

Appunti di scuola

Introduzione alla programmazione

Diomede Mazzone

2022, ver. 0.41



Sommario

Sommario	1
Introduzione alla programmazione	3
Istruzioni	3
Formalismo grafico	4
Variabili	4
Diagrammi di flusso	5
Strutture di controllo	6
Sequenza	6
Selezione	6
Iterazione	8
Ciclo di For	9
Ciclo di While	10
Ciclo Do-While	11
Strutture dati	12
Liste e matrici	12
Procedure e funzioni	13
Algoritmo	14
Kojo	14
Variabili e Valori	17
Selezione	18
Operazioni ripetute	20
Definire una funzione	21
Parametri di funzioni	23
Istruzioni basilari	27
Gestione dei colori	27
Programmazione concorrente	28
Ricorsione e frattali	30
Texture	33
Shape e block	33
Picture	39
Figure geometriche e colori	42
Twine	43
Interfaccia	44
Costruire una storia	45

Creare Condizioni	48
Immagini, sfondi e altri oggetti multimediali	52
HTML	54
TAG	55
CSS	56
Javascript	58
DOM	59
Scrivere in Javascript	59
Definizione di funzioni	61
Monitoraggio delle attività	62
Sintassi del codice	65
Bibliografia	71

Introduzione alla programmazione

Un programma, un software, o una qualunque procedura che ci permette di realizzare una determinata attività è in sostanza una sequenza di operazioni ben determinata e che produce sempre lo stesso risultato, con il vincolo però che le azioni vengono sempre realizzate nello stesso ordine.

Spesso una procedura viene definita per descrivere un'operazione complessa, da comunicare ad un altro soggetto oppure per istruire una macchina. Per evitare ambiguità nell'interpretazione della procedura descritta, è importante definire un **linguaggio formale** attraverso il quale codificare le istruzioni che si stanno illustrando. I linguaggi formali, inoltre, possono essere di varia natura, sia grafici che testuali.

Sarà possibile, quindi, definire un programma come una procedura espressa in un linguaggio che descriva l'attività necessaria alla realizzazione di qualunque tipo di “prodotto”, senza ambiguità e con un livello di dettaglio tale da essere sempre riproducibile. Questo prodotto può essere indifferentemente una pietanza da cucinare oppure un software applicativo.

Istruzioni

All'interno di una procedura si parla di istruzione quando si identifica una determinata attività da dover svolgere con precisione e senza la quale non è possibile eseguire l'istruzione successiva. Si parla di non ambiguità perché un calcolatore, che in genere è l'esecutore di un software, non è dotato di intelletto e quindi non può interpretare un'istruzione che gli viene fornita. Ad esempio, se si dovesse istruire un calcolatore per realizzare una pietanza, non si dovrebbero usare istruzioni del tipo “un pizzico di sale” oppure “sale quanto basta”, ma si dovrebbe indicare la quantità di sale in modo preciso ed universalmente riconoscibile.

Le istruzioni quindi possono essere considerate come articolazioni più o meno complesse di tre tipi di attività che si identificano nel seguente modo:

- Sequenza
- Selezione
- Iterazione

Tali strutture di controllo si definiranno in seguito.

Formalismo grafico

Per rappresentare un algoritmo risulta utile quindi rispettare delle convenzioni sintattiche così da non lasciare ambiguità di interpretazione nella lettura dello stesso. Uno dei formalismi più diffusi e comodi per la condivisione di piccole procedure è il formalismo grafico, in particolare riguardante i diagrammi di flusso.

Variabili

Una variabile è un contenitore nel quale viene conservata un'informazione. Esattamente come un contenitore fisico, una variabile può contenere un'informazione alla volta sostituendola quando gli si attribuisce un nuovo valore.

Si immagini un bicchiere vuoto, se si decide di conservare dell'acqua lo si riempie con dell'acqua, se si decide successivamente di conservare una bevanda gassata, si sostituirà l'acqua con la bevanda gassata, perdendo il liquido che conteneva.

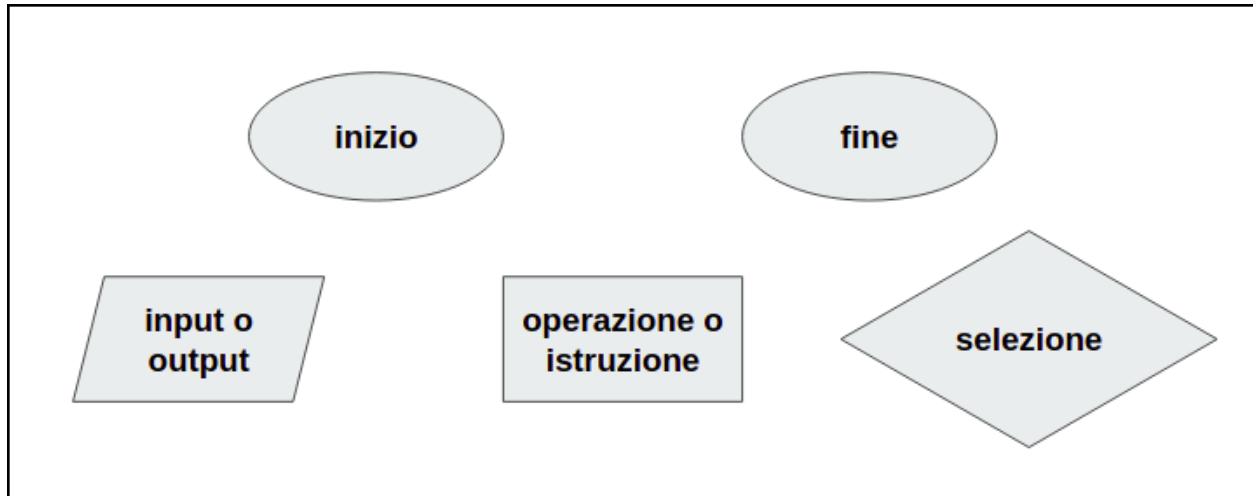
Riassumendo le caratteristiche di una variabile, si avranno due elementi distintivi:

- **NOME.** Deve avere sempre un nome significativo perchè deve far intendere il significato del valore contenuto. Se il risultato di una procedura è l'area del rettangolo, è buona norma conservare il valore della base all'interno di una variabile che si chiama **Base** ed il valore dell'altezza all'interno di una variabile che si chiama **Altezza**. In genere tutti i nomi sono validi anche se bisognerebbe rispettare alcune convenzioni, come ad esempio non iniziare il nome di una variabile con un numero. Alcune regole invece sono vincolanti, non è possibile infatti inserire spazi all'interno di un nome.
- **VALORE.** Ogni variabile deve contenere un valore. Alcuni linguaggi di programmazione prevedono la dichiarazione di variabili all'inizio del programma, anche se non si conosce il valore da inserire al suo interno. In questi casi il valore dichiarato inizialmente è quasi sempre 0.

Altro aspetto fondamentale delle variabili è la caratterizzazione del tipo di valore che possono contenere. Da questo punto di vista possiamo distinguere due tipi di variabili, **semplici** o **strutturate**. Il tipo di dato che la variabile può contenere dipende dal significato che deve assumere e dall'uso che ne deve essere fatto. Se bisogna calcolare l'area di un poligono, ad esempio, le variabili che conterranno i valori della base e dell'altezza saranno necessariamente di tipo intero o al massimo reale se il valore numerico è di tipo decimale. Se invece bisogna elaborare un testo, la variabile che dovrà contenerlo sarà di tipo stringa. Le tipologie di dati ed il loro utilizzo verranno approfondite successivamente e dipendono dal linguaggio di programmazione che comunque condividono la maggior parte dei tipi possibili.

Diagrammi di flusso

Un diagramma di flusso rappresenta, attraverso dei blocchi geometrici, le istruzioni da eseguire in modo sequenziale, dall'alto verso il basso. Inizia sempre con un blocco di tipo "Inizio" e si conclude in un altro blocco ovale di conclusione di tipo "fine".



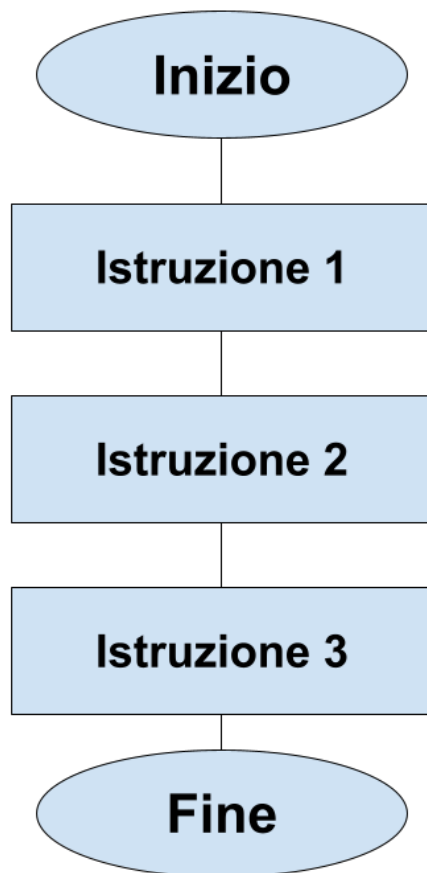
I blocchi di tipo input o output (parallelogrammo), rappresentano le istruzioni che permettono di acquisire un'informazione dall'esterno attraverso un **parametro di input** oppure di restituire verso l'esterno un valore attraverso un **parametro di output**. Un parametro è una variabile utilizzata per lo scambio di valori in ingresso o in uscita. Il blocco rettangolare, invece, permette di rappresentare un'istruzione semplice, come l'assegnazione di un valore ad una variabile, oppure gruppi di istruzioni come si potrà vedere in seguito. Un diagramma di flusso si costruisce dall'alto verso il basso, seguendo un flusso ben determinato, quindi risulta sicuramente errato un diagramma che prevede più "flussi" in uscita da un singolo blocco, mentre risulta possibile avere più flussi in ingresso.

Strutture di controllo

Sequenza

La sequenza non è altro che un insieme di istruzioni poste in un determinato ordine che non potrà essere mescolato. Si noti, quindi, che se vengono indicate le istruzioni A B e C in questo ordine, l'istruzione C non potrà essere eseguita se prima non verrà conclusa l'esecuzione dell'Istruzione B, eseguita comunque dopo la conclusione dell'Istruzione A.

Ogni elemento di una sequenza può essere semplice o complesso, perché si potrebbe racchiudere in un'unica istruzione il frutto di un'articolazione complessa di attività più semplici.



Selezione

La selezione è una struttura di controllo che permette la scelta di un percorso in alternativa ad un altro. Volendo rappresentare un algoritmo attraverso una rappresentazione visiva, potremmo

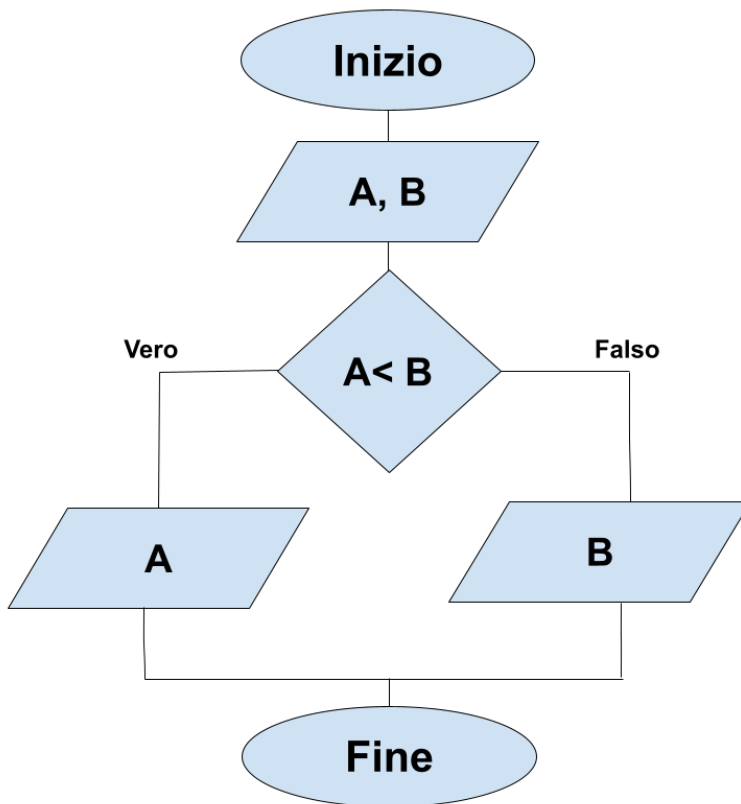
paragonarlo ad un percorso per le biglie. Lasciata cadere una biglia dall'alto, all'interno della pista, può svolgere un unico percorso e se esistono possibili biforcazioni potrà percorrere soltanto una strada, in funzione di eventuali condizioni che di volta in volta possono mutare. È immaginabile quindi che in una struttura di selezione siano presenti più rami, ma l'algoritmo può evolvere solo attraverso uno di questi in funzione di parametri che il programmatore avrà ipotizzato e definito. L'esecutore, infatti, è come una biglia, non è in grado di scegliere una strada da percorrere ma intraprenderà il percorso che in qualche modo gli viene vincolato.

Bisogna quindi caratterizzare una struttura di selezione attraverso una condizione logica che quando si valuta può essere vera o falsa, così da permettere all'esecutore di intraprendere la strada A oppure la strada B. In tal modo è possibile intuire che la parola chiave di una struttura di selezione è **SE** che generalmente viene definita come **IF** mutuando dall'inglese. L'altra parola chiave è **ALLORA (THEN)** attraverso cui è possibile indicare cosa accade se risulta vera la condizione che accompagna il SE. La terza parola chiave di una struttura di selezione è **ALTRIMENTI (ELSE)**, attraverso cui si indica l'alternativa al SE.

In breve il costrutto si può riassumere come IF - THEN - ELSE, secondo l'esempio seguente.

IF (la pasta è insipida) **THEN** metti sale
IF (c'è il sole) **THEN** metti gli occhiali **ELSE** porta l'ombrello

In un diagramma di flusso questa struttura di controllo si identifica con un rombo, all'interno del quale viene descritta la condizione che può assumere solo due valori, *vero* o *falso*. Nell'esempio che segue si può riconoscere il diagramma che identifica una selezione che restituisce in output **il più piccolo tra due numeri dati in input**.



Si noti che:

- Le linee ed i blocchi sono tutte dall'altro verso il basso.
- L'utente inserisce attraverso il parallelogrammo due valori di input che verranno associati rispettivamente alla variabile A ed alla variabile B.
- Il blocco selezione permette di verificare se A è minore di B, restituendo solo e soltanto una risposta possibile: vero o falso. In funzione di questa risposta l'algoritmo sceglie una strada oppure un'altra.
- I parallelogrammi di output restituiscono all'utente A o B in funzione della strada che il blocco selezione ha validato.

Iterazione

Una struttura iterativa, detta anche ciclo, permette di rappresentare operazioni che devono essere ripetute più di una volta, sempre nella stessa sequenza. In ogni tipo di rappresentazione delle procedure, non ha senso ripetere più volte lo stesso blocco che rappresenta un'istruzione. Se bisogna suggerire ad un amico di lanciare una palla tre volte non ha senso dire: "lancia la palla, lancia la palla, lancia la palla", verrà detto invece "lancia la palla 3 volte". Questo tipo di

struttura, quindi, permette di rappresentare ripetizioni di sequenze di istruzioni, ottimizzando spazio e tempo.

Le ripetizioni di istruzioni possono essere di varia natura, principalmente è possibile definire due tipologie: ciclo di **FOR** e ciclo di **WHILE**.

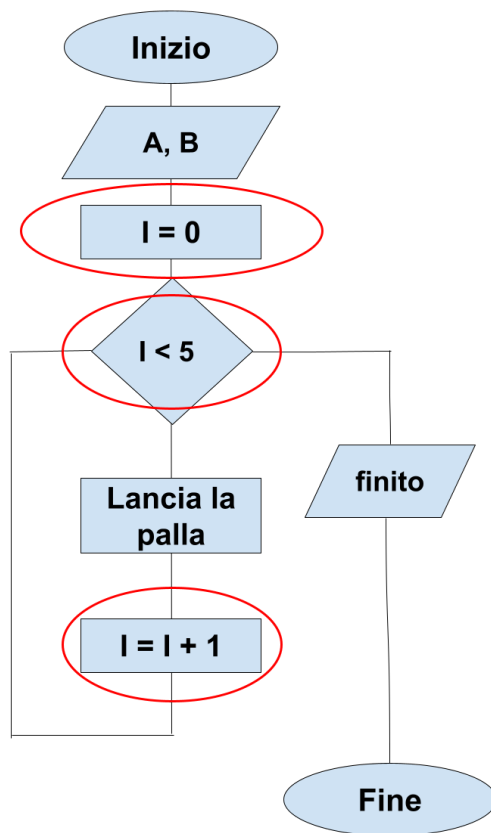
Ciclo di For

Un'istruzione che deve essere eseguita un determinato numero di volte, come nell'esempio descritto in precedenza. Questo tipo di ciclo in genere viene rappresentato nella maggior parte dei linguaggi di programmazione con la parola chiave **FOR**.

Per ripetere sei volte una operazione attraverso un ciclo di FOR, la sintassi potrebbe essere questa:

```
FOR ( i = 0; i<6; i=i+1) { operazioni da ripetere }
```

La rappresentazione in diagramma di flusso di un ciclo che realizza 4 volte l'operazione "lancia la palla" è la seguente:



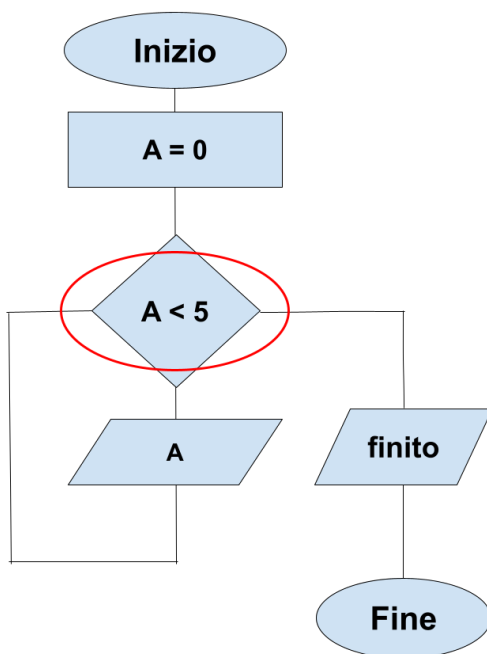
Si noti che i tre elementi evidenziati sono gli elementi alla base del ciclo:

- **Punto di inizio**, generalmente una variabile inizializzata a zero conserva il numero di esecuzioni realizzate.
- **Criterio di arresto**, la condizione da verificare affinché il ciclo possa continuare o si deve interrompere.
- **Passo di incremento**, l'istruzione che permette alla variabile di incrementare il valore contenuto in I, così da raggiungere la fine del ciclo. Senza questa istruzione il ciclo non arriverebbe mai a termine.

