

Guida Utente MALT

Sommario

- Guida Utente MALT
 - Sommario
- Introduzione
- Installazione
 - Esecuzione da codice sorgente
 - Installazione tramite Docker
- Interfaccia web di MALT
 - Funzionalità dell'interfaccia
 - Uso dell'interfaccia
- Linguaggio MALT
 - Vincoli e limitazioni
 - Variabili
 - Commenti
 - Tipi
 - Variabili Primitive Testuali
 - Variabili Lista
 - Variabili Multimediali
 - Variabili primitive testuali
 - Testo semplice (*text*)
 - Titolo e varianti
 - Identificativo del titolo
 - Blocco citazione (*blockquote*)
 - Blocco di codice (*codeblock*)
 - Concatenazione di variabili primitive testuali
 - Formattazione di variabili primitive testuali
 - Grassetto e corsivo
 - Testo cancellato e evidenziato
 - Pedici e apici
 - Variabili lista
 - Lista ordinata (*olist*)
 - Lista non ordinata (*ulist*)
 - Lista con caselle di spunta (*tlist*)
 - Lista (*list*)
 - Manipolazione di una lista
 - Aggiungere un elemento in coda (*push*)
 - Rimuovere un elemento (*remove*)
 - Variabili multimediali

- Tabella (*table*)
- Link (*link*)
- Immagine (*img*)
- Format
 - Specificatori di stringa
- Conversione tra Tipi (Type Casting)
- Cicli
 - Ciclo for
 - Ciclo for-each
- Scope delle variabili
- Funzioni (*fun*)
 - Scope delle variabili in una funzione
- Classi (*class*)

Introduzione

MALT è un linguaggio che consente di ottenere un file Markdown a partire da un nuovo linguaggio procedurale.

Markdown è un linguaggio di markup e non consente l'utilizzo di liste, cicli, classi, funzioni e tutte quelle funzionalità che sono presenti nei linguaggi di programmazione. MALT permette di unire le funzionalità tipiche dei linguaggi di programmazione con la potente formattazione e strutturazione del testo tipica invece dei linguaggi di markup.

Per poter eseguire il codice MALT è stata predisposta un'interfaccia web in cui è possibile scrivere direttamente oppure importare un file che contiene il codice che si vuole eseguire.

Installazione

Per installare ed eseguire l'ambiente che permette di eseguire il codice MALT sono disponibili diverse alternative: l'esecuzione in modalità sviluppatore direttamente dai script contenuti nel codice sorgente oppure l'esecuzione dell'ambiente pronto alla produzione tramite container Docker di immagini già costruite e pronte da utilizzare.

Esecuzione da codice sorgente

L'esecuzione da codice sorgente è consigliabile solo per utenti che conoscono Java, Maven, Node.js e NPM.

Un modo per utilizzare l'ambiente è quello di eseguire gli script utilizzati per provare il programma durante la fase di sviluppo. Questa modalità però non permette di simulare un ambiente reale con tutte le ottimizzazioni che i framework usano dopo aver eseguito il "build". Il programma si divide in due parti: analizzatore (back-end) e interfaccia (front-end).

Per il front-end è stato utilizzato Next.js che richiede l'installazione di Node.js e di npm sul sistema. Per scaricare Node.js basta andare a [questo indirizzo](#). L'installazione di Node.js solitamente già include npm.

Per il back-end è prima necessario avere installato Maven e Java JDK sul proprio PC.

Il back-end sfrutta la grammatica MALT per poter analizzare il codice. Per questo motivo bisogna prima installare *malt.jar* come repository Maven globale nel proprio PC.

Per fare ciò basta entrare nella cartella "malt_backend" ed eseguire il file *script.bat* se si sta utilizzando Windows oppure il file [script.sh](#) nel caso si stia utilizzando Linux.

Una volta installata la dipendenza, sempre nella cartella "malt_backend", è possibile eseguire il comando per avviare il back-end

```
./mvnw spring-boot:run
```

Verrà lanciato un server realizzato con Spring disponibile alla porta 8000.

Per lanciare l'interfaccia invece basta entrare nella cartella "malt_frontend" ed installare le dipendenze necessarie (solo la prima volta) con il comando

```
npm install
```

e successivamente eseguire il server con il comando

```
npm run dev
```

Una volta avviato il front-end è possibile utilizzare l'interfaccia all'indirizzo <http://localhost:3000>

Installazione tramite Docker

Le immagini Docker sono un modo pratico e veloce per creare dei container che contengono già tutto il codice e le dipendenze necessarie per eseguire l'ambiente MALT senza installare nulla oltre al motore Docker.

