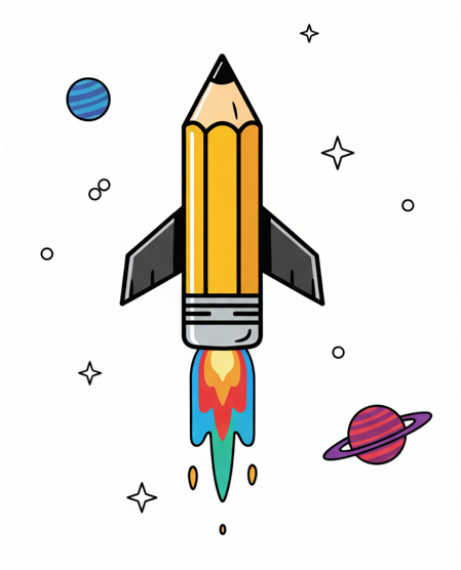
Programmare è un gioco per ragazzi

Introduzione al coding con il nuovo linguaggio Matita

Dino Accoto



Guida Matita per Istruttori

Indice

Prefazione

Matita (*Micro Accessible Tinkering Language for Instant Type-and-Play Applications*) è un linguaggio di programmazione nato da alcune constatazioni.

In primo luogo, esiste un divario tra i linguaggi a blocchi, pensati per i più giovani (ad esempio Kodu e Scratch), e i linguaggi testuali veri e propri (ad esempio Python e Java). È come passare dai LEGO ai sistemi di produzione industriale senza opzioni intermedie: nel mondo della programmazione occorre un equivalente delle stampanti 3D, cioè uno strumento di programmazione che sia testuale, ma di semplice ed immediato utilizzo.

In secondo luogo, l'ambiente di programmazione dei linguaggi più diffusi spesso non è immediatamente fruibile, perché richiede l'installazione e la manutenzione di strumenti complessi. L'uso delle sole primitive geometriche di base richiede spesso l'adozione di una libreria esterna. Un ambiente orientato all'apprendimento, invece, dovrebbe offrire un'esperienza interattiva immediata e favorire un accesso semplice alle funzionalità grafiche, utili per stimolare la curiosità e la sperimentazione. Inoltre, disegnare al computer è anche un'ottima palestra per applicare i concetti di geometria analitica.

Infine, la lingua. I linguaggi di alto livello sono nati per favorire la leggibilità del codice, poiché il linguaggio macchina è molto astratto e poco comprensibile. Tuttavia, gran parte dei linguaggi di alto livello usa l'inglese come lingua di riferimento. Ciò può creare una barriera linguistica a svantaggio degli studenti non anglofoni.

Per ogni comunicazione, scrivere a da@matitacode.com

Introduzione

Matita (*Micro Accessible Tinkering Language for Instant Type-and-Play Applications*) è un linguaggio *open source* progettato per ridurre l'attrito iniziale nell'apprendimento della programmazione.

Il linguaggio persegue l'obiettivo di rendere il codice sorgente leggibile quanto uno pseudo-codice: i comandi si avvicinano ad essere frasi compiute in linguaggio naturale (es. `scrivi "ciao"`, `metti 45 in A`, `fintanto (...) esegui`), permettendo allo studente di concentrarsi sulla logica algoritmica piuttosto che sulla sintassi.

Una caratteristica distintiva è il **bilinguismo nativo**: ogni istruzione italiana ha una controparte inglese (es. `if`/`se`, `while`/`fintanto`) utilizzabile simultaneamente nello stesso file. Ciò consente una transizione graduale verso l'inglese tecnico standard senza cambiare ambiente di sviluppo.

Un aspetto centrale della filosofia di Matita è l'**accesso immediato alla grafica**. L'uso di primitive geometriche non richiede l'importazione di librerie esterne o la configurazione di un ambiete grafico: comandi come `cerchio` o `linea` sono nativi quanto l'addizione. Questo trasforma il linguaggio in uno strumento di visualizzazione geometrica immediata.

Il codice è interpretato riga per riga, con tipizzazione dinamica e feedback immediato, favorendo uno stile procedurale e la prototipazione rapida. Sebbene il linguaggio sia Turing-completo, l'interprete include meccanismi di *loop protection* e limiti alla ricorsione per prevenire il congelamento del browser in caso di errori logici.

L'ambiente di sviluppo (IDE)

L'ambiente di riferimento, **Matita Runtime & IDE**, è tecnicamente una *Single Page Application* (SPA) contenuta in un unico file HTML. L'assenza di dipendenze da sistemi operativi specifici garantisce la totale portabilità: il file `matita.html` può essere distribuito via chiavetta USB o scaricato e utilizzato su qualsiasi computer.

L'interfaccia è suddivisa in tre aree ridimensionabili:

1. **Editor**: con *syntax highlighting* e *auto-completion* contestuale.
2. **Canvas**: un'area grafica vettoriale che gestisce automaticamente la scala e permette l'esportazione in PNG.

3. **Terminale**: per l'I/O testuale e il debugging.

Design e Confronto con altri Linguaggi

Matita si ispira alla filosofia del BASIC per l'immediatezza, ma incorpora concetti tratti da altri linguaggi per risultare moderno e utile per favorire la transizione verso linguaggi industriali.

- **Struttura del Programma** Per favorire il rigore metodologico, Matita distingue esplicitamente tra *procedure* (che eseguono azioni) e *funzioni* (che restituiscono valori). Questa separazione, ispirata al Pascal, aiuta gli studenti a comprendere la differenza semantica tra l'esecuzione di un compito e il calcolo di un risultato. I parametri di una procedura possono essere passati per riferimento.
- **Centralità delle Liste:** La lista è la struttura dati fondamentale di Matita. Essa è utilizzata non solo per collezioni ordinate di dati di tipo omogeneo o eterogeneo, ma anche per coordinate (x, y) di punti e colori (r, g, b).

  L'indicizzazione parte da 1 (come in MATLAB o Julia), una scelta ritenuta più naturale rispetto allo 0-based indexing di molti linguaggi derivati dal C.
- **Gestione dei Dati e Memoria:**
  - **Semantica di Valore:** Il comando `metti A in B` crea una copia indipendente del dato, aumentando la prevedibilità del codice. La sintassi sottolinea la validità della metafora secondo cui una variabile è una scatola con una etichetta (il suo nome) e un contenuto (il suo valore). L'*aliasing*, ossia i riferimenti condivisi alla stessa cella di memoria, è possibile ma deve essere intenzionale tramite il comando `alias`.
  - **Casting e Null:** È possibile convertire esplicitamente i tipi tramite comandi di *casting* (es. da stringa a numero). Inoltre, assegnare `null` a una variabile permette di liberare risorse, introducendo il concetto di pulizia della memoria ma senza la complessità dei puntatori.

In conclusione, Matita è un *serious toy language* che fornisce un ambiente protetto ("*sandbox*") dove sperimentare con pensiero algoritmico, matematica, logica e geometria in preparazione dell'affrontare la complessità dei linguaggi industriali.

1. Basi del Linguaggio

La programmazione è l'attività che porta a realizzare un insieme ordinato di istruzioni che un computer può eseguire per svolgere compiti specifici. A differenza del linguaggio naturale, che può essere ambiguo, la comunicazione con un computer richiede l'uso di un **linguaggio formale** preciso.

Attraverso la scrittura del **codice sorgente** (o semplicemente "codice"), il programmatore implementa degli **algoritmi**, ovvero dei procedimenti logici che trasformano i dati in ingresso (input) nei risultati desiderati (output).

1.1. L'Ambiente di Sviluppo Integrato

Un "Ambiente di Sviluppo Integrato" (in inglese IDE: Integrated Development Environment) è costituito da tutti gli strumenti necessari per scrivere un programma, eseguirlo, correggerlo e visualizzare i risultati.