Ciclo di While

Un'istruzione che deve ripetersi **FINCHÈ** non si verifica una condizione. Questo tipo di ciclo viene rappresentato solitamente con la parola chiave **WHILE**. La differenza sostanziale con il ciclo di For è che questo ciclo non ha un numero determinato di ripetizione, ma il numero di iterazioni potenzialmente può cambiare ad ogni esecuzione dell'algoritmo.

Il diagramma di flusso descritto di seguito realizza un ciclo che si ripete fino a che l'utente non inserisce un valore maggiore o uguale a 5.

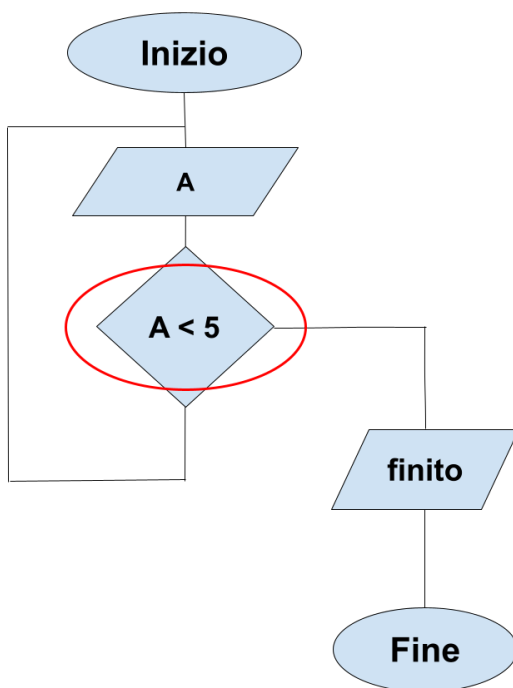


Si noti che un ciclo di `while` non ha due degli elementi fondamentali del ciclo di `for`: il punto di inizio e l'incremento, ma conserva esclusivamente **il criterio di arresto**.

Ciclo Do-While

Le strutture cicliche presentate nei paragrafi precedenti vengono definite **pre-condizione**, in quanto la condizione che verifica l'arresto del ciclo viene eseguita prima del blocco di istruzioni che deve essere ripetuto. Nel ciclo dell'esempio precedente, infatti, se la condizione del criterio di arresto non è soddisfatta, il blocco di input non verrà mai eseguito. Si provi, ad esempio, a modificare il primo blocco con `A = 6` e si scoprirà che l'utente non potrà inserire in alcun modo un valore da sottoporre a controllo.

Un ciclo **Post-condizione** invece, permette sempre di eseguire almeno una volta il blocco di istruzioni al suo interno, perchè la condizione è posta in seguito alle istruzioni da ripetere. Una tale struttura viene realizzata attraverso il costrutto che prende il nome di **do-while**. Si osservi il seguente esempio.



Si noti che:

- A differenza dell'esempio realizzato con un ciclo di While, non è necessario inizializzare la variabile A ad un valore compatibile alla condizione del criterio di arresto.
- L'utente inserirà il valore almeno una volta prima di verificare la condizione del ciclo.
- Questo algoritmo farà ripetere l'inserimento di un valore di input fino a che l'utente non scriverà un numero maggiore o uguale di 5.

Strutture dati

Le istruzioni di un programma in generale agiscono sui dati che hanno la stessa importanza delle istruzioni, quando si ragiona in modo procedurale. Un dato è un'informazione che viene elaborata oppure valutata a seconda dei casi.

Se si vuole ipotizzare la procedura che gestisce un distributore automatico di bibite, i dati su cui lavora sono relativi alle informazioni dei prodotti che deve distribuire, dei codici per la loro selezione e del relativo costo. A volte le informazioni sono invece dati da valutare, ad esempio una soglia da superare o non superare, come potrebbe essere un voto oltre il quale aver raggiunto la sufficienza o al di sotto del quale non averla raggiunta. È intuibile quindi come il modo di conservare e manipolare queste informazioni risulti estremamente importante per il buon funzionamento di una procedura, per questo motivo bisogna introdurre il concetto di **variabile**. Si tenga presente inoltre che un dato deve rappresentare sempre un'informazione, perché se non ha un significato riconoscibile perde di senso.

Liste e matrici

Quando si presenta la necessità di conservare più informazioni all'interno di una stessa variabile allora si è in presenza di variabili di tipo strutturato. Ad esempio se bisogna utilizzare un contenitore per conservare i nomi dei mesi dell'anno, non basterà più una variabile semplice, che potrà contenerne al massimo uno, ma si dovrà considerare una variabile che ad un unico nome associa 12 elementi. Una variabile strutturata di questo tipo prende il nome di **lista** (o array) che permette di identificare con un nome una struttura fatta da più elementi, tutti identificabili attraverso un indice ordinato. Per l'esempio dei mesi, quindi, alla posizione 3 (mese[3]) corrisponderà Marzo, alla posizione 1 corrisponderà Gennaio e così via. Se una variabile è stata paragonata ad un bicchiere, si potrebbe identificare la lista come un vassoio all'interno del quale sono allineati su una riga un numero di bicchieri pari al numero di elementi appartenenti alla lista. In un linguaggio come Kojo, questo tipo di strutture dati prende il nome di collezione, ma ogni linguaggio di programmazione può associargli anche altri tipi di nomi.

Procedure e funzioni

È stato introdotto il concetto di procedura facendo riferimento ad una sequenza di istruzioni volte alla risoluzione di un compito. In realtà un compito complesso può essere scomposto in più sotto problemi, si Immagini ad esempio lo chef in una cucina. Non sarà lui a provvedere alle preparazioni di tutti gli elementi che gli servono per cucinare, ma delegherà ai suoi aiutanti parte dei preparati che gli serviranno per comporre il suo piatto. Ogni aiutante, quindi, realizzerà una procedura autonomamente e restituirà il prodotto ottenuto allo chef in modo tale che si possa completare il prodotto finale.

Per quanto detto si può dire che la procedura principale potrebbe non utilizzare semplici istruzioni, ma identificare istruzioni più complesse all'interno delle quali sono previste procedure autonome, magari da richiamare più volte in più punti della procedura principale. A questo punto, considerando la possibilità che più procedure possano cooperare, risulta utile distinguere il programma **chiamante** dal programma **chiamato**. Il primo invoca il secondo e attenderà la sua conclusione, appena il programma chiamato terminerà la sua esecuzione restituirà il controllo al programma chiamante. Volendo continuare l'esempio precedente, si potrebbe dedurre che appena il collaboratore dello chef avrà terminato le sue operazioni darà la possibilità allo chef di continuare la sua procedura.

Un approccio alla programmazione di questo tipo si chiama **approccio modulare**, perché permette di dividere le attività in moduli, così da poterle richiamare ogni qualvolta le si ritiene utili. Talvolta però queste attività hanno bisogno di avere valori da elaborare, ad esempio l'aiuto cuoco ha bisogno delle uova per preparare una frittata, oppure una funzione geometrica ha bisogno dei valori della base e dell'altezza per ricavare l'area del poligono. I valori che vengono passati ad una procedura per poter funzionare si chiamano **parametri** o **argomenti** della procedura.

I parametri di input sono quei valori che servono alla procedura per evolvere, senza i quali non potrà ottenere alcun risultato. In alcuni casi, però, i parametri possono essere anche di output, perché la procedura ha bisogno di restituire un valore al programma chiamante. In questo caso la procedura prende il nome di **funzione**.

In Kojo, per far comprendere al calcolatore che vogliamo definire una procedura o una funzione bisogna utilizzare la parola chiave **def**, ma ogni linguaggio programmazione ha la propria sintassi per definire una nuova procedura. Si tenga presente che una funzione può arbitrariamente avere o non avere sia parametri di input che di output, in funzione delle necessità.

Algoritmo

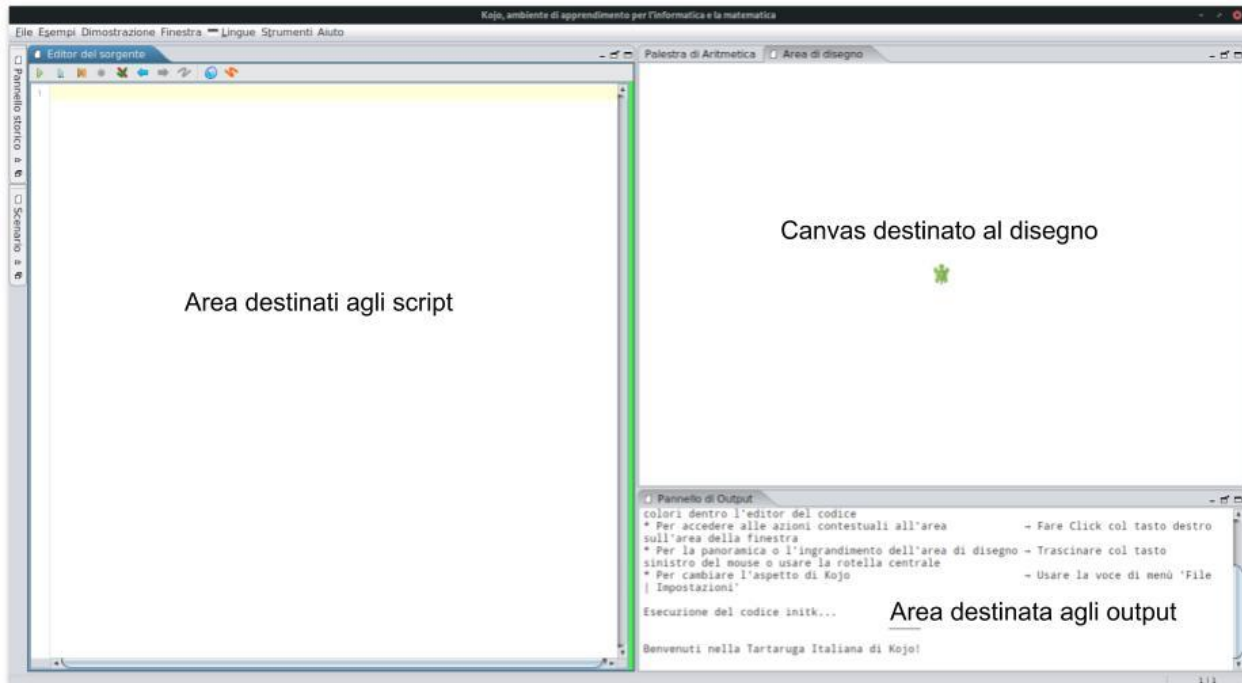
Sono stati definiti nei paragrafi precedenti gli elementi formali, strutture di controllo, procedure e funzioni, per descrivere in modo efficace un'attività senza ambiguità e con un livello sufficiente di precisione. Questi elementi **sono gli strumenti cognitivi del pensiero computazionale** attraverso il quale è possibile impiegarli in ogni ambito risulti utile applicare un ragionamento logico avente come obiettivo la soluzione di un problema.

Gli esempi citati, infatti, non riguardavano esclusivamente problemi di matematica o di logica, ma problemi riguardante tutti gli ambiti, per tale motivo non si deve ritenere questo tipo di attività come esclusiva pertinenza dell'informatica.

Sarà possibile quindi introdurre il concetto di algoritmo, cioè **una particolare sequenza di azioni, controlli e procedure con la quale si esprime e modella la soluzione di un determinato problema, qualunque esso sia.**

Kojo

Kojo è un'applicazione che permette di avvicinarsi alla sintassi legata alla programmazione di un calcolatore attraverso elementi molto semplici. Scaricata l'applicazione dal web si noterà che questa prevede più aree, in particolare una sulla sinistra all'interno della quale si inserisce il codice ed una sulla destra all'interno della quale si vede il risultato dell'esecuzione di quel codice.

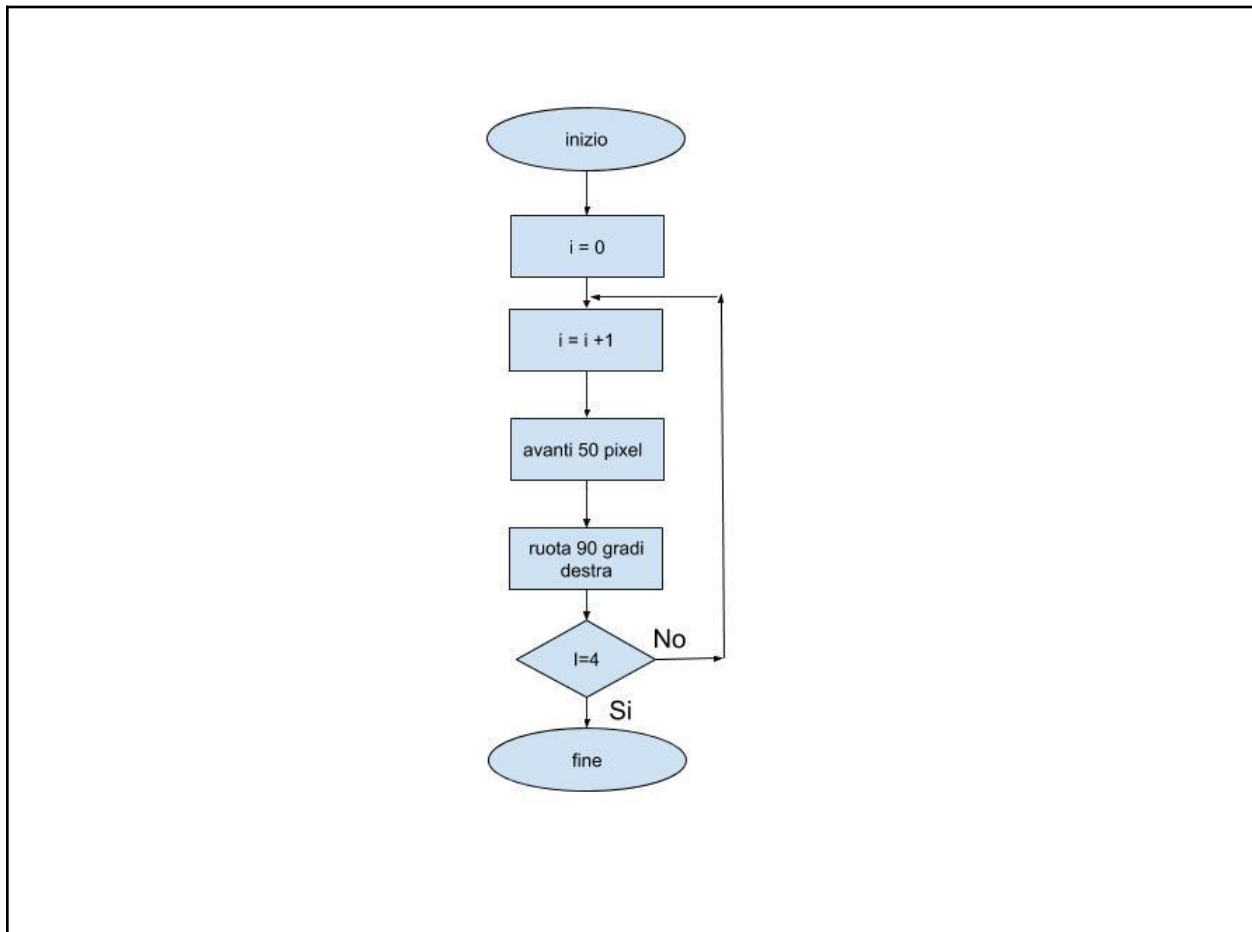


Attraverso il menù principale è possibile accedere a molti esempi, divisi per gradi di difficoltà, verranno trattati in questo testo solo gli esempi cardine per capirne il funzionamento logico.

Si immagini quindi di disegnare un quadrato, volendo descrivere tale procedura in un linguaggio informale si potrebbe scrivere:

- vai avanti
- gira a destra
- vai avanti
- gira a destra
- vai avanti
- gira a destra
- vai avanti
- gira a destra

Proprio perché è stato utilizzato un linguaggio informale, questo si presta a molte ambiguità. Non è stato infatti indicata un'unità di misura per comprendere quanto andare avanti e neppure un angolo di rotazione nel girare a destra. Volendo utilizzare un diagramma di flusso per rappresentare in modo più preciso il disegno di un quadrato, anche attraverso l'utilizzo di cicli, si dovrebbe rappresentare questo algoritmo nel modo che segue.

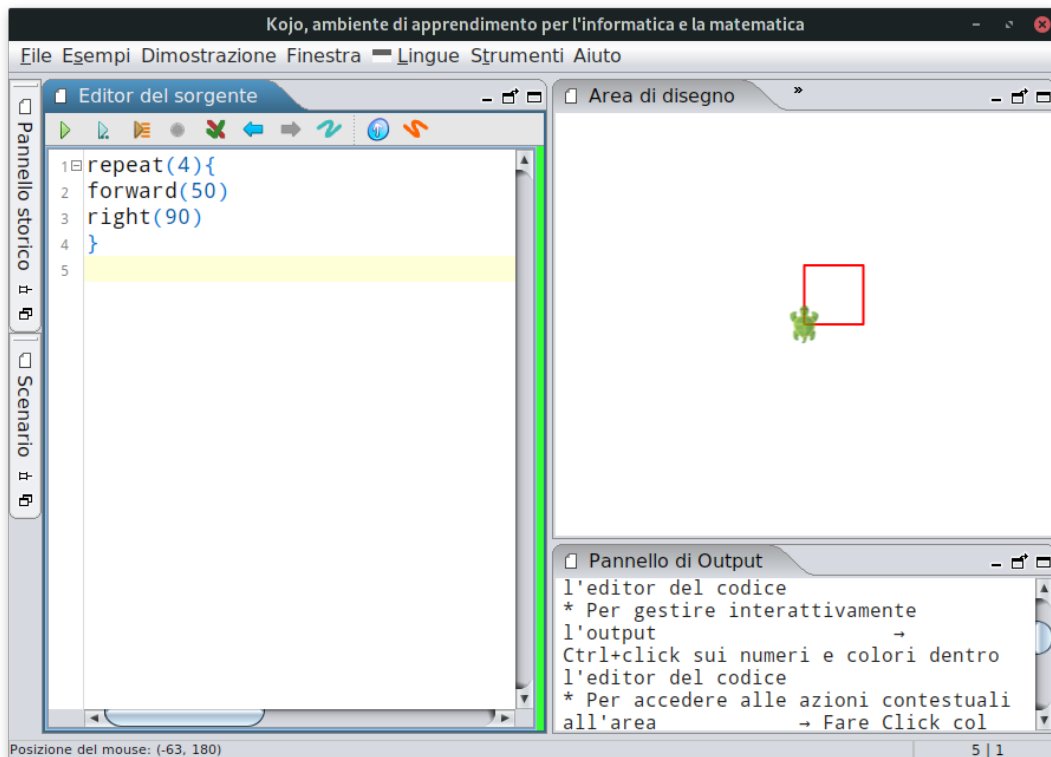


Si noti che in questo caso la struttura ciclica pone la sua condizione di verifica dopo aver svolto almeno una volta le operazioni eventualmente da ripetere. In Kojo, il diagramma di flusso precedentemente descritto, può essere codificato nel seguente modo:

```
repeat(4) {  
  forward(50)  
  right(90)  
}
```

Si noti che le parentesi graffe identificano un blocco, in questo caso il blocco di istruzioni deve essere ripetuto 4 volte, valore indicato come parametro all'interno delle parentesi tonde relative all'istruzione **repeat**.