Per completare l'installazione tramite Docker è necessario avere Docker Engine installato sul proprio computer. Se non si è esperti con i comandi Docker all'interno del terminale si consiglia caldamente di installare l'interfaccia Docker Desktop che integra il sopracitato Docker Engine. I passaggi successivi della guida faranno riferimento all'interfaccia. La guida di installazione di Docker Desktop può essere trovata a questo [link](#).

Prima di eseguire i comandi per scaricare le immagini, assicurarsi di avere in esecuzione in background Docker Engine. Se avete installato Docker Desktop basterà avviarlo e in automatico verrà eseguito anche Docker Engine.

Per scaricare l'immagine più recente del backend che esegue l'analisi lessicale, sintattica e semantica eseguire il comando

```
docker pull ghcr.io/kevin-bernardi/malt_backend:main
```

Invece, per scaricare l'immagine più recente del frontend, eseguire il comando

```
docker pull ghcr.io/kevin-bernardi/malt_frontend:main
```

Se non ci sono stati errori durante il download delle immagini dovrebbero essere comparse due immagini nella sezione "Images" di Docker Desktop.

Per creare ed eseguire i container, sempre dalla sezione "Images" fare click sul tasto "Play" in fondo alla riga dell'immagine malt_backend. Aprire la sezione "Optional settings", inserire un nome a piacere nel campo "Container name" e inserire obbligatoriamente 8080 nel campo "Host port". Il resto va lasciato vuoto. Fare la stessa cosa per l'immagine malt_frontend specificando questa volta una "Host port" a piacere.

Nella sezione "Containers" ora si potranno vedere i due container in esecuzione.

L'installazione dell'ambiente è completata ed ora sarà possibile utilizzare l'interfaccia web all'url localhost:PORT dove PORT è la "Host port" specificata nella creazione del container basato sull'immagine malt_frontend.

Interfaccia web di MALT

Funzionalità dell'interfaccia

Per poter usufruire al meglio di MALT, è stata creata un'interfaccia web che consente di eseguire il parser e/o lo scanner su un testo. Questo testo può essere digitato manualmente oppure può essere estratto da un file caricato nell'apposito spazio dell'interfaccia. Inoltre l'interfaccia consente di consultare questa guida utente in modo da supportare l'utente nelle analisi che desidera eseguire.

Uso dell'interfaccia

Una volta eseguita l'installazione dell'ambiente (descritta nel capitolo precedente), si può aprire l'interfaccia web all'url localhost:PORT.


Come prima pagina viene visualizzata l'homepage (in figura 1), in cui viene mostrato un esempio di utilizzo di MALT e dei risultati che esso può produrre. A questa pagina si può sempre tornare cliccando sul pulsante *MALT* della barra di navigazione in cima alla pagina. Per procedere con l'esecuzione concreta del parser e/o dello scanner, si può cliccare sul pulsante nella barra di navigazione in cima alla pagina oppure su quello in fondo alla pagina stessa entrambi denominati *EDITOR*.

 Homepage dell'interfaccia web di MALT *Figura 1*

Una volta entrato nella pagina *EDITOR* (in figura 2), l'utente come primo passaggio può caricare un file (con estensione *.malt*) contenente il testo da analizzare tramite il pulsante *CARICA CODICE DA FILE* oppure può scrivere direttamente il testo nell'apposito campo. Cliccando invece il pulsante *PULISCI* viene svuotata l'area di testo. Successivamente per iniziare le analisi del testo, l'utente può eseguire il parser cliccando sul pulsante *ESEGUI PARSER* oppure può eseguire il scanner cliccando sul pulsante *ESEGUI SCANNER*.

 Editor dell'interfaccia web di MALT *Figura 2*

Nel caso in cui si desidera eseguire il parser, dopo aver premuto il pulsante *ESEGUI PARSER* si visualizzano a video eventuali messaggi risultanti dalla rispettiva analisi (un esempio è mostrato nella figura 3). Inoltre al di sotto di questa area di testo è possibile decidere se mostrare la tabella globale e/o le tabelle locali di variabili, funzioni e classi.

 Parser dell'interfaccia web di MALT *Figura 3*

Invece nel caso in cui si desidera eseguire lo scanner, dopo aver premuto il pulsante *ESEGUI SCANNER* compare a video un'area di testo contenente l'analisi effettuata dallo scanner (l'applicazione dello scanner al precedente esempio è mostrata nella figura 4).