L'interfaccia utente di Matita presenta tre parti principali:

- **Editor**: L'area dedicata alla stesura del codice sorgente.
- **Area Grafica (Canvas)**: Lo spazio visivo dove vengono renderizzati i disegni e le animazioni.
- **Console**: L'interfaccia testuale per l'output di messaggi e la diagnostica.

Nella parte superiore dell'editor è disponibile un menu con alcuni pulsanti funzionali:

- **Apri / Salva**: Consentono di aprire e salvare i programmi Matita come file di testo con estensione `.txt`.
- **Scarica Matita**: Permette di salvare l'intero ambiente di sviluppo come un singolo file HTML sul proprio computer. Ciò consente di lavorare in locale, senza la necessità di continuare ad usare una connessione internet. Inoltre, vedremo che usando una copia locale diventa possibile lavorare con immagini salvate sul proprio dispositivo.
- **Aiuto**: Apre la documentazione tecnica in una nuova finestra.
- **Esegui**: Il pulsante principale. Invia al computer l'ordine di interpretare ed eseguire le istruzioni.

1.1.1. Scorciatoie da Tastiera

Per velocizzare la scrittura e il test del codice, l'IDE supporta diverse combinazioni di tasti utili:

- `Ctrl + Invio`: Funzione equivalente al pulsante **Esegui**. Avvia immediatamente l'interpretazione del programma corrente.
- `Ctrl + Spazio`: Attiva il sistema di **suggerimenti automatici** (IntelliSense). Se digitato mentre si scrive una parola, apre una lista di comandi compatibili con quanto digitato.
- TAB: Se il menu dei suggerimenti è aperto, il tasto TAB accetta e inserisce automaticamente il comando selezionato nell'editor. Inoltre, TAB è utile per indentare il codice e mantenerlo ordinato.
- ESC: Chiude la finestra dei suggerimenti o altre finestre modali (come l'Area Grafica se in primo piano), restituendo il controllo all'editor.

1.2. I Primi Comandi: `scrivi` e `cancella`

Il comando `scrivi` invia messaggi di testo alla console. Si consideri il seguente esempio:

```
scrivi "Ciao mondo"
```

Nella console apparirà l'output:
`Ciao Mondo`, preceduto dal prompt di sistema `:)`.
Matita esegue il programma *interpretando* le istruzioni sequenzialmente, dalla prima all'ultima:

```
scrivi "A"
scrivi "B"
```

Il codice precedente produrrà l'output AB.
Per inserire un'interruzione di riga, si utilizza il comando `accapo`:

```
scrivi "A"
accapo
```

```
scrivi "B"
```

Il carattere speciale di escape `\n` svolge la medesima funzione di `accapo` all'interno di una stringa. Il programma precedente è equivalente a:

```
scrivi "A\n"
scrivi "B"
```

È possibile concatenare più stringhe utilizzando l'operatore +. Esempio:

```
scrivi "Ciao " + "Mondo.\n"
```

Per ripulire la console da tutto il testo precedente, si utilizza il comando `cancella schermo`.

```
cancella schermo
scrivi "Prima riga."
accapo
scrivi "Seconda riga."
```

Un elemento sintattico fondamentale è il carattere #, che indica l'inizio di un **commento**. Il testo che segue questo carattere viene ignorato dall'interprete. I commenti sono utili per documentare il codice:

```
# questo è un commento.
metti 31 in GiorniMarzo # numero di giorni nel mese di
    marzo
# Mostriamo quanti giorni ha marzo:
scrivi "Marzo ha " + GiorniMarzo + " giorni."
```

Nota

Evidenziazione della sintassi e suggerimenti L'IDE di Matita impiega un codice colore per evidenziare i diversi elementi sintattici di un programma. Inoltre, offre suggerimenti come aiuto mnemonico per l'impiego dei costrutti.

1.2.1. Comandi multipli su una riga

Per inserire più istruzioni sulla stessa riga, è necessario separarle con un punto e virgola (;):

```
scrivi "ciao."; accapo
```

1.2.2. Singolo comando su più righe

Qualora un comando fosse troppo lungo per una singola riga, è possibile spezzarlo utilizzando una barra rovesciata, ossia il carattere "\" chiamato *backlash* in inglese, al termine della riga. L'interprete considererà le righe separate da "\" come un'unica istruzione:

```
scrivi \
"istruzione su due linee"
```

1.3. Variabili

Una variabile rappresenta un'area di memoria identificata da un nome simbolico (etichetta), destinata a contenere un valore (numero, testo, o strutture dati complesse). L'uso delle variabili permette di scrivere programmi che operano su dati dinamici, astraendo dal valore specifico contenuto.

Per dichiarare una variabile e assegnarle un valore (inizializzazione), si utilizza il comando: `metti ... in NomeVariabile`.

Si consideri il seguente esempio:

```
metti 5 in A
metti "Informatica" in B
scrivi "Il valore in A è " + A
accapo
scrivi B
```

L'output prodotto sarà:

```
:) Il valore in A è 5
:) Informatica
```

Si noti che le stringhe di testo devono essere racchiuse tra virgolette, ad esempio "Informatica".

Per leggere il contenuto di una variabile, è sufficiente scrivere il nome della variabile all'interno di un'espressione (come nell'esempio precedente con A e B). In altre parole, il nome è l'*identificatore* che permette di accedere al valore memorizzato.

In Matita i nomi delle variabili sono **case-insensitive**: `Nome`, `nome`, `nOmE`, `NOME` fanno riferimento alla stessa locazione di memoria. Un identificatore valido può contenere numeri, ma deve iniziare con una lettera. Caratteri accentati o speciali non sono ammessi per definire il nome.

Per aggiornare il valore di una variabile, si utilizza nuovamente il comando `metti`.

```
metti 100 in salute
scrivi "Salute iniziale: " + salute
accapo
metti salute - 15 in salute # Il giocatore subisce 15
    danni
scrivi "Stato aggiornato. La salute ora è: " + salute
```

1.4. Input da utente: `chiedi`

Il comando `chiedi ... dicendo ...` attiva una finestra di dialogo (prompt) che consente all'utente di inserire dati tramite tastiera. Il valore immesso viene memorizzato nella variabile specificata.

```
chiedi NomeGiocatore dicendo "Inserire nome utente:"
scrivi "Nome registrato: " + NomeGiocatore + ". Procedura
     avviata."
```

2. Matematica

2.1. Operazioni aritmetiche e Precisione

Le espressioni possono essere valutate e assegnate direttamente alle variabili.

```
metti 10 in A
metti 5 in B
metti A+B in Somma
metti A*B in Prodotto
scrivi "A + B = " + Somma; accapo
scrivi "A * B = " + Prodotto
```

L'operatore mod ("*modulo*") restituisce il resto della divisione intera. Ad esempio, `10 mod 3` restituisce 1.

```
cancella schermo
metti 39 in A
metti 7 in B
metti A mod B in C
scrivi "Il resto di " + A + "/" + B + " è " + C
```

2.2. Operatori ed espressioni

In Matita un'espressione è una combinazione di operandi (numeri, testi, liste, punti) e operatori. Il risultato può essere assegnato a una variabile o utilizzato come condizione logica.

- **Valori booleani**: `vero`, `falso`
- **Operatori logici**: e (AND), o (OR), non (NOT)
- **Confronti**:
  - Uguaglianza: `uguale a`, `vale`, `uguale`

  - Disuguaglianza: `diverso da`
  - Confronti numerici: `<`, `>`, `<=`, `>=`
- **Operatori aritmetici**: `+`, `-`, `*`, `/`
- **Resto**: `mod`
- **Potenza**: `^` (es. $2\hat{}3$)
- **Negazione (unario)**: `-X`

Esempio potenza:

```
metti 2^3 in Cubo
scrivi Cubo # Output: 8
```

2.2.1. Costanti e funzioni matematiche

Matita include diverse costanti e funzioni matematiche pronte all'uso. Una lista completa delle funzioni predefinite è disponibile nell'Appendice A.

