Dart

In 8 Hours

For Beginners
Learn Coding Fast!

Ray Yao

Matlab

In 8 Hours

For Beginners

Learn Coding Fast

Ray Yao

About the Author : Ray Yao's Team

Certified PHP engineer by Zend , USA

Certified JAVA programmer by Sun , USA

Certified SCWCD developer by Oracle , USA

Certified A+ professional by CompTIA , USA

Certified ASP . NET expert by Microsoft , USA

Certified MCP professional by Microsoft , USA

Certified TECHNOLOGY specialist by Microsoft , USA

Certified NETWORK+ professional by CompTIA , USA

www . amazon . com/author/ray-yao

Recommended Books by Ray Yao's Team

Advanced C++ in 8 Hours

Advanced Java in 8 Hours

AngularJs in 8 Hours

Asp．net Programming

Awk in 8 Hours

BootStrap in 8 Hours

C# Examples & Interview

C# Programming

C++ Examples & Interview

C++ Programming

Dart in 8 Hours

Django in 8 Hours

Erlang in 8 Hours

Go in 8 Hours

Html Css Examples & Interview

Html Css Programming

Java Examples & Interview

Java Programming

JavaScript Examples & Interview

JavaScript Programming

JQuery Examples & Interview

JQuery Programming

Jsp Servlets Programming

Kotlin in 8 Hours

Linux Command Line

Linux Examples & Interview

Lua in 8 Hours

Matlab in 8 Hours

MySql in 8 Hours

Node．Js in 8 Hours

Numpy in 8 Hours

Perl in 8 Hours

Php Examples & Interview

Php MySql Programming

PowerShell in 8 Hours

Python Examples & Interview

Python Programming

R Programming

React . Js in 8 Hours

Ruby Programming

Rust in 8 Hours

Scala in 8 Hours

Shell Scripting in 8 Hours

Swift in 8 Hours

Tcl in 8 Hours

TypeScript in 8 Hours

Visual Basic Examples & Interview

Visual Basic Programming

Vue . Js in 8 Hours

Xml Json in 8 Hours

Preface

“Matlab Programming in 8 Hours” covers all essential Matlab knowledge . You can learn complete primary skills of Matlab fast and easily . The book includes many practical examples for beginners and includes exercises for the college exam , the engineer certification exam , and the job interview exam .

Prerequisite to learn Matlab

Before learning the Matlab , you should have basic knowledge of vector , array and matrix , because Matlab works with vector , array and matrix .

Source Code for Download

This book provides source code for download; you can download the source code for better study, or copy the source code to your favorite editor to test the programs .

Table of Contents

Matlab _ Q & A

Source Code Download

Hour 1

What is Matlab Language?

Matlab is short for Matrix Laboratory . It is a commercial math software produced by The Mathworks . Matlab is a high-level technical computing language used for algorithm development , data visualization , data analysis and numerical calculation . In addition to matrix operation , draw graph , digital images and other common functions , Matlab can also be used to create user interfaces and to reference programs written in other languages .

Although Matlab is primarily used for numerical computation , it is also suitable for a variety of applications with a large number of additional toolboxes . For example , control system design and analysis , image processing , signal processing , communication and financial modeling and analysis……

In addition , Matlab also has a supporting software package Simulink , which provides a visual development environment , often used in system simulation , dynamic/embedded system development and other aspects .

Matlab is a matrix programming language that made linear algebra programming simple . Matlab has a multitude of built-in commands and mathematical functions that can help you in mathematical calculations , plotting , and executing numerical calculation methods .

The Role of Matlab

MATLAB is widely used as a computational tool in the fields of science and engineering , covering physics , chemistry , mathematics and engineering flow . Its areas of work include :

01 . Statistics and data analysis

02 . Control system design and analysis

03 . Financial modeling and analysis

04 . Signal processing and communication

05 . Image and video processing

06 . Test and measurement

07 . Computational biology

08 . Mathematics and Optimization

09 . Application deployment

10 . Database connections and reports

Matlab Editors

There three kinds of Matlab editors of choice .

1 . Matlab editor at mathworks . com

http : //www . mathworks . com/downloads/web_downloads

http s : //ww w . mathwork s . com/campaigns/products/trials/matla b . html

Note : This is a paid software or 30-days-trial software .

2 . Matlab / Octave editor at octave-online . net

https : //octave-online . net

Note : This is a free online editor .

3 . Matlab / Octave editor at tutorialspoint . com

https : //www . tutorialspoint . com/execute_matlab_online . php

Note : This is a free online editor .

You can select one of the above Matlab editors as you like .

Matlab Basic

The Matlab Editor works as a super complex calculator , it can make various simple or complicated arithmetical calculations .

Example

```
100 + 200
```

Output : ans = 300

Example

```
100 - pi
```

Output : ans = 96 . 858

Example

```
cos(0 . 5)
```

Output : ans = 0 . 87758

Example

```
100 / 0
```

Output : ans = Inf warning : division by zero

Special Variable & Constant

ans	results of operations , it means “answer”
. ^	power
eps	relative error of floating point number
i , j	imaginary unit , i2 = j2 = -1
Inf	infinite
NaN	not a number
pi	circumference ratio
%	comment symbol
;	end of a statement

Example 1.1

x = 10 ;

y = x + 20 ; % it means 10 + 20

y . ^2 % it means y^2

Output:

ans = 900

Explanation:

“ ; ” ends each statement . “%” is a comment symbol .

“ . ^” means “power” . “ans” means “answer” .

Rule of Variable Name

1 . The variable name is composed of a letter followed by any number of letters , numbers , or underscores .

2 . The variable name is case sensitive .

3 . Variable names can be of any length .

4 . Variable name cannot contain spaces ..

For example:

The **valid** variable names :

hero007

very_good

The **invalid** variable names :

007hero

very good

Variable

A variable is a container storing various values that are mutable in program runtime **.**

The variable must be initialized before use **.**

For example **:**

```
num = 100 ;    % define a variable num , initialize a value 100
```

When no result variable is specified **,** the result of the operation will be stored in the default variable “ans”

For example **:**

```
100 + 200 ;     % the result is 300 , which stores in “ans”
900 / ans       % 900 divides 300 , finally output :
 ans = 3
```

Multiple variables can be initialized simultaneously

For example **:**

```
a = 10 ; b =  20 ; c = 30 ;
a + b + c     % output :  ans = 60
```

Show About Variables

Show details of all variable that are used

```
whos
```

Example 1.2

a = 10 **;** b = 20 **;** c = 30 **;**

a + b + c **;**

whos

Output:

```
Variables in the current scop e  :

  Attr   Name   Size    Bytes Class
  ====   ====   ====    ===== =====
       a      1x1        8 double
       ans    1x1        8 double
       b      1x1        8 double
       c      1x1        8   double

 Total is 4 elements using 32 bytes
```

Note:

“ **who** ” can also show the variants that are used **,** but not in detail **.**

“ **clear** variabl_name” can remove a specified variable **.**

Statement Too Long

If a statement is too long , “ … ” symbol can turn the statement to next line .

Example 1.3

```
number1 = 1000 ;
number2 = 2000 ;
number3 = 3000 ;
number4 = 4000 ;
result = number1 + number2 ...      % to next line
+ number3 - number4
```

Output:

result = 2000

Explanation:

If a statement too long , we can use “ … ” symbol to turn the statement to next line .

Decimal Place

format long	% displays 16 decimal places
format short	% displays 4 decimal places
format bank	% displays 2 decimal places

Example 1.4

```
format long
num1 = 10 / 3
format short
num2 =  10 / 3
format bank
num3 = 10 / 3
```

Output:

num1 = 3.33333333333333

num2 = 3.3333

num3 = 3.33

Explanation:

“format long” displays 16 decimal places

“format short” displays 4 decimal places

“format bank” displays 2 decimal places

Display as Exponent

“e” symbol makes a result shown as an exponent **.**

```
format long e     % displays 16 decimal places as an exponent
format short e    % displays 4 decimal places as an exponent
```

Example 1.5

```
format long e
num1 = pi
format short e
num2 = pi
```

Output:

num1 = 3.14159265358979e+00

num2 = 3.1416e+00

Explanation:

format long e % displays 16 decimal places as an exponent

format short e % displays 4 decimal places as an exponent

Matlab File

If you have installed a Matlab editor on your local computer , you can make use of the Matlab editor to create a Matlab file , edit the Matlab code , and save the file as xxx . m . (The extension name of Matlab file is . m) . Finally , click the "Run" button to execute the Matlab file .

