scrivere dei programmi molto più flessibili e parametrici. Vediamo un esempio.

Vogliamo scrivere un programma che somma *N* numeri dati in input dall'utente. Noi che scriviamo il programma, però, non sappiamo, al momento, quanti numeri vorrà sommare l'utente (questo equivale a dire che non conosciamo in anticipo il valore di *N*). In realtà, ad ogni nuova esecuzione del programma, l'utente potrà scegliere di sommare 5 numeri, 10 numeri, 3 numeri, e così via. Scriviamo il programma e poi lo commentiamo.

La prima cosa importante da notare è che questa volta nel blocco REPEAT (ripeti-volte) non c'è un numero fissato ma la variabile **N**. Il valore di questa variabile viene scelto dall'utente:

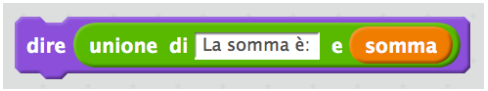
La seconda cosa da notare è che all'interno stesso del blocco REPEAT viene chiesto il nuovo numero da sommare e collocato nella variabile **nuovoNumero**. Questo viene fatto ad ogni ciclo, quindi per **N** volte.

L'istruzione 

, ad ogni ciclo, somma il nuovo numero inserito ad una somma parziale contenuta appunto nella variabile **somma**.

In particolare, la tabella seguente mostra, ad ogni ciclo, il valore contenuto nella variabile **somma** (assumiamo **N** = 3 e che i tre numeri inseriti dall'utente sono 4, 6 e 9):

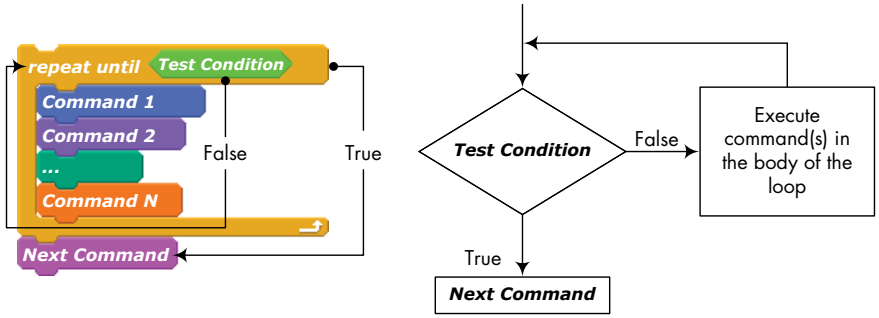
NUMERO CICLO	VALORE DELLA VARIABILE NUOVONUM	VALORE DELLA VARIABILE SOMMA ALLA FINE DEL CICLO
FOREVER non ancora iniziato	FOREVER non ancora iniziato	0
1	4	4
2	6	10
3	9	19
FOREVER terminato	9	19

L'istruzione 

viene eseguita al termine dell'istruzione REPEAT e quindi il valore di **somma** stampato è **19**, che è esattamente la somma dei tre numeri in input dati dall'utente: 4, 6 e 9.

★ REPEAT UNTIL

Immaginate di stare programmando un gioco che consiste nel fare delle domande di matematica al player (utente). Se la risposta del player è sbagliata, il gioco offrirà un'altra possibilità rifacendo la stessa domanda. In altre parole, il gioco fa dare risposte all'utente fino a che l'utente stesso non risponde esattamente. Il blocco REPEAT non è adatto perchè non conosciamo in anticipo (prima che inizi il ciclo) le risposte del giocatore. In questo caso, invece, è più utile il blocco REPEAT UNTIL (ripeti-fino a quando). Analizziamo il comportamento di questo blocco:

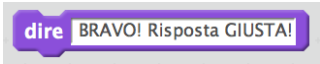


In breve, quando l'esecuzione del programma arriva al blocco REPEAT UNTIL, Scratch valuta la condizione (blocco verde) associata a questo blocco. Se la condizione è valutata **false** allora vengono eseguite le istruzioni Command 1, Command 2, ..., Command N per poi ritornare a verificare la condizione. Se quest'ultima risulta ancora **false** vengono eseguite nuovamente le istruzioni all'interno del blocco. Questa sequenza viene ripetuta fino a che la condizione non assume valore **true**. In questo caso il flusso di esecuzione "salta" il contenuto del blocco ed esegue direttamente Next Command.

