Introduzione al Linguaggio Matlab

Informatica (ICA) AA 2017 / 2018

Giacomo Boracchi

21 Settembre 2016

giacomo.boracchi@polimi.it

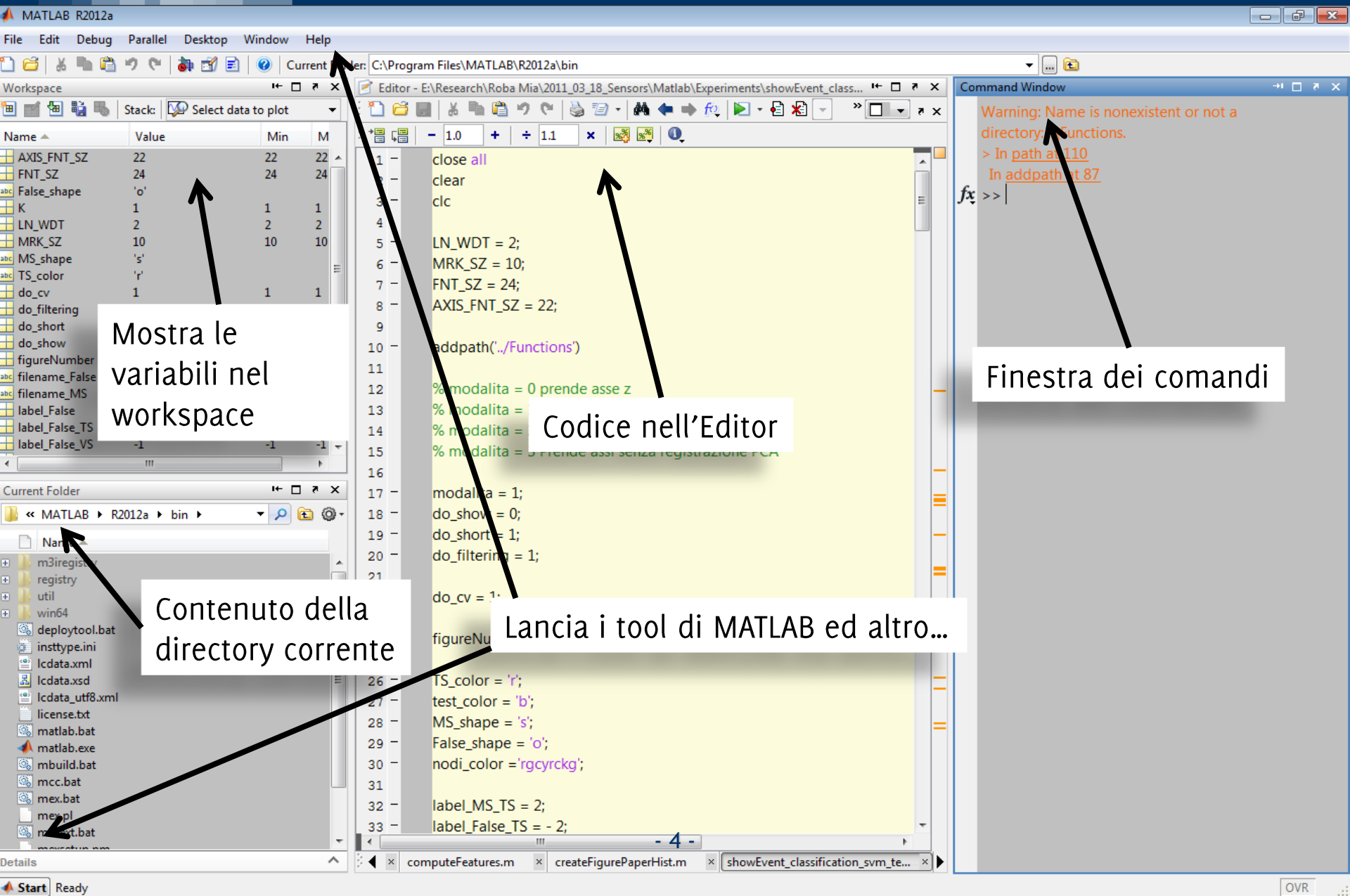
Il Matlab

MATrix LABoratory

Matlab

- Cos'è Matlab (MATrix LABoratory):
  - Un linguaggio di programmazione per calcolo numerico
  - Un Ambiente di sviluppo
- È pensato (e ottimizzato) per operare su matrici (ma include generiche funzionalità matematiche)
- Lo utilizzerete nei successivi corsi di calcolo numerico
- MATLAB è uno strumento commerciale, su licenza NON gratuita,
  - Student edition fornita dal Politecnico (maggiori dettagli a laboratorio)

Screenshot interfaccia MATLAB



Installazione di Matlab

Potete installare Matlab dal sito Polimi seguendo le istruzioni

https://home.deib.polimi.it/boracchi/teaching/InfoB/matlab_install.pdf

Installate Matlab il prima possbile, e certamente **prima del primo laboratorio**

Se avrete problemi di installazione l'ing Cogliati vi assisterà

(Credits Diego Carrera)

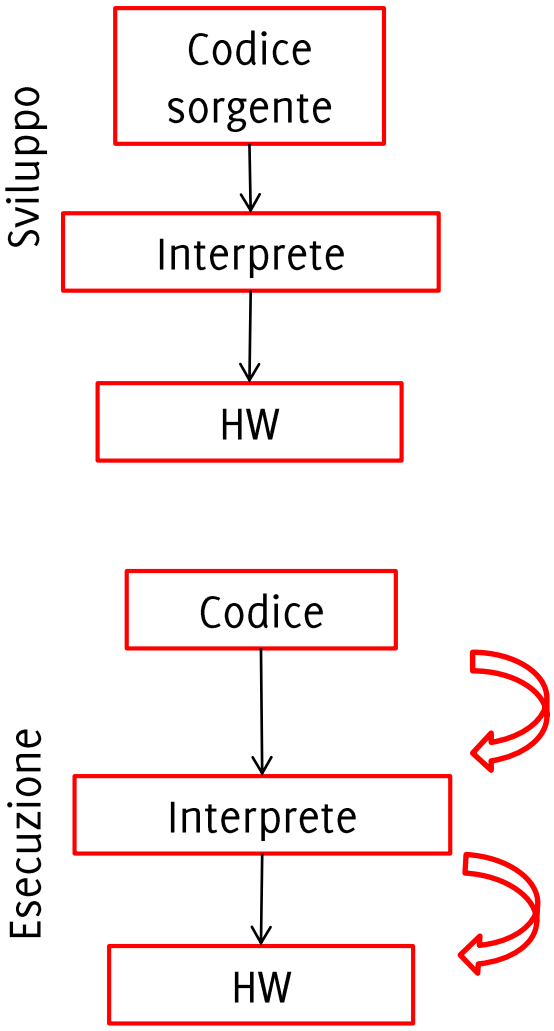
Caratteristiche del linguaggio di Matlab

- Linguaggio di alto livello
  - simile a linguaggi di programmazione C, Java, Pascal
  - possiede comandi sintetici per effettuare complesse elaborazioni numeriche
- Linguaggio interpretato, i comandi e istruzioni
  - NON sono tradotti in codice eseguibile dall'hardware
  - Ma invia istruzioni ad un altro programma, **l'interprete**, che li analizza ed esegue azioni da essi descritte

Matlab è un Linguaggio Interpretato

Linguaggi **Interpretati**:

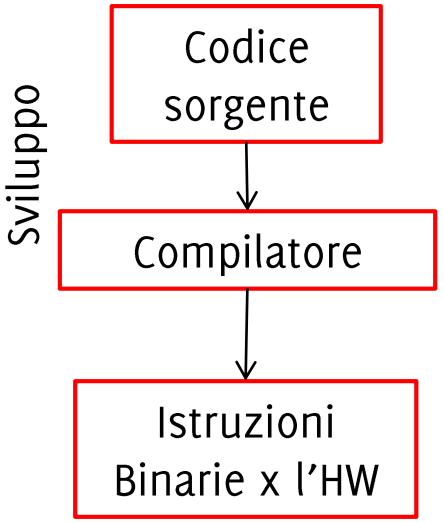
- L'interprete è un programma che **esegue** istruzioni contenute nel codice sorgente
- L'esecuzione del programma richiede la presenza del codice (talvolta il sorgente) e dell'interprete.
- I programmi sono meno efficienti di quelli compilati
- Portabilità meno pratica
- Sviluppo più facile: è possibile eseguire le istruzioni mentre si scrive il codice sorgente



Linguaggi Compilati vs Interpretati

Linguaggi **Compilati**:

- Il compilatore è un programma che **traduce** le istruzioni del codice sorgente in codice macchina (in binario)
- L'esecuzione del programma non richiede la presenza del codice sorgente, né del compilatore.
- I programmi sono efficienti
- Il programma è facilmente portabile su piattaforme analoghe



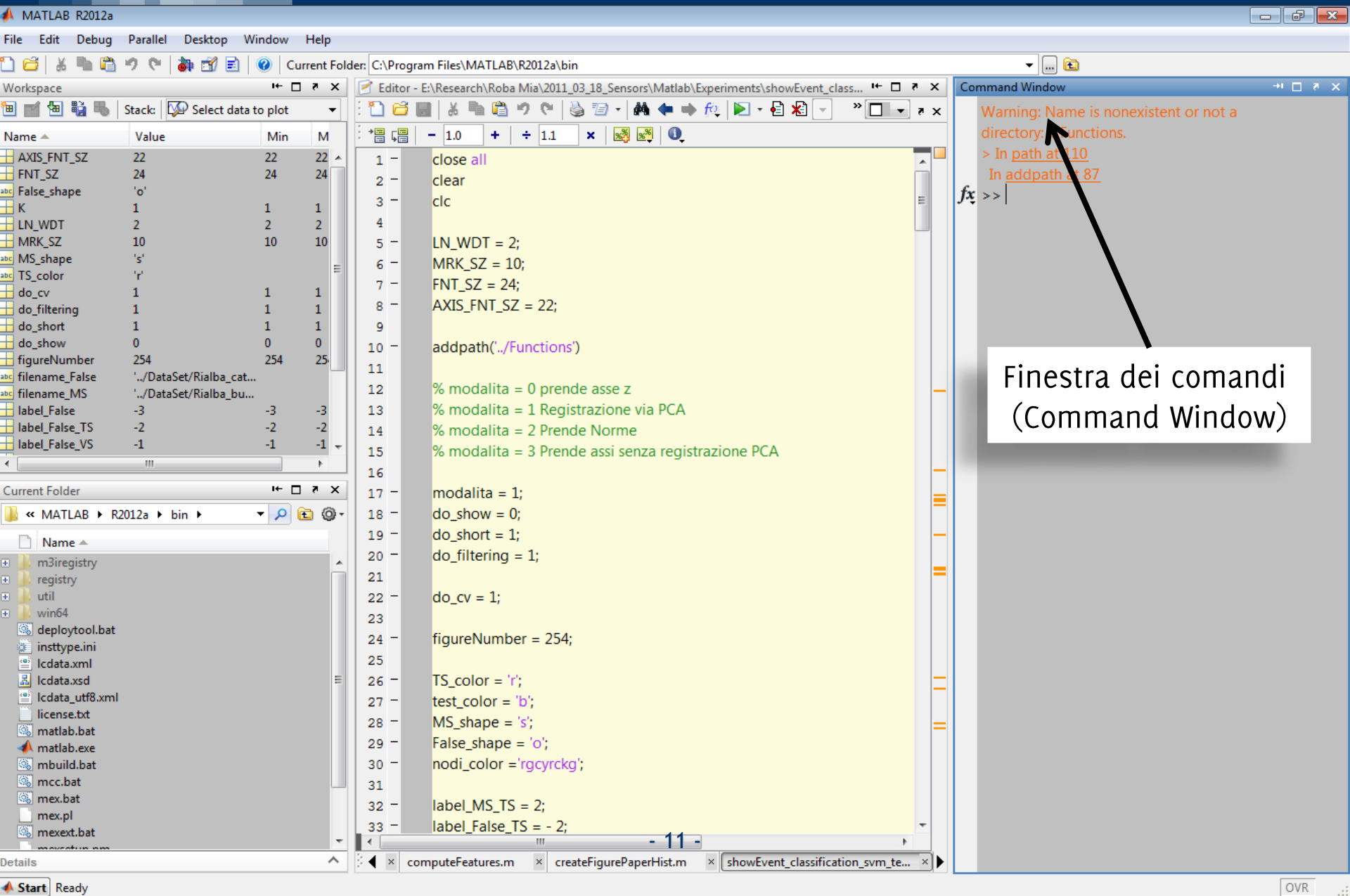
Le Istruzioni

Le Istruzioni e la Command Window

Le **istruzioni** possono essere **inviate** direttamente **all'interprete** se scritte nella command window (dopo il simbolo »)

- La command window è come una «super calcolatrice»
- La command window ha un'interfaccia testuale che inizia con **>>**

Screenshot interfaccia MATLAB



Esempio: le operazioni aritmetiche

- Nella command window è possibile eseguire qualsiasi operazione aritmetica

```
>> 5 + 7

ans =

    12

>> 5 / 7

ans =

    0.7143
```

ans è una variabile «di default» che contiene il risultato di un'istruzione che sia un assegnamento

Esempio: le operazioni aritmetiche

- Nella command window è possibile eseguire qualsiasi operazione aritmetica

Elevamento a potenza

```
>> 5 + 7

ans =

    12
```

```
>> 5 * 7

ans =

    35
```

```
>> 5 ^ 7

ans =

    78125
```

```
>> 5 / 7

ans =

    0.7143
```

```
>> 'a' + 2

ans =

    99
```

I caratteri alfanumerici si indicano con l'apice singolo: sono sempre legati agli interi mediante la tabella ASCII