Example 1.6

Open your local Matlab editor , input the code as follows :

```
a = 10 ; b = 20 ;
d = a + sin(b)
f = exp(-e)
g = a . ^2      % a . ^2 means a ²
h = b . ^2      % b . ^2 means b ²
```

Save the above as myfile . m in local computer , and click the "Run" button to execute the file .

Output:

d = 1 0 . 913
f = 0 . 065988
g = 100
h = 400

Explanation: The extension name of Matlab file is . m .

Hour 2

Data Type

int8	8 bits integer
uint8	8 bits unsigedn integer
int16	16 bits integer
uint16	16 bits unsigned integer
int32	32 bits integer
uint32	32 bits unsigned integer
int64	64 bits integer
uint64	64 bits unsigned integer
single	value of single precision
double	value of double precision
logical	value is 1 or 0 , represent true or false
char	value of a character

Example

```
format long
a = 100/3          % output a =  33 . 3333333333333
b = single(a)        % output b = 33 . 333332
c = int32(b)         % output c = 33
d = c -100           % output d = -67
e = uint16(d)        % output e = 0
```

Conversion

int2str	convert an integer to a string
str2int	convert a string to an integer
num2str	convert a number to a string
str2num	convert a string to a number
bin2dec	convert a binary to a decimal
dec2bin	convert a decimal to a binary

Example 2.1

result1 = **str2num** ('168')

result2 = **dec2bin** (result1)

Output:

result1 = 168

result2 = 10101000

Explanation:

“ **str2num** ('168')” converts the string “168” to a number **.**

“ **dec2bin** (result1)” converts the decimal 168 to a binary **.**

Check Data Type

isa	Check if it's an object a specified class
iscell	Check if it's a cell array
iscellstr	Check if it's a cell array of a string
ischar	Check if it's a character array
isfield	Check if it's a struct field
isfloat	Check if it's a float array
ishghandle	Check if it's a handler to processe graphy
isinteger	Check if it's an integer array
isjava	Check if it's a Java object
islogical	Check if it's a logical array
isnumeric	Check if it's a digital array
isobject	Check if it's a MATLAB object
isreal	Check if it's a real array
isscalar	Check if it's a scalar
isstr	Check if it's a string array
isstruct	Check if it's a strict array
isvector	Check if it's a vector

Example 2.2

```
v = 123 . 45
isfloat(v)          % output  ans = 1
isnumeric(v)        % output  ans = 1
isinteger(v)        % output  ans = 0
isvector(v)         % output  ans = 1
isscalar(v)         % output  ans = 1
```

Example 2.3

```
v = 'Go in 8 Hours'
isfloat(v)          % output  ans = 0
isnumeric(v)        % output  ans = 0
isvector(v)         % output  ans = 1
isscalar(v)         % output  ans = 0
isinteger(v)        % output  ans = 0
```

Display String Value

The function disp() can show the value of a string **.**

```
disp( ‘string’ ) ;
```

Example 2.4

```
name = 'Smith' ;
disp (name) ;
age = 18 ;
disp (['The age of ' , name , ' is ' ,   num2str(age)]) ;
```

Output:

Smith

The age of Smith is 18

Explanation:

“ **disp** (name) **;** ” shows the value of the name **.**

“num2str(age)” converts the age to string type **.**

“num2str(number)” converts number type to string type **.**

Arithmetical Operators

Operators	Running
+	add
-	subtract
*	multiply
/	divide
.^	power

Example 2.5

```
x = 100 ; y = 2 ;
v1 = x + y          % output  v1 =  102
v2 = x - y          % output  v2 =  98
v3 = x * y          % output  v3 =  200
v4 = x / y          % output  v4 =  50
v5 = x .^ y         % output  v5 =  10000
```

Arithmetical Functions (1)

Functions	Meaning
plus(a , b)	a + b
minus(a , b)	a - b
times(a , b)	a * b
idivide(a , b)	a / b (returns an integer)
power(a , b)	a . ^ b
uplus(a)	+ a
uminus(a)	- a

Example 2.6

a = 10 ; b = 2 ;

plus (a , b) % output ans = 12

minus(a , b) % output ans = 8

times(a , b) % output ans = 20

idivide(a , b) % output ans = 5

power(a , b) % output ans = 100

a + uplus(b) % output ans = 12

a + uminus(b) % output ans = 8

Arithmetical Functions (2)

Functions	Returns
mod (a , b)	• a modula number
rem (a , b)	• a remainder number
round(a)	a rounded number
ceil(num)	nearest integer greater than or equal to nu m .
floor(num)	nearest integer less than or equal to nu m .
fix(num)	rounded num to nearest integer towards zero

Example 2.7

```
a = 10 ; b = 3 ; num = a / b ;
v1 = mod (a , b)          % output  v1 =  1
v2 = rem (a , b)          % output  v2 =  1
v3 = ceil(num)            % output  v3 =  4
v4= floor(num)            % output  v4 =  3
v5 = round(num)           % output  v5 =  3
v6 = fix(num)             % output  v6 =  3
```

Logical Operators

Operators	Equivalent	Result
&&	and	1 or 0
||	or	1 or 0
~	not	1 or 0

“1” represents true **,** “0” represents false **.**

Example 2.8

```
a = 1 ; b = 0 ;
disp(['a && b returns ' , num2str(a && b)]) ;
disp(['a || b returns ' , num2str(a || b)]) ;
disp(['~a returns ' , num2str( ~ a)]) ;
disp(['~b returns ' , num2str( ~ b)]) ;
```

Output:

a && b returns 0 % && means “and”

a || b returns 1 % || means “or”

~a returns 0 % ~ means “not”

~b returns 1 % ~ means “not”

Logical Functions

Functions	Equivalent	Result
and()	&&	1 or 0
or()	||	1 or 0
not()	~	1 or 0

“1” represents true **,** “0” represents false **.**

Example 2.9

```
a = 1 ; b = 0 ;
disp(['and(a , b) returns ' , num2str( and(a,b) )]) ;
disp(['or(a , b) returns ' , num2str( or(a,b) )]) ;
disp(['not(a) returns ' , num2str( not(a) )]) ;
disp(['not(b) returns ' , num2str( not(b) )]) ;
```

Output:

```
and(a , b) returns 0        % and() means “&&”
or(a , b) returns 1         % or() means “||”
not(a) returns 0            % not() means “~”
not(b) returns 1            % not() means “~”
```

Comparison Operators

Operators	Running	Result
>	greater than	1 or 0
<	less than	1 or 0
>=	greater than or equal	1 or 0
<=	less than or equal	1 or 0
==	equal	1 or 0
~ =	not equal	1 or 0

Example 2.10

```
a = 10 ; b = 20 ;
disp(['a >= b returns ' , num2str(and(a >= b))]) ;
disp(['a == b returns ' , num2str(and(a == b))]) ;
disp(['a ~= b returns ' , num2str(and(a ~= b))]) ;
```

Output:

```
a >= b returns 0        % >= means “greater than”
a == b returns 0        % == means “equal”
a ~= b returns 1        % ~= means “not equal”
```

Check Equality

eq(a , b)	check if a is equal to b
ne(a , b)	check if a is not equal to b
gt(a , b)	check if a is greater than b
lt(a , b)	check if a is less than b
ge(a , b)	check if a is greater than or equal to b
le(a , b)	check if a is less than or equal to b

Example 2.11

```
a = 1 ; b = 0 ;
disp(['eq(a , b) returns ' , num2str( eq(a,b) )]) ;
disp(['ne(a , b) returns ' , num2str( ne(a,b) )]) ;
disp(['gt(a , b) returns ' , num2str( gt(a,b) )]) ;
disp(['lt(a , b) returns ' , num2str( lt(a,b) )]) ;
```

Output:

```
eq(a , b) returns 0        % eq() checks “equal”
ne(a , b) returns 1        % ne() checks “not equal”
gt(a , b) returns 1        % gt() checks “greater than”
lt(a , b) returns 0        % lt() checks “less than”
```

Hour 3

Print Format

printf() is used to print a value according to a specific format **.**

printf (“ forma t ” **,** variables)

The print format is as follows **:**

%d	output in a digital form
%f	output in a floating form
%c	output a single character
%s	output a string of characters

Example 3.1

printf(" **%d, %s, %f** " **,** 100 **,** 'good' **,** 3 **.** 14) **;**

Output:

100 **,** good **,** 3 **.** 140000

Explanation:

"%d **,** %s **,** %f" specifies the output format(digital **,** string **,** float) **.**

Array & Matrix

An array is a special variable containing multiple values **,** makes up a one-dimension array **.**

A matrix is a two-dimension array **.**

(1) Create an Array.