A questo punto la tartaruga, all'interno dell'area di disegno, realizzerà un quadrato.



Variabili e Valori

Per utilizzare al meglio i dati risulta utile comprendere bene il funzionamento di **variabili** e **valori**.

Si considerino le istruzioni seguenti:

```
val raggio = 10 // valore  
  
var diametro = 12 // variabile
```

- **raggio** è preceduto dalla parola chiave **val**, questo vuol dire che raggio è un **valore** pari a al numero intero 12. Un valore è **IMMUTABILE**, significa che il *raggio* non potrà essere modificato mai in nessun punto del programma.
- **diámetro** è preceduto dalla parola chiave **var**, questo vuol dire che diametro è una **variabile** che contiene il valore 12. Una variabile è un oggetto **MUTABILE**, significa che in qualunque momento può essere sostituito 12 con un altro valore.

I tipi di dati che possono essere associati a *variabili* e *valori* sono di varia natura. I principali tipi di dati che vengono trattati sono i seguenti:

- **booleani**, possono assumere solo valori di tipo *true* o *false*, caratterizzato dalla parola chiave **Boolean**.
- **interi**, numeri interi caratterizzati dalla parola chiave **Int**.
- **reali**, numeri reali caratterizzati dalla parola chiave **Float**.
- **carattere**, singolo carattere alfanumerico caratterizzato dalla parola chiave **Char**. Si utilizzano le virgolette per identificare un carattere.
- **stringa**, sequenza di caratteri caratterizzata dalla parola chiave **String**. È possibile accedere ai singoli caratteri di una stringa attraverso le parentesi tonde.

```
var parola = "casa"
```

`casa(2)` rappresenta il carattere "s", la numerazione dei caratteri parte sempre da 0

Si utilizzano le virgolette per identificare una stringa. Senza l'uso delle virgolette Kojo interpreta quella parola come una Variabile o un Valore. Ad esempio l'istruzione:

```
val parola = "casa"
```

al valore *parola* viene associata la stringa "casa". se invece l'istruzione è:

```
val parola = casa
```

al valore *parola* viene associato il contenuto della variabile *casa*.

Si noti che le variabili possono essere di tipo semplice, come quelle appena elencate e di tipo strutturato. Sinteticamente si potrebbe definire come semplice una variabile che può contenere un valore per volta, strutturata se invece può contenere più valori contemporaneamente.

Nell'esempio che segue si dichiara una variabile **sequenza** al cui interno vengono conservati tutti i valori compresi tra 1 e 100.

```
val sequenza = 1 to 100
```

Selezione

In Kojo è possibile definire anche strutture di selezione, che permettono di cambiare il flusso del programma in funzione di una o più condizioni. Per implementare quindi le strutture di controllo presentate nel capitolo precedente si osservi il seguente esercizio.

struttura-selezione.kojo

```
clear()

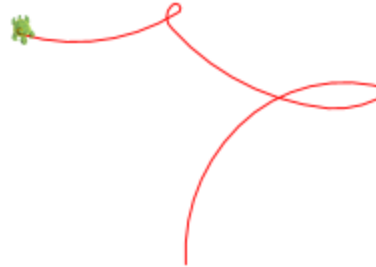
repeat (9){
  var condizione = random (1,30)

  if (condizione <= 10){

    repeat (4) {
      forward(condizione)
      right(180/condizione)
    }
  }else if(condizione <= 20){

    repeat (3) {
      forward(condizione)
      right(180/condizione)
    }
  }else{

    repeat (7) {
      forward(condizione)
      right(180/condizione)
    }
  }
}
```



Si noti quindi che:

- Viene generata una variabile in modo casuale, in funzione della quale viene generato un codice diverso.
- Ci sono tre blocchi selezione, la prima condizione che il codice verifica è **SE** il valore generato è minore di 10. Se questa condizione risulta vera svolge le istruzioni interne alla graffa e poi salterà alle istruzioni esterne alla struttura di selezione, quindi successive al blocco **ELSE**. Se fallisce la prima condizione passa al blocco successivo, identificato da **ELSE IF** (“altrimenti se”), tenendo presente che è possibile inserire un qualsiasi numero di blocchi **else if**.

In sostanza quindi una struttura selezione è identificata da almeno un blocco **IF**. Si può eventualmente ipotizzare il controllo di condizioni diverse dalla prima, attraverso l'uso di **ELSE IF**, oppure una condizione generica **ELSE**, che verrà valutata dall'interprete solo se tutte le precedenti falliscono.

Si ribadisce inoltre che i blocchi risultano mutuamente esclusivi, quindi al blocco **ELSE IF** si arriva solo e soltanto se fallisce il primo IF, così come al blocco **ELSE** si arriva solo se falliscono tutti i precedenti.

Risulta importante sottolineare che:

- Una struttura completa di selezione prevede un solo blocco **IF**, eventualmente più blocchi **else if** ed un solo blocco **else**.
- Se in una struttura di selezione metto in sequenza più blocchi **IF** (non else if) questi verranno verificati in sequenza, perchè risultano selezioni autonome una dall'altra.
- Risulta possibile innestare un blocco **if** dentro l'altro, il risultato sarà simile ad una struttura **Else if**, ma molto meno elegante.

Operazioni ripetute

Sono state introdotte le iterazioni, che nei linguaggi di programmazione prendono il nome di ciclo. Un primo approccio ai cicli in Kojo è rappresentato dalla parola **repeat(n)** dove **n** è il numero di volte in cui bisogna ripetere le istruzioni incluse nel blocco delle graffe. Nell'esempio che segue si realizza la stessa iterazione sfruttando due approcci differenti.

- Il primo metodo sfrutta l'istruzione **repeat**,
- Il secondo metodo utilizza la parola chiave **for** che, in combinazione con una variabile strutturata, permette di iterare intervalli con il primo estremo diverso da 0. Nell'esempio che segue l'iterazione parte da 5 e finisce ad 8 realizzando comunque 4 ripetizioni delle istruzioni indicate.

```
// PRIMO METODO

repeat(4) {
  avanti(100)
  destra(90)
}

// SECONDO METODO
```

```
clear()
val sequenza = 5 to 8

for ( numero <- sequenza) {
    avanti(100)
    destra(90)
}
```

Definire una funzione

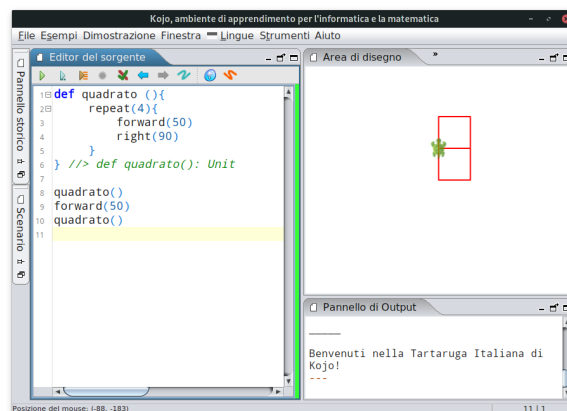
Se volessimo utilizzare le istruzioni precedenti più volte, così da disegnare più quadrati, risulta opportuno inserire quelle istruzioni all'interno di una funzione chiamata *quadrato*. Per definire tale funzione bisogna utilizzare l'identificativo **def**, seguito dal nome della funzione e dalle parentesi graffe all'interno delle quali inserire tutti i blocchi relativi a questa funzione.

È buona norma aggiungere degli spazi all'inizio prima delle istruzioni appartenenti allo stesso blocco, questo migliora la leggibilità del codice, un esempio è rappresentato nell'esercizio che segue. In Kojo, tuttavia, non è errore mettere tutte le istruzioni allineate a sinistra, come accade in altri linguaggi programmazione, ad esempio Python.

Nel codice seguente è definita la funzione *quadrato* ed è richiamata due volte all'interno dello script, infatti se si definisce la funzione e non la si richiama all'interno dell'area di scrittura del codice, questa rimarrà definita ma non chiamata. Si potrebbe quindi definire l'area di lavoro all'interno della quale si inserisce il codice come l'area all'interno della quale si deve inserire il codice del programma **principale** chiamante, prima del quale vanno inserite tutte le funzioni eventualmente da chiamare.

```
def quadrato () {
    repeat (4) {
        forward(50)
        right(90)
    }
}

quadrato()
forward(50)
quadrato()
```



Per identificare un gruppo di istruzioni raggruppabili in procedura oppure in semplici cicli, si consiglia di provare a scrivere il codice in modo strettamente sequenziale ed identificare gruppi identici di istruzioni. Si osservi il codice nell'esempio che segue:

```
clear()
setSpeed(slow)
setPenThickness(4)
setPenColor(red)
```

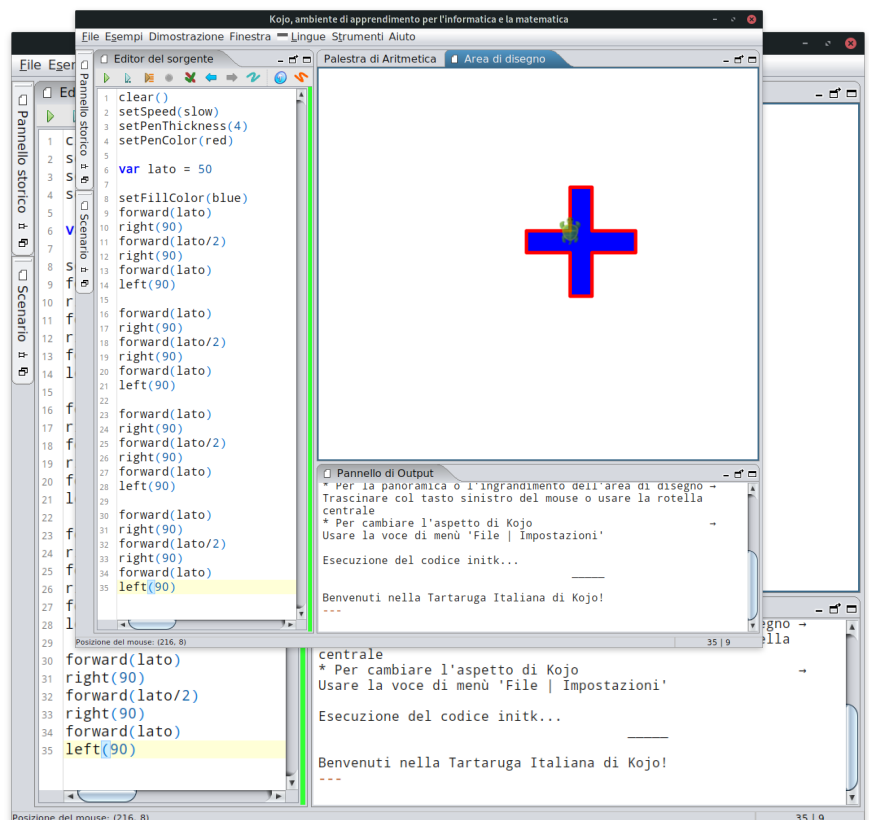
```
var lato = 50
```

```
setFillColor(blue)
forward(lato)
right(90)
forward(lato/2)
right(90)
forward(lato)
left(90)
```

```
forward(lato)
right(90)
forward(lato/2)
right(90)
forward(lato)
left(90)
```

```
forward(lato)
right(90)
forward(lato/2)
right(90)
forward(lato)
left(90)
```

```
forward(lato)
right(90)
forward(lato/2)
right(90)
forward(lato)
left(90)
```



Nel codice descritto risulta evidente come ci sia un gruppo di istruzioni che si ripete più volte, perchè ogni “ramo” della figura è uguale, ma ruotato. Questo tipo di codice, per semplificare la lettura ed ottimizzare il codice, rispettando un approccio modulare potrà essere scritto in due modi diversi. In un esercizio come quello precedente il consiglio è creare un ciclo con il blocco di istruzioni che si ripetono, successivamente definire una funzione che realizzi quello stesso blocco di istruzioni. Funzione che verrà richiamata nel ciclo della funzione principale.

raggruppando in ciclo	raggruppando in una funzione eseguita in un ciclo
<pre>clear() setSpeed(slow) setPenThickness(4) setPenColor(red) setFillColor(blue) var lato = 50 repeat(4){ forward(lato) right(90) forward(lato/2) right(90) forward(lato) left(90) }</pre>	<pre>def ramo(){ forward(lato) right(90) forward(lato/2) right(90) forward(lato) left(90) } clear() setSpeed(slow) setPenThickness(4) setPenColor(red) setFillColor(blue) var lato = 50 repeat(4){ ramo }</pre>

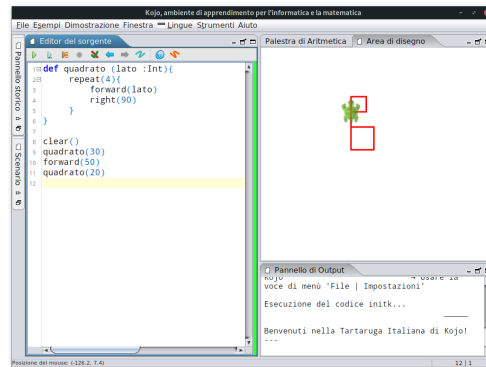
Parametri di funzioni

È stato introdotto dal punto di vista teorico il concetto di parametro, è utile quindi utilizzarlo per definire meglio la funzione quadrato. Può risultare utile infatti personalizzare la lunghezza del lato

del quadrato senza dover riscrivere un'altra funzione quadrato con un lato diverso. Per fare questo utilizzeremo un parametro di input da passare alla funzione appena creata all'interno delle parentesi tonde.

```
def quadrato (lato :Int){
    repeat (4) {
        forward(lato)
        right(90)
    }
}

clear()
quadrato(30)
forward(50)
quadrato(20)
```



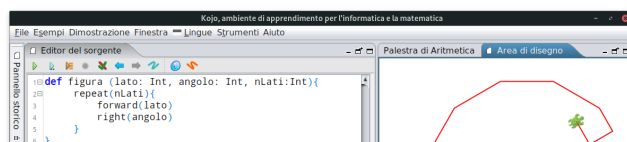
Si noti, nel codice precedente, che all'interno delle parentesi tonde non solo è indicato il nome del parametro (*lato*), ma è stato anche indicato il tipo di valore da utilizzare, in questo caso un tipo numerico intero (*Int*).

Nel programma chiamante, la funzione *quadrato* viene richiamata passando il valore numerico direttamente all'interno delle parentesi tonde, sarà *Kojo* ad interpretare quel valore come numero intero da attribuire al parametro di nome *lato*.

È possibile intuire come sia utile talvolta passare alla funzione chiamata più parametri, basterà nella sua definizione elencare i parametri di input, separati da una virgola, all'interno delle parentesi tonde. Quando verrà chiamata la funzione, ovviamente, i valori dovranno essere passati nello stesso ordine con cui sono stati dichiarati.

Nel seguente esempio si definisce una funzione **figura**, che prende in input tre parametri, il primo che rappresenta la lunghezza del lato, il secondo la rotazione dell'angolo ed il terzo il numero di lati da realizzare. Nella funzione chiamante quindi ogni qualvolta viene richiamata la funzione in *figura*, si dovrà inserire tre valori numerici di tipo intero che *Kojo* provvederà a collocare nei rispettivi parametri di input, rispettando l'ordine di scrittura.

```
def figura (lato: Int, angolo: Int, nLati: Int){
    repeat (nLati) {
        forward(lato)
        right(angolo)
    }
}
```



```
clear()
figura(100,30,5)
figura(50,90,3)
```

Se invece la funzione da realizzare deve restituire un valore al programma chiamante allora è possibile definire valori di output come nel caso che segue.

```
def divisibile(num: Int, div: Int): Boolean = {
    // % è la divisione in modulo: restituisce il resto della
    // divisione tra num e div
    (num % div) == 0
}

def verificaDiv(da: Int, a: Int)
{
    for (numero <- da to a){
        val resto = divisibile (numero, 2)
        scriviLinea(resto)
    }
}

verificaDiv(10, 100)
```

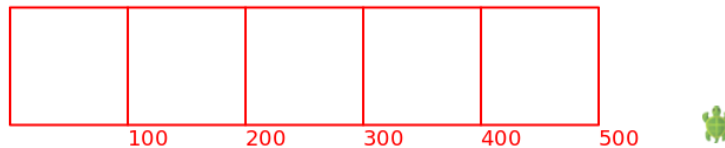
Nel seguente esempio invece si ripete un ciclo che realizza un quadrato ma sfruttando al meglio le caratteristiche del ciclo di for in luogo del più semplice repeat.

Si noti che:

- La funzione *divisibile* prende in input due valori interi e restituisce in output un valore di tipo *booleano* (*true* o *false*).
- La divisione in modulo restituisce il resto della divisione, quindi l'istruzione "*(num % div) == 0*" sarà *true* se *num* è divisibile per *div*, altrimenti restituirà *false*.
- La funzione *verificaDiv* prende in input due valori che vengono dati in pasto al ciclo di for per far variare "*numero*" dal valore contenuto in "*da*" fino al valore contenuto in "*a*". Ad

ogni iterazione “*numero*” viene passato alla funzione divisibile per verificare la divisibilità per due.

```
def quadrato(lato: Int) {  
    repeat(4){  
        forward(lato)  
        sinistra(90)  
    }  
} //> def quadrato(lato: Int): Unit  
  
def striscia(da: Int, a: Int)  
{  
    for (numero <- da to a){  
        quadrato(100)  
        var pos = numero * 100  
        scrivi(pos)  
        forward (30)  
        setPosition(pos, 0)  
    }  
} //> def striscia(da: Int, a: Int):  
Unit  
  
clear()  
  
striscia(1, 5)
```



Si noti che:

- La funzione **quadrato** prende in input il lato del quadrato da disegnare.
- La funzione **striscia** prende in input il punto di inizio del ciclo ed il termine ultimo. La variabile `numero` contiene il valore che itera all'interno del ciclo, quindi varia tra il valore “da” ed il valore “a”.
- All'interno del ciclo interno alla funzione **striscia**, la variabile **pos** viene aggiornata ad ogni iterazione, così da riposizionare il quadrato da disegnare.

Istruzioni basilari

Ogni linguaggio di programmazione prevede una serie di funzioni basilari, integrate nel sistema, che permettono di realizzare operazioni più articolate ma di uso comune. In Kojo esistono diversi approcci alla programmazione: **Turtle Graphics**, **Picture Graphics**, **Gaming**, **Robotics**. Tali approcci hanno una progressiva difficoltà nella realizzazione di codice, in questo testo verrà trattato il gruppo di istruzioni associate al primo livello: **Turtle Graphics**.