Istruzioni e Codice Sorgente

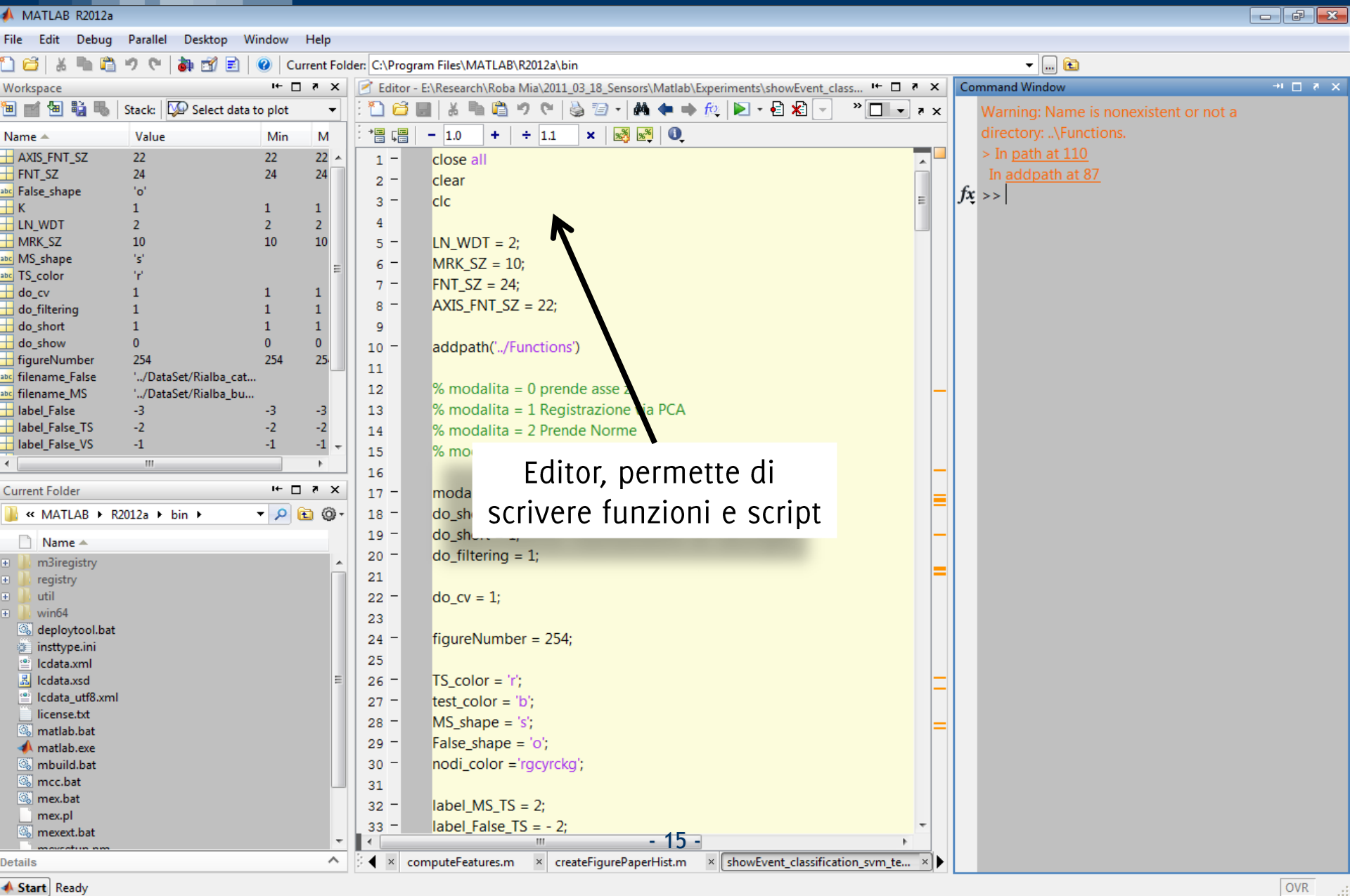
Le istruzioni possono essere contenute in un **file sorgente**, in particolare:

- uno script
- una funzione

e quindi eseguite in maniera sequenziale.

L'esecuzione di uno codice sorgente può essere visto come l'inserimento delle varie istruzioni in sequenza nella command window.

Screenshot interfaccia MATLAB



Le Variabili

le Variabili

Variabile ⇔ cella di memoria

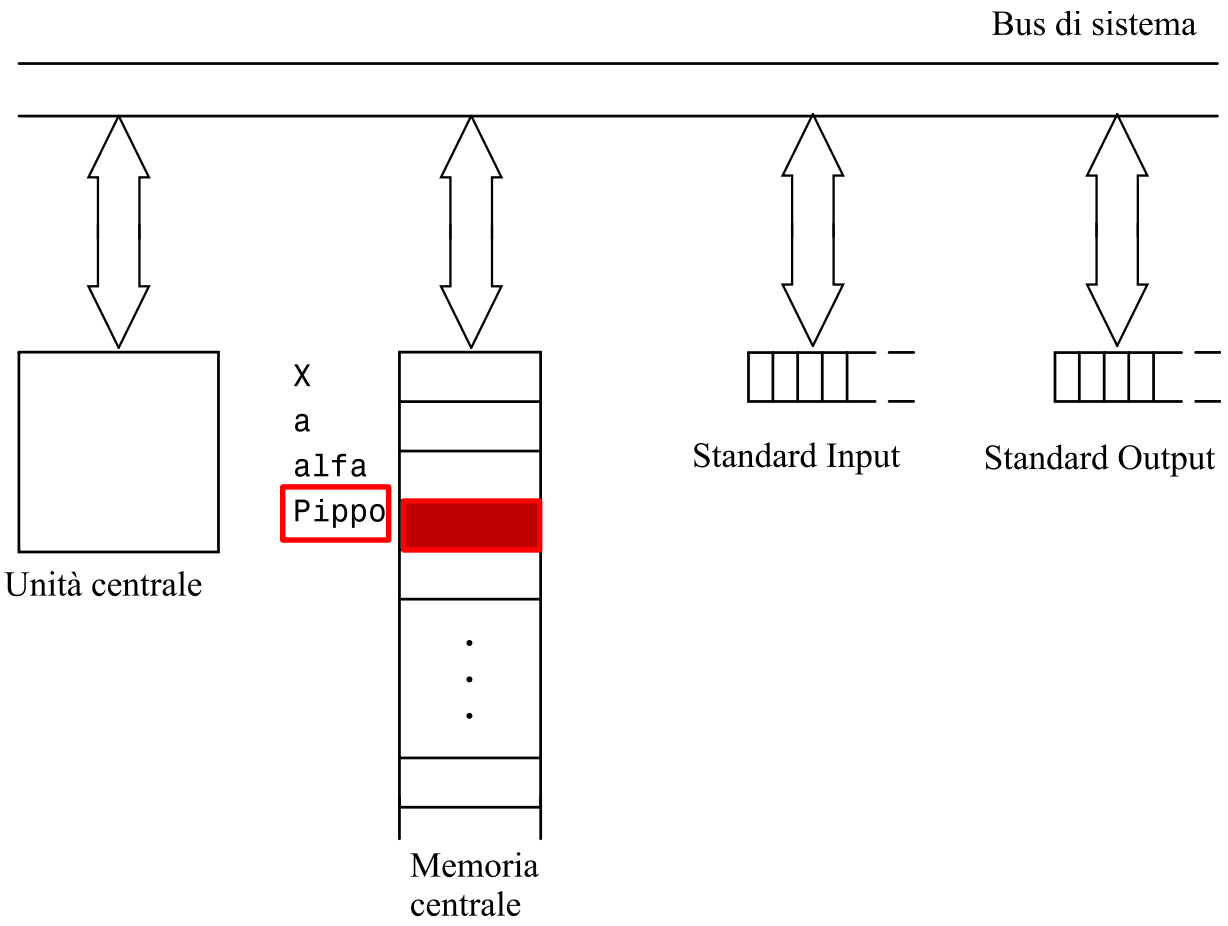
Le variabili hanno un **nome**: un **identificatore simbolico** formato da successione di **lettere, cifre e carattere _** con al primo posto una lettera.