Example 3.2

arr = [1 2 3]

Output:

```
Arr =
     1 2 3
```

(2) Create a Matrix.

Example 3.3

mat = [1 2 3 **;** 4 5 6 **;** 7 8 9]

Output:

```
mat =
     1   2   3
     4   5   6
     7   8   9
```

Collection Operations

Given A and B are two arrays **:**

Collection	**Return**
union(A **,** B)	the union of two arrays
intersect(A **,** B)	the intersect of two arrays
setdiff(A **,** B)	elements that exist in A but not in B
setdiff(A **,** B)	elements that exist in B but not in A

Example 3.4

A = [18 24 15 16 10 13 19 25 36]

B = [13 21 18 19 15 17 26 36 28]

u = **union(A, B)**

i = **intersect(A, B)**

s1 = **setdiff(A, B)**

s2 = **setdiff(B, A)**

Output:

A =

18 24 15 16 10 13 19 25 36

B =

13 21 18 19 15 17 26 36 28

u =

10 13 15 16 17 18 19 21 24 25 26 28 36

i =

13 15 18 19 36

s1 =

10 16 24 25

s2 =

17 21 26 28

Explanation:

union(A , B) returns the union of two arrays

intersect(A , B) returns the intersect of two arrays

setdiff(A , B) returns elements that exist in A but not in B

setdiff(A , B) returns elements that exist in B but not in A

If…end Statement

```
if condition……end
```

“if…end statement” executes the codes only if a specified condition is true **,** does not execute any codes if the condition is false **.**

Example 3.5

```
num = 100 ;
if num < 200
disp('The num is less than 200 ' ) ;
end
printf('The value of num is : %d ' , num) ;
```

Output:

The num is less than 200

The value of num is **:** 100

Explanation:

“ **if num < 200** ” returns true **,** so the disp(…) has been executed **.**

If…else…end Statement

```
if condition
……
else
……
end
```

"if **...** else **…** end statement" runs some code if a condition is true and runs another code if the condition is false

Example 3.6

```
num = 300 ;
if num < 200
disp('The num is less than 200 ' ) ;
else
printf('The value of num is : %d ' , num) ;
end
```

Output:

The value of num is **:** 300

Explanation:

" **if num < 200** " returns false **,** so the "else" statement has been executed **.**

if … elseif … else … end Statement

```
if condition
……
elseif condition
……
else
……
end
```

"if **…** elseif **…** else **…** end statement" runs some code if a condition is true and runs another code if the condition is false

Example 3.7

```
num = 300 ;
if num < 200
disp('The num is less than 200') ;
elseif num > 200
disp('The num is greater than 200') ;
else
disp('The num is not a number') ;
end
```

Output: The num is greater than 200

Explanation:

"else **if num > 200** " returns true **,** so the disp(…) has been executed **.**

Nested If…end Statement

There two conditions in the nested “if…end” statement **.**

Example 3.8

```
num1 = 10 ; num2 = 20 ;
if ( num1 == 10 )
if ( num2 == 20 )
disp( 'The value of num1 is 10 and num2 is 20' ) ;
end
end
```

Output:

The value of num1 is 10 and num2 is 20

Explanation:

When two “if conditions” return true **,** the “disp()” is executed **.**

Switch Statement

```
switch (variable )
  case 1   if equals this case , do this ;
  case 2   if equals this case , do this ;
  case 3   if equals this case , do this ;
  otherwise  if not equals any case , run this ;
end
```

The value of the variable will compare each case first **,** if equals one of the "case" value **;** it will execute that "case" code **.**

Example 3.9

```
fruit = 'coconut' ;
   switch(fruit)
   case 'apple'
      disp(' My favorite fruit is apple ' ) ;
   case 'banana'
       disp(' My favorite fruit is banana ' ) ;
   case 'coconut'
      disp(' My favorite fruit is coconut  ' ) ;
   case 'durian'
      disp(' My favorite fruit is durian ' ) ;
   otherwise
     disp(' No favorite fruit ' ) ;
  end
```

Output:

My favorite fruit is coconut

Explanation:

Because the value of the fruit is coconut **,** which matches the case coconut **,** so the output shows “My favorite fruit is coconut” **.**

If the condition does not match any case **,** the “otherwise” statement will be executed **.**

Nested Switch Statement

A Switch statement can be embedded in another switch statement **.** The syntax of the nested switch statement is **:**

```
switch(variable1)
case value1
……
    switch(variable2)
    case value2
    ……
    end
end
```

Example 3.10

```
x = 10 ; y = 20 ;
switch(x)
case 10
printf('\n The value of x is %d ' , x ) ;
    switch(y)
    case 20
    printf('\n The value of y is %d ' , y ) ;
    end
end
```

Output:

The value of x is 10

The value of y is 20

Explanation:

In the above example **,** we can see a switch statement is embedded inside another switch statement **.**

“\n” symbol is used to output the contents into the next line **.**

Hour 4

While Loop

```
while ( condition ) ……end
```

“while loop” loops through a block of code if the specified condition is true **.**

Example 4.1

```
num = 1 ;
while( num <= 8 )
  printf('%d' , num) ;
  num = num + 1 ;
end
```

Output:

1 2 3 4 5 6 7 8

Explanation:

“ **while(num <= 8)** ” checks if the num is less than or equal to 8 **,** when it returns true **,** the code in the while block is executed **.**

For Loop (1)

```
for variable = start_value : end_value
```

“for loop” runs a block of code repeatedly by the specified number of times .

“start_value” specifies a start value of the loop

“end_value” specifies an end value of the loop

Example 4.2

```
for num = 1:8
  printf('%d' , num) ;
end
```

Output:

1 2 3 4 5 6 7 8

Explanation:

“ **for num = 1:8** ” specifies the range of the loop from 1 to 8 .

For Loop (2)

```
for variable = start_value : step : end_value
```

“for loop” runs a block of code repeatedly by the specified number of times and step **.**

“start_value” specifies a start value of the loop

“end_value” specifies an end value of the loop

“step” specifies an interval of each loop **.**

Example 4.3

```
for num = 1:2:8
  printf('%d' , num) ;
end
```

Output:

1 3 5 7

Explanation:

“ **for num = 1:2:8** ” specifies the range of the loop from 1 to 8 **,** and step size of 2 **.**

For Loop (3)

```
for variable = [ array ]
```

“for loop” can access each element of an array

Example 4.4

```
for num = [ 0 1 3 6 8 9 ]
  printf('%d' , num) ;
end
```

Output:

0 1 3 6 8 9

Explanation:

“ **for num = [0 1 3 6 8 9]** ” iterates through each element of the array **.**

Break Statement

“break” command works inside the loop **.**

```
break ;
```

“break” keyword is used to stop the running of a loop according to the condition

.

