



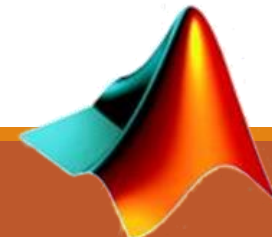
UNIVERSITÀ DEGLI STUDI DI SALERNO

Fondamenti di Informatica

Introduzione alla programmazione in MATLAB: Parte 1
M-File e Input/Output

Prof. Raffaele Pizzolante

A.A. 2016/17



MATLAB®

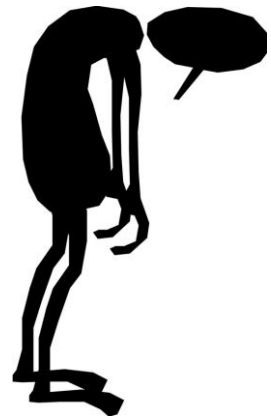
Introduzione alla programmazione in MATLAB: Parte 1

OUTLINE

- M-File
 - M-File Script
 - M-File Function
- Input/Output

M-File – 1/2

- Finora abbiamo inserito comandi, istruzioni e funzioni MATLAB direttamente mediante la **Command Window**
- Tuttavia, ciò può causare disagio, specialmente quando comandi, istruzioni e funzioni devono essere rieseguiti più volte
 - Magari in più sessioni di lavoro MATLAB distinte
 - Con leggere modifiche
 - Etc



M-File – 1/2

- Finora abbiamo inserito comandi, istruzioni e funzioni MATLAB direttamente mediante la **Command Window**
- Tuttavia, ciò può causare disagio, specialmente quando comandi, istruzioni e funzioni devono essere rieseguiti più volte
 - Magari in più sessioni di lavoro MATLAB distinte
 - Con leggere modifiche
 - Etc
- MATLAB permette di risolvere questi problemi attraverso l'utilizzo degli *M-File*



M-File – 2/2

- MATLAB consente di memorizzare una sequenza di istruzioni in un file, detto *M-File*
- In particolare, un *M-File* può essere di due tipi
 - ***M-File Script***: contiene una sequenza di comandi o istruzioni MATLAB, nella stessa forma in cui vengono scritti usando Command Window
 - ***M-File Function***: contiene nuove funzioni MATLAB definite dall'utente. In generale, tali funzioni accettano dati in input e restituiscono dati di output, come risultato della loro elaborazione



M-File Script – 1/9

- In MATLAB è possibile rieseguire comandi, istruzioni e funzioni mediante i seguenti passi
 - Creare un file (che conterrà la lista di comandi, istruzioni e funzioni)
 - Salvare il file
 - Eseguire il file
- Un file contenente una lista di comandi/istruzioni/funzioni MATLAB viene detto
 - *M-File Script*
- Ogni *M-file Script* ha l'estensione **.m**

M-File Script – 2/9

- Più precisamente, un *M-file Script* è
 - Un file esterno contenente sequenze di istruzioni MATLAB
 - Digitando il nome del file, comandi/istruzioni/funzioni prese in input da MATLAB sono ottenute direttamente da tale file
 - Utile per l'automazione di blocchi di comandi/istruzioni/funzioni MATLAB
 - Come ad esempio calcoli che è necessario eseguire più volte (manualmente) dalla Command Window

M-File Script – 3/9

- **Esempio**

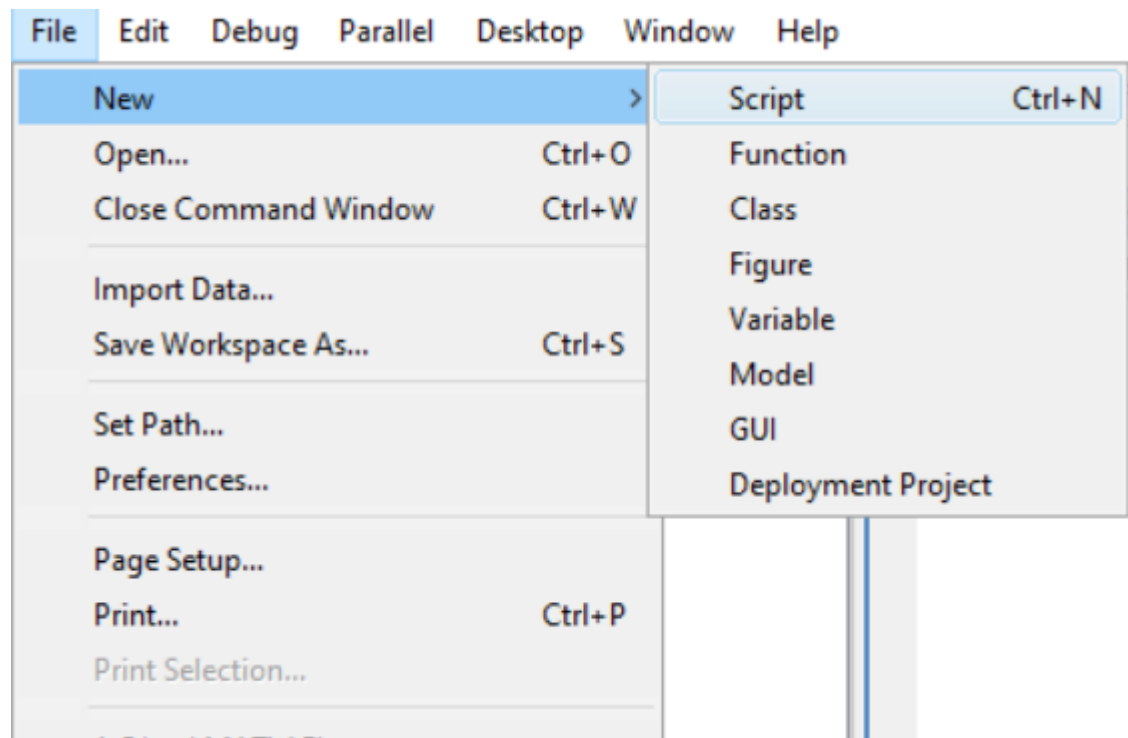
- Creare uno script (di nome **sommamat.m**) che
 - Effettua la somma di due matrici A e B
 - Salva il risultato nella matrice C ed infine lo stampa
- A e B sono definite come segue
 - $A = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}, B = \begin{bmatrix} 6 & 7 \\ 8 & 9 \end{bmatrix}$
- **SOLUZIONE** MATLAB per sommare A e B
 - Tale soluzione andrà inserita nel file **sommamat.m**

sommamat.m

```
A=[2 3;4 5];  
B=[6 7; 8 9];  
C=A+B
```

M-File Script – 4/9

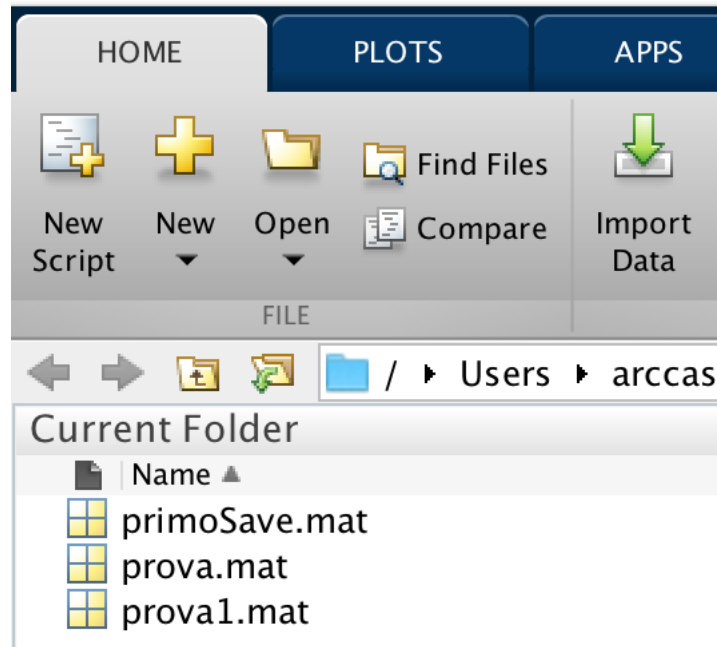
- **Creare** uno script con MATLAB



M-File Script – 4/9

(Versioni più recenti di MATLAB)

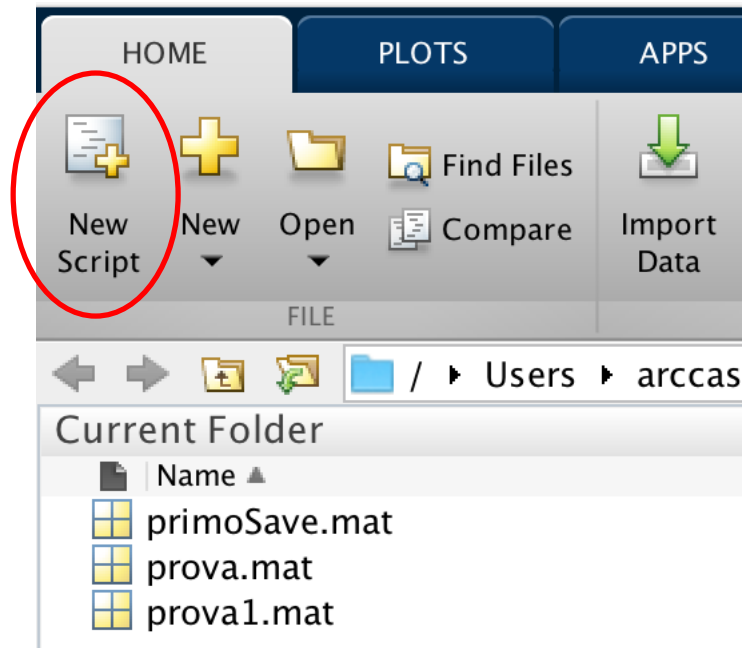
- **Creare** uno script con MATLAB



M-File Script – 4/9

(Versioni più recenti di MATLAB)