- Es. di identificatori: **`a`, `x`, `alfa`, `pippo`, `a1`, `Giuseppe`, `DopoDomani`, `velocita_massima`**
- **NB**: maiuscole distinte dalle minuscole (**`Alfa`**, **`alfa`** e **`ALPHA`** sono tre diversi identificatori).

Si parla di identificatore simbolico perché permettono di accedere ad una cella di memoria (dov'è contenuto il valore della variabile) senza sapere dove sia questa cella (indirizzo, dimensioni in memoria).

Le Variabili, identificatori simbolici

Per accedere (in lettura o scrittura) alla cella in rosso mi basta far riferimento alla variabile **`Pippo`** nel codice.



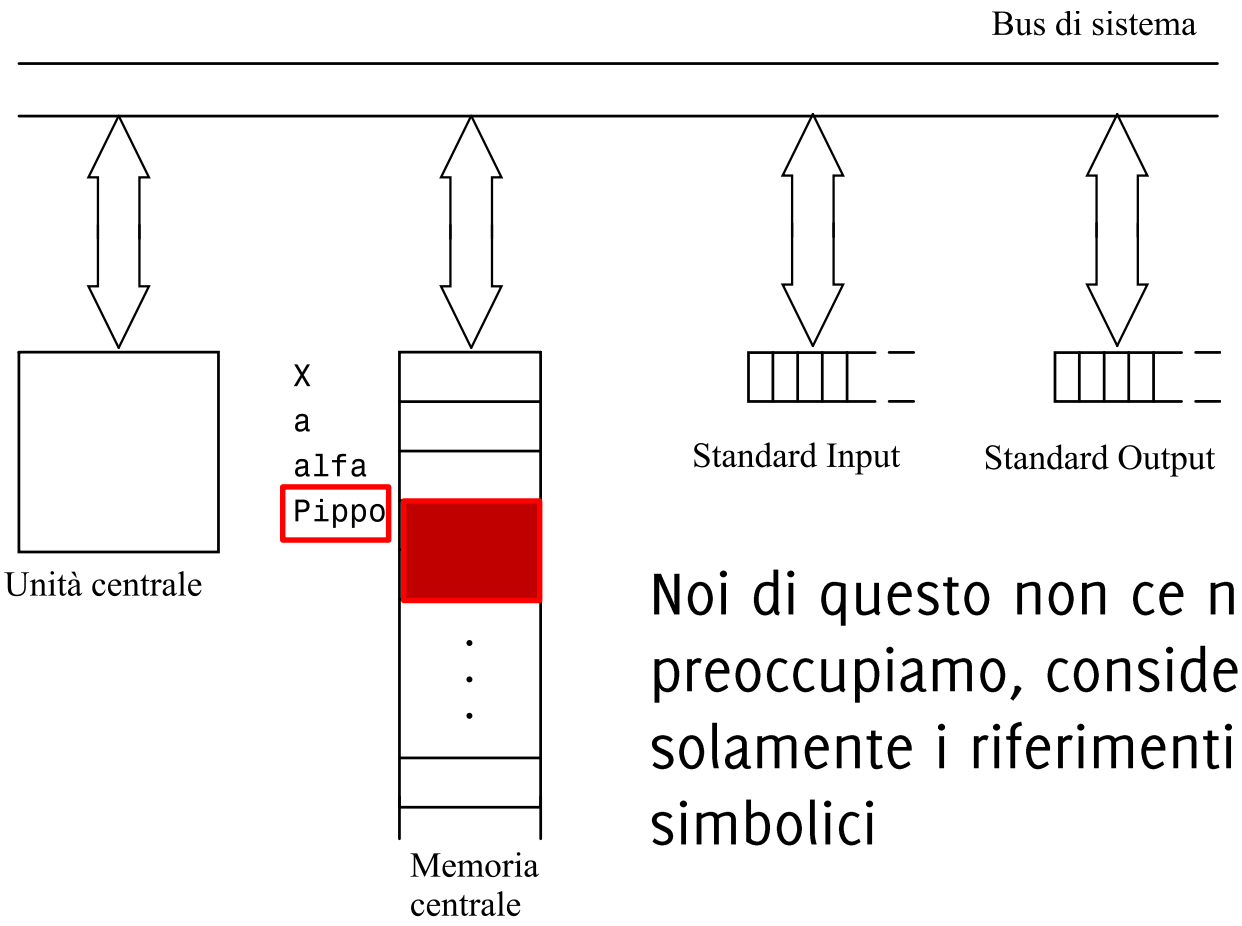
le Variabili (cnt)

Tutte le variabili hanno un **tipo**, che:

- caratterizza i valori scrivibili nella cella
- le operazioni sulla variabile
- la dimensione della cella in memoria (di questo non ce ne preoccupiamo)

Le Variabili, identificatori simbolici

A seconda del tipo è possibile che più celle facciano riferimento alla stessa variabile (**`Pippo`**)



Noi di questo non ce ne preoccupiamo, consideriamo solamente i riferimenti simbolici

Il Valore delle Variabili

Per visualizzare il valore delle variabili:

- Basta inserire il nome della variabile nella command window

```
>> a

   a =

      7
```

- Il nome della variabile non deve essere seguito da ; altrimenti la visualizzazione viene soppressa

```
>> a;

>>
```

Il Workspace

- Tutte le variabili vengono salvate nel workspace, che corrisponde allo spazio di memoria del programma
- E' possibile visualizzare le variabili ed il workspace:
  - Il comando **`whos`** (visualizza tutte le variabili)
  - Il comando **`whos nomeVariabile`** (visualizza solo **`nomeVariabile`**)
  - Il pannello del Workspace
- Per pulire il workspace e rimuovere tutte le variabili presenti si usa il comando: **`clear`**

```
>> clear
>> whos
>>
```

Nessuna variabile è più presente nel workspace

Istruzioni di Assegnamento

- E dichiarazione delle variabili

Istruzione di Assegnamento

Sintassi

```
nomeVariabile = espressione
```

Vengono eseguite le seguenti operazioni:

1. Viene valutato il valore di **`espressione`**
2. Il valore di **`espressione`** (a destra dell'uguale) viene copiato nella variabile **`nomeVariabile`** (a sinistra dell'uguale)

Istruzione di Assegnamento

Sintassi

```
nomeVariabile = espressione
```

Vengono eseguite le seguenti operazioni:

1. Viene valutato il valore di **`espressione`**
2. Il valore di **`espressione`** (a destra dell'uguale) viene copiato nella variabile **`nomeVariabile`** (a sinistra dell'uguale)

L'assegnamento è diverso dall'uguaglianza perché ha un verso ben definito: prende i valori da destra dell'uguale e li copia nella variabile a sinistra dell'uguale

```
nomeVariabile = espressione
```

Dichiarazione di Variabili Mediante Assegnamento

Quando assegno un valore ad una variabile che non è stata inizializzata (e.g., **a**), la **variabile viene creata**

```
>> a = 7

a =

     7
```

Posso quindi modificare il valore di **a** mediante successivo assegnamento

```
>> a = 15

a =

     15
```

Il nome “variabile” deriva dal fatto che queste possono cambiare valore

Note sull'Assegnamento (cnt)

In

```
nomeVariabile = espressione
```

espressione potrebbe essere:

- un'operazione algebrica, ad esempio

```
>> a = 15 / 3 + 2

a =

     7
```

- un'espressione che coinvolge una variabile

```
>> b = a - 1

b =

     6
```

Note sull'Assegnamento (cnt)

Attenzione:

```
>> a = a + 1
```

Cosa fa? Due opzioni (una è molto sbagliata)