Example 4.5

```
num = 1 ;
while ( num < 10 )
      printf( '%d' , num ) ;
      num = num+1 ;
      if( num > 5)
         break;
      end
end
```

Output:

1 2 3 4 5

Explanation:

When num>5 **,** the while loop stops **.**

Continue Statement

“continue” command works inside the loop **.**

```
continue ;
```

“continue” keyword is used to stop the current iteration **,** ignoring the following code **,** and then continue the next loop **.**

Example 4.6

```
num = 0 ;
while num < 8
num = num + 1 ;
if num == 5  continue;    % go the next while loop
end
printf('%d' , num ) ;
end
```

Output: 1 2 3 4 6 7 8

Explanation:

Note that the output has no **5.**

“if num==5 continue **;** ” means **:** When the num is 5 **,** the program will run “continue” command **,** skipping the next command “printf(‘%d’ **,** num)” **,** and then continue the next while loop **.**

A Matrix With Zero Elements

We can create a matrix whose all elements are zero **.**

zeros(row **,** column)

Example 4.7

myMatrix = **zeros(6, 8)**

Output:

myMatrix =

```
    0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0
```

Explanation:

“myMatrix = zeros(6 **,** 8)” creates a matrix with six rows of “zero” and eight columns of “zero” **.**

A Matrix With One Elements

We can create a matrix whose all elements are one **.**

ones(row **,** column)

Example 4.8

myMatrix = **ones(4, 5)**

Output:

myMatrix =

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

Explanation:

“ones(4 **,** 5)” creates a matrix with four rows of “one” and five columns of “one” **.**

Diagonal Matrix

The syntax to create a diagonal matrix is **:**

```
eye( parameter )
```

Example 4.9

eye(3)

eye(4)

Output:

```
ans =
Diagonal Matrix
   1   0   0
   0   1   0
   0   0   1

ans =
Diagonal Matrix
   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
```

Explanation:

“eye(3)” and “eye(4)” creates two diagonal matrixes **.**

An Array With Random Elements

We can create an array with random elements .

```
rand(rows , columns)
```

Example 4.10

rand(3, 4)

Output:

ans =

```
  0.116194   0.016404   0.659537   0.774605
  0.197671   0.293512   0.061329   0.138692
  0.466637   0.158825   0.069245   0.613195
```

Explanation:

“rand(3 , 4)” creates an array with random elements , three rows and four columns .

Magic Array

Magic Array means that the number of rows is equal to the number of columns .

The sum of the rows is equal to the sum of the columns .

The sum of two diagonal elements is the same .

Example 4.11

magic(3)

Output:

```
ans =
   8   1   6
   3   5   7
   4   9   2
```

Explanation:

The sum of each row is 15 .

The sum of each column is 15 .

The sum of each diagonal element is 15 .

Two-Dimensional Array

Two-dimensional array is a matrix **.**

It is easy to create a two-dimensional array **.**

```
array = [ a b c ;  d e f ;  g h i ]
```

Example 4.12

myArray = **[2 8 6; 7 1 4; 3 5 9]**

Output:

myArray =

2 8 6
7 1 4
3 5 9

Explanation:

“myArray = [2 8 6 **;** 7 1 4 **;** 3 5 9]” creates a two-dimensional array with nine elements **.**

Matrix is a two-dimensional array **.**

Hour 5

Array Functions

length(array) gets the length of an array numel(array) gets the number of the array elements ndims(array) gets the number of the array dimension

Example 5.1

```
arr = [ 1.6, 2.0, 3.28, 4.9, 5.72 ];
length (arr)
numel (arr)
ndims (arr)
```

Output:

```
ans =  5

ans =  5

ans =  2
```

Explanation:

length(arr) gets the length of an array .

numel(arr) gets the number of the array elements .

ndims(arr) gets the number of the array dimension .

Elemental Shift

The syntax to shift the array elements is as follows **:**

```
circshift ( array , [ up/down , left/right ] )
```

About the parameters [up/down , left/right] , for example **:**

[1 , 0] shifts elements down by 1 **.**

[-1 , 0] shifts elements up by 1 **.**

[0 , 1] shifts elements right by 1 **.**

[0 , -1] shifts elements left by 1 **.**

[1 , 1] shifts elements down by 1 and right by 1 **.**

[-1 , -1] shifts elements up by 1 and left by 1 **.**

[n , n] shifts elements down by n and right by n **.**

[-n , -n] shifts elements up by n and left by n **.**

Example 5.2

```
myArray = [ 1 2 3 ; 4 5 6 ; 7 8 9 ]   % the original array
X = circshift (myArray , [1 , 0])    % down shift by 1
Y = circshift (myArray , [0 , 1])    % right shift by 1
Z = circshift (myArray , [1 , 1])    % down & right shift by 1
```

Output:

myArray =

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

X =

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

Y =

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

Z =

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

Explanation:

circshift (myArray **,** [1 **,** 0]) shifts elements down by 1

circshift (myArray **,** [0 **,** 1]) shifts elements right by 1

circshift (myArray **,** [1 **,** 1]) shifts elements down & right by 1

Sort an Array (1)

The syntax to sort an array is **:**

```
sort(array)
```

Example 5.3

arr = [23 56 38 88 51 12 67 45]

sort(arr)

Output:

arr =

  23 56 38 88 51 12 67 45

ans =

  12 23 38 45 51 56 67 88

Explanation:

“ **sort(arr)** ” sorts the array “arr” **.**

Sort an Array (2)

For a two-dimensional array **,** we can sort its elements along the row or along the column respectively **.**

(1) Sort the elements along the rows

```
sort(array , 1)
```

Sort the elements along the rows by setting parameter “1” **.**

(2) Sort the elements along the columns

```
sort(array , 2)
```

Sort the elements along the columns by setting parameter “2” **.**

Example 5.4

```
myArray = [ 3 7 5 ; 6 2 9 ; 1 0 8 ]
r = sort (myArray , 1 )   % sort elements along the rows
c = sort (myArray , 2 )   % sort elements along the columns
```

Output:

myArray =

```
   3   7   5
   6   2   9
   1   0   8
```

r =

```
   1   0   5
   3   2   8
   6   7   9
```

c =

```
   3   5   7
   2   6   9
   0   1   8
```

Explanation:

sort (myArray **, 1**) sorts elements along the rows **.**

sort (myArray **, 2**) sorts elements along the columns **.**

Cell Array

Cell arrays can store different dimensions and different types of Elements **.** The syntax to create a cell array is as follows **:**

```
cell(rows , columns)
```

Example 5.5

```
c = cell(2, 4);     % create a cell array with 2 rows and 4 columns
c = {'A' , 'B' , 'C' , 'D' ; '1' , '2' , '3' , '4' , }     % sets cell array elements
```

Output:

```
c =
{
  [ 1 ,  1] = A
  [ 2 ,  1] = 1
  [ 1 ,  2] = B
  [ 2 ,  2] = 2
  [ 1 ,  3] = C
  [ 2 ,  3] = 3
  [ 1 ,  4] = D
  [ 2 ,  4] = 4
}
```

Explanation:

“ **cell(2, 4)** ” creates a cell array with 2 rows and 4 columns

Colon Symbol

MATLAB can use the colon symbol " **:** " to create a vector **,** index array and an iteration **.**

" **:** " is used to access specified rows or columns **.**

For example **:**

arr(1 **:** 3 **,** 2 **:** 5) accesses the elements from row1 to row3 **,** and column2 to column5 **.**

arr(2 **,:**) accesses the elements in row2 and all columns **.**

arr(**:,** 2 **:** 3) accesses the elements all rows and from column2 to column3 **.**

Example 5.6

myArray = [0 1 2 3 **;** 3 4 5 6 **;** 6 7 8 9]

myArray **(:,3)**

myArray **(1:2,:)**

myArray **(2:3,3:4)**

Output:

```
myArray =
   0   1   2   3
   3   4   5   6
   6   7   8   9

ans =
   2
   5
   8

ans =
   0   1   2   3
   3   4   5   6

ans =
   5   6
   8   9
```

Explanation:

(**:,** 3) accesses the elements in all rows and column3 **.**

(1 **:** 2 **,:**) accesses the elements from row1 to row2 and all columns **.**

(2 **:** 3 **,** 3 **:** 4) accesses the elements from row2 to row3 and column3 to column4

.