È possibile riassumere le tipologie di istruzioni secondo la tabella seguente:

Gruppo	descrizione
position	La posizione della tartaruga nell'area di lavoro. Tale punto nel piano viene identificato con una coppia, x e y, a partire dall'angolo in basso a sinistra.
heading	La direzione che punta la testa della tartaruga che può assumere un orientamento a 0°, 90°, 180° e 270°, andando in senso antiorario a partire dalla direzione che guarda il lato sinistro dello schermo.
pen up/down	Se la penna è "down" vorrà dire che sta scrivendo, altrimenti non lascia segni sul piano di lavoro.
pen color	Definisce il colore della penna sul foglio di lavoro
pen thickness	Il tipo di linea disegnata nell'area di lavoro
fill color	Il colore dell'area racchiusa dalla linea nell'area di lavoro.

Per quanto riguarda le possibili istruzioni si faccia riferimento alla documentazione on line di Kojo: <https://docs.kogics.net/reference/turtle.html>

Gestione dei colori

Per quanto riguarda i colori, è bene sottolineare che vengono gestiti con codifica esadecimale, vuol dire che ogni colore è associato ad una combinazione di colori **Red Green Blue**, dove ognuno di questi elementi è un numero compreso tra 0 e 255, in forma esadecimale.

Si ricorda che una codifica esadecimale prevede l'uso di 16 simboli: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, A, B, C, D, E, F, come si potrà dedurre la A corrisponde ad 11 e così via fino a F che rappresenta 15. Ad esempio, se si vuole rappresentare il colore bianco dobbiamo associare 255 a tutti e tre i colori quindi: 255, 255, 255 che in esadecimale diventa: #FFFFFF. Il cancelletto permette sistema di comprendere la codifica.

L'interpretazione della coppia avviene nel seguente modo:

$$FF = (15 * 16) + 15 \rightarrow 255$$

$$C5 = (12 * 16) + 5 \rightarrow 197$$

$$00 = (0 * 16) + 0 \rightarrow 0$$

$$FFC500 = \text{RGB}(255, 197, 0)$$

FFC500

I colori possono essere anche identificati dalla nomenclatura inglese, compreso le sfumature (ad esempio *darkMagenta*), ma in rete sarà possibile trovare convertitori automatici per rintracciare il colore scelto.

per quanto riguarda tutti i possibili utilizzi riguardo i colori si faccia riferimento alla pagina di documentazione dedicata <https://docs.kogics.net/concepts/colors.html>.

Programmazione concorrente

Per introdurre il concetto di programmazione concorrente bisogna soffermarsi sulla definizione di processo e di thread.

Un **processo** è un programma in esecuzione, quindi un qualunque software che viene avviato è un processo che evolve e che viene dato in pasto all'esecutore. L'esecuzione del browser, ad esempio, è un processo che impegnerà una determinata quantità di risorse di sistema. Nei fatti per evolvere un browser ha bisogno di memoria per conservare i dati relativi alla navigazione e di tempo di utilizzo del processore per eseguire tutte le funzioni necessarie alla navigazione. Molte di queste funzioni però potrebbero essere avviate non in una stretta sequenza, ma in concorrenza tra loro. Si pensi alle varie schede del browser all'interno delle quali vengono aperte pagine indipendenti tra loro. È intuibile, quindi, che il processo relativo al browser può essere visto come un contenitore all'interno del quale evolvono in modo autonomo diversi elementi, ciascuno dei quali relativo ad una singola scheda. Questi elementi (che in realtà sono degli algoritmi) vengono definiti **thread**, ognuno con una propria vita che inizia e finisce ogni qualvolta viene aperta o chiusa una scheda. Un programma al cui interno vengono eseguiti più 3D in contemporanea viene detto **multithread**.

Secondo la definizione appena data, ogni qualvolta si costruisce un programma si realizza un processo, che può essere monolitico oppure diviso in più sotto-algoritmi che evolvono in parallelo tra loro. Questo tipo di approccio permette al programmatore di scegliere, a secondo

del programma che si deve realizzare, se costruire un algoritmo unico oppure di farlo evolvere con più strade parallele decidendo l'ordine con cui dovranno essere eseguiti ed una eventuale relazione tra loro. Se i thread sono indipendenti, invece, l'onere di scegliere l'ordine con cui eseguirli può essere lasciato al sistema operativo all'interno del quale il programma evolverà.

In Kojo, è possibile generare più sotto algoritmi e realizzare un software multithread, semplicemente attivando più tartarughe. Nell'esempio estrapolato da quelli inclusi nell'applicazione, sono presenti più tartarughe che in parallelo realizzano una figura autonomamente.

```
// In this program, we're trying to make a synchronized drawing with
multiple turtles
clear()

// a new command to make squares
// t - the turtle that draws the square
// n - the size of the square
// delay - the turtle's animation delay; this controls the
synchronization effect
// we use runInBackground below to make the turtles run together
def square(t: Turtle, n: Int, delay: Int) = runInBackground {
  t.setAnimationDelay(delay)
  repeat(4) {
    t.forward(n)
    t.right()
  }
}

val t1 = newTurtle(0, 0)
val t2 = newTurtle(-200, 100)
val t3 = newTurtle(250, 100)
val t4 = newTurtle(250, -50)
val t5 = newTurtle(-200, -50)

square(t1, 100, 100)
square(t2, 50, 200)
square(t3, 50, 200)
```

```
square(t4, 50, 200)
square(t5, 50, 200)
```

Ricorsione e frattali

La ricorsione è una tecnica di programmazione che permette ad una funzione di richiamare se stessa durante l'esecuzione. Tale tecnica, pur essendo molto efficace in diverse occasioni, non sempre è di facile comprensione e soprattutto può nascondere errori che potrebbero portare al blocco del sistema. È necessario quindi usarla con cautela, cercando sempre di comprendere se non esista una strada migliore e più agevole per la realizzazione di un algoritmo.

Un elemento cardine della ricorsione è l'esistenza di una **condizione** che ferma il processo di ricorsione, quindi interrompe il richiamo a se stessa durante l'evoluzione e risalire la catena di richiami. Una funzione ricorsiva, infatti, risulta essere **chiamante** e **chiamata** contemporaneamente.

Si analizzi il seguente esempio, estratto dagli esempi inclusi in Kojo, per comprendere il concetto di ricorsione.

```
def tree(distance: Double) {
  if (distance > 4) {
    setPenThickness(distance/7)
                                setPenColor(cm.rgb(distance.toInt,
math.abs(255-distance*3).toInt, 125))
    forward(distance)
    right(25)
    tree(distance*0.8-2)
    left(45)
    tree(distance-10)
    right(20)
    forward(-distance)
  }
}
```

```
clear()
setSpeed(fast)
hop(-200)
tree(90)
```

Nella prima parte del codice viene definita la funzione **tree** che per essere richiamata ha bisogno di avere in input un valore reale (Double) associato alla variabile *distance*.

La prima istruzione di questa funzione è la condizione di arresto della ricorsione

```
if (distance > 4) {
```

In tal modo nella catena di richiami nel momento in cui *distance* diventerà un valore minore uguale a 4 la funzione *tree* fermerà la catena ricorsiva in quanto, osservando il codice, non esiste nessun'altra istruzione al di fuori di questo *if*.

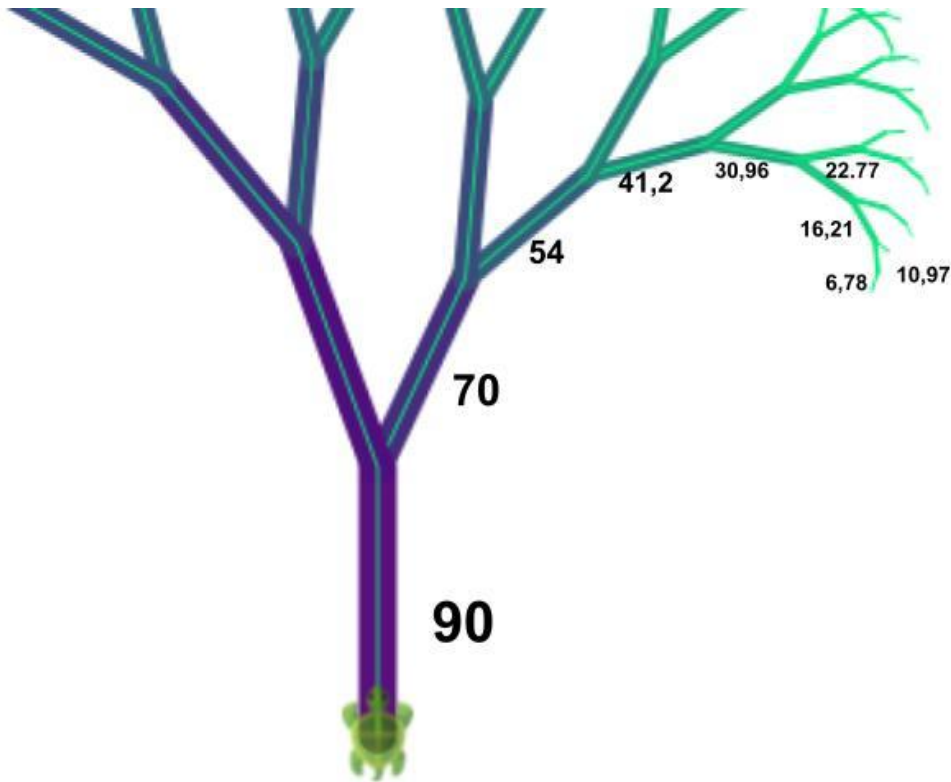
Si noti che il miglior modo per comprendere come avvengono le ricorsioni è osservare il percorso svolto dalla tartaruga a velocità ridotta, impostandola nel programma chiamante definito dopo la funzione *tree*.

Si osservi adesso il codice interno alla funzione.

Imposta lo spessore della penna ad una frazione della distanza	<code>setPenThickness(distance/7)</code>
Imposta il colore del ramo che sta disegnando in funzione della distanza	<code>setPenColor(cm.rgb(distance.toInt, math.abs(255-distance*3).toInt, 125))</code>
Disegni ramo della lunghezza ricevuta in input	<code>forward(distance)</code>
Ruota di 25 ° per iniziare il sotto ramo di destra	<code>right(25)</code>
Richiama se stessa, riducendo il valore della lunghezza acquisito in input	<code>tree(distance*0.8-2)</code>
Finita la ricorsione introdotta con l'istruzione precedente Orienta la penna per disegnare il ramo di	<code>left(45)</code>

sinistra.	
Richiama la ricorsione decrementando la distanza da disegnare per realizzare il ramo di sinistra.	<code>tree(distance-10)</code>
Ruota verso destra per partire con il sotto ramo di destra dei rami di sinistra.	<code>right(20)</code>
Ripercorre indietro il segmento appena disegnato alla conclusione di tutte le ricorsioni del ramo in oggetto.	<code>forward(-distance)</code>

Si noti che Il programma chiamante dell'esempio avvia *Tree* passando come parametro della distanza il valore 90. La prima chiamata ricorsiva passerà il valore 70, questa realizzerà la prima biforcazione verso destra. Tale chiamata, a sua volta, richiamerà ricorsivamente la funzione passando un valore pari a 54, realizzando la seconda biforcazione a destra. In questo modo verranno realizzati tutti i rami verso destra fino a che il valore da passare non risulti minore o uguale a 4 secondo l'immagine che segue, in cui ad ogni ramo è affiancato dal valore *distance* con il quale è stato realizzato. Quando il valore di *distance* scende al di sotto di 4, viene eseguita la rotazione verso sinistra e si avvia la ricorsione per la generazione di tutti i sotto rami.



Per quanto riguarda la sintassi è utile notare alcuni aspetti. Ad esempio **distance.toInt** è la conversione del valore contenuto nella variabile *distance* in un numero intero. Il valore di input infatti risultava definito reale, ma per essere passato come valore utile alla definizione di un colore deve diventare di tipo intero. Questa operazione si chiama **Casting**.

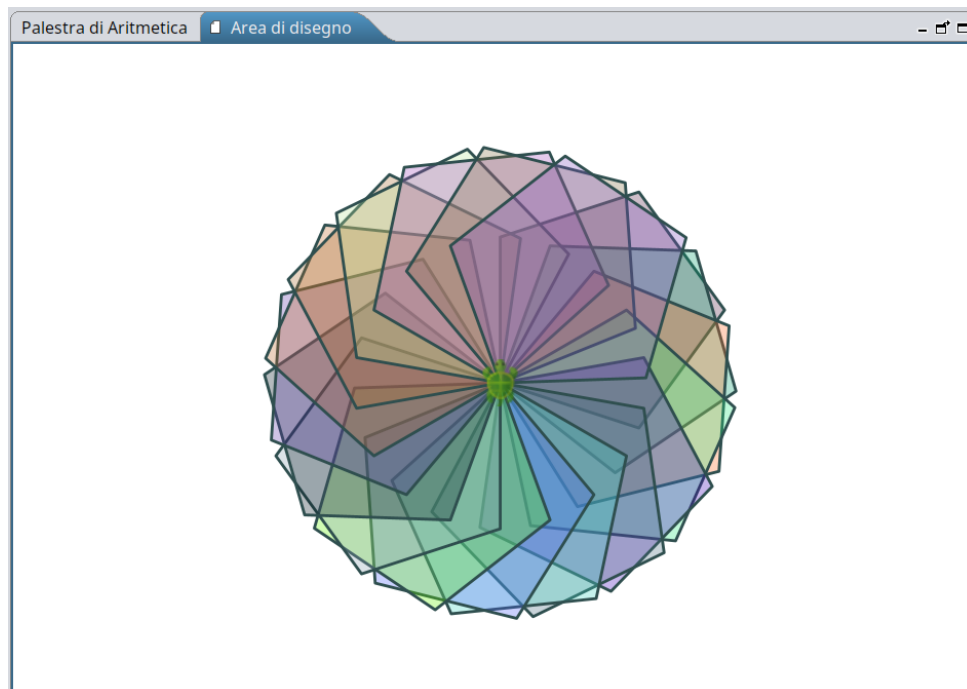
Texture

Shape e block

Quando è necessario costruire una *texture*, bisogna pensare alla figura basilare da ripetere più volte dopo aver effettuato le giuste traslazioni. Si osservi l'esempio che segue, in cui un disegno viene realizzato all'interno della funzione *Shape* e successivamente viene opportunamente posizionato nella funzione *Block*.

```
def shape() {  
  repeat(5) {  
    forward(100)  
    right(360/5)  
  }  
}
```

```
}  
}  
  
def block() {  
  setFillColor(randomColor.fadeOut(0.7))  
  shape()  
  right(20)  
}  
  
clear()  
setSpeed(fast)  
setPenColor(cm.darkSlateGray)  
repeat(18) {  
  block()  
}
```



Quando l'oggetto ruota o trasla potrebbe rendere difficile la costruzione della texture, per tale motivo esistono due funzioni destinate a salvare la posizione iniziale della tartaruga per poi ripristinarla una volta disegnato l'oggetto da ripetere, come avviene nell'esempio che segue.

```
def shape() {  
  savePosHe() // salva la posizione  
  left(45)  
  right(90, 100)  
  right(90)  
  right(90, 100)  
  restorePosHe() //ristabilisce la posizione  
}  
  
def block() {  
  setFillColor(randomColor.fadeOut(0.7))  
  shape()  
  // rotate in place  
  right(20)  
}  
  
clear()  
setSpeed(fast)  
setPenColor(cm.darkSlateGray)  
repeat(18) {  
  block()  
}
```



È possibile quindi costruire griglie complesse, come nell'esempio presentato e spiegato di seguito:

```
clear()
setBackground(white)
setSpeed(superFast)
setPenColor(cm.gray)

// Si impostano tutte le variabili che vengono utilizzate per costruire la texture
val nx = 20
val ny = 20
val cb = canvasBounds
val dx = cb.width / nx
val dy = cb.height / ny

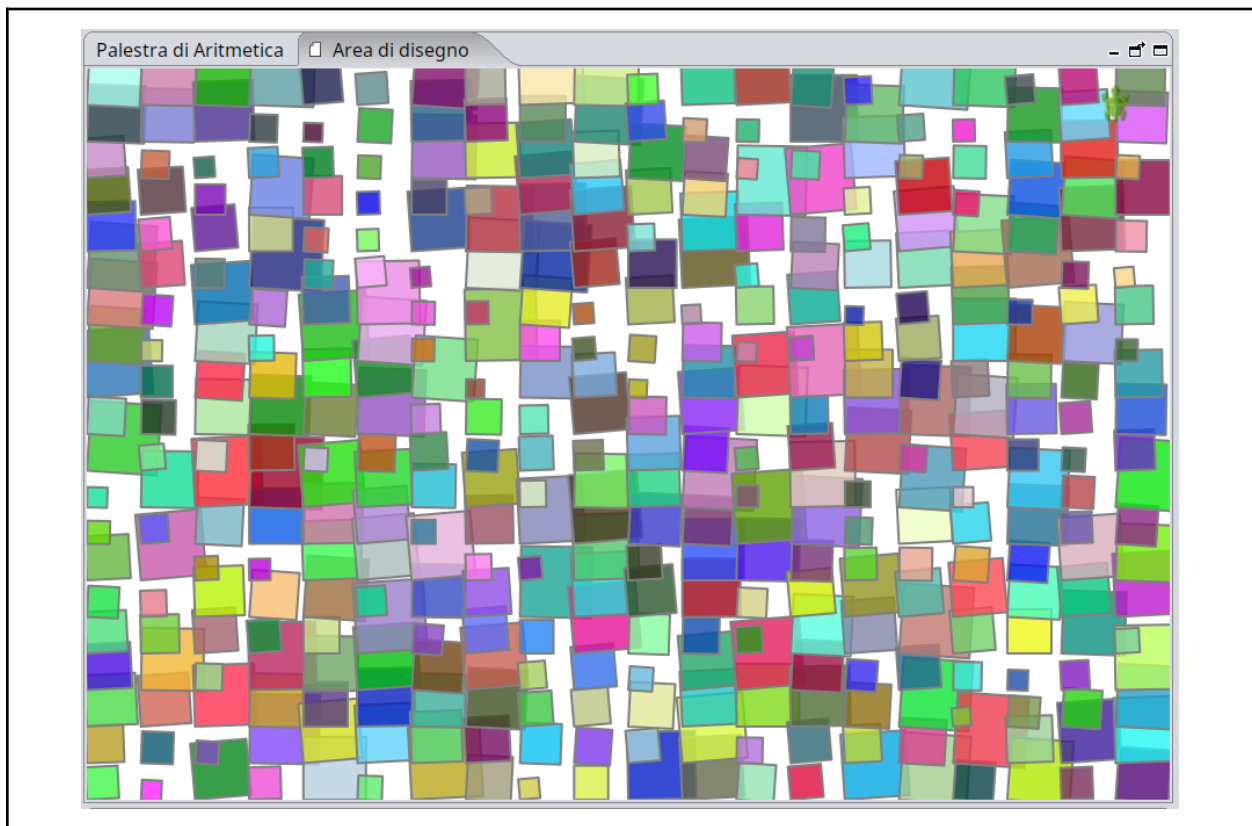
def shape() {
  savePosHe()
  right(random(-5, 5))
  val len = dy / 2 + randomDouble(dx)
  repeat(4) {
    forward(len)
    right(90)
  }
  restorePosHe()
}

def posX(gx: Int) = cb.x + gx * dx
def posY(gy: Int) = cb.y + gy * dy

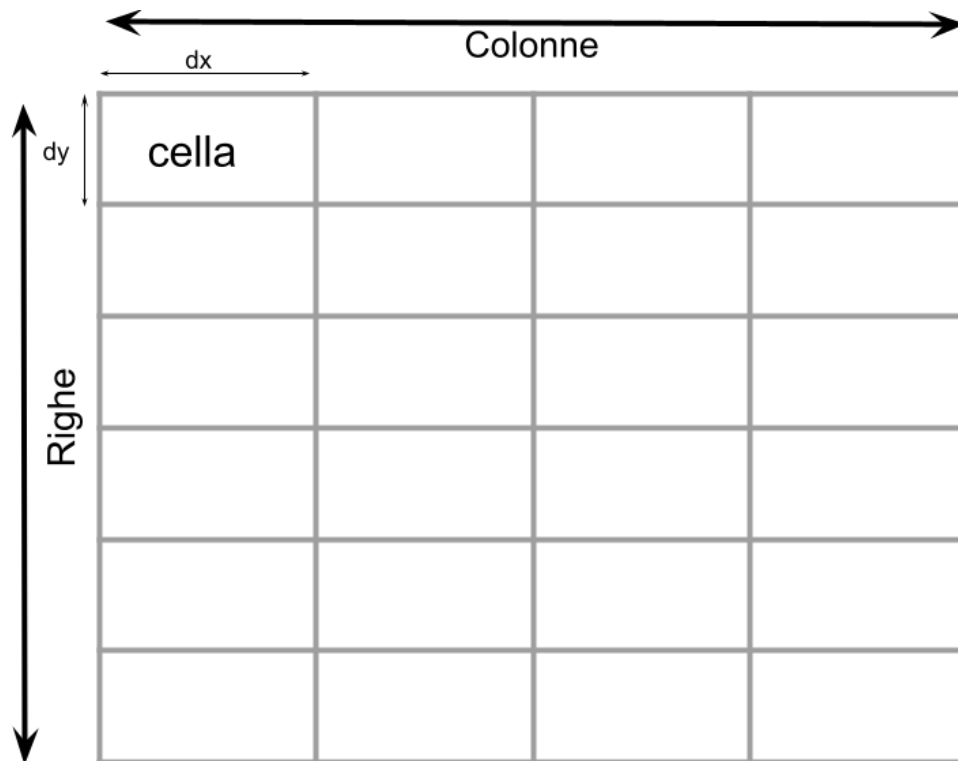
def block(gx: Int, gy: Int) {
  setPosition(posX(gx), posY(gy))
  setFillColor(randomColor.fadeOut(0.2))
  shape()
}

// funzione MAIN che realizza l'intera texture

repeatFor(0 until nx) { x =>
  repeatFor(0 until ny) { y =>
    block(x, y)
  }
}
```