- È un'equazione, senza soluzione?
  Non esiste valore di **`a`** che coincide ad **`a+1`**
- È un'operazione di assegnamento quindi
  1. Legge il valore di **`a`** a dx dell'uguale
  2. Somma a questo valore 1, il risultato è il valore di **`espressione`**
  3. Assegna il valore di **`espressione`** ad **`a`**

Note sull'Assegnamento (cnt)

Quindi,

```
>> a = 7;
>> a = a + 1

   a =

        8
```

Note sull'Assegnamento (cnt)

- Ovviamente non è possibile assegnare ad una variabile, il valore di una variabile che non esiste:

```
>> a = v
```

```
Undefined function or variable 'v'.
```

(messaggio di errore dell'interprete Matlab)

- Non è richiesto il ; al termine dell'istruzione
- Il risultato di un'operazione che non comporta un assegnamento viene assegnato alla variabile **`ans`**

Le Istruzioni di Assegnamento: i caratteri

- I caratteri alfanumerici vanno racchiusi tra apici singoli: ‘’
- Assegnamenti di un carattere ad una variabile

```
a = 'A';
a = 'z';
a = '1';
```

- **N.B: l’ultima istruzione assegna alla variabile a il valore corrispondente al carattere 1 che nella tabella ASCII corrisponde al numero 49**

I Tipi

Tipo Double

- Di default, valori numerici danno luogo a variabili di tipo **double: un** double contiene uno **scalare** espresso con doppia precisione (**64 bit**)
- È possibile vedere il tipo delle variabili mediante **whos**

whos nomeVariabile

```
>> a = 7;
>> whos a
  Name      Size    Bytes  Class     Attributes
   a        1x1         8  double
```

Tipo Char

- Una variabile di tipo char contiene uno **scalare** a 16 bit (8 bit in Octave), ciascuno dei quali rappresenta un carattere
  - Es: `Iniziale = 'G';`

Nome della variabile

Carattere 'G' corrisponde nella tabella ASCI al numer

```
whos Iniziale
Name        Size     Bytes Class     Attributes
Iniziale    1x1          2 char
```

Tipo Double (2)

- In Matlab è possibile rappresentare anche numeri **complessi**
- parti reali e immaginarie possono essere positive o negative nell'intervallo di valori $[10^{-308}, 10^{308}]$

```
>> a = [sqrt(-1)]
a =
   0 + 1.0000i

>> whos
 Name    Size     Bytes  Class    Attributes
  a       1x1     16  double      complex
```

Gestione Dinamica delle Variabili

I tipi delle variabili possono cambiare:

- mediante conversione esplicita
- mediante assegnamento: **il tipo** di una variabile è **definito dal valore** contenuto

```
>> a = 1
a =
     1.0000
>> whos a
  Name   Size    Bytes  Class     Attributes
  a      1x1     8      double
>> a = 'c';
>> whos a
  Name   Size    Bytes  Class     Attributes
  a      1x1     2      char
```

Tipo Char

- Ogni carattere corrisponde ad un numero intero.
- Il mapping è dato dalla tabella ASCII (es 'A' è 65, 'a' è 97)
- È quindi possibile sommare numeri ai caratteri, però attenzione che questo modifica il tipo della variabile

```
›› iniziale_nome = 'G'
iniziale_nome =

G

›› whos iniziale_nome
  Name              Size            Bytes  Class    Attributes
  iniziale_nome     1x1                 2  char
```

Tipo Char

- Ogni carattere corrisponde ad un numero intero.
- Il mapping è dato dalla tabella ASCII (es 'A' è 65, 'a' è 97)
- È quindi possibile sommare numeri ai caratteri, però attenzione che questo modifica il tipo della variabile

```
›› iniziale_nome = iniziale_nome + 1
iniziale_nome =

  72

›› whos iniziale_nome
  Name              Size            Bytes  Class     Attributes
  iniziale_nome     1x1                 8  double
```

Gli Operatori Aritmetici

Operatori Aritmetici

- Vi sono i soliti operatori aritmetici **+, –, *, /** e le **parentesi tonde** per definire operazioni tra i valori delle variabili

Input	Output	Commento
`1234/6`	`ans= 205.6667`	calcolo di un valore scalare
`a=1234/6`	`a = 205.6667`	assegnamento alla variabile a del risultato di 1234/6
`2/5`	`ans = 0.40000`	divisione
`5/0`	`ans = Inf`	divisione per zero
`5^2`	`ans = 25`	potenza

Operatori Aritmetici: resto della divisione intera

- Un nuovo operatore aritmetico: resto della divisione intera, o ***modulo***
  - es. **`mod(17,5)`** vale a **`2`**, **`mod(15,5)`** vale **`0`**

Operatori Aritmetici: resto della divisione intera

- Un nuovo operatore aritmetico: resto della divisione intera, o ***modulo***
  - es. **`mod(17,5)`** vale a **`2`**, **`mod(15,5)`** vale **`0`**
- Esempi:

```
a = 11; b = 4;
c = mod(a, 2);
```

Operatori Aritmetici: resto della divisione intera

- Un nuovo operatore aritmetico: resto della divisione intera, o ***modulo***
  - es. **`mod(17,5)`** vale a **`2`**, **`mod(15,5)`** vale **`0`**
- Esempi:

```
a = 11; b = 4;
c = mod(a, 2);
```

(viene scritto in **`c`** il valore **`1`**)

Operatori Aritmetici: resto della divisione intera

- Un nuovo operatore aritmetico: resto della divisione intera, o ***modulo***
  - es. **`mod(17,5)`** vale a **`2`**, **`mod(15,5)`** vale **`0`**
- Esempi:

```
a = 11; b = 4;
c = mod(a, 2);
```

(viene scritto in **`c`** il valore **`1`**)

```
a = 70; b = 5; c;
c = mod(a,(b + 2));
```

Operatori Aritmetici: resto della divisione intera

- Un nuovo operatore aritmetico: resto della divisione intera, o ***modulo***
  - es. **`mod(17,5)`** vale a **`2`**, **`mod(15,5)`** vale **`0`**
- Esempi:

```
a = 11; b = 4;
c = mod(a, 2);
```

(viene scritto in **`c`** il valore **`1`**)

```
a = 70; b = 5; c;
c = mod(a,(b + 2));
```

(viene scritto in **`c`** il valore **`0`**)

Operatori Aritmetici: resto della divisione intera

- Un nuovo operatore aritmetico: resto della divisione intera, o ***modulo***
  - es. **`mod(17,5)`** vale a **`2`**, **`mod(15,5)`** vale **`0`**

- Esempi:

```
a = 11; b = 4;
c = mod(a, 2);
```
(viene scritto in **`c`** il valore **`1`**)

```
a = 70; b = 5; c;
c = mod(a,(b + 2));
```
(viene scritto in **`c`** il valore **`0`**)

N.B: il valore di **`b`** non viene modificato, per modificare **`b`**:

```
b = b + 2; c = mod(a, b);
```