- **Costanti**: `pi` (o π), `infinito`, `nan` (Not a Number)
- **Funzioni trigonometriche e calcolo**: `abs(x)`, `sin(x)`, `cos(x)`, `radice(x)`, ecc.
- **Funzioni di arrotondamento**:
  - `arrotonda(x)`: Arrotonda all'intero più vicino.
  - `tronca(x)`: Rimuove la parte decimale senza arrotondare.

Esempio di troncamento:

```
scrivi tronca(3.9) # Output: 3
accapo
scrivi tronca(-3.9) # Output: -3
```

2.2.2. Generazione di Numeri Casuali

La generazione di numeri casuali è essenziale per simulazioni e giochi. Matita offre diverse opzioni tramite i comandi `casuale` e `dado`:

- `casuale()`: Senza argomenti, restituisce un numero reale compreso tra 0.0 (incluso) e 1.0 (escluso).
- `casuale(MAX)`: Restituisce un numero reale compreso tra 0.0 e MAX.
- `casuale(MIN, MAX)`: Restituisce un numero reale compreso tra MIN e MAX.
- `dado(N)`: Il comando simula il lancio di un dado con N facce, per cui restituisce un **numero intero** compreso tra 1 e N, estremi inclusi.

Esempi:

```
metti casuale() in Probabilita # es. 0.453
metti casuale(100) in Percentuale # es. 87.21
metti casuale(10, 20) in Temperatura # es. 14.5
metti dado(6) in Lancio # es. 4 (dà un numero intero tra
    1 e 6)
```

2.2.3. Definire la Precisione Numerica

Per controllare il numero di cifre decimali visualizzate nel comando `scrivi`, è possibile utilizzare il comando `decimali`. Questo comando influisce solo sulla *visualizzazione* dell'output in console, non sul valore interno della variabile.

```
decimali 2
metti 10 / 3 in Risultato
scrivi Risultato # Output: 3.33
accapo
decimali 5
scrivi Risultato # Output: 3.33333
```

2.2.4. Notazione Scientifica: `scientifica`

Quando Matita gestisce numeri molto grandi, il sistema può passare automaticamente alla visualizzazione in **notazione scientifica** per migliorare la leggibilità. Il formato utilizzato è "`numero e esponente`".

Per default, la soglia per l'attivazione della notazione scientifica è `1e6`. È possibile modificare questa soglia utilizzando il comando `scientifica(N)`.

Se si imposta anche un valore per `decimali`, questo influenzerà il numero di cifre mostrate nella mantissa.

```
# Impostiamo la soglia bassa per vedere l'effetto subito
scientifica(100)
metti 150 in Valore

scrivi Valore
# Output: :) 1.5e+2

# Cambiamo i decimali visualizzati nella mantissa
decimali 4
scrivi Valore
# Output: :) 1.5000e+2
```

2.2.5. Precedenza degli operatori

1. `o`
2. `e`
3. Confronti (`uguale a`, `diverso da`, `<`, ecc.)
4. `+`, `-`
5. `*`, `/`, `mod`
6. `^` (Potenza)
7. Accessi a proprietà o elementi (es. `x di P`, `elemento N di LISTA`)

3. Liste

La `lista` è la struttura dati fondamentale in Matita che permette di memorizzare una sequenza ordinata di elementi. A differenza di una variabile semplice, che contiene un singolo dato, una lista contiene molteplici dati accessibili tramite la loro posizione, detta *indice*.
Le liste in Matita sono eterogenee, ossia possono contenere tipi di dati diversi, e dinamiche, ossia la loro dimensione può variare durante l'esecuzione.

3.1. Creazione

Per istanziare una lista, si raggruppano gli elementi tra parentesi tonde:

```
metti (10, 20, 30) in miaLista
```

Per maggiore chiarezza semantica, è possibile utilizzare la parola chiave `lista`:

```
metti lista(10, 20, 30) in miaLista
metti lista() in ListaVuota
```

3.2. Operazioni sulle Liste

Matita fornisce un set di istruzioni per la manipolazione delle liste.

3.2.1. Accesso ai singoli Elementi

Per leggere un valore in una posizione specifica, si utilizza la sintassi `elemento ... di ...`. **L'indicizzazione in Matita è 1-based**: il primo elemento si trova all'indice 1, il secondo all'indice 2, ecc.

```
metti (100, 200, 300) in dati
metti elemento 2 di dati in Valore # Valore ora contiene
    200
scrivi Valore # Stampa 200
```

3.2.2. Determinazione della Dimensione

Per ottenere la cardinalità della lista (numero di elementi), si usa l'operatore `dimensione di`.

```
metti ("rosso", "verde", "blu", "giallo") in ListaColori
metti dimensione di ListaColori in N # N ora contiene 4
scrivi ListaColori + " sono " + N + " colori\n"
scrivi "il secondo colore è " + elemento 2 di ListaColori
```

L'output sarà:

```
:) rosso,verde,blu,giallo sono 4 colori
:) il secondo colore è verde
```

3.2.3. Modifica

Le liste sono mutabili attraverso specifici comandi.

Aggiunta di elementi: Il comando `aggiungi` inserisce un nuovo elemento in coda alla lista. Sintassi: `aggiungi <valore> a <nomeLista>`

```
metti lista(1, 2) in numeri
aggiungi 3 a numeri # numeri ora è (1, 2, 3)
```

Inserimento in una posizione specifica: Per inserire un valore in una posizione arbitraria, facendo slittare gli elementi successivi, si utilizza `inserisci ... in elemento ...`. Sintassi: `inserisci <valore> in elemento <indice> di <nomeLista>`

```
metti ("A", "C") in Lettere
inserisci "B" in elemento 2 di Lettere
```

```
scrivi Lettere
# Output: :) (A, B, C)
```

Rimozione in testa o in coda:

```
metti ("rosso", "verde", "blu", "giallo") in ListaColori
rimuovi primo elemento da ListaColori
scrivi ListaColori
accapo
rimuovi ultimo elemento da ListaColori
scrivi ListaColori
```

Output:

```
:) (verde, blu, giallo)
:) (verde, blu)
```

Rimozione per indice:

```
cancella schermo
metti ("A","B","C","D") in ListaLettere
rimuovi elemento 3 da ListaLettere
scrivi ListaLettere
```

Output: `:) (A, B, D)`

Aggiornamento di un elemento: Per sovrascrivere un valore in una data posizione, si combina `metti` con l'accesso all'elemento.

```
metti (10, 20, 30) in Dati
scrivi Dati; accapo
metti 99 in elemento 2 di Dati
scrivi Dati
```

Output:

```
:) (10, 20, 30)
:) (10, 99, 30)
```

3.2.4. Iterazione

Per eseguire un blocco di istruzioni per ogni elemento della lista, si utilizza il costrutto `per ogni`. La variabile specificata (es. n) assume in sequenza il valore di ogni elemento della lista. **Nota**: La variabile iteratrice contiene una *copia* del valore. Modificare n all'interno del ciclo non modifica la lista originale.

```
metti (1, 2, 3) in Numeri
per ogni n in Numeri esegui (
  metti n * 2 in Doppio
  scrivi Doppio
  accapo
)
# Output:
# :) 2
# :) 4
# :) 6
```

4. Controllo del Flusso

4.1. Costrutto `se`

Le strutture di controllo condizionali permettono di eseguire blocchi di codice solo al verificarsi di determinate condizioni. L'istruzione fondamentale è `se`. La sintassi è: `se (CONDIZIONE) allora ( ... ) altrimenti ( ... )`.

Per le condizioni si utilizzano gli operatori di confronto: `uguale a`, `vale`, >, <, >=, <=, !=.

Si consideri l'esempio seguente, che verifica l'idoneità per la patente:

```
cancella schermo
chiedi Eta dicendo "Quanti anni hai?"
se (Eta >= 18) allora (
  scrivi "È possibile ottenere la patente."
) altrimenti (
  scrivi "Devi aspettare ancora " + (18 - Eta) + " anni."
)
```

4.2. Cicli

I cicli (o loop) permettono di iterare un blocco di istruzioni più volte, ottimizzando la stesura del codice.