Lo script divide l'area di lavoro in mattonelle della dimensione preimpostata di nx e ny . In tal modo la superficie viene divisa in una griglia, accessibili riga per colonna. In ogni cella verrà disegnato un quadrato generato in modo casuale così come spiegato di seguito.



Il software può essere così spiegato:

- **CanvasBounds** è un oggetto relativo al riquadro di lavoro, all'interno del quale si visualizza il disegno. Vuol dire che il disegno occuperà tutto lo spazio utile, in funzione della grandezza della finestra di *kojo*. **cb**, quindi, è l'oggetto che rappresenta l'area di lavoro, che contiene al suo interno, tra gli altri, due attributi: *width* per larghezza e *height* per l'altezza. Tali attributi vengono utilizzati per definire le variabili **dx** e **dy**.
- **dx** e **dy** sono definite come le dimensioni di rettangoli che dividono la larghezza ed altezza dell'area di lavoro in 20 "mattonelle" di dimensione *nx* e *ny*.
- **shape()**. La funzione *shape* (che salva la posizione iniziale di applicazione del disegno e la riusa alla fine della funzione) realizza un quadrato, con caratteristiche "uniche" perchè casuali ad ogni realizzazione:
 - ruota l'angolo di applicazione di un valore casuale compreso tra -5 e 5 gradi;
 - **val len = dy / 2 + randomDouble(dx)** genera una lunghezza del lato uguale alla metà del valore *dy* sommato ad un numero reale casuale compreso tra 0 e *dx*;
 - realizza un quadrato di lunghezza *len*, che risulterà quindi casuale e ruotato sul primo angolo in modo non predeterminato.

- **posx(gx: Int)** e **posy(gy: Int)** sono due funzioni che definiscono un valore posizionale. Prendono in input due valori interi, *gx* e *gy*, e restituiscono una posizione. Tale posizione è calcolata in modo da posizionare la tartaruga sullo spigolo in basso a sinistra della mattonella in cui creare il quadrato.
- **block(gx: Int, gy: Int)** genera il blocco prendendo in input l'angolo in basso a sinistra della mattonella.
 - **setPosition(posx(gx), posy(gy))** posiziona la tartaruga nel punto in cui si deve generare shape.
 - **setFillColor(randomColor.fadeOut(0.2))** definisce un colore casuale, al quale si applica la funzione *fadeOut* che ne modifica la trasparenza.
- La funzione principale (MAIN) realizza due cicli innestati, uno dentro l'altro.
 - **repeatFor(0 until nx) { x =>** è un ciclo che va da 0 a *nx*, associando ad *x* il valore che assume di ripetizione in ripetizione. Il ciclo delle *x* permette di passare di mattonella in mattonella di attraverso le colonne.
 - **repeatFor(0 until ny) { y =>** ad ogni iterazione del ciclo in *x* c'è un ciclo in *y*, che associa ad *y* un valore compreso tra 0 e *ny* ad ogni iterazione, così da poter attraversare le righe.
 - All'interno di entrambi i cicli si richiama la funzione *block*, passandogli i valori di *x* e *y* trovati, perchè la coppia (*x,y*) identifica riga e colonna della mattonella all'interno della quale disegnare il quadrato.

Picture

Kojo mette a disposizione un modulo per la gestione delle immagini. In questo modo non è solo possibile disegnare percorsi, ma anche costruire vere e proprie immagini che possono essere allineate, sovrapposte, traslate e in generale manipolate. Creare immagini permette inoltre di avvicinarsi all'animazione ed alla costruzione di giochi di tipo Arcade, perchè sarà possibile trattarle come oggetti che interagiscono tra loro.

Si osservi il seguente esempio:

```
// pulisce l'ambiente di lavoro
clear()
```



```
// costruiamo le immagini. Ogni immagini viene salvata in una
variabile
val pic1 = Picture {
    setPenColor(red)
    setPenThickness(6)
    setFillColor(yellow)
    repeat(6) {
        forward(150)
        right(60)
    }
}
// seconda immagine
val pic2 = Picture {
    setPenColor(yellow)
    setPenThickness(6)
    setFillColor(red)
    repeat(4) {
        forward(160)
        right(90)
    }
}

// crea una immagine con una scritta
val scritta = Picture {
    setPenFontSize(25)
    setPenColor(cm.darkRed)
    write("Kojo Picture!")
}

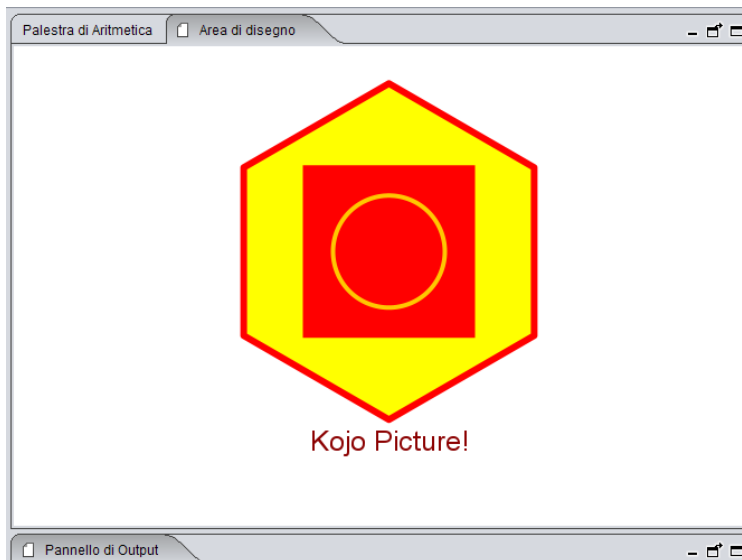
// crea una immagine con solo una cornice
val cerchio = Picture {
    setPenColor(orange)
    setPenThickness(4)
    right(360, 50)
}

// crea una immagine trasparente per lasciare uno spazio tra le
immagini
val gap = Picture {
    setPenColor(noColor)
    repeat(4) {
        forward(100)
        right(90)
    }
}
```

```
// sovrappone le immagini, nell'ordine in cui vengono passate come input
val sovrapponi = picStackCentered(pic1, pic2, cerchio)

// allinea le immagini passate in input rispetto al centro
val allinea = picColCentered(scritta, sovrapponi, gap)

// disegna la colonna creata al centro della scena
drawCentered(allinea)
```



Si noti come ogni variabile venga costruita a partire dalla struttura **Picture**. Tale struttura è identificata da un blocco (parentesi graffe) all'interno del quale si determinano le caratteristiche fisiche dell'immagine da creare. In sostanza gli stessi percorsi che la tartaruga realizza attraverso i movimenti noti, se vengono realizzati all'interno di un blocco Picture, creano un'immagine. Ogni variabile, quindi, può essere spostata, sovrapposta o manipolata come una figura autonoma attraverso apposite funzioni. Nell'esempio precedente si possono identificare:

- **picStackCentered()**: sovrappone tutte le immagini passate come input, il numero delle immagini può variare di volta in volta. L'ordine con cui vengono inserite le immagini determina sovrapposizioni diverse, perchè l'ordine con cui vengono passate le immagini corrisponde all'ordine con cui vengono sovrapposte. In sostanza la sovrapposizione delle immagini determina una pila (si immagini una pila di piatti) che in informatica viene definita "Stack". Il risultato finale è un'immagine a sua volta, che fonde tutte quelle sovrapposte e che potrà essere salvata in una nuova variabile.

- **picColCentered()**: Prende in input un numero variabile di immagini e le allinea tutte rispetto al loro centro, dall'alto verso il basso. Il risultato finale è un'immagine a sua volta, che fonde tutte quelle allineate.
- **drawCentered()**: Disegna l'immagine passata come input al centro dell'area di lavoro.

Figure geometriche e colori

Nel paragrafo precedente sono state create immagini a partire dal movimento della tartaruga, così da generare figure geometriche. **Picture**, in quanto “modulo” da cui prendere funzioni ed elementi, mette a disposizione la possibilità di costruire figure geometriche note. In tal modo non sarà necessario costruire il percorso della tartaruga in una funzione a parte prima di generare un'immagine, ottimizzando anche il tempo di esecuzione dello script; basterà quindi richiamare l'apposita funzione come negli esempi che seguiranno. Risulta invece necessario utilizzare il movimento della tartaruga nel momento in cui bisognerà realizzare percorsi articolati e non regolari, quindi immagini personalizzate e non standard.

È da sottolineare che *Picture* nasce con lo scopo di creare immagini da utilizzare come oggetti figurativi in applicazioni particolari, per tale motivo una volta disegnata un'immagine non è possibile ripeterla più volte, ma solo traslarla. Un'immagine, infatti, è un oggetto che può esistere in un'unica copia nell'ambiente di lavoro.

Le principali funzioni relative alle Pictures sono quelle mostrate in tabella:

Function	Description
Picture {turtle drawing code}	Crea una Immagine a partire da un percorso della tartaruga.
Picture.line(width, height)	Crea una linea indicando lunghezza e spessore.
Picture.rectangle(width, height)	Crea un rettangolo indicando base ed altezza.
Picture.circle(radius)	Crea un cerchio con il centro in (0,0) e raggio da passare alla funzione.
Picture.ellipse(xRadius, yRadius)	Crea una ellisse che prende in input la il fuoco sull'asse X e Y, il centro è in (0, 0)
Picture.ellipseInRect(width, height)	Crea un'ellisse ma attraverso i valori di base ed altezza del rettangolo circoscritto.
Picture.point	Crea l'immagine di un punto.

Function	Description
Picture.arc(radius, angle)	Crea un arco, attraverso il raggio e l'angolo. Il centro è in (0, 0).
Picture.text(string)	Crea un'immagine con il testo.
Picture.hgap(width)	Crea un'immagine invisibile della larghezza indicata.
Picture.vgap(width)	Crea un'immagine invisibile dell'altezza indicata.
Picture.image(fileName)	Crea un'immagine con un'immagine da file.
Picture.image(url)	Crea un'immagine a partire da un percorso url.
Picture.image(image)	Crea un'immagine a partire da un'altra immagine.

Twine

Twine (<https://twinery.org/>) è un ambiente nel quale approcciarsi al coding, anche senza dover necessariamente utilizzare un linguaggio di programmazione nel senso classico del termine. È stato più volte detto che il coding non è semplicemente o unicamente programmazione, per questo motivo è possibile applicarlo nella realizzazione di testi, senza rispettare la linearità classica che verrebbe implicitamente imposta da un qualunque software di scrittura. Se un testo sfugge ad una lettura lineare, vuol dire che deve prendere la forma di **ipertesto**, tipico del web.

Un testo che realizza salti necessita di strumenti diversi da un semplice editor e sarà necessario conoscere linguaggi detti di **markup**, che non sono linguaggi di programmazione, ma un modo diverso di scrivere e formattare testi. In questo modo sarà possibile organizzare la struttura del lavoro, ma anche realizzare salti e connessioni. Il più noto linguaggio di markup comprensibile dai Browser è l'HTML. Se un ipertesto viene integrato con un linguaggio di programmazione come **Javascript**, sarà possibile anche realizzare salti condizionati da una sezione all'altra rendendo il lavoro particolarmente interattivo, perché permette allo scrittore di definire flussi di narrazione in funzione delle scelte del lettore, realizzando storytelling particolarmente complessi.

Per la stesura di un ipertesto è possibile quindi utilizzare diversi linguaggi, che concorrono al prodotto finale:

- **HTML**, per la struttura delle pagine. È un linguaggio di markup che permette al software di lettura la comprensione della formattazione del testo.

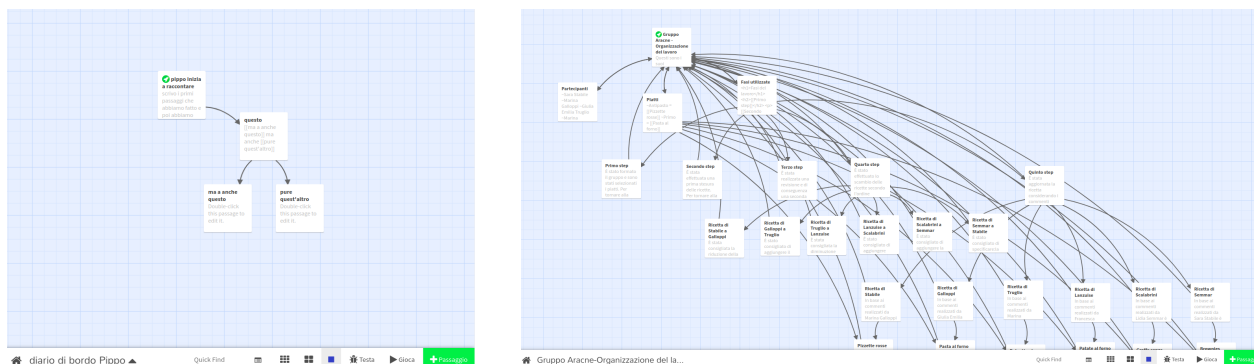
- **JAVASCRIPT**, un linguaggio di programmazione completo comprensibile ai Browser per realizzare condizioni più articolate di semplici salti ipertestuali.
- **CSS**, fogli di stile che permettono un'estetica più accattivante del testo e delle pagine realizzate in html.

Fatte queste premesse è possibile introdurre *Twine*, utile alla realizzazione di testi non lineari, costruendo un prodotto html e javascript, senza necessariamente conoscere la sintassi di questi due linguaggi. Twine può quindi essere considerato uno strumento di **storytelling digitale**.

Interfaccia

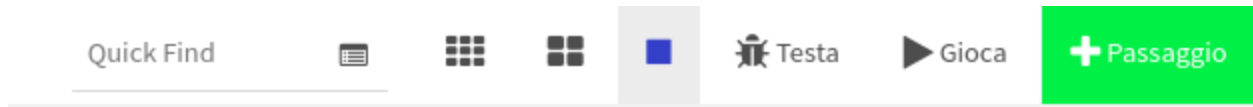
Dal sito ufficiale di Twine è possibile installare la versione legata al proprio sistema operativo, ma è anche possibile utilizzare la versione on line, così da non dover installare nulla. Non è possibile lavorare parallelamente tra più utenti come avviene con le app in cloud, tuttavia è possibile esportare il file del lavoro svolto, così da condividerli con altri utenti che potranno importarli e modificarli nella propria installazione (o utilizzo on line).

È stato descritto l'output di un lavoro svolto nell'applicazione, ossia contenuti testuali collegati tramite ipertesto, ma non è stato ancora affrontato come viene gestita la produzione di tali contenuti. Bisogna quindi sottolineare che Twine offre una sostanziale differenza di visualizzazione del lavoro in produzione con il lavoro in esecuzione. Per la produzione risulta infatti utile procedere come in una mappa concettuale, quindi con "elementi" posti in connessione tra loro. L'interfaccia di Twine permette di costruire singoli elementi all'interno dei quali scrivere e personalizzare il testo oggetto dell'elemento, visualizzando contemporaneamente le relazioni tra i diversi blocchi. Nelle immagini che seguono è possibile vedere le relazioni tra i blocchi, visualizzati per titoli e collegamenti.

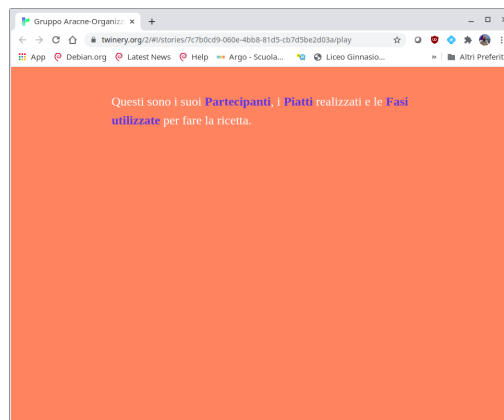


L'interfaccia permette una visualizzazione sintetica degli elementi, fornendo un menù, in basso a destra, utile alla loro gestione e visualizzazione. In particolare il tasto "Gioca" permette di avviare

il primo blocco così da iniziare la navigazione esplorando tutti i blocchi, seguendo le scelte dell'utente. È possibile anche avviare i singoli blocchi in modalità test, permettendo di visualizzare gli elementi “architetturali” dei blocchi, ossia tag html, variabili e tutti gli elementi che verranno introdotti successivamente. Per avviare in questa modalità i blocchi è sufficiente avviarli tramite il tasto che si visualizza al passaggio del mouse nell'interfaccia di modifica.