Script

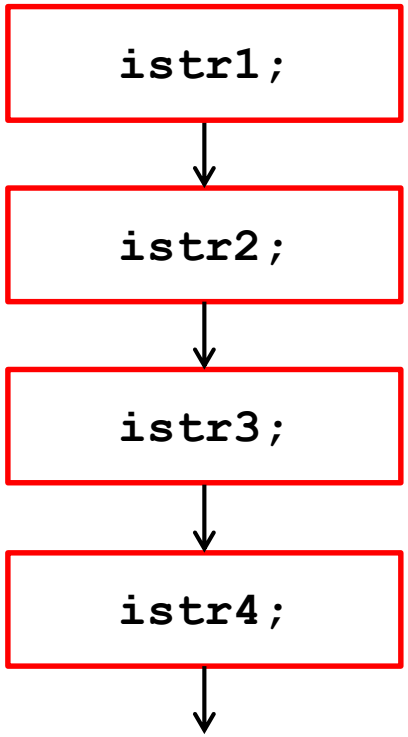
- Sequenze di istruzioni Matlab

Script (m-file)

- Uno script è un **file di testo** contenente una **sequenza di comandi MATLAB**
  - non deve contenere caratteri di formattazione (solo testo puro)
  - viene salvato con estensione .m

- I comandi all'interno di uno script sono eseguiti **sequenzialmente**, come se fossero scritti nella finestra dei comandi
  - Per eseguire il file (quindi la sequenza di istruzioni che contiene) si digita il suo nome (senza .m)
  - I risultati appaiono nella finestra dei comandi (se non usiamo il ;)

Sequenze di istruzioni




```
istr1;

istr2;

istr3;

istr4;

...
```

La Sequenza di Istruzioni

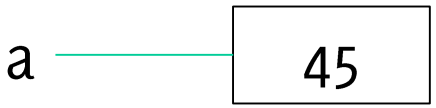
- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale**, dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

```
1. Istruzione1;
2. Istruzione2;
3. ….
4. IstruzioneN;
```

- *Es:*

```
a = 45;
z = 5;
x = (a - z) / 10;
```

- Stato della memoria



La Sequenza di Istruzioni

- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale**, dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

```
1. Istruzione1;
2. Istruzione2;
3. ….
4. IstruzioneN;
```

- *Es:*

```
a = 45;
z = 5;
x = (a - z) / 10;
```

- Stato della memoria

La Sequenza di Istruzioni

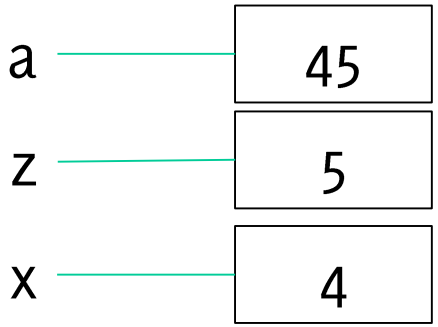
- In C, le istruzioni dei programmi sono eseguite in maniera **sequenziale**, dalla prima all'ultima
- Terminata la *i-sima* istruzione, si esegue la *(i+1)-sima*

```
1. Istruzione1;
2. Istruzione2;
3. ….
4. IstruzioneN;
```

- *Es:*

```
a = 45;
z = 5;
x = (a - z) / 10;
```

- Stato della memoria



Esecuzione di uno script

- Uno script viene mandato in esecuzione (lanciato) inserendone il nome nella command window (senza estensione .m)
- Occorre collocarsi nel folder contenente lo script prima di lanciarlo
- È possibile mandare in esecuzione lo script anche premendo F5 dall'editor o utilizzando il bottone apposito nell'interfaccia grafica

Istruzioni negli Script e ‘;’

- Le istruzioni **possono terminare** con ‘;’ ma non è obbligatorio
- Di default, il risultato di ogni istruzione viene visualizzato nella command window.
- Il ‘;’ **blocca la visualizzazione** del risultato dell’istruzione
  - Maggiore velocità di esecuzione
  - Visualizzazione più compatta
- Regola di buona programmazione
  - Inserire sempre il ‘;’, a meno che non si voglia ispezionare il valore di una variabile a scopo di debugging

Input/output

Acquisizione Dati da Tastiera (input)

- Funzione input

```
valore = input(stringaDaVisualizzare);
```

- Matlab stampa a video la **stringaDaVisualizzare** e **attende** un **input in formato Matlab**
  - Un numero (i.e., uno scalare)
  - Un carattere (delimitato da apici singoli)
  - Array, se racchiuso tra [e], oppure
  - Stringa, se racchiusa tra ‘ e ’, oppure
  - Una qualsiasi espressione Matlab

Vedremo nel seguito

- Il dato inserito dall’utente viene memorizzato nella variabile **valore**
- se **stringaDaVisualizzare** è una sequenza di caratteri, questa deve essere racchiusa tra apici singoli

Scrittura con `fprintf`

Esempio:

```
fprintf('\nInserire a:');
```

Sintassi:

```
fprintf (stringaControllo);
```

- ***`stringaControllo`*** sequenze di caratteri (i.e., stringa) delimitata da doppi apici singoli ‘ ‘.
- Possono essere
  - **caratteri normali** (lettere, cifre, pungeggiatura)
  - caratteri speciali (es, vai a capo)
  - Placeholders (e.g. ‘%d’ per il contenuto di variabili)

I caratteri nella ***`stringaControllo`*** vengono riportati a schermo.

Stampa dei Risultati

Usiamo il commando **fprintf**

```
fprintf('Il valore di pi e`%f \n', pi);
```

Stringhe di formato:

- `%d` (interi),
- `%e` (formato esponenziale),
- `%f` (formato decimale),
- `%g` (il più corto tra il formato esponenziale e decimale)

Per i caratteri si usa

- `%c` (un solo carattere

Scrittura su Standard Output: `fprintf`

Esempio:

```
fprintf('\n %d + %d = %d', a, b, a+b);
```

Sintassi:

```
fprintf (stringaControllo, elementiStampa);
```

- ***stringaControllo*** può contenere
  - **caratteri di stampa** (**normali** o **speciali**)
  - **caratteri di conversione** (segnaposto, convertono valori di variabili in caratteri per la stampa)
- ***elementiStampa*** elenco di **variabili**, **espressioni composte**, o costanti separati da virgole

Ogni elemento di ***elementiStampa*** viene convertito in caratteri e associato ai caratteri di conversione in ***stringaControllo*** in **nell'ordine con cui appare**.

stringaControllo:

Alcuni **caratteri speciali per la stampa**

- `'\n'` manda a capo
- `'\t'` spazia di un «tab»

Alcuni **caratteri di conversione**

- `%d` intero decimale
- `%f` numero reale
- `%c` carattere
- `%s` sequenza di caratteri (stringa)

Scrittura su Standard Output: `printf`

Esempi:

```
>> cat_dipend = 1;

>> stip_medio = 35623.5;

>> fprintf('Lo stipendio annuo dei
dipendenti di categoria %d è pari a $%f\n',
cat_dipend, stip_medio);
```

Scrittura su Standard Output: `printf`

Esempi:

```
>> cat_dipend = 1;
>> stip_medio = 35623.5;
>> fprintf('Lo stipendio annuo dei
dipendenti di categoria %d è pari a $%f\n',
cat_dipend, stip_medio);
```

Nella stampa **`%d`** verrà sostituito dal valore di **`cat_dipend`** mentre **`%f`** verrà sostituito dal valore di **`stip_medio`**.

L'abbinamento è dovuto **esclusivamente all'ordine** con cui appaiono i caratteri di conversione e le variabili (non al tipo).