4.2.1. Iterazione Definita

Per eseguire istruzioni un numero prestabilito di volte, si utilizza il ciclo `indice`. Viene creata una variabile contatore locale (spesso denominata `i`) che viene incrementata automaticamente. Sintassi: `indice i va da INIZIO a FINE ed esegui ( ... )`

```
indice i va da 1 a 5 ed esegui (
  scrivi "Iterazione numero " + i
  accapo
)
```

Output:

```
:) Iterazione numero 1
:) Iterazione numero 2
:) Iterazione numero 3
:) Iterazione numero 4
:) Iterazione numero 5
```

4.2.2. Iterazione Indefinita

Questo ciclo viene eseguito fintanto che la condizione specificata rimane vera. È idoneo quando il numero di iterazioni non è noto a priori. Sintassi: `fintanto (CONDIZIONE) esegui ( ... )`

Esempio di conto alla rovescia:

```
cancella schermo
metti 10 in conto_alla_rovescia
fintanto (conto_alla_rovescia > 0) esegui (
  scrivi conto_alla_rovescia
  accapo
  metti conto_alla_rovescia - 1 in conto_alla_rovescia
  aspetta 1000 ms
)
scrivi "Avvio."
```

Il programma decrementa la variabile ogni secondo e termina quando la condizione `conto_alla_rovescia > 0` diventa falsa.

4.3. Interrompere l'esecuzione: `stop`

Il comando `stop` arresta immediatamente l'esecuzione dell'intero programma. È spesso utilizzato all'interno di una struttura condizionale per terminare un ciclo o un gioco al verificarsi di una determinata condizione.

```
se (ViteRimaste uguale a 0) allora (
  scrivi "Game Over"
  stop
)
```

4.4. Gestione del Tempo

4.4.1. Aspettare del tempo: `wait`

L'esecuzione delle istruzioni è sequenziale e immediata. Tuttavia, in contesti come le animazioni, è necessario introdurre delle pause. Il comando `aspetta` sospende l'esecuzione per un determinato intervallo. L'unità di misura è il millisecondo (ms). Esempio:

```
aspetta 45 ms
```

Questa istruzione impone al processore una pausa di 45 millisecondi prima di passare all'istruzione successiva.

4.4.2. Cronometro: `tick` e `tock`

Per misurare il tempo trascorso, Matita fornisce un cronometro integrato.

- `tick`: È un comando che avvia o resetta il cronometro al momento attuale.
- `tock`: È una variabile speciale che contiene il numero di millisecondi trascorsi dall'ultimo comando `tick`.

Esempio di misurazione:

```
scrivi "Misurazione avviata..."
accapo
tick # Avvia il cronometro
aspetta 1500 ms
scrivi "Tempo trascorso: " + tock + " ms"
```

Nota: Se si tenta di leggere il valore di `tock` senza aver prima eseguito `tick`, il programma segnalerà un errore.

5. Manipolazione di Testo e Numeri

Matita offre semplici comandi per analizzare e trasformare stringhe e numeri trattandoli come liste di caratteri o cifre.

5.1. Trasformare una stringa in lista

Il comando `scomponi` divide una stringa o un numero nei suoi singoli caratteri, restituendo una lista.

```
metti "Ciao" in Testo
metti scomponi(Testo) in ListaCaratteri
scrivi ListaCaratteri
# Output: :) (C, i, a, o)
```

5.2. Scomposizione Specifica

Il comando `scomposizione(valore, TIPO)` offre un controllo maggiore rispetto a `scomponi`. I tipi target disponibili sono: `CIFRE`, `CARATTERI` e `PAROLE`.

```
# Dividere una frase in parole
metti "Matita è divertente" in Frase
metti scomposizione(Frase, "PAROLE") in ListaParole
scrivi ListaParole
# Output: :) (Matita, è, divertente)

# Dividere un numero in cifre
metti 123.45 in Numero
metti scomposizione(Numero, "CIFRE") in ListaCifre
scrivi ListaCifre
# Output: :) (1, 2, 3, ., 4, 5)
```

5.3. Composizione

L'operazione inversa alla scomposizione è la `composizione(lista, TIPO)`, che unisce gli elementi di una lista in un singolo valore. I tipi target disponibili sono `TESTO` e `NUMERO`.

```
metti ("M", "a", "r", "e") in Lettere
metti composizione(Lettere, "TESTO") in Parola
scrivi Parola
# Output: :) Mare

# Creare un numero da cifre
metti (3, ".", 1, 4) in Cifre
metti composizione(Cifre, "NUMERO") in Valore
scrivi Valore * 2 # Possiamo fare calcoli perché è un
    numero
# Output: :) 6.28
```

5.4. Conversione di Tipo (Casting)

Talora è necessario trattare un numero come se fosse testo, ad esempio per concatenarlo ad altre parole, o -viceversa- per usarlo in un calcolo matematico. Il comando `cast` esegue l'operazione di conversione di tipo. Il termine *casting* deriva dall'inglese *"to cast"*, ossia gettare in uno stampo, ed è universalmente diffuso nei linguaggi di programmazione per indicare la trasformazione del tipo di un dato.

- `cast(NUMERO)`: Restituisce una **stringa** contenente il numero.
- `cast(STRINGA)`: Restituisce un **numero** (supporta interi positivi, negativi, decimali e notazione scientifica).

Esempio:

```
cancella schermo
# Test da Stringa a Numero
metti "10" in S1
metti "3.14" in S2
metti "-50" in S3
metti "1.2e3" in S4  # 1200 in notazione scientifica

metti cast(S1, "numero") + cast(S2, "numero") + cast(S3,
```

```
    "numero") + cast(S4, "numero") in Totale
scrivi "Somma dei cast: " + Totale
accapo

# Test da Numero a Stringa (Concatenazione)
metti 500 in N1
metti 12.5 in N2
# Somma le stringhe (concatenazione) invece dei numeri
scrivi "Concatenazione: " + cast(N1, "testo") + cast(N2,
    "testo")

# Output atteso:
# :) Somma dei cast: 1163.14
# :) Concatenazione: 50012.5
```

5.5. Funzione Unicode

Unicode è uno standard internazionale che assegna un codice numerico univoco a ogni carattere o simbolo usato nei sistemi informatici, incluse le lettere di tutti gli alfabeti, e le Emoji.

Il comando `unicode(x)` agisce come un ponte tra numeri e simboli:

- **Da Numero a Simbolo**: Se l'argomento è un numero, restituisce il carattere associato.
- **Da Simbolo a Numero**: Se l'argomento è una stringa, restituisce il codice numerico del primo carattere.

Esempio:

```
# Esempio 1: Generare un simbolo dal codice (Razzo)
metti unicode(128640) in SimboloRazzo
scrivi "Verso l'infinito... " + SimboloRazzo
# Output: :) Verso l'infinito...
accapo

# Esempio 2: Ottenere il codice da un simbolo (Faccina)
metti unicode("[icona]") in CodiceFaccina
scrivi "Il codice della faccina è: " + CodiceFaccina
# Output: :) Il codice della faccina è: 128578
```

6. Procedure e Funzioni

L'utilizzo di cicli consente di risolvere molti problemi. Tuttavia, all'aumentare della complessità, diventa necessario organizzare il codice in modo *modulare* per favorirne leggibilità e manutenibilità.

Matita supporta la programmazione procedurale permettendo la definizione di comandi personalizzati. Per evitare la duplicazione del codice e incapsulare la logica, si utilizzano le **Procedure** e le **Funzioni**:

- Una **Procedura** è un blocco di istruzioni che esegue un compito ben definito. Si raggruppano in procedure quelle istruzioni che possono essere utilizzate in più punti del codice. Le procedure possono applicare le stesse operazioni su diversi parametri, forniti come input. Una procedura esegue istruzioni ma non produce un valore in uscita.
- Una **Funzione** esegue operazioni e restituisce un valore (output) utilizzabile in altre espressioni tramite il comando `restituisci`.