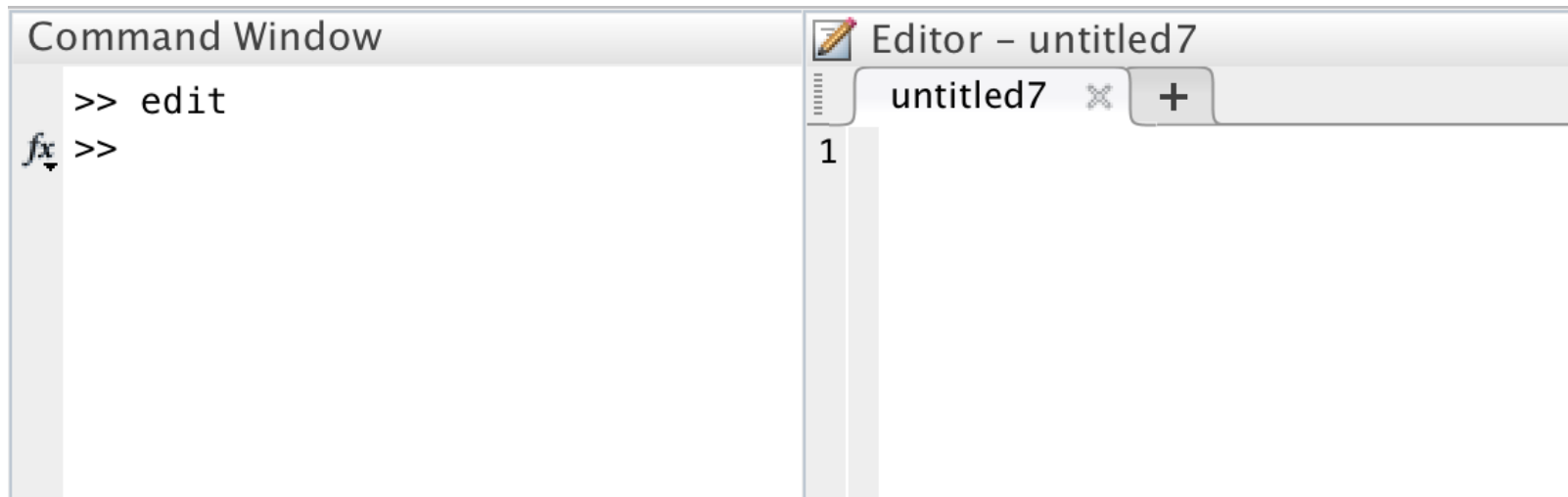
- **Creare** uno script con MATLAB



M-File Script – 4/9

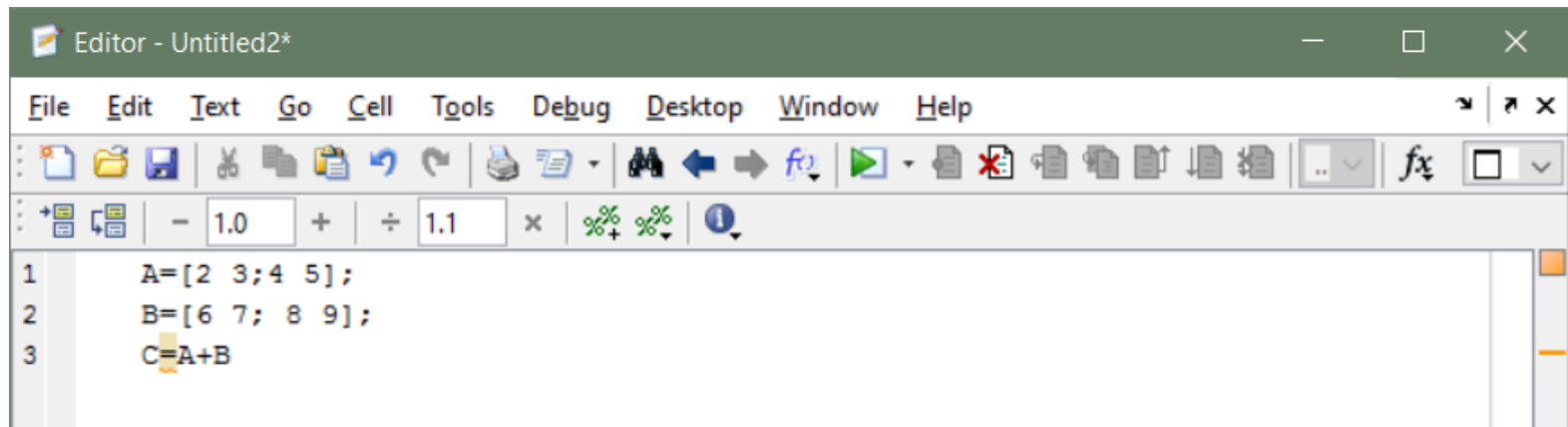
(Utilizzando la Command Window)

- **Creare** uno script con MATLAB utilizzando il comando **edit**



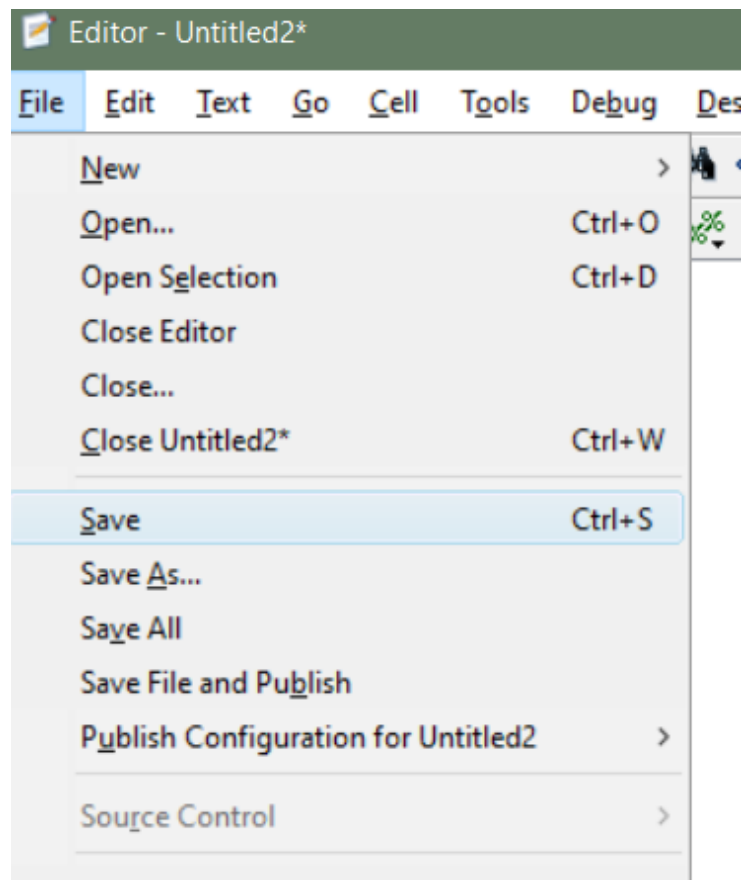
M-File Script – 5/9

- Editor di *M-File Script*
- Inserire le istruzioni MATLAB mediante l'Editor di *M-file Script*



M-File Script – 6/9

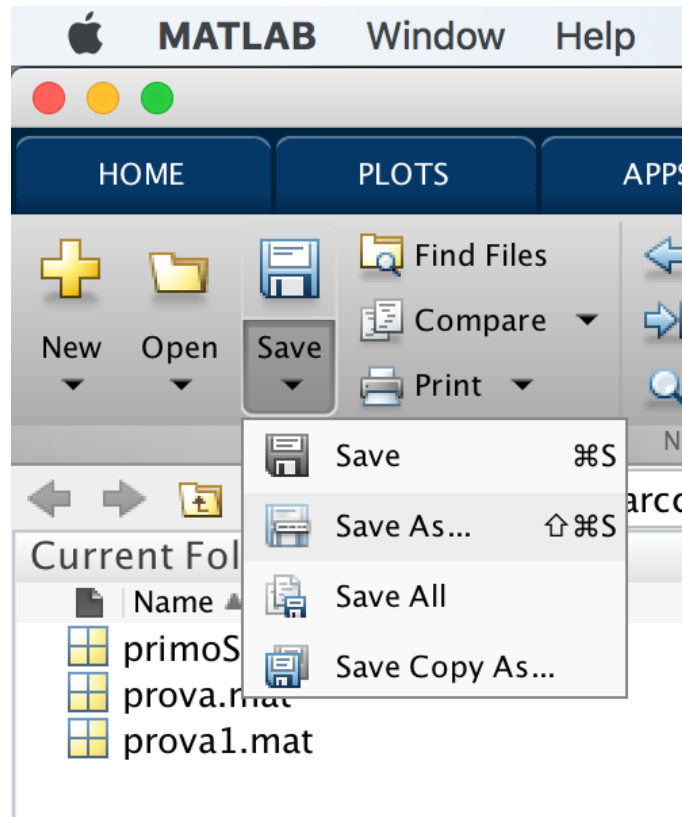
- **Salvare l'M-File Script** (il .m viene automaticamente aggiunto)



M-File Script – 6/9

(Versioni più recenti di MATLAB)

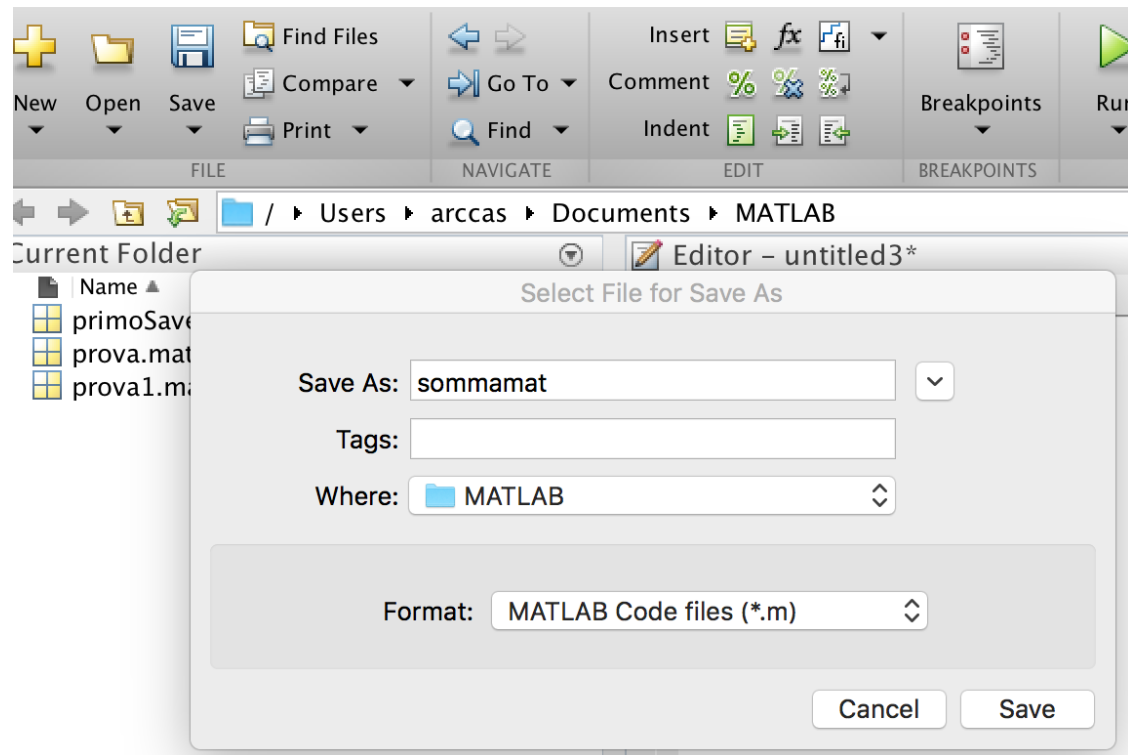
- **Salvare** l'*M-File Script* (il .m viene automaticamente aggiunto)