Scrittura su Standard Output: `printf`

Esempi:

```
>> cat_dipend = 1;

>> stip_medio = 35623.5;

>> fprintf('Lo stipendio annuo dei
dipendenti di categoria %d è pari a $%f\n',
cat_dipend, stip_medio);
```

```
Lo stipendio annuo dei dipendenti di
categoria 1 è pari a $35623.500000
```

Scrittura su Standard Output: `printf`

Esempi:

```
>> iniz_nome = 'G';

>> iniz_cognome = 'B';

» fprintf('Questo programma è stato scritto
da \n%c%c\n\nBuon lavoro!\n', iniz_nome,
iniz_cognome);
```

Scrittura su Standard Output: `printf`

Esempi:

```
>> iniz_nome = 'G';
>> iniz_cognome = 'B';
» fprintf('Questo programma è stato scritto
da \n%c%c\n\nBuon lavoro!\n', iniz_nome,
iniz_cognome);

Questo programma è stato scritto da
GB


Buon lavoro!
```

Un primo semplice programma

Programma 1

Scrivere un programma che calcola il vostro anno di laurea come l'anno di nascita + 30 e visualizza a schermo il risultato

Programma 1

Scrivere un programma che calcola il vostro anno di laurea come l'anno di nascita + 30 e visualizza a schermo il risultato

```
clear
clc
% richiedo all'utente anno di nascita
anno = input('Inserire anno di nascita');
% calcolo anno di laurea
annodiLaurea = anno + 30;
% visualizzo a schermo l'anno di laurea
fprintf('Sei nato nel %d laurerai nel %d',
anno, annodiLaurea);
```

Programma 1

Scrivere un programma che calcola il vostro anno di laurea come l'anno di nascita + 30 e visualizza a schermo il risultato

```
clear
clc
% richiedo all'utente anno di nascita
anno = input('Inserire anno di nascita');
% calcolo anno di laurea
annodiLaurea = anno + 30;
% visualizzo a schermo l'anno di laurea
fprintf('Sei nato nel %d laurerai nel %d',
anno, annodiLaurea);
```

Pulisce il workspace

Programma 1

Scrivere un programma che calcola il vostro anno di larea come l'anno di nascita + 30 e visualizza a schermo il risultato

```
clear
clc
% richiedo all'utente anno di nascita
anno = input('Inserire anno di nascita');
% calcolo anno di laurea
annodiLaurea = anno + 30;
% visualizzo a schermo l'anno di laurea
fprintf('Sei nato nel %d laurerai nel %d',
anno, annodiLaurea);
```

Pulisce la command window

Programma 2

Scrivere un programma che richiede all'utente due numeri e ne calcola la somma

E qualche programma più elaborato

Esempio di programmi

- Scrivere un programma che prende in ingresso una temperatura in Fahrenheit e la trasforma in Celsius

$$C = 5/9 * (F - 32)$$

Esempio

Scrivere un programma che richiede due caratteri che vengono salvati in opportune variabili.

Il programma poi scambia i contenuti delle variabili e ne stampa i valori.

Vi ricordate?
Algoritmo per invertire il contenuto di A e B

1. Prendi un terzo bicchiere C
2. Rovescia il contenuto del bicchiere A nel bicchiere C
3. Rovescia il contenuto di B in A
4. Rovescia il contenuto di C in B

Algoritmo per scambiare i valori di due variabili A e B (con le variabili a volte non occorre il bicchiere C)

Tipo di Dato Logico

Tipo di Dato Logico

- È un tipo di dato che può avere solo due valori
  - true (vero) 1
  - false (falso) 0
- I valori di questo tipo possono essere generati
  - direttamente da due funzioni speciali (true e false)
  - dagli operatori relazionali
  - dagli operatori logici
- I valori logici occupano un solo byte di memoria (i numeri ne occupano 8)

Esempi

```
>> a = true;
>> whos a
 Name   Size    Bytes  Class     Attributes
  a     1x1       1    logical
```

a è un vettore 1x1 che occupa 1 byte e appartiene alla classe "tipo logico"

```
>> a = 1>7
a =
     0
```

Operatori Relazionali

- Operano su tipi numerici o stringhe.
- Possono essere usati per confrontare
  - due scalari
  - due vettori aventi la stessa dimensione
- Forma generale: a OP b
  - a,b devono essere variabili (o espressioni) della stessa dimensione
  - OP: `==, ~=, >, >=, <, <=`
- Esempi:
  - `3 < 4` true(1)
  - `3 == 4` false(0)
  - `'A' < 'B'` true(1)

Note

- **Attenzione:** non confondere == e =
  - == è un operatore di confronto
  - = è un operatore di assegnamento
- La precisione finita può produrre errori con == e ~ =
  - sin(0) == 0 → 1
  - sin(pi) == 0 → 0
  - eppure logicamente sono vere entrambe!!
- Per i numeri piccoli conviene usare una soglia
  - abs(sin(pi) – sin(0)) < = eps
- eps è una variabile built-in in Matlab e vale 2.2204e-16
- pi è una variabile built-in ed è una buona approssimazione di π

Precedenze tra gli Operatori

Ogni espressione viene valutata nel seguente ordine:

- operatori aritmetici
- operatori relazionali da sinistra verso destra
- NOT (~)
- AND (**&** e **&&**) da sinistra verso destra
- OR (| e ||) e XOR da sinistra verso destra

Richiamo, Tabelle di Verità

a	b	a &&b	a ||b	~a	xor(a,b)
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Or esclusivo: vero quando è vera
solo uno delle due espressioni coinvolte
Xor(a,b) == a || b && (~(a AND b))

Operatori Logici: Forma Generale

- Operatori binari: **AND** (&&, oppure &, oppure and), **OR** (||, oppure |, oppure or), **XOR** (xor):

  a OP1 b per la notazione simbolica
  OP(a,b) per la notazione testuale

- Operatori unari: NOT (~):

  OP2 a

- a,b possono essere variabili, costanti, espressioni da valutare, scalari o vettori (dimensioni compatibili)
- Valori numerici di a, b vengono interpretati come logici:
  - o come falso
  - tutti i numeri diversi da o come vero

Differenza tra `&&` e `&` (e tra || e |)

`&&` è l'operatore short-circuit per logicals scalari

- Quando calcola **`a && b`** non valuta **`b`** se **`a`** è falso

|| è l'operatore short-circuit per logicals scalari

- Quando calcola **`a || b`** non valuta **`b`** se **`a`** è vero

`&` e || si applicano anche a vettori di logicals (vedremo poi)

`&` può essere conveniente. Si pensi di dover valutare se la seguente espressione è vera:

«ho meno di 15 anni» **`&`** «I giochi olimpici invernali del 1924 si sono tenuti in Francia»