 Scanner dell'interfaccia web di MALT *Figura 4*

In caso di necessità, l'utente ha sempre la possibilità di consultare la guida utente di MALT (in figura 5) tramite il pulsante *GUIDA* nella barra di navigazione in cima a ciascuna pagina.

 Guida dell'interfaccia web di MALT *Figura 5*

Linguaggio MALT

Nei seguenti capitoli verranno esposte le funzionalità del linguaggio in ordine di difficoltà partendo dalle variabili e arrivando alla fine ad utilizzare tutto quello che serve per affrontare i casi d'uso più comuni.

Vincoli e limitazioni

In MALT sono presenti alcuni vincoli e limitazioni:

- le funzioni, le classi e i metodi delle classi vengono eseguiti nel momento in cui vengono definiti e non nel momento delle loro chiamate;
- i cicli vengono eseguiti soltanto per un'iterazione e non per tutte le iterazioni che soddisfano la condizione del ciclo stesso;
- non è ancora disponibile la generazione del testo Markdown.

Variabili

La dichiarazione di variabili avviene in questo modo:

```
tipo nome_variabile = valore;
```

Ecco un esempio:

```
text t = "sono un testo";
```

Analizziamo parola per parola questa istruzione:

- *text*: tipo della variabile. *text* rappresenta testo semplice (vedi sotto quali sono i tipi disponibili)
- *t*: nome della variabile
- *"sono un testo"*: valore assegnato alla variabile

Il nome della variabile deve iniziare con una lettera e può contenere solo lettere, numeri e underscore (_).

Il simbolo '=' permette di assegnare il valore presente dopo il simbolo di uguaglianza alla variabile presente prima del simbolo.

Ogni istruzione deve terminare con il simbolo ';'.

Riassumendo, l'istruzione permette di assegnare il valore *"sono un testo"* alla variabile *t* di tipo *text*.

Il valore assegnabile ad una variabile dipende dal suo tipo.

Una sintassi alternativa consiste nell'assegnare il valore della variabile in un momento successivo rispetto alla sua dichiarazione al posto di fare tutto sulla stessa riga:

```
text t;
```

```
// ALTRO CODICE...
```

```
t = "testo";
```

Commenti

MALT permette di inserire sia commenti su un'unica riga sia commenti multilinea.

Nel primo caso la notazione è:

```
// commento...
```

Invece nel caso di commenti multilinea serve scrivere:

```
/* commenti...  
commenti... */
```

Tipi

MALT è un linguaggio un po' diverso dagli altri linguaggi classici. Qui non esistono variabili che contengono valori numerici (int, float, double...) perché MALT è incentrato completamente sulla manipolazione testuale.

Invece di avere molti tipi che gestiscono valori numerici, qui esistono molti tipi che gestiscono stringhe di testo. Come visto sopra, per assegnare il tipo di una variabile basta scriverlo prima del nome nella dichiarazione.

I tipi sono organizzati in 3 diverse categorie che verranno esposte di seguito.

Variabili Primitive Testuali

Tipo	Descrizione
text	Testo semplice
title	Titolo
s1title	Sotto titolo 1
s2title	Sotto titolo 2
s3title	Sotto titolo 3
s4title	Sotto titolo 4
s5title	Sotto titolo 5
blockquote	Blocco citazione
codeblock	Blocco di codice

Variabili Lista

Tipo	Descrizione
olist	Lista ordinata
ulist	Lista non ordinata
tlist	Lista con casella di spunta
list	Lista / Array

Variabili Multimediali

Tipo	Descrizione
table	Tabella
link	Link
image	Immagine

Nei successivi paragrafi verrà spiegato l'utilizzo di tutti i tipi appena elencati.

Variabili primitive testuali

Vengono definite variabili primitive testuali tutte quelle variabili a cui sono assegnabili direttamente una stringa di testo. Il tipo `text` visto prima ne è un esempio.

Le stringhe di testo possono contenere lettere, numeri, simboli, spazi, ...

Le stringhe devono essere racchiuse da doppie virgolette (").

L'aggettivo "primitive" si riferisce al fatto che questi tipi possono contenere una ed una sola stringa di testo e non ad esempio una lista di stringhe o una tabella che contiene stringhe.