M-File Script – 6/9

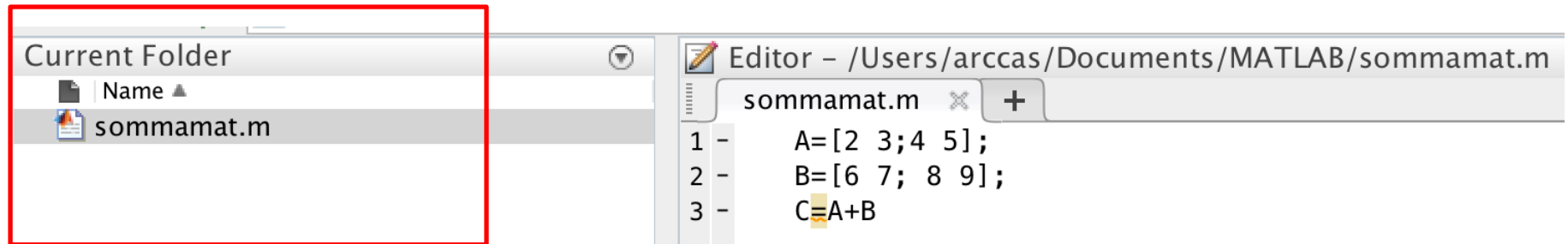
(Versioni più recenti di MATLAB)

- **Salvare l'M-File Script** (il .m viene automaticamente aggiunto)



M-File Script – 6/9

- **Salvare l'M-File Script** (il .m viene automaticamente aggiunto)



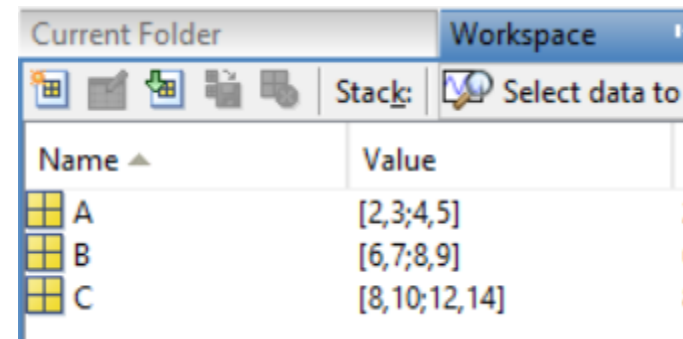
M-File Script – 7/9

- **Eseguire l'M-File Script** mediante la Command Window
 - L'M-File Script deve essere memorizzato nella Current Directory
 - Per **eseguirlo** è sufficiente digitare nella Command Window il **nome del file script (senza estensione .m)**
- **Esempio**
 - Supponiamo di aver memorizzato il file script dell'esempio precedente nella Current Directory, con il nome di **sommamat.m**

```
>> sommamat
```

```
C =
```

```
      8      10  
     12     14
```



The screenshot shows the MATLAB Command Window and Workspace. The Command Window displays the output of the 'sommamat' script, which is a 2x2 matrix C. The Workspace window shows the variables A, B, and C, each with its corresponding value.

Name	Value
A	[2,3;4,5]
B	[6,7;8,9]
C	[8,10;12,14]

M-File Script – 8/9

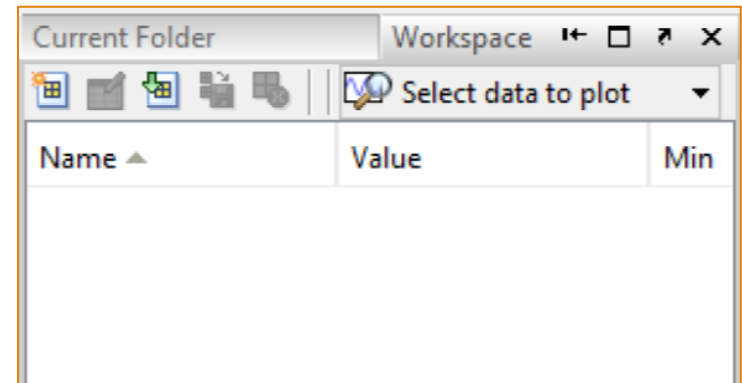
- Gli *M-File Script* possono operare su variabili esistenti nel Workspace, oppure possono crearne di nuove
- Tutte le variabili che vengono create da tali script rimangono nel Workspace e possono essere usate per effettuare ulteriori calcoli

Esempio 1

media.m

```
mean = (a + b + c) / 3
```

Script per calcolare
la media di 3 numeri



Workspace

M-File Script – 8/9

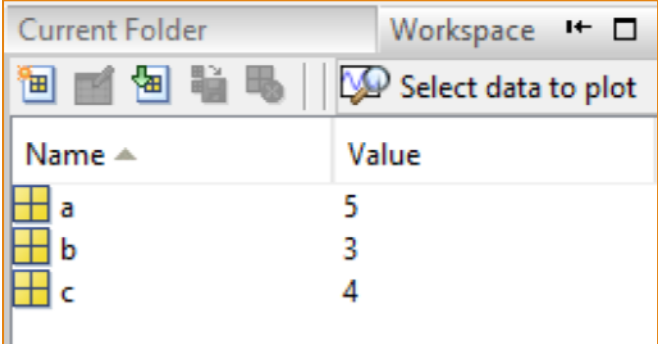
- Gli *M-File Script* possono operare su variabili esistenti nel Workspace, oppure possono crearne di nuove
- Tutte le variabili che vengono create da tali script rimangono nel Workspace e possono essere usate per effettuare ulteriori calcoli

Esempio 1

media.m

```
mean = (a + b + c) / 3
```

Script per calcolare
la media di 3 numeri



The screenshot shows the MATLAB Workspace window. It has a toolbar with icons for 'Current Folder', 'Workspace', and 'Select data to plot'. Below the toolbar is a table with two columns: 'Name' and 'Value'. The table lists three variables: 'a' with value 5, 'b' with value 3, and 'c' with value 4. The word 'Workspace' is written vertically on the right side of the window.

Name	Value
a	5
b	3
c	4

Variabili presenti nel
Workspace prima
dell'esecuzione dello script

Definisco tre variabili a, b, c

Command Window

```
>> a = 5; b = 3; c = 4;
```

Command Window

- Gli *M-File Script* possono operare su variabili esistenti nel Workspace, oppure possono crearne di nuove
- Tutte le variabili che vengono create da tali script rimangono nel Workspace e possono essere usate per effettuare ulteriori calcoli

Esempio 1

media.m

```
mean = (a + b + c) / 3
```

Script per calcolare
la media di 3 numeri

Eseguo lo script
chiamato media

```
Command Window

>> a = 5; b = 3; c = 4;
>> media

mean =

    4
```

Command Window

Current Folder		Workspace	
		Select data to plot	
Name		Value	
a		5	
b		3	
c		4	
mean		4	

Workspace

Variabile inserita nel
Workspace dopo
l'esecuzione dello
script

M-File Script – 8/9

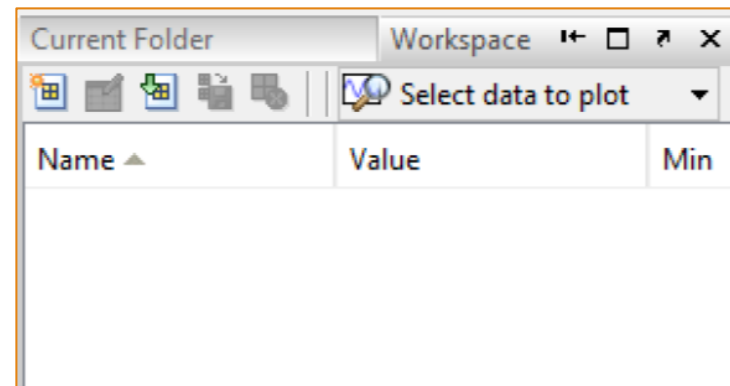
- Gli *M-File Script* possono operare su variabili esistenti nel Workspace, oppure possono crearne di nuove
- Tutte le variabili che vengono create da tali script rimangono nel Workspace e possono essere usate per effettuare ulteriori calcoli

Esempio 2

media.m

```
a = 5;  
b = 3;  
c = 4;  
mean = (a + b + c) / 3
```

Script per calcolare
la media di 3 numeri



Workspace

M-File Script – 8/9

- Gli *M-File Script* possono operare su variabili esistenti nel Workspace, oppure possono crearne di nuove
- Tutte le variabili che vengono create da tali script rimangono nel Workspace e possono essere usate per effettuare ulteriori calcoli

Esempio 2

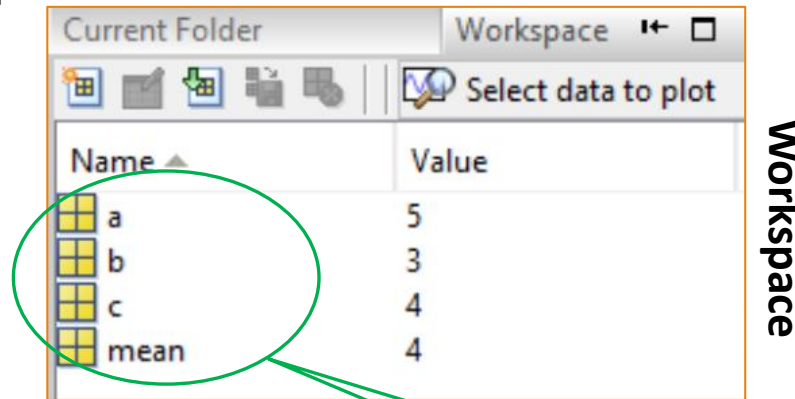
media.m

```
a = 5;  
b = 3;  
c = 4;  
mean = (a + b + c) / 3
```