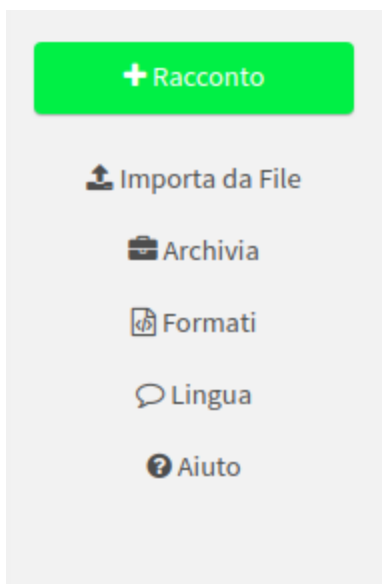


La visualizzazione dell'output è una pagina web che interpreta il primo blocco, collegato a quelli successivi tramite link di ipertesto, esattamente come avverrebbe in un sito web html.



Costruire una storia

Per iniziare una storia è sufficiente aggiungere il primo elemento, attraverso la pagina principale del software utilizzando il tasto “**+Racconto**” nella barra laterale.



Gli altri tasti principali permettono di:

- **Importa da File.** importare tutti i lavori svolti e archiviati precedentemente.
- **Archivia.** Tutte le storia in un archivio
- **Formati.** Il tipo di storia da scrivere, riguarda la sintassi da usare considerando che ne esistono diverse. In questo testo ne verrà presentata una sola ed è quella di default del software.

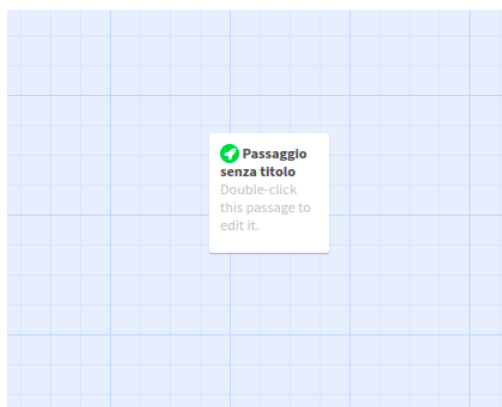
Avviato il racconto viene chiesto il nome, sempre modificabile, così da iniziare a scrivere il primo blocco.



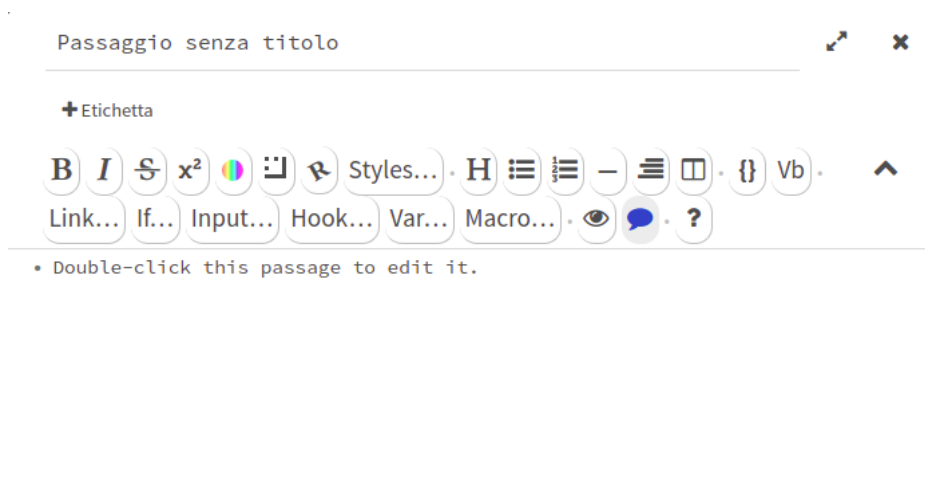
Qual'è il nome del Racconto?
(successivamente potrà essere modificato)

✕ Annulla

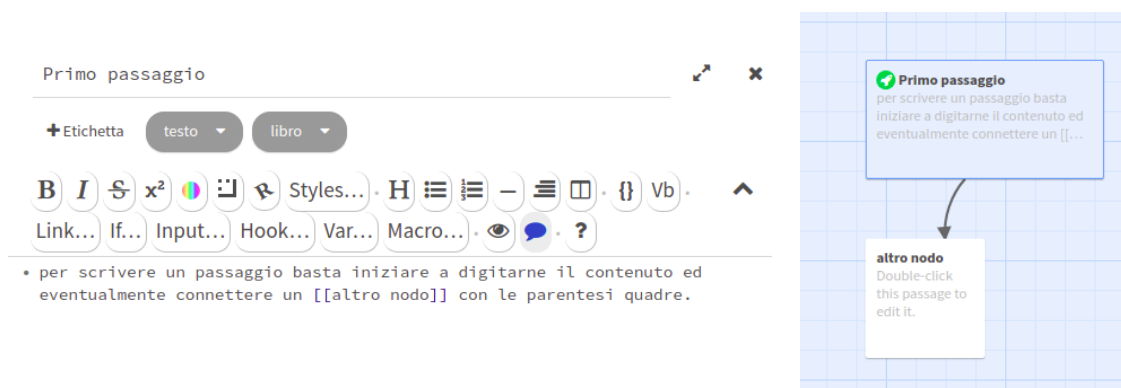
+ Aggiungi



Cliccando due volte sul blocco, oppure selezionando la voce modifiche che appare al passaggio del mouse, sarà possibile iniziare a scrivere con un editor di testo semplificato.



Le singole voci del menu rimandano alle strutture classiche di una pagina html, in questa sede non le spiegheremo nel dettaglio perchè risulta molto più utile provare a sperimentare. Risulta invece utile sottolineare le caratteristiche del nodo: Il titolo, le Etichette e la possibilità di creare connessioni ad altri nodi. Basterà, infatti, racchiudere tra doppie parentesi quadre una parole affinché Twine generi un altro nodo e le connessione che arrivano ad esso.



Osservando il risultato ottenuto dal link generato dalle doppie parentesi quadre è evidente come, con l'incremento di altri nodi, si viene a generare una mappa strutturata della storia che si sta costruendo. Se un nodo deve essere raggiunto da più blocchi, sarà sufficiente inserire il nome di quel nodo all'interno di parentesi quadre in tutti i blocchi da cui deve essere raggiunto. È necessario inoltre che il nome risulti essere esattamente corrispondente al blocco da raggiungere, perché la sintassi legata ai nomi è molto vincolata.

Per visualizzare l'anteprima del lavoro svolto basterà cliccare sull'apposito tasto del menu che appare al passaggio del mouse sul singolo blocco, Si avvierà, in questo modo, la lettura delle pagine HTML scritte all'interno del browser che si sta utilizzando.

Creare Condizioni

Gli strumenti visti fino a questo punto permettono di creare storie a partire da una mappa concettuale e relazionale tra gli elementi della storia. In tal modo sarà possibile realizzare una storia con collegamenti ipertestuali tra gli elementi, permettendo salti tra un blocco e l'altro, lasciando al lettore la scelta della strada da percorrere. Un tale approccio permette di superare i limiti che vengono imposti dall'organizzazione di un racconto secondo lo schema classico, cioè con una stesura di tipo sequenziale. A questo elemento di innovazione, twine introduce anche la possibilità di innestare condizioni logiche all'interno di un blocco, così da attivare o disattivare la visualizzazione di altri elementi raggiungibili dall'utente.

L'introduzione di vere e proprie istruzioni di tipo logico all'interno del testo, introduce sintassi riconducibile a un vero e proprio linguaggio di programmazione che prende il nome di **Harlowe**. Di seguito verrà presentato un esempio che rende possibile la gestione di una condizione all'interno di un blocco, ma per approfondire tutti gli aspetti legati a questo approccio è possibile consultare la pagina dedicata a tutte le soluzioni possibili (<https://twine2.neocities.org/>). Al link indicato, inoltre, è possibile trovare anche tutti i tipi di formattazione del testo ottenibili attraverso una sintassi semplice e intuitiva.

Si supponga di scrivere un racconto all'interno del quale si debba vincolare l'accesso ad una stanza, il lettore potrà aprire la porta solo dopo aver trovato la chiave dentro un baule.

Si inizi a scrivere il primo blocco della storia, all'interno del quale va indicata la variabile da valorizzare.

The screenshot shows the Twine editor interface. At the top, a text block is titled "prima stanza". Below the title, there are two dropdown menus: "Etichetta" (set to "testo") and "libro". Below these are two rows of icons for text formatting and logic. The first row includes icons for bold (B), italic (I), strikethrough (ABC), superscript (x²), color (rainbow), background color (grid), link (chain), styles (text box), heading (H), list (bulleted), ordered list (numbered), indent (margin), outdent (margin), table (table), code (code block), and variable (Vb). The second row includes icons for link (Link...), if (If...), input (Input...), hook (Hook...), variable (Var...), macro (Macro...), eye (visibility), speech bubble (comment), and a question mark (?). Below the icons, there is a list of three bullet points:

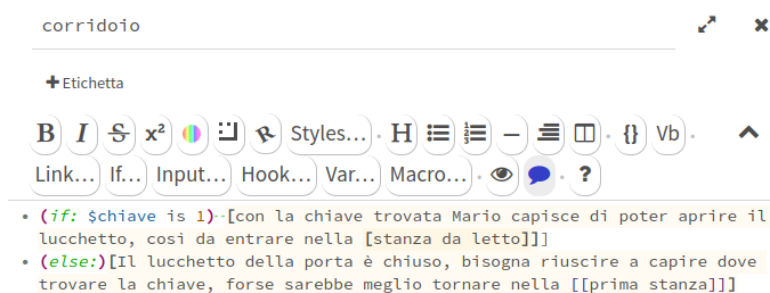
- Nella stanza ci sono alcune porte, tutte di forma diversa.
- La prima `[[porta]]` è grande e piena di borchie, dallo spioncino Mario intuisce che la stanza alle spalle è buia.
- La seconda porta permette di accedere ad un `[[corridoio]]`, è socchiusa e può vedere una porta verde con un grande lucchetto.

Nella storia si vuole vincolare l'accesso alla porta verde al ritrovamento di una chiave, dentro la stanza buia, relativa al blocco *porta*.



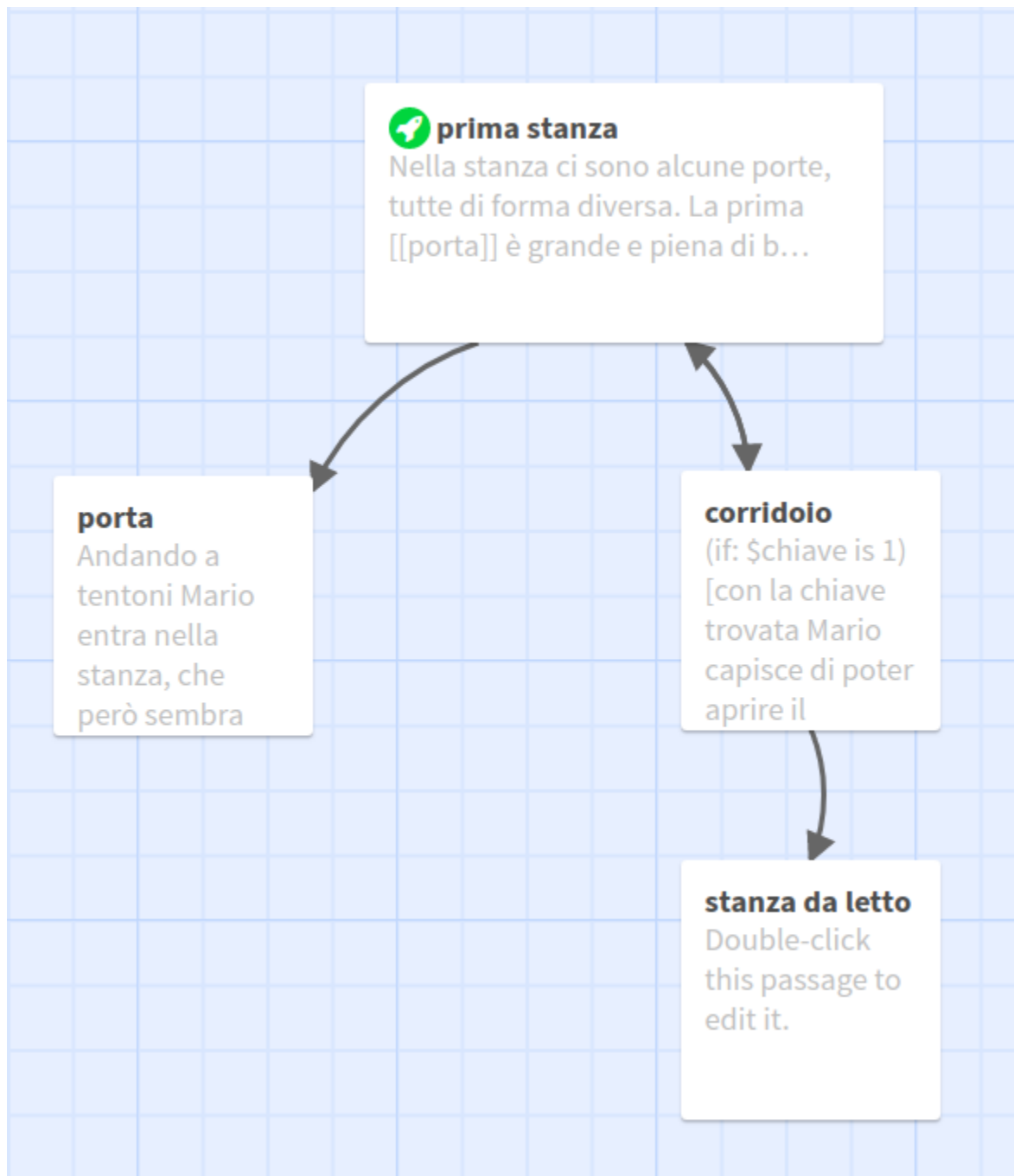
L'istruzione tra parentesi tonde nell'ultima riga è leggibile in questo modo: imposta (SET) la variabile CHIAVE (il simbolo di \$ suggerisce a Twine che la parola chiave è una variabile) al valore 0 (TO).

A questo punto sarà possibile, nel blocco del *corridoio*, permette di aprire il lucchetto solo se si è entrati nel blocco *porta*.




Si noti che la selezione del testo da visualizzare avviene tramite il costrutto IF- THEN - ELSE. Se - IF- la variabile assume un determinato valore -THEN - visualizza il testo tra parentesi quadre. Altrimenti - ELSE - visualizza il testo indicato tra le parentesi quadre che seguono.

I collegamenti che si ottengono sono i seguenti:



Le pagine visualizzate dal lettore saranno quelle nelle immagini seguenti e dipendono dal percorso che ha seguito.



Nella stanza ci sono alcune porte, tutte di forma diversa.
La prima **porta** è grande e piena di borchie, dallo spioncino Mario intuisce che la stanza alle spalle è buia.
La seconda porta permette di accedere ad un **corridoio**, è socchiusa e può vedere una porta verde con un grande lucchetto.

Se entra direttamente nel corridoio visualizza la seguente pagina:

Il lucchetto della porta è chiuso, bisogna riuscire a capire dove trovare la chiave, forse sarebbe meglio tornare nella **prima stanza**

A questo punto, entrando nella prima stanza e nella successiva porta troverà la chiave.



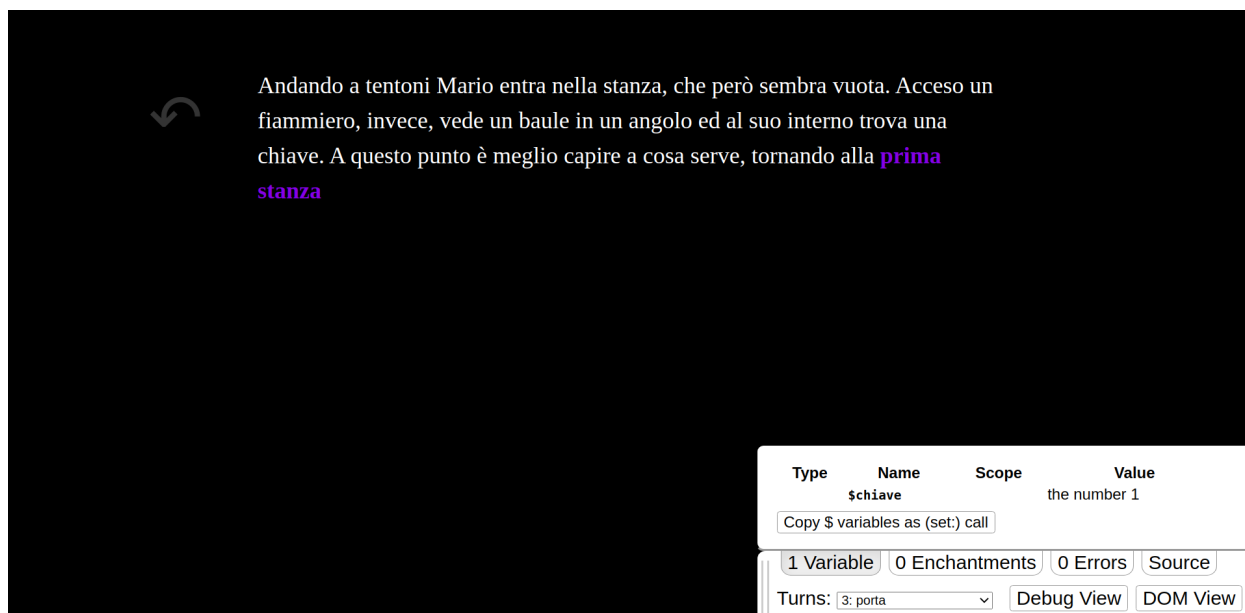
Andando a tentoni Mario entra nella stanza, che però sembra vuota. Acceso un fiammifero, invece, vede un baule in un angolo ed al suo interno trova una chiave.

Accedendo adesso al corridoio si potrà aprire quel lucchetto, il testo visualizzato sarà il seguente.



con la chiave trovata Mario capisce di poter aprire il lucchetto, così da entrare nella **stanza da letto**

Si noti inoltre che è possibile visualizzare i blocchi in fase di test, selezionando il relativo tasto che si visualizza al passaggio del mouse. In questo modo sarà possibile visualizzare le variabili attive ed il valore contenuto in basso a destra, così da controllare il flusso delle informazioni.



A questo punto è comprensibile come sia possibile generare racconti non lineari attraverso l'uso di salti e selezioni, strutture di programmazione viste in precedenza.