I tipi delle variabili primitive testuali sono i seguenti:

- `text`
- `title`
- `s1title`
- `s2title`
- `s3title`
- `s4title`
- `s5title`
- `blockquote`
- `codeblock`

L'assegnamento per queste variabili funziona in questo modo:

```
tipo_pr_test nome_var = "valore testuale";
```

dove *tipo_pr_test* è uno dei tipi elencati sopra.

Per `codeblock` esiste un parametro opzionale in più che può essere inserito tra il tipo e il nome della variabile nella dichiarazione. Consulta la sezione dei `codeblock` per saperne di più.

Come già detto, l'assegnazione e la dichiarazione posso essere separate.

Verranno ora presentati in dettaglio questi tipi.

Testo semplice (*text*)

text rappresenta del testo semplice, tutto quello che non è un titolo, una tabella, un'immagine, un codeblock...

Titolo e varianti

I tipi *title*, *s1title*, *s2title*, *s3title*, *s4title*, *s5title* fanno parte della categoria dei titoli.

Tra MALT e Markdown c'è la seguente corrispondenza:

MALT	Markdown
title t = "Titolo";	# titolo
s1title t = "Titolo";	## titolo
s2title t = "Titolo";	### titolo
s3title t = "Titolo";	#### titolo
s4title t = "Titolo";	##### titolo
s5title t = "Titolo";	##### titolo

Identificativo del titolo

È possibile aggiungere un identificativo al titolo inserendo `{{#id}}` tra il nome della variabile e il simbolo "=" come in questo esempio:

```
s1title titolo {#titolo} = "Titolo con identificativo";
```

Blocco citazione (*blockquote*)

Un *blockquote* è un blocco di testo dove viene inserita una citazione o del testo importante che si deve notare.

```
blockquote bc = "Testo importante!";
```

corrisponde in Markdown a:

```
< Testo importante!
```

che verrà mostrato come:

```
| Testo importante!
```

Blocco di codice (*codeblock*)

`codeblock` rappresenta un blocco di codice con supporto a syntax highlighting. La dichiarazione di un `codeblock` si differenzia dalle altre variabili primitive testuali per la presenza di un parametro opzionale nella dichiarazione. Questo parametro opzionale serve per specificare il linguaggio che viene scritto all'interno del blocco di codice per effettuare syntax highlighting in modo corretto.

Ecco un esempio di un blocco di codice con codice Java al suo interno:

```
codeblock java cb = "  
    public class Malt {  
        public static void main(String[] args) {  
            System.out.println("Testo");  
        }  
    }  
";
```

Visto che il parametro per la specifica del linguaggio è opzionale avremmo anche potuto scrivere

```
codeblock cb = "  
    ....  
";
```

senza ovviamente sfruttare il syntax highlighting.

Concatenazione di variabili primitive testuali

In MALT è possibile eseguire la concatenazione di più variabili primitive testuali (sia tramite il nome di queste variabili sia tramite il loro valore) durante un assegnamento seguendo la notazione del seguente esempio:

```
text t1 = ("zero" + "zero"); // t1 vale "zero zero"  
  
text s = "uno";  
text t2 = (s + "due"); // t2 vale "uno due"  
  
blockquote p = "tre";  
s3title t3 = (s + p); // t3 vale "uno tre"  
// tutti i tipi primitivi testuali possono essere utilizzati in una concaten  
// MALT converte automaticamente i tipi  
// Ulteriori dettagli sulla conversione dei tipi è spiegata nell'apposito pa
```



Formattazione di variabili primitive testuali

Il valore delle variabili primitive testuali può essere formattato seguendo le stesse regole di Markdown.

Grassetto e corsivo

- Grassetto: ****** o **__** attorno al testo;
- Corsivo: *** o *_* attorno al testo;
- Grassetto e corsivo: ******* o **___** attorno al testo;

```
text t = "*Stringa* **con** ***formattazione***";
```

corrisponde in Markdown a:

Stringa **con formattazione**

Testo cancellato e evidenziato

- Testo cancellato: ~~~~~~ attorno al testo;
- Testo evidenziato: **==** attorno al testo;

```
text t = "~~Testo cancellato~~ e ==testo evidenziato==";
```

corrisponde in Markdown a:

~~Testo cancellato~~ e testo evidenziato

Pedici e apici

- Pedice: _~ attorno al pedice;
- Apice: [^] attorno all'apice;

```
text t1 = "Formula bruta del glucosio: C ~6~ H ~12~ O ~6~";
text t2 = "Isotopi dell'idrogeno: prozio ^1^H, deuterio ^2^H, trizio ^3^H";
```

```
// gli spazi attorno ai simboli ~ e ^ non sono necessari
```



corrispondono in Markdown a:

Formula bruta del glucosio: C₆ H₁₂ O₆

Isotopi dell'idrogeno: prozio ¹H, deuterio ²H, trizio ³H

Variabili lista

Vengono definite variabili lista tutte quelle variabili che hanno una lista di testi come valore.