Script per calcolare
la media di 3 numeri

Eseguo lo script
chiamato media

```
Command Window  
  
>> media  
  
mean =  
  
4
```



Name	Value
a	5
b	3
c	4
mean	4

Workspace

Variabili inserite nel
Workspace dopo
l'esecuzione dello
script

M-File Script – 9/9

- ***M-File Script***

- **VANTAGGI:**

- È possibile modificare (se necessario) comandi/istruzioni/funzioni nel file una sola volta, ed eseguire tale file (script) più volte

- **SVANTAGGI:**

- Tutte le variabili create all'interno dello script sono aggiunte al Workspace, e questo può portare a problemi indesiderati
 - *Ad Esempio*
 - Alcune variabili già esistenti nel Workspace vengono sovrascritte
 - Lo stato di alcune variabili già esistenti nel Workspace viene modificato
 - Etc

Commenti in MATLAB – 1/3

- Gli *M-File Script* (ma anche gli *M-File Function*) possono contenere qualsiasi serie di istruzioni/comandi/funzioni MATLAB, ma anche **commenti**
- Qualsiasi testo che segue un segno di percentuale (%) su una data linea è detto testo di **commento** ed è mostrato in **verde**.
- **I commenti**
 - Possono apparire
 - Su linee distinte rispetto alle istruzioni MATLAB
 - Alla fine di una istruzione MATLAB
 - Non vengono processati da MATLAB
- L'aggiunta di commenti è essenziale per la comprensione di programmi costituiti da un gran numero di istruzioni
 - A maggior ragione se il programma deve essere compreso da persone diverse dal suo autore

```
% Esercizio 1 dell'Esercitazione 2
% matrice
- format bank % per notazione con 2 cifre decimali (maggiori dettagli in seguito)
- m = [ 5 5.50 6.50 6 6.25
      40 43 37 50 45
      1000 1100 1000 1200 1100 ];

% a)
- guadagno_operaio_settimana == m(1,:) .* m(2,:)

% b)
- guadagno_operai_settimana == sum(guadagno_operaio_settimana)

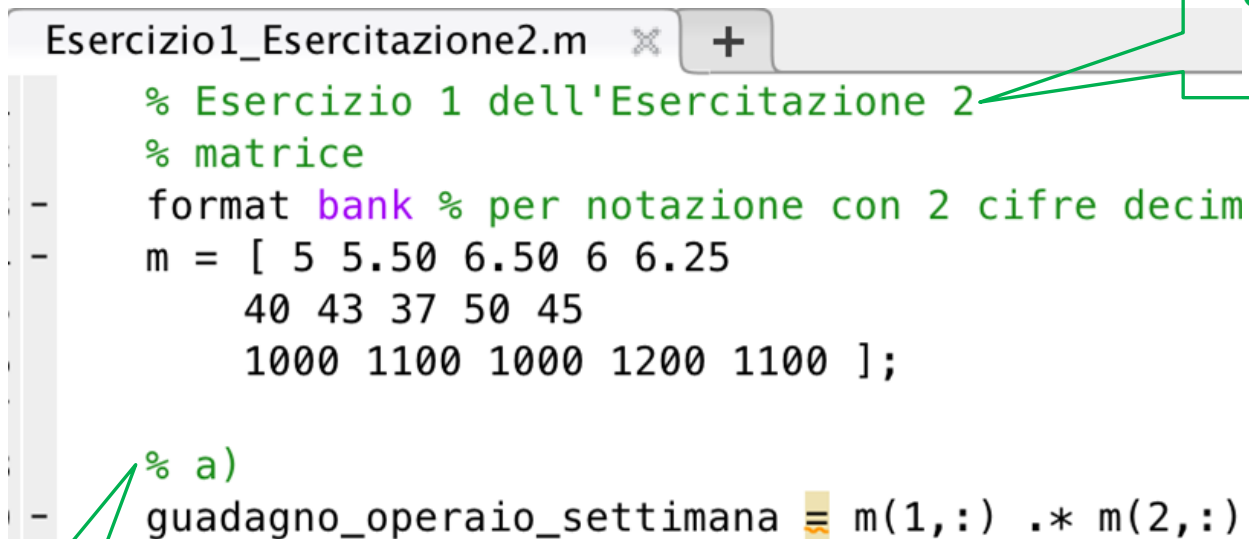
% c)
- pezzi_prodotti == sum(m(3,:))

% d)
- costo_medio_pezzo == guadagno_operai_settimana / pezzi_prodotti

% e)
- ore_totali = sum(m(2,:));
- ore_medie_pezzo == ore_totali / pezzi_prodotti

% f)
- format short % maggiori dettagli in seguito
- efficienza_operai == m(2,:) ./ m(3,:)
- operaio_piu_efficiente == max(efficienza_operai)
- operaio_meno_efficiente == min(efficienza_operai)
```

Commenti in MATLAB – 2/3



```
Esercizio1_Esercitazione2.m  x  +  
% Esercizio 1 dell'Esercitazione 2  
% matrice  
- format bank % per notazione con 2 cifre decim  
- m = [ 5 5.50 6.50 6 6.25  
      40 43 37 50 45  
      1000 1100 1000 1200 1100 ];  
  
% a)  
- guadagno_operai_settimana = m(1,:) .* m(2,:)
```

Commento su
linea distinta

Commento su
linea distinta

Commenti in MATLAB – 3/3

Commento a fine
istruzione

```
% f)
- format short % maggiori dettagli in seguito
- efficienza_operai = m(2,:) ./ m(3,:)
- operaio_piu_efficiente = max(efficienza_operai)
- operaio_meno_efficiente = min(efficienza_operai)
```

Funzioni – 1/4

- Una funzione è un **segmento (blocco) autonomo di programma** che esegue **un compito specifico**
- In termini più formali, una funzione (detta anche subroutine, metodo, procedura o sottoprogramma) è una **porzione di codice** all'interno di un programma più ampio, che svolge un **compito specifico** e può essere relativamente indipendente dal resto del codice
- Le **funzioni** rappresentano le **basi per costruire programmi più complessi**

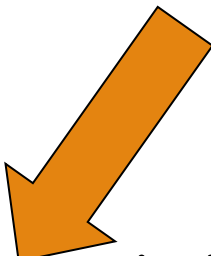
Input

$\mathbf{x} =$

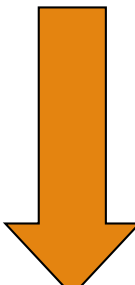
80	90	95	100	98	88	92
----	----	----	-----	----	----	----

Funzioni

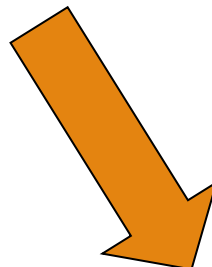
$y = \text{mean}(\mathbf{x})$



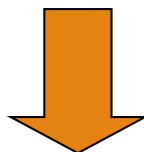
$y = \text{sum}(\mathbf{x})$



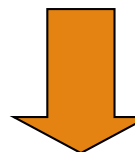
$y = \text{max}(\mathbf{x})$



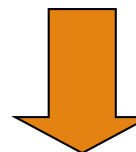
Output



91.85



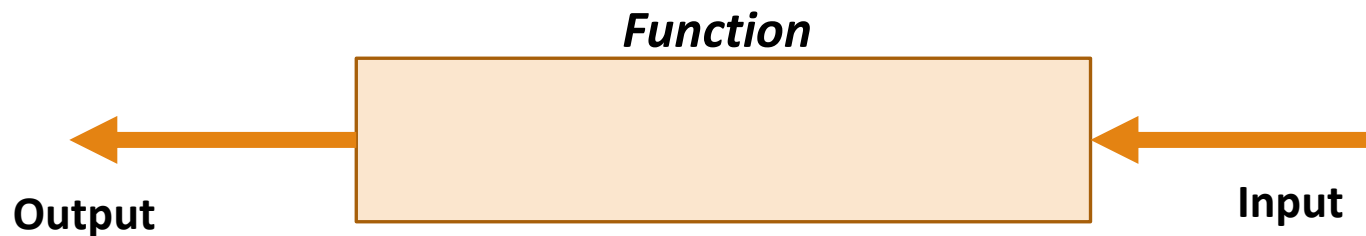
643



100

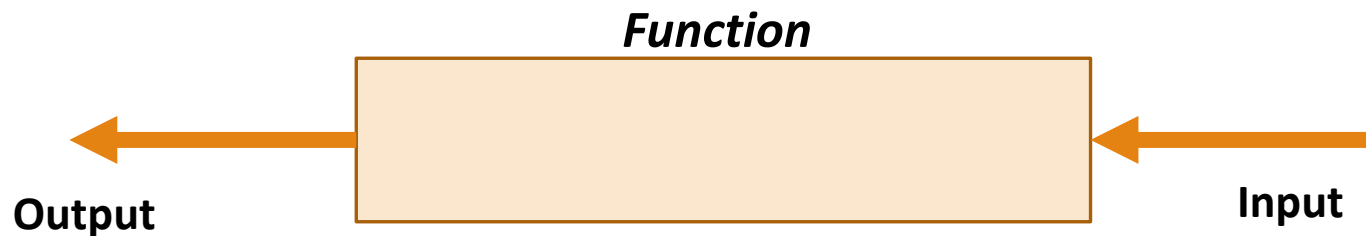
Funzioni – 2/4

- Una funzione è un **segmento (blocco) autonomo di programma** che esegue **un compito specifico**