Access Cell Array

The syntax to access a cell array is **:**

```
c(startRow : endRow , startCol : endCol)
```

“startRow **:** endRow” means from startRow to endRow

“startCol **:** endCol” means from startCol to endCol **.**

Finally a new cell array will be created **.**

For example **:**

c(2 **:** 3 **,** 3 **:** 6) accesses the elements from row2 to row3 **,** and from column3 to column6 **.** Finally a new cell array will be created **.**

Example 5.7

```
c = cell(2 , 4) ;
c = {'A' , 'B' , 'C' , 'D' ; '1' , '2' , '3' , '4' , }
result = c(1:2, 2:3)
```

Output:

```
c =
{
  [1 , 1] = A
  [2 , 1] = 1
  [1 , 2] = B
  [2 , 2] = 2
  [1 , 3] = C
  [2 , 3] = 3
  [1 , 4] = D
  [2 , 4] = 4
}
result =
{
  [1 , 1] = B
  [2 , 1] = 2
  [1 , 2] = C
  [2 , 2] = 3
}
```

Explanation:

" **c(1:2, 2:3)** " accesses the elements from row1 to row2 **,** and column2 to column3 **.** In the end **,** a new cell array has been created **.**

Step Length

We can specify the step length in a range **.**

startValue **: step :** endValue

Example 5.8

0 **:2:** 8

30 **: -5 :** 10

Output:

```
ans =
   0   2   4   6   8

ans =
   30   25   20   15   10
```

Explanation:

“0 **:2:** 8” accesses the elements from 2 to 8 **,** step length is 2 **.**

“30 **: -5 :** 10” accesses the elements from 30 to 10 **,** and the step length is -5 **.**

Hour 6

Lower & Upper

lower() upper()

lower() can convert a string to lowercase **.**

upper() can convert a string to uppercase **.**

Example 6.1

```
str1="Vue in 8 Hours"
str2="Lua in 8 Hours"
lower(str1)
upper(str2)
```

Output:

ans = vue in 8 hours

ans = LUA IN 8 HOURS

Explanation:

lower(str1) converts str1 to lowercase **.**

upper(str2) converts str2 to uppercase **.**

strfind & strrep

```
strfind(string , ‘substring’)
strrep(string , ‘oldstring’ , ‘newstring’)
```

strfind() looks for a substring , and returns an index .

strrep() replaces the oldstring with newstring .

Example 6.2

```
str1="Vue in 8 Hours"
str2="Lua in 8 Hours"
strfind(str1, 'Vue')
strrep(str2, 'Lua', 'Dart')
```

Output:

ans = 1

ans = Dart in 8 Hours

Explanation:

strfind(str1 , 'Vue') looks for ‘Vue’ in str1 . “ans=1” means index 1 .

strrep(str2 , 'Lua' , 'Dart') replaces ‘Lua’ with ‘Dart’ in str2 .

strcat & strjoin

```
strcat(string1 , string2)
strjoin(array , delimiter)
```

strcat() connects two strings **.**

strjoin() connects all elements of an array with a delimiter **.**

Example 6.3

```
s1 = 'Shell Scripting' ;
s2 = ' in 8 Hours' ;
s3 = strcat(s1,s2)

arr = {'A' , 'B' , 'C' , 'D' , 'E'} ;
s4 = strjoin(arr, '-')
```

Output:

s3 = Shell Scripting in 8 Hours

s4 = A-B-C-D-E

Explanation:

“ **strcat(s1,s2)** ” connects s1 with s2 **.**

“ **strjoin(arr, '-')** ” connects all elements in “arr” with a delimiter “-” **.**

strcmp & strtok

```
strcomp(string1 , string2)
strtok(string)
```

“strcomp(string1 , string2)” compares two strings . If they are equal , returns 1 . If they are not equal , returns 0 .

“strtok(string)” returns the first substring up to the first whitespace in the string .

Example 6.4

```
str1 = 'Rust in 8 Hours'
str2 = 'Dart in 8 Hours'
comp = strcmp(str1, str2)
s1 = strtok(str1)
s2 = strtok(str2)
```

Output:

comp = 0

s1 = Rust

s2 = Dart

Explanation:

“ **strcmp(str1, str2)** ” compares str1 with str2 .

“ **strtok(str)** ” returns the first substring in the str .

String Functions

blanks	create a string of blank characters
cellstr	create string cell arrays
char	converted to character arrays
iscellstr	check if it is a string cell array
ischar	check if it is a char cell array
sprintf	format the data into a string
isletter	check if it is an element in alphabetical order
isspace	check if it is a space character
isstrprop	check if it is is a specificed-type string
sscanf	read formatted data from a string
strsplit	split strings at the specified delimiter
symvar	check symbolic variables in expressions
regexp	matching regular expressions(case sensitive)
regexpi	matching regular expressions(case insensitive)
regexprep	replacing strings with regular expressions
strcmp	compare strings (case sensitive)
strcmpi	compare strings (case insensitive)
strtrim	delete leading & trailing spaces from string
strjust	alignment character array

Anonymous Function

A function is a reusable block of code **.**

The syntax to declare an anonymous function is as follows **:**

```
handle = @(arg) expression
```

The @ operator is used to create the function handle **.**

(arg) is used to accept the come-in parameter

Example 6.5

```
a = 10 ; b = 20 ;
handle = @(x) a*x + b/x ;      % anonymous function
x = 2 ;
y = handle(x)
```

Output:

y = 30

Explanation:

“handle = @(x)” creates a function handle **.**

(x) accepts the come-in parameter “2” **.**

Define a Function

The syntax to define a function is as follows **:**

```
function out = functionName(in)
```

“out” is an output parameter **.** “in” is an input parameter **.**

“out” passes the result to output **.**

Example 6.6

```
function out = myFun(in)    % define a function myFun
   out = (in * 2) ;
end
myFun(10)      % call myFun
```

Output:

ans = 20

Explanation:

“myFun(10)” calls the function **.**

“function out = myFun(in)” define a function “myFun” **.**

Note:

Some online editors may do not support this kind function **.**

Nested Function

A function can be embedded inside another function **.**

```
function out = fun1(in)      % outer function
    function fun2       % inner function
    ……
    end
end
```

Example 6.7

```
function [out] = fun1 (a , b , c)
     function fun2
     d = 10 ;
     end
fun2 ;       % call fun2
out = ( a + b + c ) * d ;
end
fun1(1 , 2 , 3)          % call fun1
```

Output: ans = 60

Explanation:

“fun1()” is an outer function **,** “fun2()” is an inner function **.**

Some online editors may do not support this kind function **.**

Vector

A vector is a matrix with either one row or one column **.**

The syntax to define a row vector and column vector is **:**

```
[ v1  v2  v3  v4  v5 ]    % define a row vector with spaces
[ v1 , v2 , v3 , v4 , v5 ]    % define a row vector with ‘ , ’
[ v1 ; v2 ; v3 ; v4 ; v5 ]    % define a column vector with ‘ ; ’
```

Example 6.8

```
a = [1  2  3  4  5]      % define a row vector
b = [1, 2, 3, 4, 5]      % define a row vector
c = [1; 2; 3; 4; 5]      % define a column vector
```

Output:

```
a =
   1   2   3   4   5
b =
   1   2   3   4   5
c =
   1
   2
   3
   4
   5
```

Reference Vector Elements

Three syntaxes to reference vector elements are as follows **:**

```
vectorName(index)       % reference the element in the index
vectorName(index1 : index2)    % from index1 to index2
vectorName( : )       % reference all elements
```

Example 6.9

```
v = [10 , 20 , 30 , 40 , 50]
v(2)       % reference the element in the second index
v(2:4)     % reference the elements from index2 to index4
v(:)     % reference all elements
```

Output:

```
ans =  20
ans =  20   30   40
ans =
   10
   20
   30
   40
   50
```

Vector Add & Subtract

When adding or subtracting two vectors **,** the elements of both vectors must have the same type and length **.**

Example 6.10

```
A = [26 , 12 , 47 , 33 , 50] ;
B = [16 , 35 , 29 , 59 , 35] ;
num1 = A + B;
num2 = A - B;
disp(num1) ;
disp(num2) ;
```

Output:

42 47 76 92 85

10 -23 18 -26 15

Explanation:

“A + B” means the elements in A plus the elements in B **.**

“A - B” means the elements in A subtracts the elements in B **.**

Vector Multiplication

The way of the vector multiplication is that one number multiplies each element of the vector respectively **.**

Example 6.11

```
v = [ 13 36 45 28 ] ;
mum = 2 * v
```

Output:

```
 26   72   90   56
```

Explanation:

“2 * v” means that 2 multiplies each element respectively **.**

Hour 7

Vector Transpose

Vector transposing can convert a row vector to a column vector **,** or convert a column vector to a row vector **.**

The single quote (') is a transposing operator **.**

```
rowVector = colVector '     %
colVector is converted to rowVector
colVector = rowVector '     %
rowVector is converted to colVector
```

Example 7.1

```
r = [ 1 2 3 4 ] ;
colVector = r';       % converted to a column vector
disp(colVector) ;
c = [1 ; 2 ; 3 ; 4] ;
rowVector = c';       % converted to a row vector
disp(rowVector) ;
```

Output:

```
1
2
3
4
1   2   3   4
```

Row Vector Append

We can create a new vector or new matrix by appending a row vector to another row vector **.**

```
newVector = [ rowVector1 , rowVector2 ]
newMatrix = [ rowVector1 ; rowVctor2 ]
```

Example 7.2

```
v1 = [ 0 , 1 , 2 , 3 , 4 ] ;
v2 = [ 5 , 6 , 7 , 8 , 9 ] ;
newVector = [ v1, v2 ]
newMatrix = [ v1; v2 ]
```

Output:

newVector =

0 1 2 3 4 5 6 7 8 9

newMatrix =

0 1 2 3 4
5 6 7 8 9

Explanation:

[v1 , v2] create a new vector by using (,)

[v1 ; v2] create a new matrix by using(;)

Column Vector Append

We can create a new vector or new matrix by appending a column vector to another column vector **.**

```
newVector = [ colVector1 ; colVector2 ]
newMatrix = [ colVector1 , colVctor2 ]
```

Note:

If append a row vector **,** using (**,**) to create a new vector **,** using (**;**) to create a new matrix **.**

If append a column vector **,** using (**;**) to create a new vector **,** using (**,**) to create a new matrix **.**

Example 7.3

```
v1 = [0 ; 1 ; 2 ; 3 ; 4 ] ;
v2 = [5 ; 6 ; 7 ; 8 ; 9 ] ;
newVector = [v1;v2]
newMatrix = [v1,v2]
```

Output:

```
newVector =
     0
     1
     2
     3
     4
     5
     6
     7
     8
     9

newMatrix =
     0     5
     1     6
     2     7
     3     8
     4     9
```

Explanation:

[v1 **;** v2] create a new vector by using (**;**)

[v1 **,** v2] create a new matrix by using (**,**)

Vector Dot Product

In mathematics , the dot product is also called the scalar product , which is an algebraic operation in two equal-length sequence that returns a single number . (About detail vector knowledge , please reference other related mathematical books .)

The syntax to get the dot product of two vectors is :

```
dot(v1 , v2) ;
```

Example 7.4

```
v1 = [1 , 3 , -5] ;
v2 = [4 , -2 , -1] ;
dp = dot(v1, v2);
disp('The dot product is : ') ;
disp(dp) ;
```

Output:

The dot product is :

3

Explanation:

" **dot(v1, v2)** " gets the dot product of two vectors .

Vector Magnitude

The size of the vector , namely the length of the vector is called the vector magnitude . (About detail vector knowledge , please reference other related mathematical books .)

The syntax to get the vector magnitude is :

```
sqrt(sum(v . * v))
```

Example 7.5

```
v = [1 : 2 : 10] ;
disp('The magnitude is : ') ;
disp( sqrt(sum(v.* v)) ) ;
```

Output:

The magnitude is :

12 . 845

Explanation:

" **sqrt(sum(v.* v))** " gets the magnitude the vector .

Linspace Vector

Linspace vector means the vector with evenly space elements .

MATLAB allows us to create the linspace vector with evenly spaced elements .

The syntax to create linspace is :

```
linspace(firsVal , lastVal , totalElements)
```

“firstVal” means the first value . “lastVal” means the last value .

“totalElement” means that there’re total number elements in the new linspace vector .

Example 7.6

```
linspaceVector = linspace(1,2,5)
```

Output:

```
linspaceVector =
   1 . 0000   1 . 2500   1 . 5000   1 . 7500   2 . 0000
```

Explanation:

“linspace(1 , 2 , 5)” creates a linspace vector , its first element is 1 . 0000 , the last element is 2 . 0000 , there are total 5 elements in the linspace vector .

Matrix

In mathematics , a Matrix is a set of complex or real numbers arranged in a rectangular array .

(About detail Matrix knowledge , please reference other related mathematical books .)

There are the following rules for creating matrices in MATLAB :

1 .

Matrix elements must be within [ ] .

2 .

Matrix elements are separated by spaces or commas (,) .

3 .

Matrix's rows are separated by semicolons .

4 .

Matrix's elements can be a number , variable , expressions or functions .

5 .

The dimensions of the matrix need not be defined in advance .

Create a Matrix

The syntax to create a matrix is as follows **:**

```
myMatrix = [v11  v12  v13 ; v21  v22  v23 ; v31
v32  v33]
```

Matrix elements are separated by spaces

Matrix's rows are separated by semicolons **.**

The following example shows how to create a 5x4 matrix **.**

Example 7.7

```
myMatrix = [ 1 2 3 4; 2 3 4 5; 3 4 5 6; 4 5 6 7; 5 6 7
8 ]
```

Output:

```
myMatrix =
   1   2   3   4
   2   3   4   5
   3   4   5   6
   4   5   6   7
   5   6   7   8
```

Explanation:

The result is a matrix with 5 rows x 4 columns elements **.**

Matrix Add & Subtract

When two matrices are added or subtracted **,** the number of rows and columns in one matrix must equal the number of rows and columns in the other matrix **.**

Example 7.8

```
mat1 = [ 9 8 7 ; 6 5 4 ; 3 2 1 ] ;
mat2 = [ 1 2 3 ; 4 5 6 ; 7 8 9 ] ;
result1 = mat1 + mat2
result2 = mat1 - mat2
```

Output:

```
result1 =
   10   10   10
   10   10   10
   10   10   10
result2 =
    8    6    4
    2    0   -2
   -4   -6   -8
```

Explanation:

“mat1 + mat2” means mat1 pluses mat2 **.**

“mat1 - mat2” means mat1 subtracts mat2 **.**

Matrix Multiplication

When two matrices are multiplied , the number of rows in the first matrix should equal the number of columns in the second matrix .

To multiply matrix by matrix , we compute the dot product of rows and columns .

The dot product is the result of multiplying symmetric elements together and adding them together .

But in Matlab , we can use the syntax of matrix multiplication to get the result easily . The syntax of matrix multiplication is :

```
dotProduct = matrix1 * matrix2
```

Example 7.9

```
matrix1 = [ 1 2 3 ; 4 5 6 ]
matrix2 = [ 7 8 ; 9 10 ; 11 12 ]
dotProduct = matrix1 * matrix2
```

Output:

```
matrix1 =
   1   2   3
   4   5   6

matrix2 =
   7   8
   9  10
  11  12

dotProduct =
  58  64
 139 154
```

Explanation:

“ **matrix1 * matrix2** ” means matrix1 multiplies matrix2 **.**

The result should have the number of rows in the first matrix and the number of columns in the second matrix **.**

Hour 8

Matlab Division (/)

In Matlab division , the operands of the two matrices must have the same number of rows and columns .

The syntax of the Matlat right division is :

```
result = matrix1 / matrix2
```

Example 8.1

```
mat1 = [ 1 2 3 ; 4 5 6 ; 7 8 9 ] ;
mat2 = [ 9 8 7 ; 6 5 4 ; 3 2 1 ] ;
result = mat1 / mat2
```

Output:

```
result =
   0.83333  -0.33333  -1.50000
   1.33333  -0.33333  -2.00000
   1.83333  -0.33333  -2.50000
```

Explanation:

" **mat1 / mat2** " means that mat1 is right divided mat2 .