Quindi, al contrario delle variabili primitive testuali, le variabili lista non contengono direttamente una stringa ma una lista di una o più stringhe.

I tipi che definiscono le variabili lista sono le seguenti:

- *oList*: lista ordinata

- *ulist*: lista non ordinata
- *tlist*: lista con casella di spunta
- *list*: lista pura

Tra i quattro tipi non ci sono differenze per quanto riguarda la loro dichiarazione o assegnamento.

All'interno della lista possono esserci soltanto stringhe o variabili primitive testuali, che a loro volta verranno risolte in stringhe.

Ecco un esempio:

```
s3title numero_due = "due";  
olist lista_ordinata = ["uno", numero_due, "tre"];  
  
// corrisponde a ["uno", "due", "tre"]
```

Andiamo a vedere più nel dettaglio i vari tipi per variabili lista.

Lista ordinata (*olist*)

```
olist animali = ["cane", "gatto", "coniglio", "gallina"];
```

in Markdown viene trasformato in:

1. cane
2. gatto
3. coniglio
4. gallina

Lista non ordinata (*ulist*)

```
ulist frutta = ["mela", "pera", "pesca", "arancia"];
```

in Markdown viene trasformato in:

- mela
- pera
- pesca
- arancia

Lista con caselle di spunta (*tlist*)

```
tlist cose_da_fare = ["pulire", "_x_lavorare", "mangiare", "dormire"];
```

in Markdown viene trasformato in:

- ☐ pulire
- ☒ lavorare
- ☐ mangiare
- ☐ dormire

Automaticamente, i valori della lista sono convertiti in righe senza spunta.

Aggiungendo il simbolo speciale `_x_` all'inizio della stringa viene visualizzata la spunta.

Lista (*list*)

Rispetto a *olist*, *ulist* e *tlist*, la *List* non viene trasformata in una lista in Markdown ma è una lista "pura", una lista che viene solamente utilizzata per raccogliere un insieme di stringhe da usare all'interno del linguaggio in funzioni, cicli ecc...

Manipolazione di una lista

MALT permette di aggiungere un elemento in coda e di rimuovere un elemento dato il suo indice.

Aggiungere un elemento in coda (*push*)

Utilizzando la funzione `push(Lst, eL)` è possibile aggiungere la variabile primitiva testuale *eL* alla lista *Lst*.

```
olist animali = ["cane", "gatto"];
text nuovo_elemento = "gallina";

push(animali, nuovo_elemento);

// animali ora contiene ["cane", "gatto", "gallina"]
```

Rimuovere un elemento (*remove*)

Utilizzando la funzione `remove(Lst, idx, res)` è possibile rimuovere dalla lista *Lst* il valore di indice *idx* e salvare l'elemento rimosso nella variabile *res*.

Se si vuole rimuovere l'ultimo elemento senza specificare l'indice basta inserire il simbolo `"_"` nel campo *idx*.

Se non si vuole salvare l'elemento estratto basta usare il simbolo `"_"` nel campo *res*.

```
ulist animali = ["cane", "gatto", "gallina"];

text rimosso;
remove(animali, 1, rimosso);

// animali ora contiene ["cane", "gallina"] e rimosso contiene "gatto"
```

```
ulist animali = ["cane", "gatto", "gallina"];

text rimosso;
remove(animali, _, rimosso); // rimuovo l'ultimo elemento

// animali ora contiene ["cane", "gatto"] e rimosso contiene "gallina"
```

```
ulist animali = ["cane", "gatto", "gallina"];

remove(animali, _, _); // rimuovo l'ultimo elemento e non lo salvo

// animali ora contiene ["cane", "gatto"]
```

Variabili multimediali

Le variabili multimediali sono tre tipi di variabili che permettono di aggiungere tre nuovi contenuti al documento Markdown: tabelle, link e immagini.

Tabella (*table*)

Una tabella in MALT viene dichiarata e utilizzata nel modo seguente:

```
table tabella = [$l,$c,$r] ([ "uno", "due", "tre"], [ "quattro", "cinque", "sei"], [
```



che viene trasformata in Markdown come:

uno	due	tre
quattro	cinque	sei
sette	otto	nove

La dichiarazione è simile a quanto visto fino ad ora: prima dell'uguale viene dichiarata una variabile *tabella* di tipo *table*.