Funzioni – 2/4

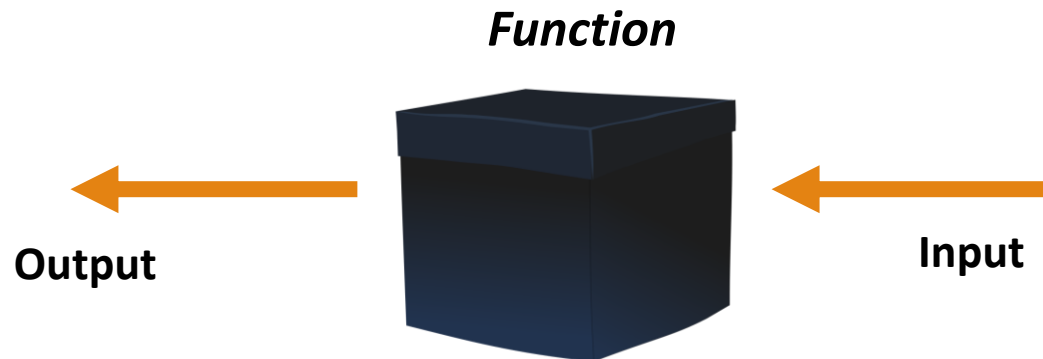
- Una funzione è un **segmento (blocco) autonomo di programma** che esegue **un compito specifico**



- Una funzione può
 - Accettare uno o più (ma anche zero) **argomenti in *input***
 - Restituire uno o più (ma anche zero) **argomenti in *output***

Funzioni – 3/4

- Una funzione può essere vista come una sorta di “***black box***”
- Una funzione comunica con il “mondo esterno” soltanto usando le proprie variabili di input e output
- Il suo codice sorgente ed il suo Workspace (**stato**) risultano nascosti al chiamante



Funzioni – 4/4

- ***Perché usare le funzioni?***

- ***Riusabilità***

- Una funzione può essere usata più volte, senza necessità di riscrivere ogni volta il codice sorgente (istruzioni) che essa contiene

- ***Leggibilità del codice***

- Un programma che risolve un problema complesso, può essere suddiviso più sotto-programmi (funzioni), ognuno dei quali risolve un sotto-problema (*divide-et-impera*)

- ***Gestibilità del codice***

M-File Function – 1/11

- MATLAB mette già a disposizione diverse funzioni, dette *funzioni built-in*
 - Alcune delle quali sono state utilizzate nelle lezioni precedenti
 - Ad esempio **max**, **sum**, **sqrt**, etc
- Inoltre, MATLAB permette all'utente di creare proprie funzioni, dette funzioni **user-defined**

M-File Function – 2/11

- Vediamo come creare in MATLAB una funzione *user-defined*
- Sintassi per creare una *funzione* definita dall'utente

```
function [variabili di output] = nome_funzione(variabili di input)
    <corpo_funzione>
end
```

M-File Function – 2/11

```
function [variabili di output] = nome_funzione(variabili di input)
    <corpo_funzione>
end
```

- Le **variabili di output** sono quelle i cui valori vengono calcolati dalla funzione, utilizzando i valori delle **variabili di input**
- Le **variabili di output** sono racchiuse tra parentesi quadre (che sono facoltative quando c'è un solo output)
- Le **variabili di input** devono essere racchiuse tra parentesi tonde
- La parola **function** nella riga di definizione della funzione deve essere scritta in lettere minuscole

M-File Function – 2/11

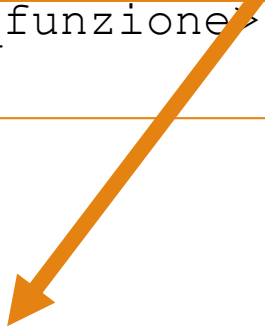
- Vediamo come creare in MATLAB una funzione *user-defined*
- **Sintassi** per creare una *funzione* definita dall'utente

```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)
    <corpo_funzione>
end
```

M-File Function – 2/11

- Vediamo come creare in MATLAB una funzione *user-defined*
- Sintassi per creare una *funzione* definita dall'utente

```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)  
    <corpo_funzione>  
end
```

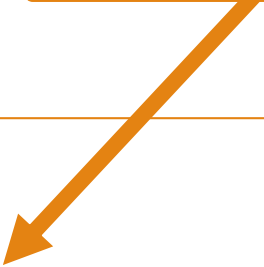


Output

M-File Function – 2/11

- Vediamo come creare in MATLAB una funzione *user-defined*
- **Sintassi** per creare una *funzione* definita dall'utente

```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)  
    <corpo_funzione>  
end
```



Input

M-File Function – 2/11

- Vediamo come creare in MATLAB una funzione *user-defined*
- Sintassi per creare una *funzione* definita dall'utente

```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)  
    <corpo_funzione>  
end
```



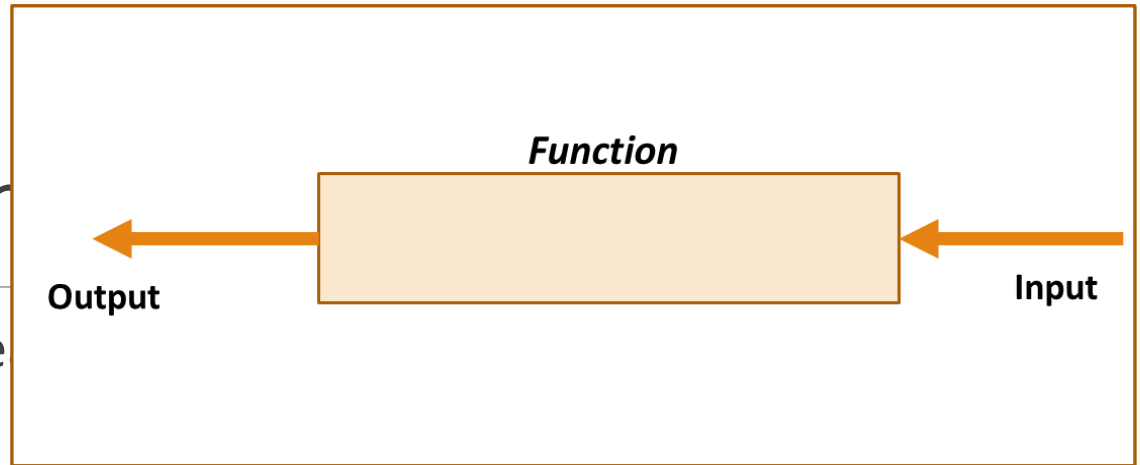
Output



Input

M-File Fun

- Vediamo come cre



- Sintassi per creare una *funzione* definita dall'utente

```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)  
    <corpo_funzione>  
end
```

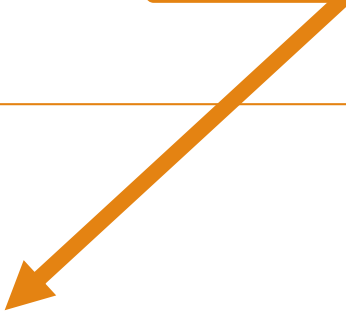
Output

Input

M-File Function – 2/11

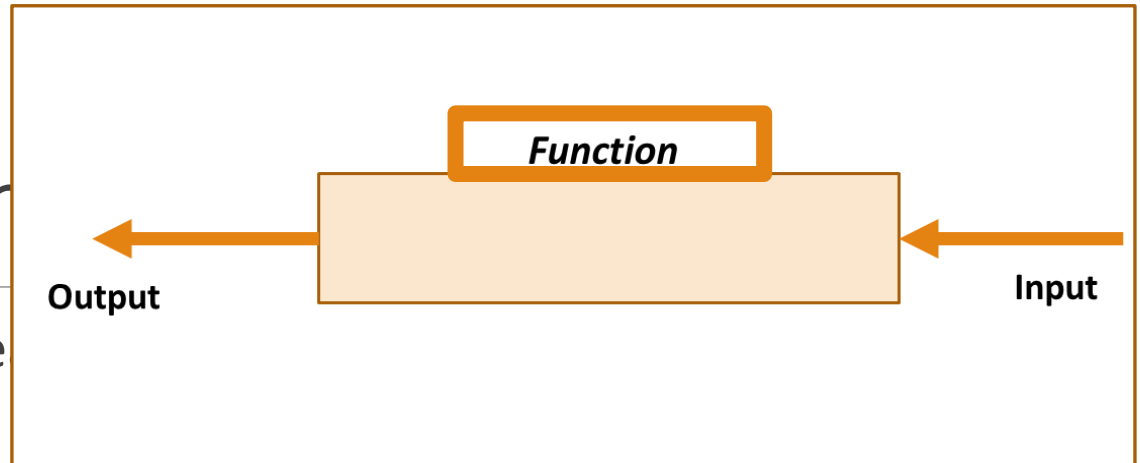
- Vediamo come creare in MATLAB una funzione *user-defined*
- Sintassi per creare una *funzione* definita dall'utente

```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)  
    <corpo_funzione>  
end
```



Nome della funzione

M-File Fun



- Vediamo come cre
- Sintassi per creare una *funzione* definita dall'utente

```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)  
    <corpo_funzione>  
end
```

Nome della funzione

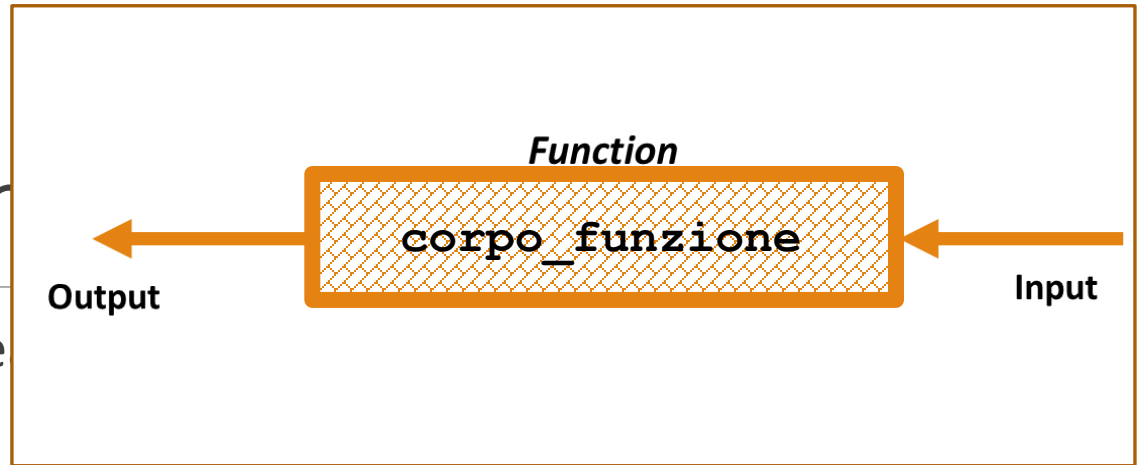
M-File Function- 2/11

- Vediamo come creare in MATLAB una funzione *user-defined*
- Sintassi per creare una *funzione* definita dall'utente

```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)
    <corpo_funzione>
end
```