Matlab Division (\ )

In Matlab division , the operands of the two matrices must have the same number of rows and columns .

The syntax of the Matlat left division is :

```
result = matrix1 \ matrix2
```

Example 8.2

```
mat1 = [ 1 2 3 ; 4 5 6 ; 7 8 9 ] ;
mat2 = [ 9 8 7 ; 6 5 4 ; 3 2 1 ] ;
result = mat1 \ mat2
```

Output:

```
result =
  -5.83333  -5.33333  -4.83333
  -0.33333  -0.33333  -0.33333
   5.16667   4.66667   4.16667
```

Explanation:

" **mat1 \ mat2** " means that mat1 is left divided mat2 .

One Number & Matrix

The arithmetical calculation between one number and a matrix is very simple , the method is the one number respectively operates each element of the matrix .

For example :

num + [a b c ; d e f] means :

num+a , num+b , num+c ;

num+d , num+e , num+f ;

Example 8.3

```
mat = [ 20 21 22 ; 23 24 25 ; 26 27 28 ] ;
num = 2 ;
v1 = mat + num
v2 = mat - num
v3 = mat * num
v4 = mat / num
```

Output:

```
v1 =
   22   23   24
   25   26   27
   28   29   30

v2 =
   18   19   20
   21   22   23
   24   25   26

v3 =
   40   42   44
   46   48   50
   52   54   56

v4 =
   10.000   10.500   11.000
   11.500   12.000   12.500
   13.000   13.500   14.000
```

Explanation:

The above example shows the arithmetical calculation between one number and each matrix element **.** Very simple **.**

Matrix Transpose

Matrix transpose can convert matrix rows to matrix columns **,** or convert matrix columns to matrix rows **.**

The single quote (') is a transposing operator **.**

```
rows = columns '     % columns are converted to rows
columns = rows '     % rows are converted to columns
```

Example 8.4

```
original = [ 1 2 3 ;  4 5 6 ; 7 8 9]
transposed = original '
```

Output:

original =

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

transposed =

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

Explanation:

The single quote (') is a transposing operator **.**

Join Matrices

Join Matrices means that two matrices are connected together **,** and become a new matrix **.**

Join Matrices are divided into horizontal connections and vertical connections **.**

[**,**] joins matrices horizontally **.**

[**;**] join matrices vertically **.**

The syntax to join two matrices is as follows **:**

```
[ matrix1 , matrix2 ]      % join matrices horizontally
[ matrix1 ; matrix2 ]      % join matrices vertically
```

Example 8.5

```
mat1 = [ 11 12 13 ; 14 15 16 ; 17 18 19]
mat2 = [ 21 22 23 ; 24 25 26 ; 27 28 29]
rowJoin = [ mat1, mat2 ]
columnJoin = [ mat1; mat2 ]
```

Output:

```
mat1 =
   11   12   13
   14   15   16
   17   18   19
mat2 =
   21   22   23
   24   25   26
   27   28   29
rowJoin =
   11   12   13   21   22   23
   14   15   16   24   25   26
   17   18   19   27   28   29
columnJoin =
   11   12   13
   14   15   16
   17   18   19
   21   22   23
   24   25   26
   27   28   29
```

Explanation:

[mat1 **,** mat2] joins matrices horizontally **.**

[mat1 **;** mat2] joins matrices vertically **.**

Matrix Determinant

The determinant of a matrix is a special number that can be computed from a square matrix **.**

The syntax to compute the matrix determinant is **:**

```
det(matrix)
```

Example 8.6

```
mat = [ 1 2 3 ; 4 5 6 ; 7 8 9 ]
det(mat)
```

Output:

mat =

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

ans = 0

Explanation:

“ **det(mat)** ” computes the determinant of the matrix **.**

Inverse Matrix

The inverse matrix M in Matlab is written as M^{-1}.

But in Matlab **,** the syntax to get an inverse matrix is **:**

```
inv(matrx)
```

Example 8.7

```
mat = [ 4 7 ; 2 6 ]
inv(mat)
```

Output:

```
mat =
   4   7
   2   6

ans =
   0.60000  -0.70000
  -0.20000   0.40000
```

Explanation:

“inv(mat)” returns an inverse matrix “mat” **.**

Linear Graph

Three steps to draw a graph in Matlab :

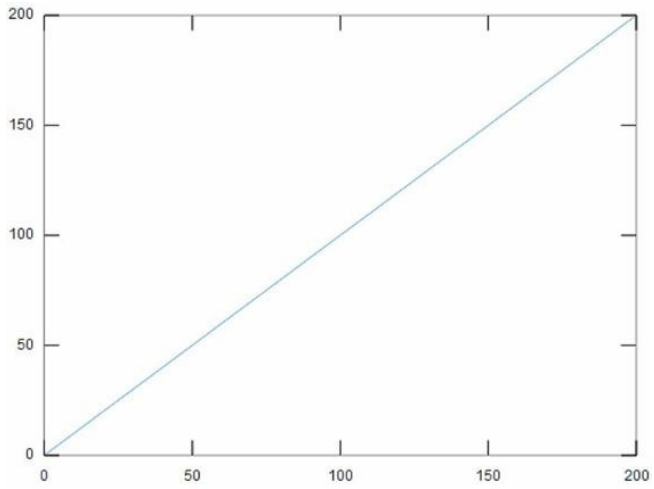
1 . Define the range of the X coordinates
2 . Define a function y = f(x) , e . g . linear function , sin function
3 . Call the function plot(x , y)

In the following example , we will draw a linear graph .

Example 8.8

```
x = [ 0 : 10 : 200 ] ;      % define the range of x
y = x ;                     % define a linear function
plot(x , y)                 % call plot(x , y) to draw the graph
```

Output:



Arc Graph

In the following example , we will draw an arc graph .

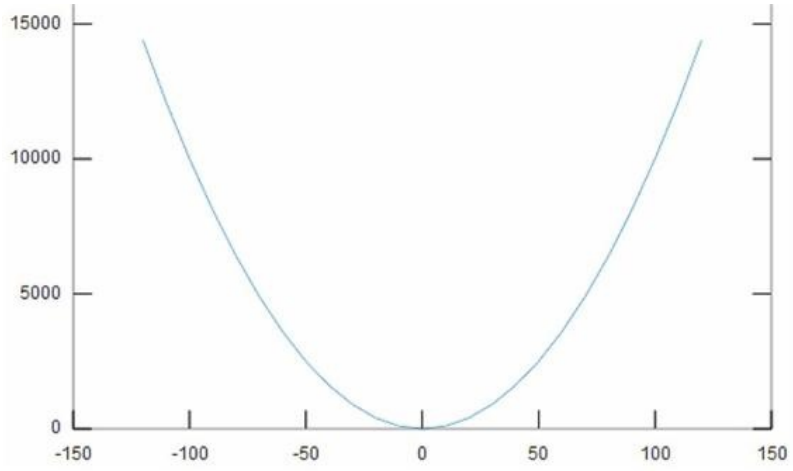
The arc function is $y = x^2$;

We can use “y = x . ^2” to represent the “$y = x^2$ ” in Matlab .