Vediamo un esempio:

Il programma precedente è un gioco nel quale il giocatore deve calcolare l'area di un quadrato con la misura del lato generata casualmente dal programma stesso. Il trucco sta nel fatto che il risultato corretto viene calcolato da Scratch che lo confronta con la risposta dell'utente. Se i due valori sono uguali

vuol dire che la risposta del giocatore è corretta mentre se i due valori sono discordanti allora la risposta del giocatore non è corretta. Nel secondo caso, la condizione all'interno del blocco REPEAT UNTIL (ripeti fino a quando) restituisce valore **false** e, quindi, i blocchi all'interno possono essere eseguiti, rifacendo la stessa domanda al giocatore:

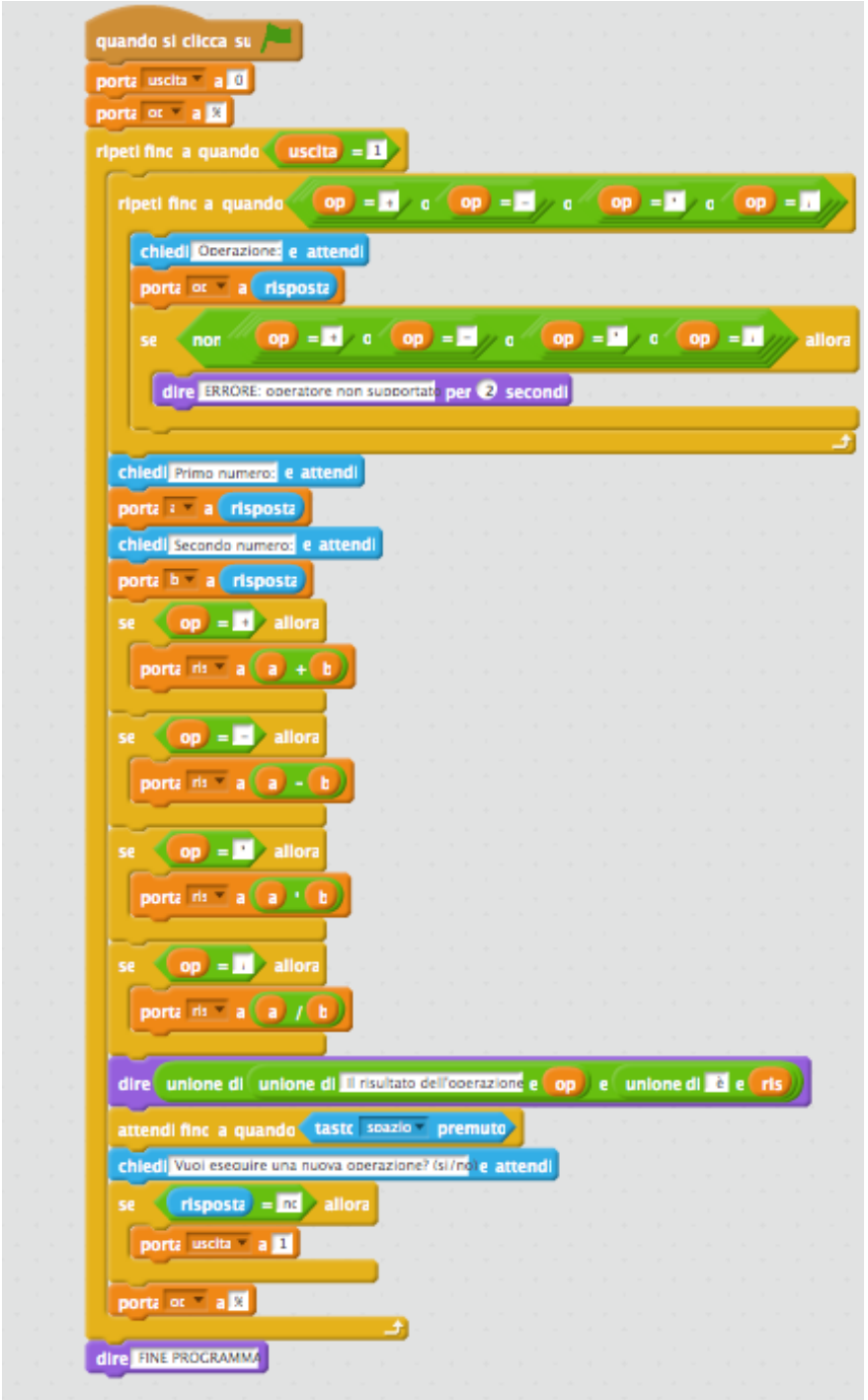
Nel primo caso, la condizione all'interno del blocco REPEAT UNTIL (ripeti fino a quando) restituisce valore **true** e, quindi, i blocchi all'interno non vengono eseguiti. In questo caso Scratch esegue direttamente l'istruzione 

.

★ CICLI ANNIDATI

In casi reali è molto frequente l'esigenza di utilizzare i cicli annidati: ovvero un'istruzione di ciclo all'interno di un'altra.

Vediamo direttamente un caso reale in cui è possibile utilizzare il suddetto annidamento.



Il programma precedente fa uso di due cicli annidati.

Quello più esterno si interrompe quando la risposta dell'utente alla domanda **Vuoi eseguire una nuova operazione?** risulta essere uguale a **no**. Tale ciclo permette al programma di eseguire più di un'operazione e di fermarsi se richiesto dall'utente.

Il ciclo più interno, invece, permette di gestire il caso in cui l'utente indica un'operatore non gestito dal programma. In questo caso viene richiesto di selezionare nuovamente l'operatore da utilizzare. Il ciclo termina se l'operatore indicato è tra quelli gestiti dal programma (+, -, *, /).