M-File Fun

- Vediamo come cre



- Sintassi per creare una *funzione* definita dall'utente

```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)
    <corpo_funzione>
end
```

M-File Function – 3/11

- **Esempio 1** (Area Triangolo Equilatero)

```
function area = area_triangolo_equilatero(lato)

area = sqrt(3)/4 * lato^2;

end
```

Definizione (o dichiarazione) della funzione

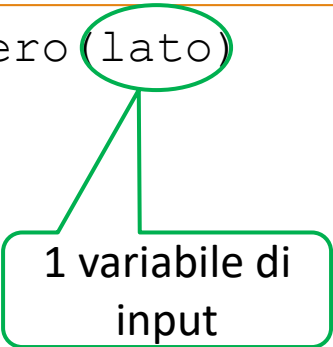
M-File Function – 3/11

- **Esempio 1** (Area Triangolo Equilatero)

```
function area = area_triangolo_equilatero(lato)

area = sqrt(3)/4 * lato^2;

end
```



Definizione (o dichiarazione) della funzione

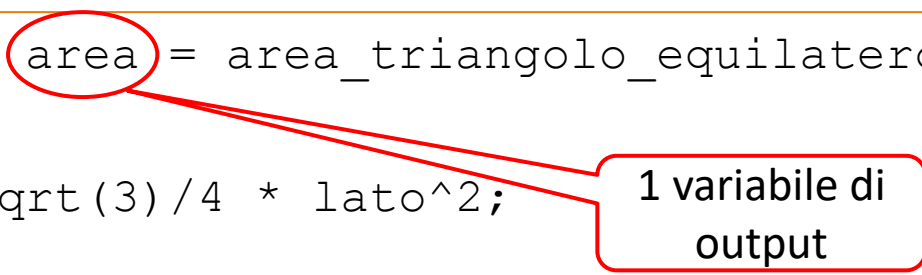
M-File Function – 3/11

- **Esempio 1** (Area Triangolo Equilatero)

```
function area = area_triangolo_equilatero(lato)

area = sqrt(3)/4 * lato^2;

end
```



Definizione (o dichiarazione) della funzione

M-File Function – 3/11

- **Esempio 1** (Area Triangolo Equilatero)

```
function area = area_triangolo_equilatero(lato)
    area = sqrt(3)/4 * lato^2;
end
```

The diagram highlights the output variable 'area' in the function definition and the assignment statement. A red box labeled '1 variabile di output' points to the 'area' variable in both lines of code.

Definizione (o dichiarazione) della funzione

N.B. I nomi delle variabili di output presenti nella definizione della funzione devono essere identici a quelli delle variabili in cui sono memorizzati i valori (calcolati) che la funzione deve restituire come output

M-File Function – 3/11

- **Esempio 1** (Area Triangolo Equilatero – Con Commenti)

```
function area = area_triangolo_equilatero(lato)

%La funzione prende in input la lunghezza di un lato e
%restituisce in output l'area del triangolo

%L'area del triangolo equilatero può essere calcolata
%dividendo per 4 la radice quadrata di 3; il risultato
%ottenuto da tale divisione deve essere moltiplicato per
%la dimensione del lato, elevata al quadrato

area = sqrt(3)/4 * lato^2;

end
```

M-File Function – 4/11

- **Esempio 2** (Area Sfera)

```
function area = area_sfera(raggio)

area = 4 * pi * raggio^2;

end
```

Definizione (o dichiarazione) della funzione

M-File Function – 5/11

- **Esempio 3** (Area e Volume Sfera)

```
function [area, volume] = area_volume_sfera(raggio)

area = area_sfera(raggio);
volume = 4/3 * pi * raggio^3;

end
```

Definizione (o dichiarazione) della funzione

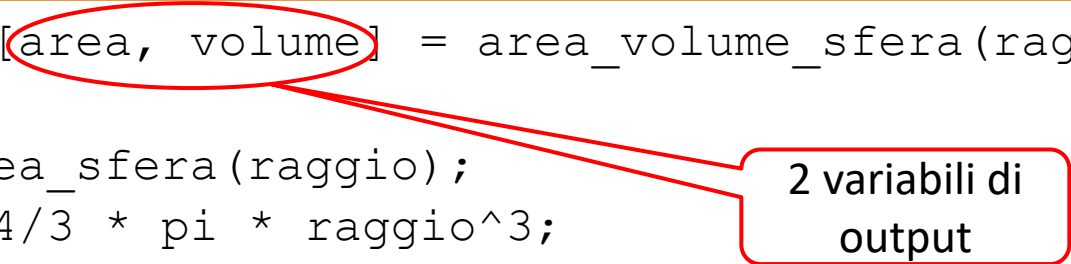
M-File Function – 5/11

- **Esempio 3** (Area e Volume Sfera)

```
function [area, volume] = area_volume_sfera(raggio)

area = area_sfera(raggio);
volume = 4/3 * pi * raggio^3;

end
```



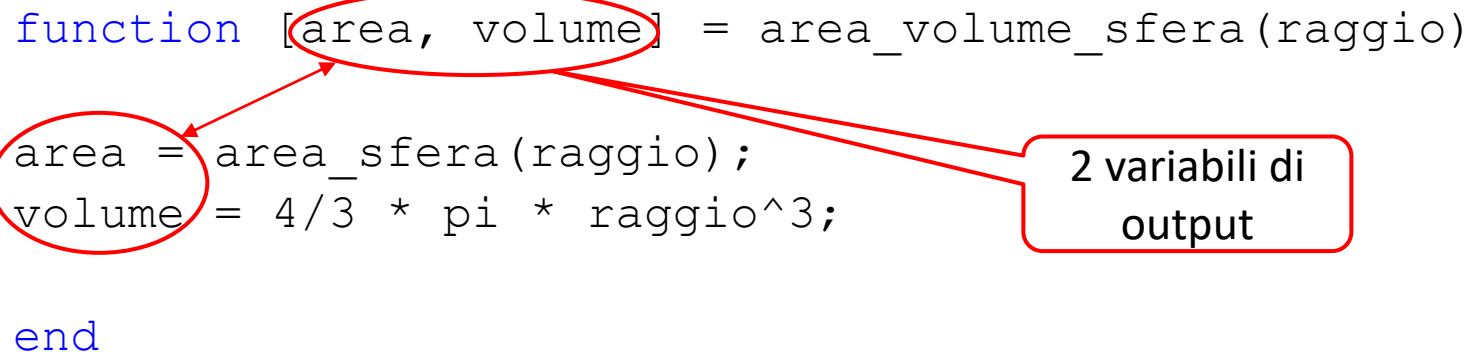
2 variabili di output

Definizione (o dichiarazione) della funzione

M-File Function – 5/11

- **Esempio 3** (Area e Volume Sfera)

```
function [area, volume] = area_volume_sfera(raggio)
area = area_sfera(raggio);
volume = 4/3 * pi * raggio^3;
end
```



2 variabili di output

Definizione (o dichiarazione) della funzione

N.B. I nomi delle variabili di output presenti nella definizione della funzione devono essere identici a quelli delle variabili in cui sono memorizzati i valori (calcolati) che la funzione deve restituire come output

M-File Function – 5/11

- **Esempio 3** (Area e Volume Sfera)

```
function [area, volume] = area_volume_sfera(raggio)

area = area_sfera(raggio); → invocazione a un'altra funzione
volume = 4/3 * pi * raggio^3;

end
```

- **Osservazione:** all'interno di una funzione è possibile invocare una o più funzioni *user-defined* e/o funzioni *built-in* di MATLAB
- **NOTA:** Le funzioni *user-defined*, per poter essere invocate, devono essere state precedentemente memorizzate (salvate) nel relativo *M-File Function*
 - Vediamo come...

M-File Function – 6/11

- Le funzioni *user-defined*, per poter essere invocate, devono essere state precedentemente memorizzate (salvate) nel relativo *M-File Function*

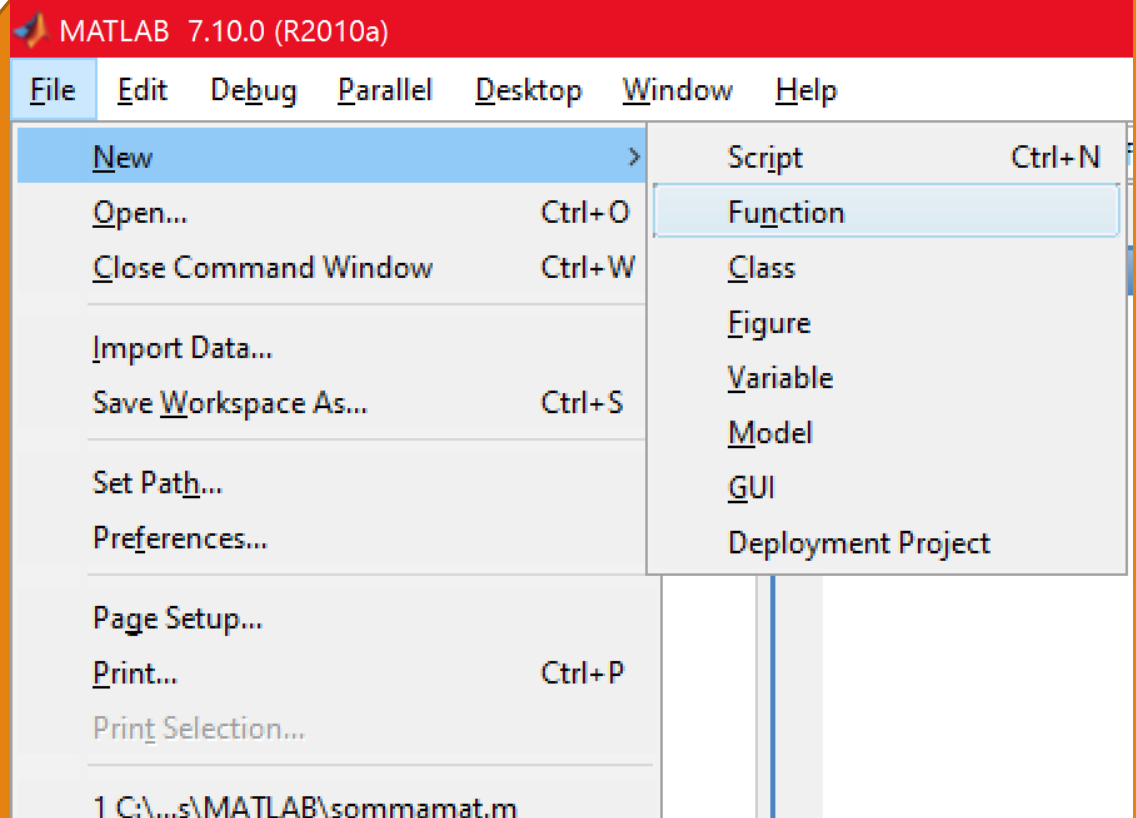
```
function [out1,out2,...,outN] = nome_funzione(in1,in2,...,inM)
    <corpo_funzione>
end
```