Importante: L'interprete Matita legge ed esegue il codice riga per riga. Pertanto, è fondamentale che ogni procedura o funzione siano definite **prima** della riga in cui avviene la prima chiamata.

6.1. Procedure

Esempio di definizione e chiamata di una procedura:

```
# Definizione
definisci procedura saluta(nome) (
  scrivi "Ciao, " + nome + ". Benvenuto in Matita."
  accapo
)

# Chiamata
saluta("Alex")
saluta("Maria")
```

6.2. Funzioni

Esempio di definizione e chiamata di una funzione:

```
definisci funzione Raddoppia(numero) (
  metti numero * 2 in risultato
  restituisci risultato
)

metti 7 in Numero
metti Raddoppia(Numero) in Doppio
scrivi "Il doppio di " + Numero + " è " + Doppio
accapo
```

Nota

Costrutti identificativi e funzioni built-in. Alcune espressioni sono *identificative*: richiamano una parte o una proprietà di qualcosa che esiste già (es. `x di puntoP`, `elemento 3 di miaLista`, `dimensione di miaLista`). Altre espressioni sono *generative* e producono un nuovo valore (funzioni built-in), ad esempio `punto(10,20)`, `lista(1,2,3)`, `sin(angolo)`, `radice(9)`.

6.3. Ricorsione

La ricorsione è una tecnica di programmazione in cui una funzione invoca se stessa. È particolarmente efficace per risolvere problemi che possono essere suddivisi in sottoproblemi identici ma di dimensione inferiore. L'esempio classico è il calcolo del fattoriale ($n!$).

```
definisci funzione fattoriale(n) (
  # Caso base: condizione di terminazione.
  se (n uguale a 0) allora (
    restituisci 1
  ) altrimenti (
    # Passo ricorsivo
    restituisci n * fattoriale(n - 1)
  )
```

```
)

metti 5 in Numero
metti fattoriale(Numero) in Risultato
scrivi Numero + "! = " + risultato # Mostrerà 120
```

Se la catena di chiamate ricorsive è eccessiva, si verifica un overflow dello stack (limite di ricorsione superato).

7. Gestione della memoria

7.1. Scope delle variabili

Lo **scope** è l'ambito di visibilità di una variabile. Nei blocchi di codice annidati (dentro `esegui`, `fintanto`, `se`, `per ogni`, ecc.), le variabili create con `metti` sono **locali** al blocco stesso.

Le variabili definite nel flusso principale sono dette **globali**. Una variabile globale è leggibile all'interno di un blocco locale, ma se si tenta di assegnare un valore a una variabile con lo stesso nome nel blocco locale, viene creata una nuova variabile locale che "nasconde" quella globale (*shadowing*). In questi casi, Matita emette un avviso:

```
:) Avviso: La variabile locale 'Numero' sta nascondendo una
```

Si consideri il seguente esempio:

```
cancella schermo
metti 1 in Numero
fintanto (Numero < 6) esegui (
  metti Numero + 1 in Numero
  scrivi Numero
  accapo
)
# ora siamo fuori dal blocco locale
scrivi "Valore finale: " + Numero
```

Output:

```
:) Avviso: La variabile locale 'Numero' sta nascondendo una
:) 2
:) 3
:) 4
:) 5
:) 6
:) Valore finale: 1
```

Come si evince, la variabile `Numero` all'interno del ciclo è distinta da quella esterna. Al termine del ciclo, la variabile globale mantiene il suo valore originale, ossia 1.

7.1.1. La direttiva `globale`

Per modificare una variabile globale all'interno di uno scope locale, è necessario dichiararla esplicitamente con la direttiva `globale`:

```
cancella schermo
metti 1 in Numero
fintanto (Numero < 6) esegui (
  globale Numero
  metti Numero + 1 in Numero
  scrivi Numero
  accapo
)
# ora siamo fuori dal blocco
scrivi "Valore finale: " + Numero
```

Output:

```
:) 2
:) 3
:) 4
:) 5
:) 6
:) Valore finale: 6
```

È possibile dichiarare più variabili globali contemporaneamente separandole con una virgola:

```
globale Nome1, Nome2 # più variabili separate da virgole
```

7.2. Semantica per valore

L'interprete Matita adotta, per impostazione predefinita, una **Semantica di Valore** (Value Semantics). Ogni operazione di assegnazione tramite il comando `metti` comporta una **Deep Copy** (copia profonda) del dato.

Ciò significa che assegnare una lista o un oggetto complesso a una nuova variabile crea un clone indipendente e distinto in memoria.

```
metti (1, 2, 3) in A
metti A in B      # B è una copia clale di A
metti 99 in elemento 1 di B
# A rimane (1, 2, 3), B diventa (99, 2, 3)
```

Questa scelta progettuale rende la gestione dei dati sicura e prevedibile.

Accesso per riferimento: `alias`

In diversi linguaggi, fra cui Python e Java, l'istruzione `B = A` fa sì che entrambe le variabili puntino allo stesso oggetto in memoria. Si dice che tali linguaggi adottano implicitamente una **semantica di riferimento** e A e B sono alias, ossia repliche, una dell'altra. Un alias non contiene dunque dati propri, ma agisce come un puntatore verso un'altra variabile target.

Per chiarezza, Matita adotta di default la **semantica di valore**, analogamente al C e C++. In Matita l'istruzione `metti A in B` **duplica** in B il valore di A, effettuando una *copia profonda* di A. Il risultato sarà che esisteranno due variabili distinte, ma col medesimo contenuto iniziale.

Per creare un riferimento condiviso, ad esempio per permettere la condivisione dei dati o l'ottimizzazione delle risorse, Matita introduce il concetto di **aliasing esplicito**.

Un alias viene creato tramite richiesta esplicita attraverso il comando: `B alias A`, dove A è una variabile esistente.

Qualsiasi operazione di lettura o scrittura eseguita su B viene intercettata dall'interprete e reindirizzata sulla variabile A. Il seguente codice:

```
metti 4 in A
metti 56 in (B alias A)
scrivi "A: " + A + "; B: " + B
```

produce come output:

```
:)  A: 56; B: 56
```

7.2.1. Passaggio per Riferimento a Procedure: `inout`

Matita supporta due modalità di passaggio dei parametri ad una procedura.

1. **Passaggio per valore (default):** Il valore dell'argomento viene copiato nello scope locale della procedura. Le modifiche non si propagano al chiamante.

2. **Passaggio per riferimento (tramite la parola chiave *inout*):** Utilizzando la parola chiave `inout`, l'interprete crea un **alias locale** che punta alla variabile passata dal chiamante.

```
definisci procedura Incrementa(inout X) (
    metti X + 1 in X  # Modifica direttamente la variabile esterna
)
```

7.2.2. Liberare memoria: `null`

La parola chiave `null` indica un valore speciale corrispondente ad *assenza di dati*. Tale parola chiave consente di svuotare una variabile.

Scrivendo ad esempio `metti null in VAR`, il nome VAR viene scollegato dal suo contenuto precedente, il che segnala al computer che quel contenuto non serve più e la memoria che occupava può essere liberata.

L'uso di `null` è utile per liberare la memoria del computer quando oggetti pesanti, come immagini o liste molto lunghe, non sono più necessari.

8. Elementi Grafici

L'area grafica utilizza un sistema di riferimento cartesiano. La posizione di ogni elemento è definita da una coppia di coordinate (x, y). L'origine $(0, 0)$ si trova al centro dell'area. Valori positivi di x indicano posizioni a destra, valori negativi a sinistra. Valori positivi di y indicano posizioni in alto, valori negativi in basso.