Il valore assegnato è invece composto da due parti.

La prima parte è una lista che deve contenere un numero di specificatori pari al numero di colonne della tabella. Gli specificatori descrivono come devono essere allineati i testi all'interno delle celle della colonna corrispondente.

Gli specificatori sono 3:

Specificatore	Descrizione
\$l	Allineamento a sinistra
\$c	Allineamento al centro
\$r	Allineamento a destra

Questa prima parte è interamente opzionale. Se non viene specificata, viene utilizzato l'allineamento a sinistra per tutte le celle della tabella.

La seconda parte invece descrive il contenuto della tabella. Tra le parentesi tonde vengono scritte tante liste quante sono le righe della tabella ed ogni lista è lunga tanto quanto il numero di colonne della tabella. La prima lista corrisponde all'intestazione della tabella.

Tutte le liste devono avere la stessa lunghezza (non ci possono essere righe con un numero diverso di colonne dalle altre)

Link (*link*)

Un link si definisce nel modo seguente:

```
link link1 =1 ("https://google.it", "Google");
```

corrisponde in Markdown a:

[Google](https://google.it)

(Questo link potrebbe non funzionare in alcuni formati di visualizzazione. Nei casi in cui funziona, corrisponde ad un link che porta a Google)

N.B. Per l'assegnamento di link si utilizza =1 invece di =

Il primo parametro è l'url a cui il link deve reindirizzare. Il secondo parametro è il testo visibile del link.

Immagine (*img*)

Un'immagine si definisce come mostrato di seguito:

```
img foto =i ("/percorso/immagine.jpg","Disasalia immagine");  
img foto_web =i ("https://sitofoto.com/foto_albero.png", "Didasalia immagin
```

N.B. Per l'assegnamento di immagini si utilizza `=i` invece di `=`

Il primo parametro è un percorso locale se si desidera utilizzare un'immagine presente sul computer. Mentre se si desidera utilizzare un'immagine non locale, si può inserire l'url della foto come nella seconda riga dell'esempio.

Il secondo parametro permette di specificare una didascalia dell'immagine.

Format

In MALT è stato introdotto un nuovo tipo di istruzione che consente di inserire testo che può essere formattato diversamente in base ai specificatori utilizzati. Un esempio di questa istruzione è:

```
text str = "MALT è %i nuovo %b di programmazione.";  
text a = "un";  
text b = "linguaggio";  
format(res,str,a,b)
```

che consente di ottenere:

MALT è *un* nuovo **linguaggio** di programmazione.

Nella format del precedente esempio, *res* corrisponde alla variabile in cui salvare il risultato di format mentre *str* rappresenta la stringa da formattare (che contiene gli specificatori). Invece *a* e *b* sono le variabili che vanno sostituite ai specificatori presenti in *str* seguendo l'ordine in cui sono riportati.

Il numero di specificatori nella stringa da formattare deve corrispondere al numero di variabili riportati nella format (esclusi gli argomenti del risultato e della stringa da formattare).

Specificatori di stringa

Gli specificatori di stringa sono 4:

Specificatore	Descrizione
%b	Testo in grassetto
%i	Testo in corsivo
%ib	Testo in grassetto e corsivo
%t	Testo senza formattazione

Conversione tra Tipi (Type Casting)

MALT è un linguaggio abbastanza libero nella compatibilità tra tipi diversi e la conversione è sempre **implicita**.

La conversione tra tipi è consentita solamente tra tipi della stessa categoria se la categoria è primitiva testuale o lista.

Le variabili di tipo multimediale rappresentano oggetti troppo diversi per avere un qualche tipo di conversione che funzioni quindi l'assegnamento su queste variabili funziona solo quando il loro tipo è identico.