- **Salvare una funzione *user-defined* in un *M-File Function***
 - Il nome della funzione (**nome_funzione**) deve essere uguale al nome del file in cui sarà salvata tale funzione
 - Ad es., se il nome della funzione è **media**, tale funzione deve essere salvata nel file **media.m** (N.B. MATLAB suggerisce già il nome corretto da dare alla funzione)

M-File Fun

- Le funzioni *user-defined* state precedentemente *Function*

```
function [out1,out2,...  
    <corpo_funzione>  
end
```

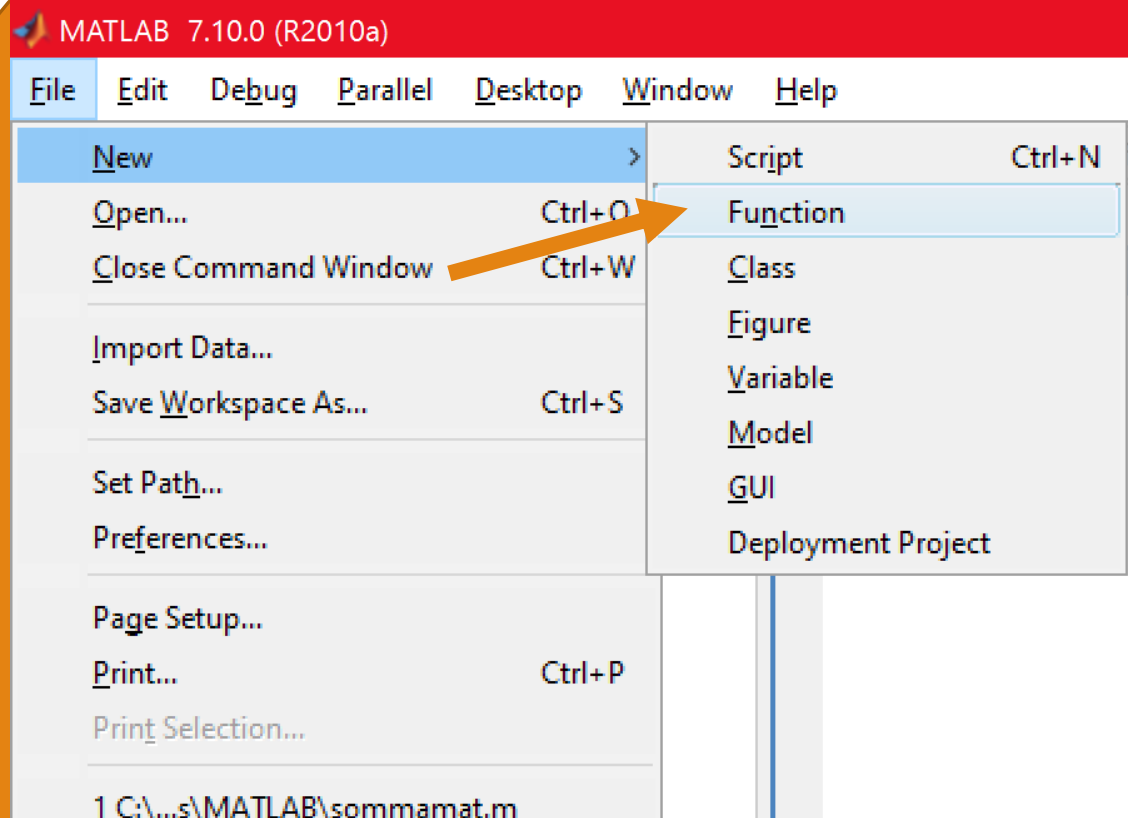


- Salvare una funzione *user-defined* in un M-File Function**
 - Il nome della funzione (**nome_funzione**) deve essere uguale al nome del file in cui sarà salvata tale funzione
 - Ad es., se il nome della funzione è **media**, tale funzione deve essere salvata nel file **media.m** (N.B. MATLAB suggerisce già il nome corretto da dare alla funzione)

M-File Fun

- Le funzioni *user-defined* state precedentemente *Function*

```
function [out1,out2,...  
    <corpo_funzione>  
end
```



- Salvare una funzione *user-defined* in un M-File Function**
 - Il nome della funzione (**nome_funzione**) deve essere uguale al nome del file in cui sarà salvata tale funzione
 - Ad es., se il nome della funzione è **media**, tale funzione deve essere salvata nel file **media.m** (N.B. MATLAB suggerisce già il nome corretto da dare alla funzione)

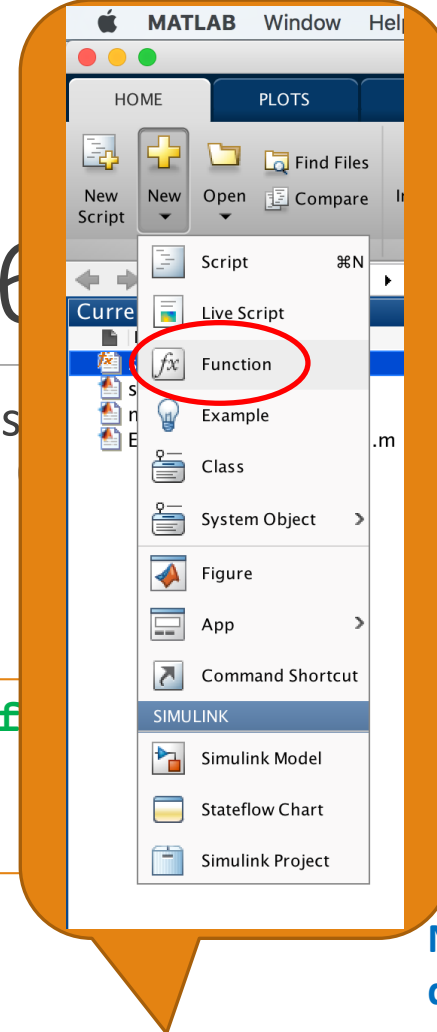
M-File Function —

- Le funzioni *user-defined*, per poter essere state precedentemente memorizzate *Function*

```
function [out1,out2,...,outN] = nome_f  
    <corpo_funzione>  
end
```

no essere
o *M-File*

, ..., inM)



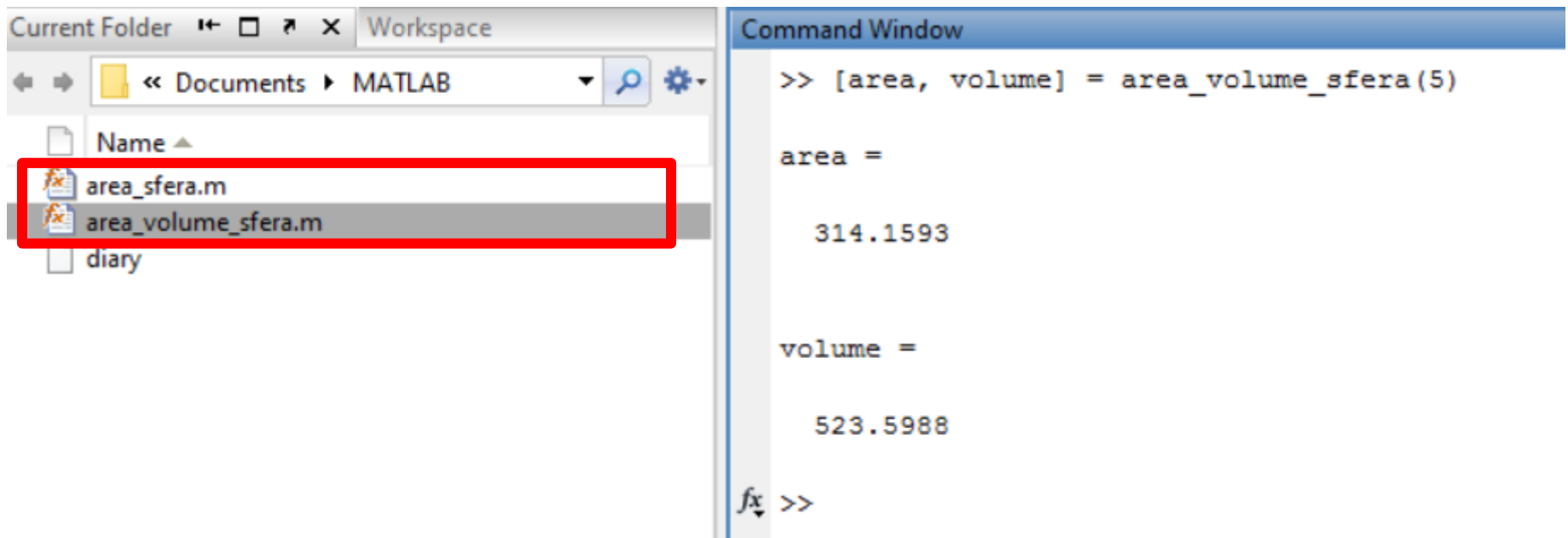
- Salvare una funzione *user-defined* in un *M-File Function***
 - Il nome della funzione (**nome_funzione**) deve essere uguale al nome del file in cui sarà salvata tale funzione
 - Ad es., se il nome della funzione è **media**, tale funzione deve essere salvata nel file **media.m** (N.B. MATLAB suggerisce già il nome corretto da dare alla funzione)

M-File Function – 7/11

- Un *M-file Function* ha estensione **.m** ed il suo contenuto deve iniziare con la parola chiave `function`
 - Seguita da eventuali parametri di input e di output
- Ogni *M-file Function* ha un proprio workspace, **separato dal Workspace mostrato in MATLAB**
 - Tutte le variabili all'interno dell'*M-file Function* vengono dette **"locali"** ad esso
 - Esistono soltanto all'interno della funzione stessa
 - Non vengono viste dall'ambiente MATLAB o da altre eventuali *M-file Function* chiamanti

