

VECTOR OPERATIONS

Q1) Generate two vectors u and v by typing the code given below.

$$v1 = [1 \ 2 \ 3 \ 4 \ 10]$$
$$v2 = [-1 \ -2 \ -3 \ -4 \ -10]$$

Code:

```
%Q1
disp("Q1");
v1=[1 2 3 4 10];
v2=[-1 -2 -3 -4 -10];
disp(v1);
disp(v2);
```

Output:

Workspace		New to MATLAB? See resources for Getting Started.	
Name ^	Value		
v1	[1,2,3,4,10]	>> Experiment_1	
v2	[-1,-2,-3,-4,-10]	Q1	
		1 2 3 4 10	
		-1 -2 -3 -4 -10	
		fx >>	

Q2) Concatenate two vectors to form a new vector v3. Use $v3 = [u, v]$ or $v3 = [u \ v]$

Code:

```
%Q2
disp("Q2");
v3=[v1 v2];
disp(v3);
```

Output:

Workspace	
Name ^	Value
v1	[1,2,3,4,10]
v2	[-1,-2,-3,-4,-10]
v3	[1,2,3,4,10,-1,...

>> Experiment_1
Q2
1 2 3 4 10 -1 -2 -3 -4 -10
fx >>

Q3) Type help ones and help zeros. Create vectors v4 of 20 zeros and v5 of 20 ones using the command. Use the size or length command to identify the size of the created vector.

Code:

```
%Q3
disp("Q3");
%help ones;
%help zeros;
v4=zeros(1,20);
v5=ones(1,20);
%help size;
sz=size(v5);
len=length(v5);
fprintf("Size of v5 is %d\n",sz);
fprintf("Length of v5 is %d\n",len);
```

Output:

Workspace	
Name ^	Value
len	20
sz	[1,20]
v1	[1,2,3,4,10]
v2	[-1,-2,-3,-4,-10]
v3	[1,2,3,4,10,-1,...
v4	1x20 double
v5	1x20 double

>> Experiment_1
Q3
Size of v5 is 1
Size of v5 is 20
Length of v5 is 20
fx >>

Q4) Create a vector v6 of length 20 with every entry equal to 5.

Code:

```
%Q4
disp("Q4");
v6=5*ones(1,20);
disp(v6);
```

Output:

Workspace		>> Experiment_1																			
Name ^	Value	Q4																			
v6	1x20 double	Columns 1 through 19																			
		5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	
		Column 20																			
		5																			

Q5) Add two vectors v1 and v2 by using v1 + v2. Assign it to v7.

Code:

```
%Q5
disp("Q5");
v7=v1+v2;
disp(v7);
```

Output:

Workspace		>> Experiment_1				
Name ^	Value	Q5				
v1	[1,2,3,4,10]	0	0	0	0	0
v2	[-1,-2,-3,-4,-10]					
v7	[0,0,0,0,0]					

Q6) Add a constant to all entries of v7 vector. Try $v7 + 5$

Code:

```
%6
disp("Q6");
v7=v7+5;
disp(v7);
```

Output:

Workspace	
Name ^	Value
v1	[1,2,3,4,10]
v2	[-1,-2,-3,-4,-10]
v7	[5,5,5,5,5]

>> Experiment_1
Q6
5 5 5 5 5
fx >> |

Q7) Try 1:2:10. Observe the result. Use this to create vectors

$v8 = [1 \ 2 \ 3 \ \dots \ 10]$

$v9 = [25 \ 20 \ 15 \ \dots -20]$

Code:

```
%7
disp("Q7");
v8=1:2:10;
fprintf("v8 is:");
disp(v8);
```

Output:

Workspace	
Name ^	Value
v1	[1,2,3,4,10]
v2	[-1,-2,-3,-4,-10]
v8	[1,3,5,7,9]

>> Experiment_1
Q7
v8 is: 1 3 5 7 9
fx >> |

Q8) Pick the third value in vector v1. Use v1(3). Note that the numbering of entries is from 1 to length(v1). Try v1(6) and v1(0).

Code:

```
%8
disp("Q8");
fprintf("%d %d %d\n",v1(3),v1(5),v1(1));
v8=1:1:10;
v9=25:-5:-20;
disp(v8);
disp(v9);
```

Output:

Workspace		>> Experiment_1										
Name ^	Value	Q8										
v1	[1,2,3,4,10]	3	10	1								
v2	[-1,-2,-3,-4,-10]				1	2	3	4	5	6	7	8
v8	[1,2,3,4,5,6,7,...]											9
v9	[25,20,15,10,...]											10
					25	20	15	10	5	0	-5	-10
												-15
												-20

Q9) Obtain v10 from v1 by removing the third entry.

Code:

```
%9
disp("Q9");
v10=v1;|
v10(3)=[ ];
disp(v10);
```

Output:

Workspace		>> Experiment_1				
Name ^	Value	Q9				
v1	[1,2,3,4,10]					
v10	[1,2,4,10]	1	2	4	10	
v2	[-1,-2,-3,-4,-10]					

Q10) Obtain v11 from v1 by picking only the odd indexed entries.