Ecco alcuni esempi di conversioni:

```
text testo = "testo";
list numeri = ["uno", "due", "tre"];

s3title titolo = testo;      // CONSENTITO: sia titolo (tipo: s3title) che te
codeblock cb = testo;        // CONSENTITO: sia cb (tipo: codeblock) che test
codeblock cb2 = titolo;      // CONSENTITO: sia cb2 (tipo: codeblock) che tit
blockquote bq = cb2;         // CONSENTITO: sia bq (tipo: blockquote) che cb2
title titolo2 = numeri;      // ERRORE: titolo2 (tipo: title) è primitiva tes

list l1 = cb;                // ERRORE: l1 (tipo: list) è lista mentre cb (ti
list l2 = numeri;            // CONSENTITO: sia l2 (tipo: list) che numeri (t
ulist ul = numeri;           // CONSENTITO: sia ul (tipo: ulist) che numeri (
tlist tl = ul;               // CONSENTITO: sia tl (tipo: tlist) che ul (tipo

table tabella = testo;       // ERRORE: tabella (tipo: table) è multimediale
img foto = cb;               // ERRORE: foto (tipo: img) è multimediale mentr
img foto2 = i ("/img/foto.png", "Foto");
img foto3 = foto2;           // CONSENTITO: sia foto3 (tipo: img) che foto2 (

table tabella2 = foto3;      // ERRORE: tabella2 (tipo: table) e foto3 (tipo:
```

Cicli

In MALT è possibile inserire cicli, in particolare cicli for. Essi possono essere strutturati in due modi differenti: nel primo caso si ripetono le istruzioni in base ad un certo contatore che rappresenta il numero delle iterazioni, mentre nel secondo caso le iterazioni corrispondono alla lunghezza della lista.

Ciclo for

Il ciclo for permette di iterare le istruzioni in esso contenute un numero di volte pari al valore del contatore passato nella condizione. Ne viene fornito un breve esempio:

```
text i;

for(i, 3){

    // istruzioni da ripetere

}
```

In ciascuna iterazione del ciclo al contatore *i* viene assegnato un valore a partire da 0 e fino al valore 3 - 1 (valore massimo - 1).

Ciclo for-each

Il ciclo for-each consente di eseguire le istruzioni contenute un numero di volte pari alla lunghezza della lista che si sta iterando. Inoltre, ad ogni iterazione si ha a disposizione il valore dell'elemento corrispondente della lista. Ecco un breve esempio:

```
list lista_frutta = ["mela", "pera", "banana", "arancia"];
text frutto;

for(frutto in lista_frutta){

    // istruzioni da ripetere

}
```

Ad ogni iterazione la variabile *frutto* assumerà un valore diverso:

- Prima iterazione: mela
- Seconda iterazione: pera
- Terza iterazione: banana
- Quarta iterazione: arancia

Il valore della variabile *frutto* potrà poi essere utilizzato per effettuare altre operazioni.

Scope delle variabili

Lo scope di una variabile definisce l'area in cui quella variabile è valida, definita e visibile.

In MALT non esistono parole chiave per estendere o limitare lo scope.

Lo scope di una variabile si estende sempre a partire dalla dichiarazione fino all'ultima istruzione del blocco dove è stata dichiarata.

Fino ad ora sono definibili solo due possibili blocchi: il blocco globale e il blocco del ciclo for.

Il blocco globale è un blocco sempre presente e si estende per tutto il codice. Il blocco del ciclo for è invece definito da un ciclo for e si estende per tutte le istruzioni ripetute dal ciclo for.

```
// inizio blocco globale
text testo_globale;

for(i, 5){
  // inizio blocco del ciclo for
  text testo_locale;

  // altre istruzioni...

  //fine blocco del ciclo for
}

// fine blocco globale
```

Nell'esempio la variabile *testo_globale* è definita nel blocco globale ed è visibile in tutto il codice (dopo la dichiarazione).

La variabile *testo_locale* è invece definita nel blocco del ciclo for ed è visibile soltanto all'interno del ciclo dopo la dichiarazione della variabile.

Per questo motivo le variabili dichiarate fuori dai cicli vengono definite "globali" mentre le variabili dichiarate nei cicli vengono definite "locali".

Funzioni (*fun*)

In MALT si possono definire delle funzioni utilizzando la notazione utilizzata nel seguente esempio:

```
fun esempio (text tx, title tl) {
  // altre istruzioni...

  return tx;
}
```

Nella funzione dell'esempio si hanno 2 argomenti, ma essi potrebbero anche non essere presenti oppure potrebbero essere uno o più.

Anche l'istruzione con *return* può essere opzionale, ma essa consente di restituire in output il valore di una variabile (nell'esempio la variabile *tx* di tipo *text*) oppure direttamente una stringa.