Le coordinate sono rappresentate come liste di due numeri. È possibile utilizzare la sintassi esplicita `punto(x, y)`. Per accedere alle singole componenti di un punto, si utilizzano le proprietà `x di` e `y di`.

```
metti punto(10, 44) in MioPunto
metti x di MioPunto in CoordinataX
scrivi CoordinataX
```

Per modificare una singola coordinata, si usa il comando `imposta`. Questa operazione è fondamentale per gestire il movimento degli oggetti.

```
# Sposta il punto 10 passi in giù senza cambiare la x
imposta y di MioPunto a (y di MioPunto - 10)
scrivi y di MioPunto # Mostrerà 34
```

8.1. Colori e sfondi

Matita utilizza il modello di colore RGB. I colori sono definiti tramite una lista di tre valori (R, G, B), rispettivamente rappresentanti le percentuali di Rosso, Verde e Blu, variabili da 0 a 100, estremi inclusi.

Esempio con impostazione dello sfondo:

```
metti (68, 85, 90) in bluCielo
imposta colore bluCielo
```

```
dipingi sfondo
```

Per cancellare il contenuto dell'area grafica, ad esempio prima di disegnare un nuovo fotogramma in un'animazione, si utilizza il comando `cancella grafica`.

8.2. Disegnare Forme di Base

Dopo aver impostato il colore corrente, è possibile tracciare primitive grafiche.

Cerchio: Sintassi `cerchio raggio R in POS`. Disegna un disco pieno.

```
imposta colore (100, 87, 0)
cerchio raggio 60 in (120, 80)
```

Circonferenza: Sintassi `circonferenza raggio R spessore S in POS`. Disegna solo il contorno.

```
imposta colore (0, 0, 0)
circonferenza raggio 25 spessore 4 in (-200, -50)
```

Linea: Sintassi `linea spessore S da P1 a P2`. Traccia un segmento tra due punti.

```
imposta colore (0, 0, 0)
linea spessore 2 da (-200, -75) a (-200, -160)
```

Arco: Sintassi `arco raggio R spessore S da P1 a P2`. Disegna un arco di cerchio di dato raggio, passante per i due punti dati. Dal momento che esistono al più due circonferenze di dato raggio passanti per due punti assegnati nel piano, il codice disegnerà l'arco che va dal primo al secondo punto ruotando in verso antiorario. Ciò implica che, se si inverte l'ordine dei due punti, si otterrà un arco a concavità invertita.

```
imposta colore (100, 0, 0)
arco raggio 240 spessore 12 da (-220, -160) a (220, -160)
```

Esempio composito:

```
cancella grafica

metti 15 in S       # Spessore di ogni arco
metti 260 in R      # Raggio del colore più esterno (Rosso)
metti -160 in Y     # Altezza dell'orizzonte (dove finisce il prato)

# --- CIELO E SFONDO ---
imposta colore (68, 85, 90)
dipingi sfondo

# --- PRATO ---
imposta colore (16, 65, 27)
rettangolo da (-360, -250) a (360, Y)

# --- SOLE ---
imposta colore (100, 87, 0)
cerchio raggio 70 in (250, 180)

# --- ARCOBALENO ---
# 1. Rosso
imposta colore (90, 0, 0)
arco raggio R spessore S da (R, Y) a (-R, Y)
metti R - S in R  # Rimpicciolisco il raggio per il prossimo colore

# 2. Arancione
imposta colore (90, 50, 0)
arco raggio R spessore S da (R, Y) a (-R, Y)
metti R - S in R

# 3. Giallo
imposta colore (90, 90, 0)
arco raggio R spessore S da (R, Y) a (-R, Y)
metti R - S in R

# 4. Verde
imposta colore (0, 70, 0)
arco raggio R spessore S da (R, Y) a (-R, Y)
metti R - S in R

```

```
# 5. Blu
imposta colore (0, 0, 90)
arco raggio R spessore S da (R, Y) a (-R, Y)
metti R - S in R

# 6. Indaco / Viola
imposta colore (40, 0, 70)
arco raggio R spessore S da (R, Y) a (-R, Y)

# --- PALLONCINO ---
imposta colore (80, 0, 90) # Un viola più acceso
cerchio raggio 25 in (-100, -50)
linea spessore 2 da (-100, -75) a (-100, Y)
```

8.3. Graffiti (Testo Grafico)

Il comando `graffiti` permette di inserire testo direttamente nell'area grafica.

```
imposta colore (10, 10, 10)
graffiti grandezza 30 in punto(0, 0) "Centro."
```

8.4. Ridisegnare una scena

Il comando `ridisegna` forza l'aggiornamento dell'area grafica (canvas). Questo comando è essenziale per visualizzare i cambiamenti avvenuti nella scena memorizzata in memoria.

8.5. Creare Animazioni

Per ottenere il movimento e l'animazione degli oggetti in Matita, si sfrutta il principio della persistenza della visione: una serie di immagini statiche, leggermente diverse tra loro, mostrate in rapida successione, crea l'illusione del movimento. La struttura tipica di un ciclo di animazione prevede i seguenti passaggi:

1. Iniziare un ciclo (solitamente `fintanto (vero)`).

2. Cancellare la grafica precedente (con `cancella grafica` o `dipingi sfondo`).
3. Aggiornare le coordinate degli oggetti (ad esempio spostando un punto).
4. Disegnare gli oggetti nelle nuove posizioni.
5. Forzare l'aggiornamento grafico con `ridisegna` (opzionale se si usa `aspetta`, che lo fa implicitamente).
6. Attendere un breve intervallo di tempo (es. `aspetta 16 ms` per circa 60 fotogrammi al secondo).

Esempio di animazione semplice:

```
metti 0 in X
fintanto (X < 300) esegui (
  metti X + 2 in X
  cancella grafica
  cerchio raggio 20 in punto(X, 0)
  aspetta 16 ms
)
```

8.6. Input da tastiera

Per realizzare interazioni in tempo reale, si verifica lo stato dei tasti. L'espressione `tasto "NOME_TASTO" viene premuto` restituisce `vero` se il tasto indicato è attualmente premuto. I tasti direzionali sono mappati come UP, DOWN, LEFT, RIGHT. Altri tasti di controllo: SPACE, ENTER, ESC, TAB, BACKSPACE, SHIFT, CTRL, ALT. I caratteri alfanumerici si indicano semplicemente con il loro simbolo ("A", "x", ecc.).

Esempio di controllo di un oggetto tramite tastiera:

```
metti punto(0,0) in posGiocatore
metti 300 in lim

fintanto (vero) esegui (

  se (tasto "UP" viene premuto) allora (
    imposta y di posGiocatore a (y di posGiocatore + 5)
  )
```

```
  se (tasto "DOWN" viene premuto) allora (
    imposta y di posGiocatore a (y di posGiocatore - 5)
  )
  se (tasto "RIGHT" viene premuto) allora (
    imposta x di posGiocatore a (x di posGiocatore + 5)
  )
  se (tasto "LEFT" viene premuto) allora (
    imposta x di posGiocatore a (x di posGiocatore - 5)
  )

  # Uscita di emergenza
  se (tasto "x" viene premuto) allora (stop)

  # Disegno
  cancella grafica
  imposta colore (0, 100, 100)
  rettangolo da (-lim, -lim) a (lim, lim)

  imposta colore (0, 50, 100)
  cerchio raggio 20 in posGiocatore

  ridisegna

  # Refresh rate (circa 60 fps)
  aspetta 16 ms
)
```

8.7. Etichette grafiche

Per ottimizzare il codice e riutilizzare gruppi di forme, è possibile definire delle **etichette**. Un'etichetta raggruppa una serie di istruzioni di disegno in un unico oggetto riutilizzabile. Sintassi: `etichetta (...) come NOME`.