Usare **`&&`** e **`&`** permette di evitare di andare a cercare su Wikipedia

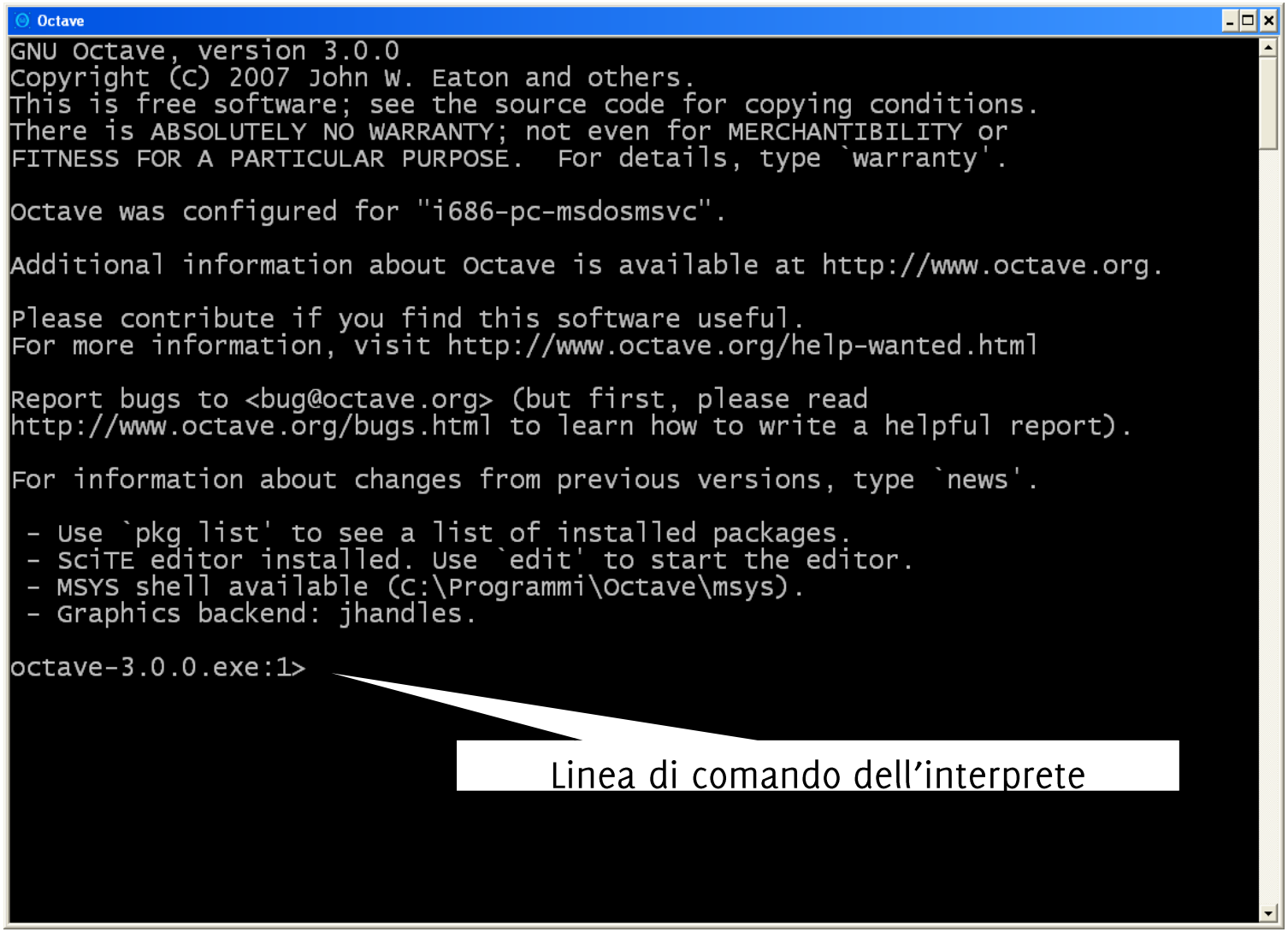
…

Alternative a Matlab

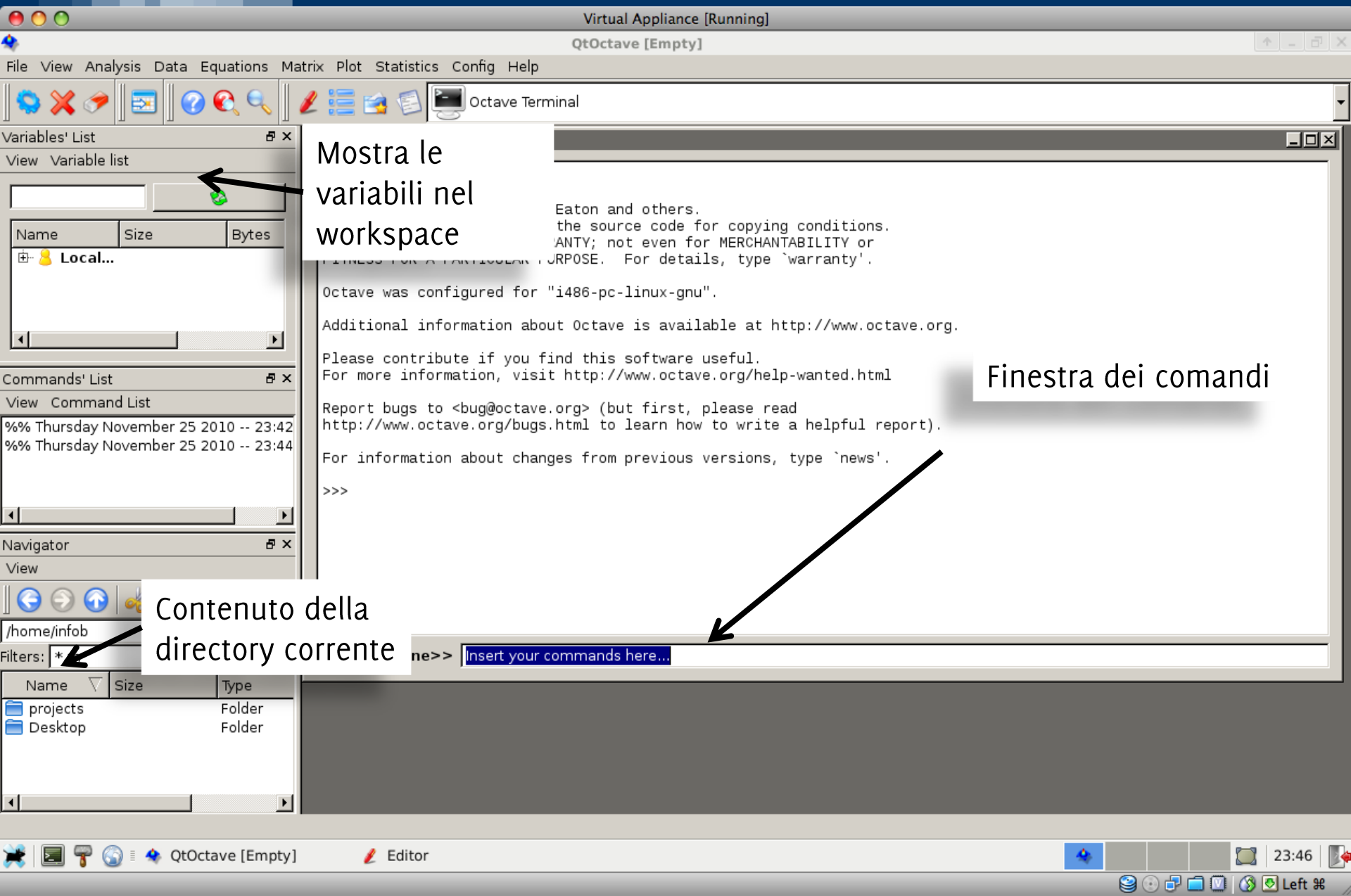
Octave

- Esiste una soluzione alternativa: Octave
  - identico nella concezione, molto simile per gli aspetti operativi
  - disponibile gratuitamente, vedi www.gnu.org/software/octave/
  - Occorre installare un'interfaccia grafica qtoctave

Screenshot dell'interfaccia OCTAVE



Screenshot dell'interfaccia grafica QtOctave



Mostra le variabili nel workspace

Finestra dei comandi

Contenuto della directory corrente