Una volta creata una funzione globale, essa può essere chiamata come nel prossimo esempio. Questa chiamata può avvenire seguendo due vincoli:

1. Si deve avere un numero di parametri nella chiamata pari al numero di argomenti della funzione;
2. Ciascun parametro della chiamata deve presentare un tipo uguale o convertibile a quello del rispettivo argomento della funzione.

```
text tx1 = "Testo";
title t11 = "Titolo";

// definizione della funzione "esempio"
fun esempio (text tx, title t1) {
  //altre istruzioni...

  return tx;
}

esempio(tx1,t11); // chiamata della funzione "esempio"
```

Scope delle variabili in una funzione

Anche per le funzioni, così come per i cicli for visti prima, viene creato un blocco. Lo scope delle variabili dichiarate all'interno della funzione si estende a partire dalla dichiarazione fino all'ultima istruzione del blocco.

Di seguito viene mostrato un esempio:

```

// inizio blocco globale

text t1 = "TestoGlobale";
for(i, 3){
    // inizio blocco for

    text t1f = "TestoGlobaleFor";
    // ...

    // fine blocco for
}

fun esempio (text tx, title t1) {
    // inizio blocco della funzione "esempio"

    text t2 = "TestoLocale";
    for(i, 3){
        // inizio blocco for

        text t2f = "TestoLocaleFor";
        // ...

        // fine blocco for
    }

    return tx;

    // fine blocco della funzione "esempio"
}

// fine blocco globale

```

La variabile `t1` è globale e quindi risulta sempre visibile. Invece la variabile `t1f`, poichè è stata definita all'interno di un ciclo `for`, risulta visibile soltanto nel ciclo stesso e di conseguenza è locale al `for`. Le precedenti considerazioni cambiano leggermente per le variabili `t2` e `t2f` siccome sono state definite in due livelli diversi all'interno di una funzione. La variabile `t2` è stata definita all'interno della funzione `esempio` e quindi risulta visibile solo nella funzione (variabile locale alla funzione). Invece la variabile `t2f`, poichè è stata definita in un ciclo `for` a sua volta all'interno di una funzione, risulta visibile soltanto nello specifico ciclo `for` della funzione `esempio` (variabile locale al ciclo `for` della funzione).

Classi (*class*)

MALT consente di definire delle classi. Questo è possibile tramite la notazione del seguente esempio:

```

class ClasseEsempio {

    text campo1;
    text campo2;

    fun metodo1 (text t1){
        // ...
    }

    fun metodo2 (){
        // ...
    }

    // ...

}

```

In una classe si possono definire campi e metodi. I campi non sono altro che variabili locali alla classe e quindi visibili solo all'interno di essa. I metodi sono delle funzioni legate alla classe. Le variabili definite al loro interno seguono le stesse regole di visibilità di una normale funzione.

I metodi hanno accesso ai campi della classe ma i primi sono visibili globalmente e possono essere chiamati ovunque nel codice utilizzando la dot notation in due modi diversi in base alla posizione della chiamata: all'interno della classe del metodo o esternamente.

Nel caso in cui si volesse chiamare il metodo all'interno della classe in cui è stato definito, la chiamata prevede l'utilizzo della parola chiave *this* seguita dal nome del metodo. Ne viene riportato un esempio

```

class Obj {
    text txlocal = "Testo locale";
    title tllocal = "Titolo locale";
    fun print (text tx, title tl) {
        //altre istruzioni...

        return tx;
    }

    this.print(txlocal,tllocal); // chiamata all'interno della stessa classe
                                // del metodo print
}

text txglobal;

this.print(txglobal, txglobal); // ERRORE!, non siamo all'interno
                                // della classe del metodo print

```

Invece, nel caso in cui si volesse chiamare il metodo all'esterno della classe in cui è stato definito, la chiamata prevede l'utilizzo del nome della classe seguito dal nome del metodo. Un esempio di

questa chiamata è riportato di seguito:

```
text txglobal = "Testo globale";
title tlglobal = "Titolo globale";

class Obj {
  fun print (text tx, title tl) {
    //altre istruzioni...

    return tx;
  }
}

Obj.print(txglobal,tlglobal);
```

Se si proviene da un linguaggio orientato agli oggetti è importante notare che al contrario di questi, in MALT non è possibile creare oggetti chiamando il costruttore della classe. In questo linguaggio, la classe fa solo da "collezione" di variabili (campi) e funzioni (metodi). Inoltre i campi sono sempre e solo locali alla classe (visibilità "private") e i metodi sono sempre e solo pubblici e accessibili ovunque (visibilità "public").

In MALT, non esistendo il concetto di oggetto, non esiste nemmeno il concetto di costruttore / distruttore.