Esempio di definizione e utilizzo:

```
# Definizione del modello
etichetta (
  # Corpo
  imposta colore (100, 100, 100) # Bianco
  cerchio raggio 30 in (0, 0)
  cerchio raggio 20 in (0, 45)
  cerchio raggio 15 in (0, 75)

  # Dettagli
  imposta colore (0, 0, 0) # Nero
```

```
    cerchio raggio 2 in (-5, 78)
    cerchio raggio 2 in (5, 78)
    cerchio raggio 2 in (0, 50)
    cerchio raggio 2 in (0, 40)

    # Naso
    imposta colore (100, 60, 0) # Arancione
    triangolo con vertici (0, 75), (0, 73), (8, 74)
) come PupazzoDiNeve

imposta colore (50,50,50)
dipingi sfondo
disegna PupazzoDiNeve in punto(150, 0)
disegna PupazzoDiNeve in punto(-100, 0)
```

8.8. Trasformazioni: Scala e Rotazione

Gli oggetti etichettati possono subire trasformazioni geometriche.

8.8.1. Scalatura

Il comando `scala` ridimensiona l'etichetta. Un fattore 2 raddoppia le dimensioni, 0.5 le dimezza. **Nota**: L'operazione di scalatura è permanente per l'etichetta su cui viene applicata.

```
scala PupazzoDiNeve di 0.5
disegna PupazzoDiNeve in punto(0, 100)
disegna PupazzoDiNeve in punto(0, -100)
```

8.8.2. Rotazione

È possibile ruotare un oggetto rispetto al centro del suo bounding box. L'angolo è espresso in gradi sessagesimali (0-360), con rotazione antioraria per valori positivi.

```
ruota PupazzoDiNeve di 45 gradi e posiziona in (300, -300)
```

8.9. Immagini

È possibile importare file immagine esterni per utilizzarli come oggetti grafici o sfondi. I file devono risiedere nella stessa directory del file `matita.html`.

8.9.1. Immagini come Etichette

Un'immagine caricata viene trattata come un'etichetta e può quindi essere disegnata, scalata e ruotata.

```
importa "NomeFile.png" come gattino
disegna gattino in (0, 0)
scala gattino di 0.5
ruota gattino di 15 gradi e posiziona in (-120, -60)
```

8.9.2. Immagini come Sfondi

Per impostare un'immagine come sfondo dell'intera area grafica:

```
carica "erba.png" scala 0.5 come sfondo
dipingi sfondo
```

8.10. Esercizio guidato: faccina interattiva

Il seguente codice integra definizione di etichette, input utente e ciclo di rendering per creare un oggetto controllabile.

```
# Definiamo i colori
metti (100, 100, 100) in Bianco
metti (0, 0, 0) in Nero
metti (100, 90, 10) in Giallo

# Dimensioni dello sfondo bianco
metti 300 in lim

# Definiamo la Faccina come oggetto grafico (etichetta)
etichetta (
```

```
  imposta colore Giallo
  cerchio raggio 120 in punto(0, 0)
  imposta colore Nero
  circonferenza raggio 120 spessore 8 in punto(0, 0)
  cerchio raggio 12 in punto(-40, 30)
  cerchio raggio 12 in punto(40, 30)
  arco raggio 60 spessore 8 da (-50, -25) a (50, -25)
) come Faccina

# Inizializzazione variabili
metti punto(0, 0) in Posizione
metti 0 in Angolo

cancella schermo
scrivi "Controlli: Tasti Freccia (Muovi), A/D (Ruota), X
    (Esci)"
accapo

# Ciclo principale di gioco
fintanto (vero) esegui (
  # Variabili globali che modifichiamo nel loop
  globale Posizione, Angolo

  se (tasto "x" viene premuto) allora (stop)

  # Movimento
  se (tasto "UP" viene premuto) allora (
    imposta y di Posizione a (y di Posizione + 5)
  )
  se (tasto "DOWN" viene premuto) allora (
    imposta y di Posizione a (y di Posizione - 5)
  )
  se (tasto "LEFT" viene premuto) allora (
    imposta x di Posizione a (x di Posizione - 5)
  )
  se (tasto "RIGHT" viene premuto) allora (
    imposta x di Posizione a (x di Posizione + 5)
  )

  # Rotazione
  se (tasto "a" viene premuto) allora (
    metti Angolo + 5 in Angolo
  )
  se (tasto "d" viene premuto) allora (
    metti Angolo - 5 in Angolo
  )

  # Disegno del frame corrente
  cancella grafica
  imposta colore Bianco
```

```
    rettangolo da (-lim, -lim) a (lim, lim)
    ruota Faccina di Angolo gradi e posiziona in Posizione

    ridisegna
    aspetta 16 ms
)
```

A. Riferimento Rapido di Matita

Di seguito viene presentato un riepilogo schematico di tutti i comandi disponibili nel linguaggio Matita v0.3.1.

A.1. Comandi Generali e Sistema

Comando	Descrizione
`metti VALORE in NOME`	Assegna un valore a una variabile (locale per default nei blocchi).
`globale NOME`	Dichiara che le variabili specificate fanno riferimento allo scope globale.
`VAR2 alias VAR1`	Crea un alias: VAR2 punta alla stessa cella di memoria di VAR1.
`scrivi VALORE`	Stampa un valore nella console di output.
`accapo`	Inserisce un'interruzione di riga nella console.
`cancella schermo`	Pulisce tutto il testo presente nella console.
`aspetta N ms`	Sospende l'esecuzione del programma per N millisecondi.
`stop`	Interrompe immediatamente l'esecuzione del programma.

A.2. Matematica e Numeri

Comando / Funzione	Descrizione
`decimali N`	Imposta il numero di cifre decimali visualizzate con `scrivi`.
`scientifica(N)`	Imposta la soglia sopra la quale i numeri usano la notazione scientifica (es. 1e+6).
`casuale()`	Restituisce un numero decimale tra 0.0 e 1.0.
`casuale(MAX)`	Restituisce un numero decimale tra 0 e MAX.
`casuale(MIN, MAX)`	Restituisce un numero decimale tra MIN e MAX.
`dado(N)`	Restituisce un numero **intero** tra 1 e N.
`arrotonda(X)`	Arrotonda X all'intero più vicino.
`tronca(X)`	Rimuove la parte decimale di X senza arrotondare.
`radice(X)`	Calcola la radice quadrata di X.
`abs(X)`	Restituisce il valore assoluto di X.
`sin(X) / cos(X)`	Funzioni trigonometriche (input in radianti).
`min(Lista) / max(Lista)`	Trova il valore minimo o massimo in una lista o serie di argomenti.

A.3. Logica e Controllo di Flusso

Comando	Descrizione
`se (C) allora (...) altrimenti (...)`	Esecuzione condizionale. Il ramo `altrimenti` è opzionale.
`indice I va da A a B ed esegui (...)`	Ciclo definito (For). Incrementa o decrementa I automaticamente.
`per ogni E in L esegui (...)`	Ciclo Foreach. Esegue il blocco per ogni elemento E nella lista L.
`fintanto (C) esegui (...)`	Ciclo indefinito (While). Ripete finché la condizione C è vera.

A.4. Input Utente e Tempo

Comando	Descrizione
`chiedi NOME dicendo "TESTO"`	Mostra un prompt all'utente e salva la risposta nella variabile NOME.
`tasto "T" viene premuto`	Restituisce `vero` se il tasto specificato è attualmente premuto.
`tick`	Avvia o resetta il cronometro interno.
`tock`	Variabile speciale: restituisce i millisecondi trascorsi dall'ultimo `tick`.

A.5. Funzioni e Procedure

Comando	Descrizione
`definisci procedura NOME(p) (...)`	Definisce una procedura (azione senza ritorno).
`definisci procedura NOME(inout p) (...)`	Definisce una procedura con passaggio parametri per riferimento.
`definisci funzione NOME(p) (...)`	Definisce una funzione. Deve contenere `restituisci`.
`restituisci VALORE`	Esce dalla funzione e restituisce il valore al chiamante.
`NOME(arg1, arg2)`	Chiamata standard a procedura o funzione.
`NOME con (arg1, arg2)`	Sintassi alternativa per la chiamata di procedura.