Immagini, sfondi e altri oggetti multimediali

Le storie costruite con Twine sono in sostanza pagine html collegate attraverso link diretti, quindi collegamenti ipertestuali. Per tale motivo l'inserimento di oggetti multimediali all'interno della storia (come immagini, video e audio) non può avvenire tramite inclusione all'interno del prodotto, come verrebbe gestito un documento di testo; gli inserimenti di "oggetti" esterni possono invece essere gestiti attraverso l'utilizzo dei **tag html** predisposti all'utilizzo di risorse esterne al file. Ogni pagina HTML del web, infatti, include al suo interno soltanto il testo formattato attraverso l'uso corretto dei tag (e dei fogli di stile CSS che verranno presentati in seguito), tutte le risorse incluse sono in realtà oggetti salvati altrove e richiamati attraverso appositi tag che includono il percorso diretto alla singola risorsa.

Date le premesse è buona norma salvare tutti gli oggetti multimediali all'interno di una cartella accessibile dal web, ad esempio un servizio Cloud come Drive, Onedrive o Dropbox, rendendo tale cartella pubblica e accessibile tramite link diretto ai singoli file inclusi.

La sintassi per includere un'immagine è la seguente:

```

```

A differenza degli altri tag html, quello che riguarda le immagini non ha un tag di apertura ed uno di chiusura del blocco, ma il link all'immagine viene inserito all'interno delle parentesi ****.

Esistono molti parametri che possono essere inclusi all'interno del tag, i principali sono indicati nell'esempio precedente:

- **src**: è la sorgente (src viene da source) del file, il link dal quale ricavarsi l'immagine. Risulta evidente che è questo il motivo per il quale è necessario rendere pubblica la cartella che contiene la singola immagine da visualizzare.
- **width, height**: sono le dimensioni di altezza e larghezza, che dimensionano l'immagine secondo le proprie necessità, indipendentemente dalla grandezza reale della fotografia da integrare.
- **alt**: è un parametro che permette di leggere un commento o una descrizione della fotografia al passaggio del mouse. Quello che verrà associato al parametro **alt** sarà visibile, quindi, solo se l'utente passa e ferma il mouse sull'immagine, senza cliccare.

Si tenga presente che quando si parla di URL della risorsa da includere, si intende il percorso diretto e pubblico a quella risorsa; tale percorso si può realizzare assoluto o relativo.

Un **path** (percorso) si intende **assoluto** quando il link inizia con l'origine del percorso, se pubblico nel web inizierà con "http", se all'interno del proprio PC Windows inizierà con C:\.

Un **path** si intende **relativo** quando si considera la cartella di partenza del path quella da dove è richiamata la risorsa da includere.

Per chiarire bene il concetto si considerino le cartelle del proprio computer, immaginando di salvare le pagine HTML nella cartella:

C:\Documents\Users\Sites

Il percorso che porta alla cartella *Sites* in questo caso è assoluto, perché è descritto a partire dal punto iniziale C:\

Se si volessero raccogliere tutte le immagini all'interno di una cartella *image*, il suo Path assoluto diventerebbe C:\Documents\Users\Sites\image

il Path assoluto, invece, della pagina html riferito al progetto potrebbe essere questo:

C:\Documents\Users\Sites\index.html

A questo punto per inserire link alle immagini raccolte (all'interno del file index.html) è possibile scegliere una delle due strade come descritto nell'esempio che segue.

```
// PATH ASSOLUTO



// PATH RELATIVO


```

Si noti che:

- Il path relativo implica la creazione della cartella *image* all'interno della cartella che contiene il file html che include fotografie in *image*.
- Nel path relativo non deve essere incluso il carattere “\” come primo carattere del PATH, Altrimenti lo considererà assoluto.
- Se non si è sicuri di gestire in modo appropriato le cartelle è sempre conveniente utilizzare Path assoluti così da evitare problemi di risorse non raggiungibili.
- Se l'immagine non è nel proprio pc ma è in rete, il percorso da inserire deve contenere “http://...” altrimenti non verrà riconosciuto.

Analogamente alle immagini è possibile inserire video ed audio attraverso i seguenti tag:

```
<video src="sorgente video" width="640" height="480">
</video>

<audio src="the URL of your video" autoplay>
```

Si noti come il parametro *autoplay* eseguirà il file audio appena verrà aperta la pagina.

HTML

Nei paragrafi precedenti è stato presentato un prodotto che a tutti gli effetti risulta essere un sito web, composto da pagine html collegate tra loro. È possibile quindi definire un sito web come uno o più file html connessi da collegamenti ipertestuali e visualizzabile attraverso un browser web. Tale applicazione, infatti, è la finestra attraverso la quale si naviga nel web e quindi in

internet. Per costruire un sito web, quindi, è necessario conoscere i linguaggi comprensibili da un generico browser per visualizzare contenuti o per interagire con essi. Si noti inoltre che i browser sono tanti (Mozilla Firefox, Google Chrome, Edge, Safari, Opera, ecc.) e non tutte le istruzioni sono uniformemente comprese dai diversi applicativi, ma questo è un aspetto che verrà esplicitato più avanti.

Si è parlato di linguaggi di programmazione, in realtà per quanto riguarda il web risulta essere una semplificazione eccessiva. Una pagina HTML è un file all'interno del quale sono organizzati i contenuti visualizzabili dall'utente e formattati secondo la scelta estetica di chi crea la pagina. Questo implica che il browser legge il contenuto di una pagina HTML ed interpreta esclusivamente l'organizzazione del testo o di contenuti in generale, senza eseguire alcuna istruzione e senza possibilità di far interagire l'utente, se il sito è costruito con il solo utilizzo di HTML.

È possibile a questo punto dare la definizione di HTML: **HyperText Markup Language**, un linguaggio per costruire ipertesti tramite **marcatori** (detti tag), perché definisce contenitori attraverso **TAG** così da permettere al browser di comprendere la formattazione del testo e l'organizzazione dei contenuti. In verità questa definizione di linguaggio HTML che non permette interazione con l'utente, ma solo la possibilità di visualizzare contenuti, è stata messa relativamente in discussione con la versione 5 rilasciata da relativamente poco tempo. Tali aspetti però non riguardano gli argomenti che verranno trattati.

TAG

All'interno di una pagina gli elementi vengono definiti dai TAG, quindi bisogna comprendere come gestire questi elementi e come racchiudere porzioni di testo al loro interno. Ad esempio, se si volesse indicare il titolo di una pagina si potrebbe procedere nel seguente modo:

```
<h1> Esercizio di Twine</h1>
<h2> Questa è una storia </h2>

<p>
qui si potrebbe scrivere tutta la storia...
</p>
```

Si noti che:

- I tag vengono definiti tra parentesi angolari e poi chiusi aggiungendo uno *slash* “/” all'interno delle parentesi. Il testo assumerà un font e una grandezza in funzione del tag utilizzato.

- Nell'esempio illustrato sono stati utilizzati i tag **H1**, **H2** e **p** che corrispondono a **headline 1**, **headline 2** e **paragraph**. Nello specifico, quindi, è possibile dedurre che al tag `<h1></h1>` `<h2></h2>` corrispondono i titoli che vanno da 1 a 6 ed al tag `<p>..</p>` corrisponda tutto il testo che viene associato al corpo della pagina.
- Non esiste un'interruzione di linea, questo vuol dire che i browser visualizzano il testo lungo un'unica riga, andando a capo solo alla chiusura del Tag. Esistono comunque tecniche per permettere la visualizzazione della pagina e del testo in funzione della grandezza della finestra del Browser nella quale si sta visualizzando.

Se si volesse invece modificare il tipo di font, la grandezza o il colore di un elemento “forzando” la visualizzazione di default associata a quel tag, è possibile farlo indicando determinati parametri all'interno dei tag. Un esempio di modifica di stile è presente nel seguente esempio:

```
<p style="color:red">This is a paragraph.</p>
<p style="color:blue">This is another paragraph.</p>
```

CSS

Quando si costruisce un sito in HTML tutte le pagine sono risorse autonome, file separati collegati attraverso link. Questa struttura, estremamente dinamica per certi versi, ha come aspetto negativo la necessità di personalizzare l'aspetto grafico di ogni singolo elemento, senza poter definire una sola volta le caratteristiche estetiche: colore del font, sfondi, ecc.

Un modo possibile per definire in modo univoco le caratteristiche degli elementi di una pagina (quindi dei tag) è l'uso del CSS, Cascading Style Sheets. In sostanza è possibile definire il CSS come un linguaggio attraverso cui gestire il design di una pagina web, i cui contenuti sono scritti in HTML.

Secondo questa logica è possibile intuire che una volta scritto un foglio di stile, è possibili estendere il suo l'utilizzo a più pagine web, rendendo omogenea la struttura grafica all'interno di un sito anche molto complesso, senza dover necessariamente modificare tutti i tag di tutte le pagine.

È possibile approcciarsi ai fogli di stile in 3 modi diversi: **interni**, **esterni**, **in linea**.

- I fogli di stile **esterni** permettono l'associazione dello stesso foglio a più pagine.
- I fogli di stile **interni** controllano l'aspetto di una singola pagina.
- I fogli di stile **in linea** controllano solo un elemento su di una singola pagina, anche una sola parola.

È necessario comunque specificare che se ad una pagina HTML sono associati più fogli di stile quelli in linea avranno precedenza sui fogli interni, i quali a loro volta ce l'avranno sugli esterni.

Per scrivere un foglio di stile sarà sufficiente usare un editor di testo semplice come blocco note e generare un file di testo con estensione .css.

Un esempio di parametri è il seguente:

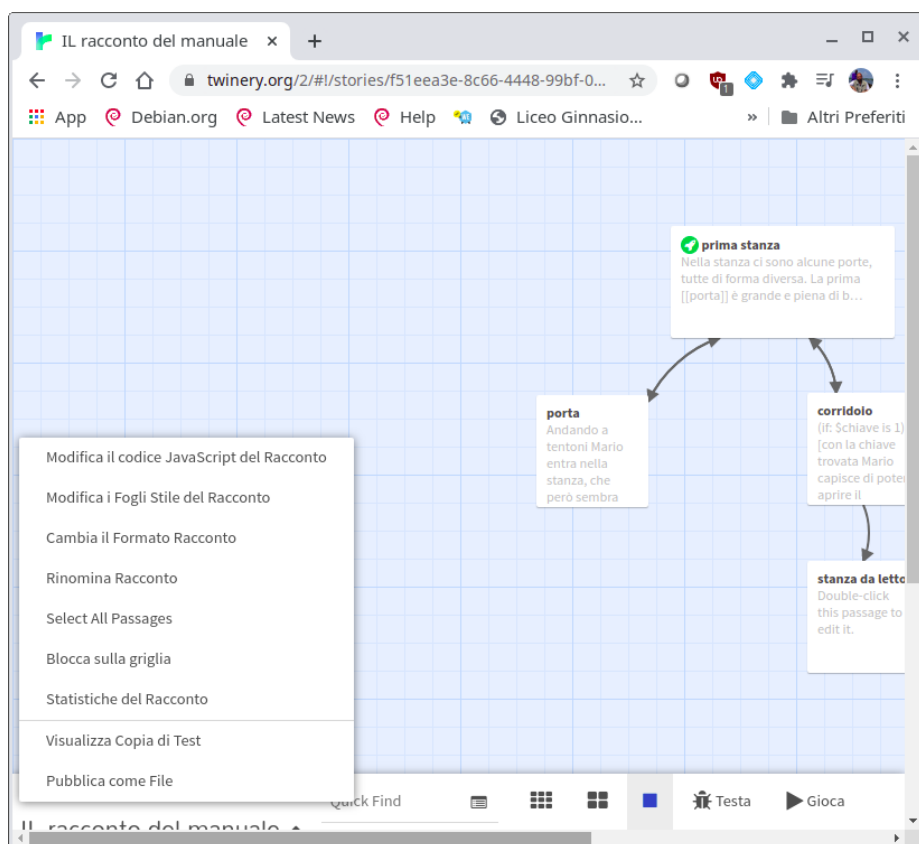
```
body {  
  background-color: lightblue;  
}  
  
h1 {  
  color: white;  
  text-align: center;  
}  
  
p {  
  font-family: verdana;  
  font-size: 20px;  
}
```

Si noti che:

- Al tag HTML indicato viene associato lo stile racchiuso nel blocco tra le parentesi graffe.
- I parametri vengono passati secondo la sintassi: PARAMETRO: VALORE, chiudendo l'istruzione con il "punto e virgola".

Per un dettaglio sui parametri possibili si faccia riferimento alle guide reperibili on line, una di queste è <https://www.w3schools.com/css/default.asp>

In Twine, è possibile inserire parametri CSS attraverso il menu in basso a sinistra, alla voce "Modifica i fogli di stile del racconto", come indicato in foto.



Si aprirà una finestra all'interno della quale è possibile inserire le personalizzazioni grafiche da applicare a tutto il racconto. Si tenga presente che alcune scelte non saranno assunte da Twine in quanto saranno prioritarie quelle di default.

Javascript

È stato detto più volte che attraverso l'uso dell'html è possibile strutturare una pagina di contenuti ed attraverso l'uso dei fogli di stile CSS è possibile organizzarne l'aspetto grafico. Questi strumenti però non permettono l'interazione con l'utente e soprattutto restituiscono un prodotto statico senza alcuna possibilità di modificare in modo dinamico i contenuti. JavaScript, invece, è un vero e proprio linguaggio di programmazione che, attraverso i classici costrutti presenti in un algoritmo, permette la modifica dinamica dei contenuti di una pagina e l'interazione con l'utente. È comprensibile a questo punto immaginare che tutti i siti e le piattaforme web di uso comune, abbiano una forte componente scritta in JavaScript. In questo testo verranno trattati solo gli aspetti più semplici di tale linguaggio, per approfondire invece gli aspetti più articolati si faccia riferimento alla documentazione ufficiale: <https://www.w3schools.com/js>

DOM

Riprendendo gli esempi trattati in precedenza, una pagina web è un file di testo con estensione `.html`, all'interno del quale le informazioni sono classificate attraverso l'uso dei *tag*. È stato visto come la grafica viene gestita attraverso un altro file, con estensione `.css`, all'interno del quale vengono definiti i parametri grafici relativi agli oggetti presenti nella pagina. Per introdurre elementi dinamici realizzati attraverso JavaScript si procederà con lo stesso approccio, è possibile farlo inserendo il codice all'interno della pagina HTML oppure scrivendo il codice all'interno di un altro file con estensione `.js` allegato alla pagina html, esattamente come è stato fatto per il file CSS.

È possibile interagire in modo dinamico sugli elementi della pagina perché JavaScript è in grado di accedere agli elementi strutturali definiti dai vari tag html. questa funzionalità è realizzabile grazie al cosiddetto **HTML DOM** (Document Object Model): modello ad oggetti del documento. In sostanza si considera la pagina come un insieme di oggetti, aggregati a vari livelli attraverso i tag html e classi css.

È possibile quindi immaginare una pagina come un intero oggetto oppure, ad esempio, il singolo tag *p* che assume gli attributi di una determinata classe CSS. Gli oggetti visti in questo modo hanno delle proprietà: il contenuto, lo stile, e tutte le caratteristiche che contraddistinguono quel singolo oggetto. JavaScript è in grado di agire in modo dinamico su tali proprietà. Per approfondire gli elementi del Dom si faccia riferimento alla guida ufficiale <https://www.w3schools.com/jsref>.

Scrivere in Javascript

Esattamente come per il CSS è possibile scrivere un file esterno alla pagina HTML all'interno del quale inserire tutto il codice utile. Tale file può essere scritto con un qualsiasi editor di testo, l'importante è rinominare l'estensione in modo corretto: `(.js)`. All'interno della sezione `<head>` del file html bisognerà poi linkare questo file in modo che il browser possa usarlo e decodificarlo al momento opportuno. L'altra strada, più semplice, è quella di inserire il codice JavaScript all'interno della sezione `<head>` del file html racchiudendo la all'interno dei tag `<script>` `</script>` nel seguente modo:

```
<script type="text/javascript">  
  
...
```

```
</script>
```

Per semplicità, nell'esempio che segue si è scelto di inserire il codice JavaScript all'interno della pagina HTML senza allegare un file esterno. Si supponga di voler inserire all'interno della pagina due tasti, alla pressione dei quali il contenuto di alcuni blocchi cambi in modo dinamico. Nella seguente pagina sono stati inseriti *tag span* per differenziare i blocchi dispari da quelli pari, sarà utilizzato questo *tagging* per modificare gli attributi di tali oggetti attraverso la pressione dei tasti.

```
<div class="grid-container">
  <div class="grid-item"><span class="dispari">quadrato 1</span></div>
  <div class="grid-item"><span class="pari">quadrato 2</span></div>
  <div class="grid-item-3"><span class="dispari">quadrato 3</span></div>
  <div class="grid-item-4"><span class="pari">quadrato 4</span></div>
  <div class="grid-item"><span class="dispari">quadrato 5</span></div>
  <div class="grid-item"><span class="pari">quadrato 6</span></div>
  <div class="grid-item"><span class="dispari">quadrato 7</span></div>
  <div class="grid-item"><span class="pari">quadrato 8</span></div>
  <div class="grid-item-9"><span class="dispari">quadrato 9</span></div>
</div>
```

In fondo alla pagina vengono inseriti i tasti, in modo da permettere all'utente di modificare gli oggetti precedentemente costruiti. Per inserire i tasti bisogna utilizzare il tag **<button>**.

```
<div class="grid-item"><span cla
<div class="grid-item"><span cla
<div class="grid-item-9"><span c
</div>

<button>Blocchi pari</button>
<button>Blocchi dispari</button>
```

La pagina visualizzata dall'utente sarà la seguente:

costruisci una tabella.

Usa la corretta gestione delle colonne per costruire la seguente tabella.

quadrato 1	quadrato 2	quadrato 3
quadrato 4	quadrato 5	quadrato 6
quadrato 7	quadrato 8	quadrato 9

Ovviamente cliccando sui tasti non succede nulla, perché sono stati predisposti gli elementi di interazione, ma non è stato definito cosa dovesse accadere qualora fossero selezionati. Per fare questo è necessario andare a definire delle funzioni all'interno del tag `<script>` presente nel *HEAD* della pagina.

Definizione di funzioni

Ricordiamo che una funzione è un algoritmo che esegue una sequenza di istruzioni. Per definire una funzione in javascript, bisogna dichiararla attraverso la parola chiave ***function*** ed il nome che le si vuole attribuire, nel nostro caso *pari* e *dispari*. Una funzione, in quanto “sottoalgoritmo”, può prevedere valori di input necessari all’esecuzione delle istruzioni definite. Per tale motivo i nomi di funzioni devono essere seguiti dalle parentesi tonde che, come in questo caso, possono essere anche vuote. le funzioni che si stanno creando, infatti, non hanno bisogno di nessun valore per poter essere richiamate, ma per essere definite funzioni è necessario comunque utilizzare le parentesi tonde. Tutte le istruzioni che fanno capo ad una funzione devono essere racchiuse all'interno delle parentesi graffe, le quali definiscono il blocco delle istruzioni che fanno capo solo alla funzione che si sta dichiarando.

```
<script type="text/javascript">
```

```
    function pari(){  
    }  
    function dispari(){  
    }
```

```
</script>
```

Una volta dichiarate le funzioni, bisogna associare il singolo bottone alla funzione che si intende richiamare. È possibile associare l'esecuzione di una funzione a molti eventi come il passaggio del mouse, alla pressione continua del pulsante, un semplice Click e tutte quelle tipologie di interazioni che normalmente avvengono durante la navigazione web. L'esempio che si sta costruendo permetterà di eseguire le funzioni al click del mouse sul pulsante scelto. Questo tipo di evento si chiama **"onclick"** e dovrà essere associato al tasto scelto.

```
<button onclick="pari();" >Blocchi pari</button>  
<button onclick="dispari();" >Blocchi dispari</button>
```

Si noti che:

- l'evento **onclick** viene inserito all'interno dell'apertura del tag BUTTON.
- La funzione viene inserita tra le virgolette "" che racchiudono il codice javascript da eseguire, potenzialmente può eseguire anche più di una funzione o addirittura si potrebbe inserire tutto il codice della funzione tra le virgolette "" e dopo il segno di assegnazione '='.
- Ogni istruzione da eseguire al click del mouse deve essere chiusa dal punto e virgola, altrimenti l'interprete JavaScript non riconosce un'istruzione di senso compiuto generando un errore.