Example 8.9

```
x = [ -120 : 10 : 120 ] ;      % define the range of x
y = x . ^2 ;                   % define an arc function
plot(x , y)                    % call plot(x , y) to draw the graph
```

Output:



Sine Graph

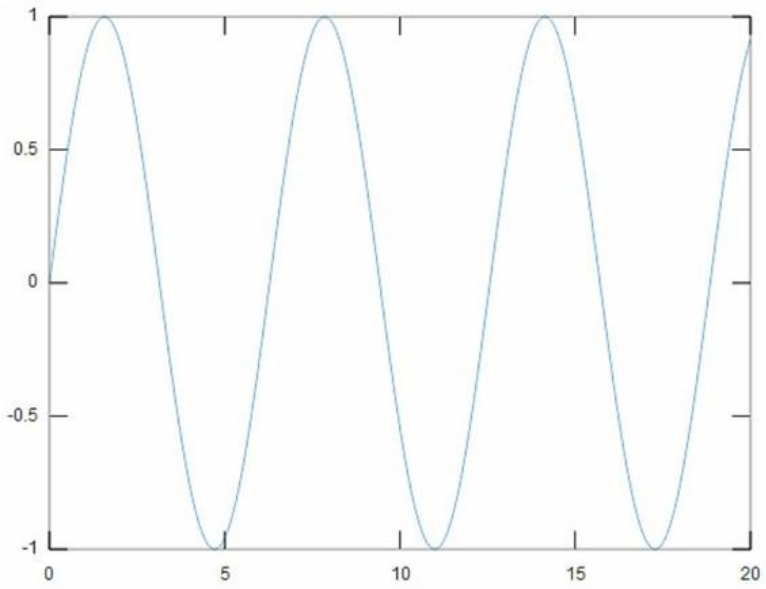
In the following example , we will draw a sine graph .

The sine function is y = sine(x) ;

Example 8.10

```
x = [0 : 0 . 02 : 20] ;      % define the range of x
y = sin(x) ;                 % define a sine function
plot(x , y)      % call plot(x , y) to draw the graph
```

Output:



Matlab

Q & A

Questions

Please fill in the correct answer **:**

01 **.**

About the Special Variables **:**

ans	results of operations **,** it means “answer”
fill in	power
eps	relative error of floating point number
i **,** j	imaginary unit **,** i2 = j2 = -1

A **.** **

B **.** squ

C **.** *=

D **.** .^

02 **.**

What is the output based on the following code?

result = dec2bin(str2num('2'))

A **.** 2

B **.** 0

C **.** 10

D . 1

03 .

What is the output based on the following code?

A = [32 24 75 56 18 35]

B = [20 91 18 67 26 80]

i = intersect(A , B)

A . 20

B . 18

C . 56

D . 91

04 .

num = 1 ;

while(num <= 8)

  printf('% **fill in** ' , num) ; % format output

  num = num + 1 ;

end

A . a

B . b

C . c

D . d

05 .

length(array) gets the length of an array

numel(array) gets the number of the array elements

fill in (array) gets the number of the array dimension

A . numdim

B . arraydim

C . ndims

D . arrayd

06 .

arr = {'A' , 'B' , 'C' , 'D' , 'E'} ;

s = **fill in** **(arr, '-')** % connect all elements in arr

A . strjoin

B . arrjoin

C . strconcat

D . arrconcat

07 .

Vector transposing can convert a row vector to a column vector , or convert a column vector to a row vector .

The symbol (**fill in**) is a transposing operator **.**

A **.** *

B **.** @

C **.** #

D **.** '

08 **.**

mat1 = [1 2 3 **;** 4 5 6 **;** 7 8 9] **;**

mat2 = [9 8 7 **;** 6 5 4 **;** 3 2 1] **;**

result = **mat1 fill in mat2** % right divide

A **.** /

B **.** \

C **.** %

D **.** div

09 **.**

If a statement too long **,** we can use " **fill in** " symbol to turn the statement to next line **.**

A **.** >

B **.** <

C **. ...**

D **. ;**

10 .

What is the output based on the following code :

a = 100 ; b = 3 ;

idivide(a , b)

A . 33 . 33333

B . 33

C . 1

D . 0

11 .

What is the output based on the following code?

A = [18 24 15 16 30 73 69 25 36]

B = [73 30 24 69 15 25 88 36 18]

s = setdiff(B , A)

A . 18

B . 69

C . 88

D . 30

12 .

“myMatrix = **fill in** (6 , 8)” creates a matrix with six rows of “zero” and eight columns of “zero” .

A . zeros

B . zero

C . nils

D . nil

13 .

fill in (myArray , [1 , 1]) shifts elements down & right by 1 .

A . shift

B . downright

C . matrixshift

D . circshift

14 .

“ **fill in** (string)” returns the first substring up to the first whitespace in the string .

A . strsub

B . strwhitespace

C . strtok

D . strfirst

15 .

v1 = [1 , 3 , -5] ;

v2 = [4 , -2 , -1] ;

dp = **fill in (v1, v2);** % get the dot product

disp('The dot product is : ') ;

disp(dp) ;

A . dotproduct

B . dotprod

C . dotpr

D . dot

16 .

mat1 = [11 12 13 ; 14 15 16 ; 17 18 19]

mat2 = [21 22 23 ; 24 25 26 ; 27 28 29]

columnJoin = **fill in** & join matrix vertically

A . [mat1 , mat2]

B . [mat1 ; mat2]

C . [mat1 . mat2]

D . [mat1 : mat2]

17 .

What is the output based on the following code?

format bank

num = 10 / 3

A . 3 . 3

B . 3 . 33

C . 3 . 333

D . 3 . 3333

18 .

What is the output based on the following code?

a = 3 ; b = 2 ;

num = a / b ;

v = fix(num)

A . 3

B . 2

C . 1

D . 0

19 .

What is the output based on the following code?

A = [18 28 15 16 21 13 19 21 36]

B = [13 21 18 19 15 17 26 36 28]

s = setdiff(A , B)

A . 13

B . 14

C . 15

D . 16

20 .

What is the output based on the following code?

```
for num = 1:2:8
  printf('%d' , num) ;
end
```

A . 1 4 7

B . 2 5 8

C . 3 6 9

D . 1 3 5 7

21 .

```
c = fill in (2, 4);      % create a cell array with 2 rows and 4 columns
c = {'A' , 'B' , 'C' , 'D' ; '1' , '2' , '3' , '4' , }
```

A . cell

B . array

C . vector

D . matrix

22 .

a = 10 ; b = 20 ;

handle = fill in (x) a*x + b/x ;

% create an anonymous function

x = 2 ;

y = handle(x)

A . #

B . @

C . $

D . &

23 .

Linspace vector means the vector with evenly space elements .

The syntax to create linspace is :

linspace(firsVal , lastVal , totalElements)

A . linearspaces

B . linspaces

C . linspace

D . space

24 .

Three steps to draw a graph in Matlab :

1 . Define the range of the X coordinates 2 . Define a function y = f(x) , e . g . linear function , sin function 3 . Call the function **fill in** (x , y)

A . draw

B . graph

C . picture

D . plot

Answers

01 . D	09 . C	17 . B
02 . C	10 . B	18 . C
03 . B	11 . C	19 . D
04 . D	12 . A	20 . D
05 . C	13 . D	21 . A
06 . A	14 . C	22 . B .
07 . D	15 . D	23 . C
08 . A	16 . B	24 . D

Source Code Download Link:

https://forms.aweber.com/form/98/2130886598.htm

Source Code Download

Recommended Books by Ray Yao's Team

Advanced C++ in 8 Hours
Advanced Java in 8 Hours
AngularJs in 8 Hours
Asp.net Programming
Awk in 8 Hours
BootStrap in 8 Hours
C# Examples & Interview
C# Programming
C++ Examples & Interview
C++ Programming
Dart in 8 Hours
Django in 8 Hours
Erlang in 8 Hours
Go in 8 Hours
Html Css Examples & Interview
Html Css Programming
Java Examples & Interview
Java Programming
JavaScript Examples & Interview
JavaScript Programming
JQuery Examples & Interview
JQuery Programming
Jsp Servlets Programming
Kotlin in 8 Hours
Linux Command Line
Linux Examples & Interview
Lua in 8 Hours
Matlab in 8 Hours
MySql in 8 Hours
Node.Js in 8 Hours
Numpy in 8 Hours
Perl in 8 Hours
Php Examples & Interview
Php MySql Programming
PowerShell in 8 Hours
Python Examples & Interview
Python Programming
R Programming
React.Js in 8 Hours
Ruby Programming
Rust in 8 Hours
Scala in 8 Hours
Shell Scripting in 8 Hours
Swift in 8 Hours
Tcl in 8 Hours
TypeScript in 8 Hours
Visual Basic Examples & Interview
Visual Basic Programming
Vue.Js in 8 Hours
Xml Json in 8 Hours

Source Code Download Link:

https://forms.aweber.com/form/98/2130886598.htm