A.6. Grafica: Forme e Colori

Comando	Descrizione
`imposta colore (R, G, B)`	Imposta il colore corrente (valori 0-100).
`cancella grafica`	Rimuove tutti gli oggetti dal canvas.
`dipingi sfondo`	Riempie lo sfondo con il colore corrente.
`ridisegna`	Forza l'aggiornamento grafico (utile nelle animazioni).
`cerchio raggio R in (X,Y)`	Disegna un cerchio pieno.
`circonferenza raggio R spessore S...`	Disegna il contorno di un cerchio.
`rettangolo da (X1,Y1) a (X2,Y2)`	Disegna un rettangolo pieno.
`linea spessore S da (X1,Y1) a (X2,Y2)`	Disegna un segmento.
`triangolo con vertici P1, P2, P3`	Disegna un triangolo pieno.
`arco raggio R spessore S da P1 a P2`	Disegna un arco di cerchio.
`graffiti grandezza G in (X,Y) "TESTO"`	Scrive del testo nell'area grafica.

A.7. Grafica: Oggetti e Immagini

Comando	Descrizione
`etichetta (...) come NOME`	Raggruppa una sequenza di istruzioni grafiche in un oggetto riutilizzabile.
`importa "file.png" come NOME`	Carica un'immagine esterna come oggetto grafico.
`carica "file.png" scala K come sfondo`	Imposta un'immagine come sfondo del canvas.
`disegna NOME in (X,Y)`	Disegna un'istanza dell'etichetta o dell'immagine.
`scala NOME di K`	Ridimensiona permanentemente l'oggetto NOME.
`ruota NOME di G gradi e posiziona in P`	Disegna l'oggetto ruotato nella posizione specificata.

A.8. Liste e Manipolazione Dati

Comando	Descrizione
`metti lista(A, B, ...) in NOME`	Crea una lista con gli elementi specificati.
`dimensione di NOME`	Restituisce il numero di elementi nella lista.
`elemento N di NOME`	Legge il valore alla posizione N (partendo da 1).
`metti X in elemento N di NOME`	Sostituisce il valore all'indice N.
`aggiungi X a NOME`	Inserisce X in coda alla lista.
`inserisci X in elemento N di NOME`	Inserisce X alla posizione N, traslando gli altri.
`rimuovi elemento N da NOME`	Rimuove l'elemento in posizione N.
`rimuovi primo/ultimo elemento da NOME`	Rimuove la testa o la coda della lista.
`scomponi(VALORE)`	Divide una stringa/numero in una lista di singoli caratteri.
`scomposizione(VAL, TIPO)`	Divide in base al TIPO (`CIFRE`, `CARATTERI`, `PAROLE`).
`composizione(LISTA, TIPO)`	Unisce una lista in un `NUMERO` o `TESTO`.
`cast(VALORE)`	Converte un Numero in Stringa e viceversa.
`unicode(X)`	Converte Codice Numerico ↔ Simbolo.

B. Errori Comuni e Soluzioni

B.1. Risoluzione dei Problemi

- **"Non trovo una variabile di nome ..."**: Verificare che la variabile sia stata inizializzata con `metti VALORE in NOME` prima di essere letta e controllare eventuali errori di battitura.
- **"Cambio una variabile in un ciclo, ma dopo il ciclo ha il suo vecchio valore."**: È un problema di *scope*. Per modificare una variabile esterna da un blocco locale (ciclo o procedura), utilizzare la direttiva `globale NOME` all'inizio del blocco.
- **"L'indice ... non è valido per la lista."**: Tentativo di accesso a un indice minore di 1 o maggiore della dimensione corrente della lista. Ricorda che in Matita gli indici partono da 1.
- **"Coordinata non valida..."**: Sintassi errata nel parametro di posizione grafica. Usare `in (x, y)` oppure `in punto(x,y)`.
- **"Il loop ha superato il numero massimo di iterazioni."**: È stato rilevato un probabile ciclo infinito (es. `fintanto (vero)...` senza `stop`). Verificare la condizione di uscita.
- **"L'immagine importata non si carica."**: Verificare che il nome del file sia corretto e che risieda nella stessa cartella del file `matita.html` (se usato in locale) o che l'URL sia accessibile.
- **"Parentesi non bilanciate."**: C'è una disparità tra parentesi aperte e chiuse. Controllare anche di aver chiuso tutte le virgolette delle stringhe.
- **"Argomenti non validi per la chiamata di procedura."**: Utilizzare la sintassi `Nome(arg1, arg2)` oppure `Nome con (arg1, arg2)`. Verificare che il numero di argomenti corrisponda alla definizione.
- **"La funzione ... ha terminato senza un 'return'."**: Il flusso del programma è arrivato alla fine di una funzione senza incontrare `restituisci`. Assicurarsi che tutti i percorsi logici (anche dentro i `se`/`altrimenti`) restituiscano un valore.

- **"Avviso: La variabile locale 'Nome' sta nascondendo una variabile globale..."**: Conflitto di nomi (Shadowing). Stai creando una nuova variabile locale con lo stesso nome di una esistente. Rinominare la variabile locale o usare `globale Nome` se l'intento era modificare quella esterna.

B.2. Tabella dei Messaggi di Errore

Di seguito la tabella dei messaggi diagnostici generati dall'interprete Matita.

Tipo	Significato
Avviso	Shadowing di variabile globale in scope locale senza direttiva `globale`.
Avviso	Alias creato (aliasing): due nomi di variabile puntano ora alla stessa cella di memoria.
Errore	Variabile non trovata (non inizializzata o errore di battitura).
Errore	Funzione o Procedura già esistente (tentativo di ridefinizione).
Errore	Limite di ricorsione superato (Stack Overflow).
Errore	Argomenti non validi per la chiamata di procedura (sintassi).
Errore	Testo inatteso dopo il nome della procedura.
Errore	Parentesi non corrispondenti (bilanciamento errato).
Errore	Numero di argomenti errato per la procedura/funzione (Arity Mismatch).
Errore	Funzione terminata senza il comando `restituisci`.
Errore	Durata dell'attesa (`aspetta`) non valida (deve essere ≥ 0).
Errore	Operazione consentita solo su liste applicata a un tipo diverso.
Errore	Indice della lista non valido (Out of Bounds).
Errore	Tentativo di rimuovere elementi da una lista vuota.
Errore	Tentativo di impostare proprietà (x, y) su una variabile non punto.
Errore	Valore per una coordinata non numerico.
Errore	Il messaggio per `chiedi` deve essere un testo.
Errore	Colore non valido (usare lista r,g,b con valori 0-100).
Errore	Nome file sfondo non valido (deve essere stringa).

Continua nella pagina successiva...

Tipo	Significato (continua)
Errore	Fattore di scala sfondo non valido (deve essere positivo).
Errore	Impossibile caricare l'immagine (file non trovato o corrotto).
Errore	Parametri forma geometrica non numerici.
Errore	Raggio o spessore non positivi.
Errore	Sintassi `graffiti` malformata.
Errore	Contenuto graffito non testuale.
Errore	Sintassi `etichetta` o `importa` malformata.
Errore	Etichetta già esistente o etichetta non trovata.
Errore	Angolo di rotazione o scala non numerici.
Errore	Loop `per ogni` applicato a un non-lista.
Errore	Loop infinito rilevato (superato limite iterazioni).
Errore	Comando sconosciuto o sintassi non riconosciuta.
Errore	Coordinate non valide (usare punto o lista x,y).
Errore	Triangolo richiede esattamente 3 vertici.
Errore	Impossibile usare `tock`: cronometro non avviato con `tick`.

Nota

In alcuni casi rari, se l'errore è molto specifico o interno al motore JavaScript, il messaggio potrebbe apparire in inglese. Fare riferimento alla traduzione italiana documentata in questa guida per la risoluzione.