Monitoraggio delle attività

Così come è stato scritto il codice dell'esempio, la pressione dei bottoni non genera ancora nulla, perché le funzioni al loro interno non hanno alcuna istruzione.

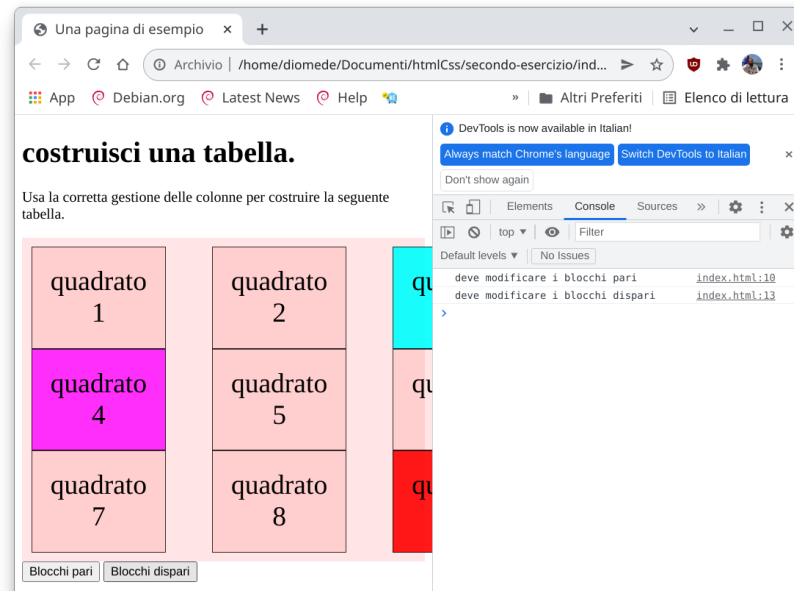
Prima di procedere alla realizzazione di qualcosa di concreto è bene predisporre le funzioni al monitoraggio di ciò che accade. Risulta molto utile, infatti, scrivere software che possa interagire con il programmatore in modo nascosto all'utente, attraverso l'uso di una interfaccia che viene chiamata **console**. In tal modo è possibile monitorare quello che succede senza dover necessariamente svelare queste attività all'utente della pagina web. Per fare ciò bisogna inserire una determinata istruzione all'interno delle funzioni che permetta la scrittura all'interno della *console* di messaggi comprensibili al programmatore.

```
<script type="text/javascript">

    function pari(){
        console.log("deve modificare i blocchi pari")
    }
    function dispari(){
        console.log("deve modificare i blocchi dispari")
    }

</script>
```

Per visualizzare la console bisogna selezionare l'opzione **ispeziona** con il tasto destro del mouse quando si visualizza la pagina in modalità utente.

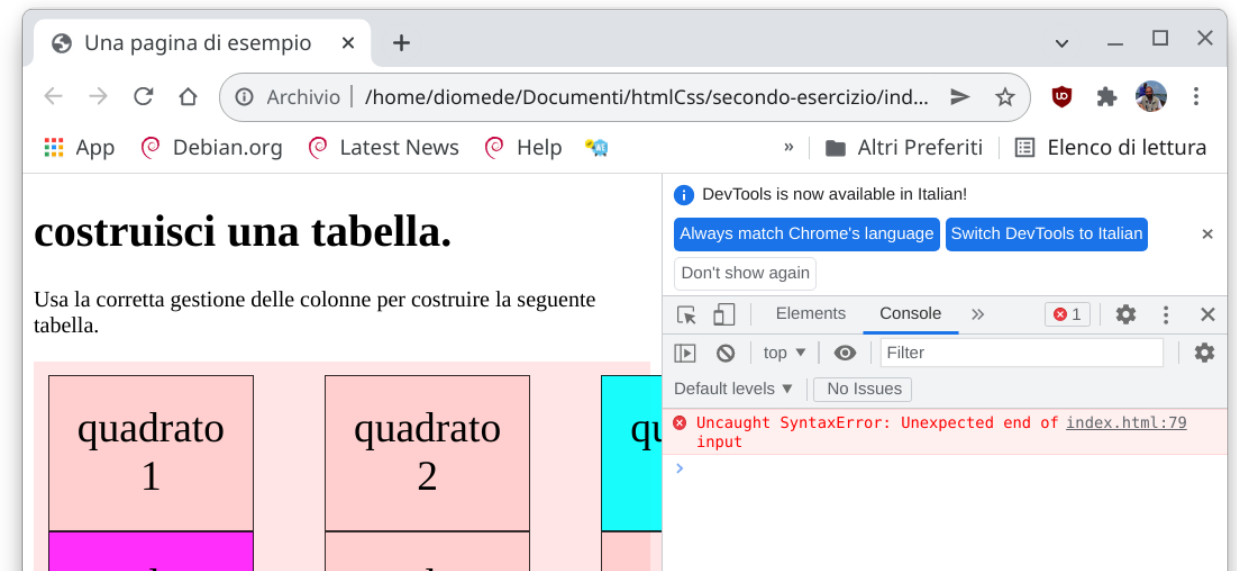


Come si può vedere, sul lato destro emerge una sezione divisa in *palette*, una di queste è la *console*. Al click del mouse su entrambi i tasti appare il messaggio che si è deciso di far stampare all'interno delle funzioni JavaScript. In tal modo è possibile tracciare l'andamento di un software anche complesso, senza per questo rendere noto all'utente della pagina web quello che accade.

A titolo di esempio, si ipotizzi di commettere un errore anche banale, come dimenticare di chiudere una parentesi tonda della funzione richiamata all'interno del tag Button.

```
<button onclick="pari(" >Blocchi pari</button>
<button onclick="dispari();" >Blocchi dispari</button>
```

La console restituirà immediatamente l'errore indicando la riga del file risultata errata.



Sintassi del codice

A questo punto possiamo ricapitolare gli elementi costruiti:

- Struttura html.
- Tag CSS per la gestione degli oggetti grafici.
- Funzioni JavaScript dichiarate e predisposte al monitoraggio delle attività.
- Inserimento dei tasti attraverso i tag `BUTTON` ed il richiamo corretto delle funzioni desiderate.

Resta soltanto da concludere l'implementazione delle attività che si ritiene di dover far realizzare alle funzioni dichiarate. Come anticipato nell'introduzione, JavaScript riesce a visualizzare tutti gli elementi del Dom, questo significa che attraverso funzioni predefinite può selezionare gli elementi che non interessano. Nell'implementazione della funzione quindi bisogna andare a dichiarare una variabile, all'interno della quale salvare l'elenco di tutti gli elementi che si intendono modificare.

```
<script type="text/javascript">

    function pari(){
        console.log("deve modificare i blocchi pari");

        blocchi_pari = document.getElementsByClassName('pari');
        console.log(blocchi_pari);
    }
    function dispari(){
        console.log("deve modificare i blocchi dispari");

        blocchi_dispari = document.getElementsByClassName('dispari');
        console.log(blocchi_dispari);
    }

</script>
```

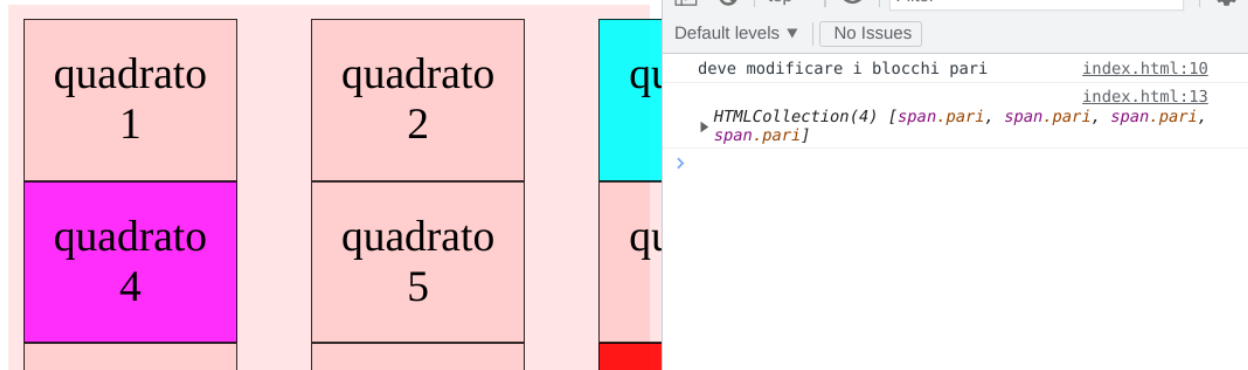
Si noti che:

- Le variabili **blocchi_pari** e **blocchi_dispari** conterranno le rispettive liste di blocchi.
- Per identificare i blocchi, JavaScript mette a disposizione una funzione **getElementByClassName()** il cui nome lascia intendere cosa permette di fare: preleva dal document tutti gli elementi il cui nome della classe è quello indicato tra le parentesi tonde. Si noti che la funzione è applicata all'oggetto documento (l'intero file *html*) attraverso l'uso del *punto*, che separa la funzione dell'oggetto sul quale applicarla.
- Il nome della classe viene indicato tra due apici ' ', in tal modo JavaScript riconosce il nome come una parola da passare in input alla funzione.
- Il *console.log()* richiamato dopo l'uso delle variabili, restituirà alla console la lista dei blocchi ricavati. Si noti che tra le parentesi tonde del *console.log()* viene passata la variabile appena creata, per tale motivo non vengono utilizzate le virgolette ""; in tal modo JavaScript capirà che *blocchi_dispari*, ad esempio, è una variabile e non una parola da scrivere nella console.

L'output che si vedrà quindi nella console è quello mostrato in figura.

costruisci una tabella.

Usa la corretta gestione delle colonne per costruire la seguente tabella.



Si noti che la lista degli oggetti riconosciuti è Quello che comunemente viene considerato un *Array* oppure detto *Vettore*. La lista è chiusa tra parentesi quadre ed ogni elemento è separato attraverso l'uso delle virgole. Una variabile di questo tipo è un insieme di oggetti che possono essere prelevati uno alla volta, per applicare le modifiche ipotizzate.

Per conoscere quanti elementi sono stati trovati dobbiamo richiamare una proprietà del vettore ricavato: **length**. Tale proprietà viene richiamata sempre utilizzando il punto separatore, così come è stato fatto prima. Restituiamo in *console* tale valore.

```
<script type="text/javascript">

function pari(){
    console.log("deve modificare i blocchi pari");

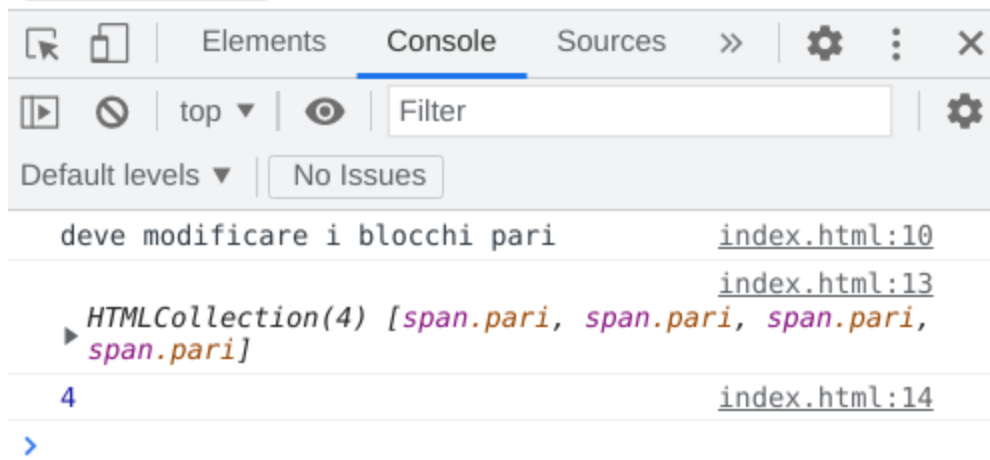
    blocchi_pari = document.getElementsByClassName('pari');
    console.log(blocchi_pari);
    console.log(blocchi_pari.length);
}

function dispari(){
    console.log("deve modificare i blocchi dispari");

    blocchi_dispari = document.getElementsByClassName('dispari');
    console.log(blocchi_dispari);
    console.log(blocchi_dispari.length);
}

</script>
```

L'output in console, alla pressione del tasto "pari", sarà il seguente:



È stato detto che una lista permette di fare una scansione di tutti i suoi elementi, vuol dire che il primo elemento corrisponde ad un indice 0, il secondo all'indice 1 e così via fino a **length-1**. Nell'esempio realizzato gli elementi pari sono 4, quindi gli indici vanno da 0 a 3.

Per permettere a javascript di prelevare e modificare tutti gli oggetti bisogna realizzare un costrutto iterativo, in particolare verrà implementato un ciclo precondizionato: **il ciclo di for**.

La sintassi per realizzare il ciclo descritto è la seguente:

```
function pari(){
    console.log("deve modificare i blocchi pari");

    blocchi_pari = document.getElementsByClassName('pari');
    console.log(blocchi_pari);
    console.log(blocchi_pari.length);

    var i;
    for (i=0; i < blocchi_pari.length; i++){
        // ...
    }
}
```

Si noti che:

- Il ciclo utilizza una variabile *ausiliaria* **i** dichiarata subito dopo il ciclo.

- Il ciclo **for** è costituito da tre istruzioni racchiuse tra parentesi tonde e separate da punti virgola:
 - **Il punto di inizio**, assegnamo 0 alla variabile *i*.
 - **Il criterio di arresto**, deve fermarsi all'ultimo elemento.
 - Ad ogni iterazione si **incrementa il valore della *i***.
 - Tra le parentesi graffe si racchiudono le istruzioni da ripetere ad ogni iterazione.

Nell'esempio scelto si è deciso di modificare lo sfondo dei blocchi selezionati, per ottenere questo risultato bisogna modificare l'attributo relativo allo sfondo, un blocco alla volta.

```
function pari(){
  console.log("deve modificare i blocchi pari");

  blocchi_pari = document.getElementsByClassName('pari');
  console.log(blocchi_pari);
  console.log(blocchi_pari.length);

  var i;
  for (i=0; i < blocchi_pari.length; i++){

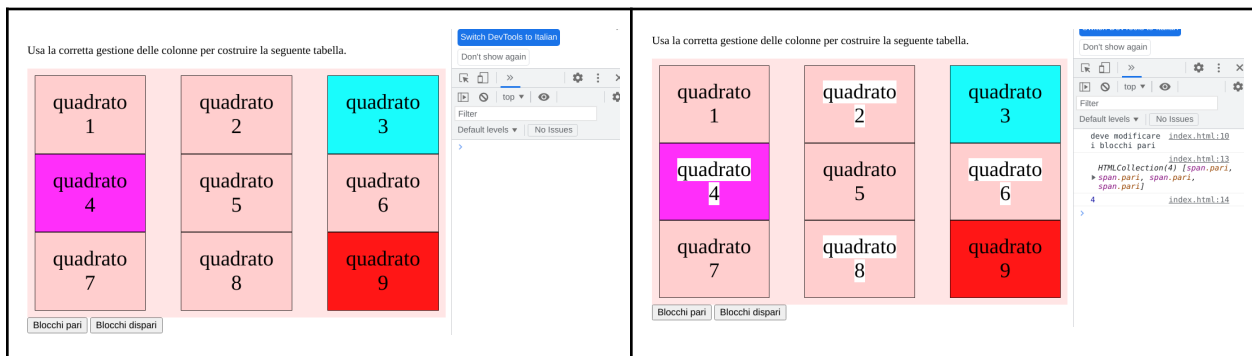
    blocchi_pari[i].style.backgroundColor = "white";
  }
}
```

si noti che:

- È stato anticipato che **blocchi_pari** è una lista di oggetti, accessibili attraverso un indice che va inserito tra parentesi quadre. La variabile *i* permette di selezionarne uno alla volta iterazione dopo iterazione.
- L'attributo viene modificato tramite l'operatore ' = ', nel caso specifico l'attributo è **backgroundColor** appartenente al gruppo **style** dell'oggetto selezionato nella lista. Si noti come l'uso del "punto" permette di identificare ed associare tutti gli elementi elencati.

Nella tabella che segue si può apprezzare cosa accade alla selezione del *button*.

PRIMA	DOPO
-------	------



L'inconveniente di questa realizzazione è che una volta selezionato i blocchi pari, rimarranno evidenziati anche quando si selezionano quelli dispari e viceversa. Risulta utile, quindi, ripristinare lo status iniziale alla pressione dell'altro tasto. Per realizzare questa soluzione si può aggiungere un ciclo che azzeri le modifiche ai blocchi diversi da quelli selezionati, la funzione dispari, ad esempio, sarà la seguente:

```
function dispari(){
    console.log("deve modificare i blocchi dispari");

    blocchi_dispari = document.getElementsByClassName('dispari');
    console.log(blocchi_dispari);
    console.log(blocchi_dispari.length);

    var i;
    for (i=0; i < blocchi_dispari.length; i++){
        blocchi_dispari[i].style.backgroundColor = "white";
    }

    blocchi_pari = document.getElementsByClassName('pari');
    var i;
    for (i=0; i < blocchi_pari.length; i++){

        blocchi_pari[i].style.backgroundColor = "";
    }
}
```

A questo punto è stata costruita una pagina html con elementi CSS grafici e con una modifica dinamica degli stessi elementi grafici, integrando HTML, CSS e Javascript.

Bibliografia

<https://www.kogics.net/kojo>

<https://docs.kogics.net/turtle-index.html>

<https://docs.kogics.net/ideas/turtle-shape-block.html#quick-recap>

<https://docs.kogics.net/picture-index.html>

<https://twinery.org/>

<https://twinery.org/cookbook/>

<https://www.w3schools.com/css/default.asp>

<https://docs.kogics.net/reference/picture.html>

<https://docs.kogics.net/concepts/computing-essentials.html>

<https://docs.kogics.net/gaming-index.html>

<https://docs.kogics.net/reference/scala.html#data>

<https://www.w3schools.com/js>

<https://www.w3schools.com/jsref>