M-File Function – 8/11

- Una volta memorizzata la funzione *user-defined* nel relativo *M-File Function* (nella Current Directory), tale funzione può essere invocata dalla *Command Window* di MATLAB



M-File Function – 8/11

- Dopo che è stata creata (dichiarata), una funzione può essere utilizzata (invocata), fornendogli in input gli opportuni parametri

```
function area = area_triangolo_equilatero(lato)

area = sqrt(3)/4 * lato^2;

end
```

Dichiarazione

```
>> area_triangolo_equilatero(3)

ans =

    3.8971
```

Invocazione

M-File Function – 8/11

- Dopo che è stata creata (dichiarata), una funzione può essere utilizzata (invocata), fornendogli in input gli opportuni parametri

```
function area = area_triangolo_equilatero(lato)

area = sqrt(3)/4 * lato^2;

end
```

Dichiarazione

```
>> area_triangolo_equilatero(3)

ans =

    3.8971
```

Invocazione

M-File Function – 8/11

- **N.B. È importante notare la differenza tra definizione (Dichiarazione) della funzione ed uso (Invocazione) della funzione stessa**

```
function area = area_triangolo_equilatero(lato)

area = sqrt(3)/4 * lato^2;

end
```

Dichiarazione

```
>> area_triangolo_equilatero(3)

ans =

    3.8971
```

Invocazione

M-File Function – 8/11

- I valori restituiti in output da una funzione possono essere assegnati a variabili
 - Che saranno visibili nel Workspace di MATLAB

```
function mySum = sumTwoNums (a,b)

mySum = a+b;

end
```

```
>> somma = sumTwoNums (3,4)

somma =

    7
```

Workspace	
Name ▲	Value
 somma	7

M-File Function – 8/11

- I valori restituiti in output da una funzione possono essere assegnati a variabili
 - Che saranno visibili nel Workspace di MATLAB

```
>> [area, volume] = area_volume_sfera(6)
```

```
area =  
    452.3893  
volume =  
    904.7787
```

Workspace	
Name ▲	Value
 area	452.3893
 volume	904.7787

M-File Function – 9/11

- **Parametri formali**

- I parametri formali sono quelli indicati in fase di dichiarazione della funzione

- ***Esempio***

```
function [area, volume] = area_volume_sfera(raggio)  
.  
.  
.  
end
```

- raggio è un parametro formale di input della funzione area_volume_sfera

M-File Function – 10/11

- **Parametri attuali**

- I parametri attuali sono quelli indicati in fase di invocazione della funzione

- ***Esempio***

```
.  
.   
area = area_sfera(raggio);  
.   
.   
. 
```

- *In questo caso, raggio è un parametro attuale di input della funzione invocata area_sfera*

M-File Function – 11/11

- I parametri possono essere di qualsiasi tipo
 - Array, matrici, scalari, etc..
- I parametri attuali vengono associati a quelli formali tenendo conto della posizione
 - Il primo parametro attuale viene associato al primo parametro formale, il secondo attuale al secondo formale, etc..
- È necessario che l'invocazione a una funzione avvenga con un numero di parametri attuali uguale al numero dei parametri formali

M-File Function – 11/11

Dichiarazione

```
function area_triangolo = areaTriangolo(base, altezza)  
  
area_triangolo = (base * altezza) / 2;  
  
end
```

2 parametri formali:
corrispondenti
rispettivamente alla
base ed all'altezza

N.B. Il primo parametro attuale deve corrispondere al primo parametro formale, il secondo parametro attuale deve corrispondere al secondo parametro formale e così via...

Invocazione

```
>> area_triangolo = areaTriangolo(5, 3)  
  
area_triangolo =  
  
7.5000
```

2 parametri attuali:
corrispondenti
rispettivamente alla
base ed all'altezza

Input/Output – 1/6

- MATLAB fornisce vari comandi che permettono di ottenere l'input degli utenti e formattare i dati di output (i risultati ottenuti eseguendo i comandi di MATLAB)

Input/Output – 2/6

- Con il comando `input` è possibile ottenere un input da parte dell'utente tramite il prompt del Command Window
- *Esempio*

```
>> x = input('Inserisci x: ')
```

Input/Output – 2/6

- Il comando **input** visualizza un testo sullo schermo, aspetta che l'utente digiti qualcosa e poi memorizza l'input nella variabile specificata
- *Esempio*

```
>> x = input('Inserisci x: ')
```

Variable in cui
verrà memorizzato
l'input digitato
dall'utente

Stringa mostrata a video

Input/Output – 2/6

- Con il comando **input** è possibile ottenere un input da parte dell'utente tramite il prompt del Command Window
- *Esempio*

```
>> x = input('Inserisci x: ')\nInserisci x:
```

Input/Output – 2/6

- Con il comando **input** è possibile ottenere un input da parte dell'utente tramite il prompt del Command Window
- *Esempio*

```
>> x = input('Inserisci x: ')\nInserisci x:
```



Attende l'input dell'utente

Input/Output – 2/6

- Con il comando **input** è possibile ottenere un input da parte dell'utente tramite il prompt del Command Window
- *Esempio*

```
>> x = input('Inserisci x: ')\nInserisci x: 45\n\nx =\n\n    45
```

Input/Output – 2/6

- Con il comando **input** è possibile ottenere un input da parte dell'utente tramite il prompt del Command Window
- *Esempio*

```
>> x = input('Inserisci x: ')
```

```
Inserisci x: 45
```

```
x =
```

```
45
```

Memorizza nella variabile **x** il
valore preso in input, ovvero, **45**

Input/Output – 3/6

- *Esempio 2*

```
>> a = input('a: ');  
a: 125  
>> b = input('b: ');  
b: 270  
>> c = input('c: ');  
c: 391  
>> average = (a+b+c)/3  
  
average =  
  
262
```

Input/Output – 3/6

Command Window

```
>> help input
```

input Prompt for user input.

RESULT = **input**(PROMPT) displays the PROMPT string on the screen, waits for input from the keyboard, evaluates any expressions in the input, and returns the value in RESULT. To evaluate expressions, **input** accesses variables in the current workspace. If you press the return key without entering anything, **input** returns an empty matrix.

STR = **input**(PROMPT,'s') returns the entered text as a MATLAB string, without evaluating expressions.

Come al solito, maggiori informazioni sul comando possono essere ottenute utilizzando il comando **help**

Input/Output – 4/6

- Generalmente vengono utilizzati due modi per mostrare l'output in MATLAB
 - `disp`
 - `fprintf`
- Il comando `disp` ha il vantaggio di essere molto semplice da utilizzare, ma fornisce un controllo limitato su ciò che può essere mostrato in output
- Il comando `fprintf` è estremamente completo nella gestione dell'output, ma non è di facile utilizzo
 - Possibilità di specificare numerose opzioni riguardanti come verrà visualizzato l'output (maggiori informazioni digitando il comando `help fprintf`)

Input/Output – 4/6

- Esempi di utilizzo del comando `disp`

```
>> disp('stringa')  
stringa
```

```
>> disp(['stringa1', 'stringa2', 'stringa3'])  
stringa1stringa2stringa3
```

```
>> disp(['stringa ', num2str(10)])  
stringa 10
```

Input/Output – 4/6

- Esempi di utilizzo del comando `disp`

```
>> disp('stringa')  
stringa
```

```
>> disp(['stringa1', 'stringa2', 'stringa3'])  
stringa1stringa2stringa3
```

```
>> disp(['stringa ', num2str(10)])  
stringa 10
```

IMPORTANTE: `num2str` trasforma un numero in una stringa, in modo che possa essere stampata da `disp`

Input/Output – 5/6

- Esempio 1 (Utilizzo comando `disp`)

```
>> a = 46; b = 35; c = 100;
>> disp('Stamperò il valore di a e la somma b + c')
Stamperò il valore di a e la somma b + c
>> disp(['Il valore della variabile a è :', num2str(a)])
Il valore della variabile a è :46
>> disp(['La somma b + c è:', num2str(b+c)])
La somma b + c è:135
```

Input/Output – 6/6

- È possibile decidere il formato di visualizzazione dei risultati prodotti da MATLAB, mediante il comando **format**
 - Il comando **format** determina l'aspetto dei numeri sullo schermo
- MATLAB utilizza molte cifre significative nei suoi calcoli, ma raramente servono tutte
- Il formato standard di visualizzazione di MATLAB utilizza quattro cifre decimali

Input/Output – 6/6

- È possibile decidere il formato di visualizzazione del risultato tramite il comando **format**
 - **format short**: 4 cifre decimali (formato standard o di *default*); Ad es., 13.6745
 - **format long** : 16 cifre. Ad es., 17.27484029463547
 - **format short e**: 5 cifre (4 decimali) più l'esponente. Ad es., 6.3792e+03
 - **format long e**: 16 cifre (15 decimali) più l'esponente. Ad es., 6.379243784781294e-04
 - **format bank**: 2 cifre decimali. Ad es., 126.73
 - **format +**: Positivo, negativo o zero. Ad es., +
 - **format rat**: Approssimazione razionale. Ad es., 43/7
 - **format compact**: Elimina le righe vuote
 - **format loose**: Annulla l'effetto di **format compact**

Input/Output – 6/6

È possibile decidere il formato di visualizzazione del risultato tramite il comando `format`

`format short`: 4 cifre decimali (formato standard o di *default*);
Ad es., 13.6745

`format long`: 16 cifre. Ad es., 17.27484029463547

- `format short e`: 5 cifre (4 decimali) più l'esponente. Ad es., 6.3792e+03
- `format long e`: 16 cifre (15 decimali) più l'esponente. Ad es., 6.379243784781294e-04

`format bank`: 2 cifre decimali. Ad es., 126.73

N.B. In questo contesto `e` non rappresenta il numero `e` di *Nepero*, che è alla base dei logaritmi naturali, ma l'iniziale della parola "esponente"

`format compact`: Elimina le righe vuote

`format loose`: Annulla l'effetto di `format compact`

Riferimenti

- Capitolo 1
 - Paragrafo 1 (**Comandi di formattazione**)
 - Paragrafi 4 [**File script ed Editor/Debugger**] e 5 [**La guida di MATLAB**]
- Capitolo 3
 - Paragrafi 1 [**Funzioni matematiche di base**] e 2 [**Funzioni definite dall'utente, fino a Varianti nella chiamata di una funzione (incluso)**]