Code:

```
%10
disp("Q10");
v11=v1(1:2:length(v1));
disp(v11);
```

Output:

Workspace		>> Experiment_1		
Name ^	Value	Q10		
v1	[1,2,3,4,10]	1	3	10
v11	[1,3,10]			
v2	[-1,-2,-3,-4,-10]			

fx >>

Q11) Obtain the transpose of v1 and save it as v1t .

Code:

```
%11
disp("Q11");
v1t=v1';
disp(v1t);
```

Output:

Workspace		>> Experiment_1		
Name ^	Value	Q11		
v1	[1,2,3,4,10]	1		
v1t	[1;2;3;4;10]	2		
v2	[-1,-2,-3,-4,-10]	3		
		4		
		10		

Q12) Try the operations $v1 * v1t$ and $v1t * v1$

Code:

```
%12
disp("Q12");
vtt1=v1t*v1;
v1tt=v1*v1t;
disp(vtt1);
disp(v1tt);
```

Output:

Workspace		>> Experiment_1				
Name ^	Value	Q12				
v1	[1,2,3,4,10]	1	2	3	4	10
v1t	[1;2;3;4;10]	2	4	6	8	20
v1tt	130	3	6	9	12	30
v2	[-1,-2,-3,-4,-10]	4	8	12	16	40
vtt1	5x5 double	10	20	30	40	100
		130				

Q13) Try the operation $v1. * v1$. Also try $v1.^v2$ and $v1./v2$.

Code:

```
%13
disp("Q13");
vq=v1.*v1;
disp(vq);
vmul=v1.^2;
disp(vmul);
vdiv=v1./v2;
disp(vdiv);
or r/
```

Output:

Workspace		>> Experiment_1				
Name ^	Value	Q13				
v1	[1,2,3,4,10]	1	4	9	16	100
v2	[-1,-2,-3,-4,-10]	1	4	9	16	100
vdiv	[-1,-1,-1,-1,-1]	-1	-1	-1	-1	-1
vmul	[1,4,9,16,100]					
vq	[1,4,9,16,100]					

Q14) Add all the entries of v1 and store it in s

Code:

```
%14
disp("Q14");
s=sum(v1);
disp(s);
```

Output:

Workspace		>> Experiment_1	
Name ^	Value	Q14	
s	20	20	
v1	[1,2,3,4,10]		
v2	[-1,-2,-3,-4,-10]		

fx >> |

Q15) Find the average of entries in v1

Code:

```
%15
disp("Q15");
avg=mean(v1);
disp(avg);
```

Output:

Workspace		>> Experiment_1	
Name ^	Value	Q15	
avg	4	4	
v1	[1,2,3,4,10]		
v2	[-1,-2,-3,-4,-10]		

fx >>

Q16) Try `v1 == 3`. Observe the result.

Code:

```
%16
disp("Q16");
k=v1==3;
disp(k);
```

Output:

Workspace			
Name ^	Value		
<input checked="" type="checkbox"/> k	1x5 logical		
<input type="checkbox"/> v1	[1,2,3,4,10]		
<input type="checkbox"/> v2	[-1,-2,-3,-4,-10]		

>> Experiment_1
Q16
0 0 1 0 0
fx >>

Q17) Find the number of entries in `v9` greater than or equal to 15. Do not use any loops.

Code:

```
%17
disp("Q17");
k=v1>=4;
disp(k);
```

Output:

Workspace			
Name ^	Value		
<input checked="" type="checkbox"/> k	1x5 logical		
<input type="checkbox"/> v1	[1,2,3,4,10]		
<input type="checkbox"/> v2	[-1,-2,-3,-4,-10]		

>> Experiment_1
Q17
0 0 0 1 1
fx >>

Q18) Provide two different methods without using loops to find the sum of entries in v9 which are greater than 15. Hint: Use '*' and '.*'

Code:

```
%18
disp("Q18");
vprod=ones(length(v9),1)*v9;
disp(vprod);
vv=vprod+vprod';
vans=vv>=15;
disp(vans);

vgreater=(v9>=15).*v9;
ss=sum(vgreater);
disp(vgreater);
disp(ss);
```

Output:

The screenshot shows the MATLAB workspace and command window. The workspace contains variables: ss (60), v9 (1x10 double), vans (10x10 logical), vgreater (10x10 double), vprod (10x10 double), and vv (10x10 double). The command window shows the output of the code, including the display of v9, vprod, vans, vgreater, and ss.

Workspace:

Name	Value
ss	60
v9	[25,20,15,10,...] 1x10 double
vans	10x10 logical
vgreater	[25,20,15,0,0,...] 10x10 double
vprod	10x10 double
vv	10x10 double

Command Window:

```
>> Experiment_1
Q18
25    20    15    10     5     0    -5   -10   -15   -20
25    20    15    10     5     0    -5   -10   -15   -20
25    20    15    10     5     0    -5   -10   -15   -20
25    20    15    10     5     0    -5   -10   -15   -20
25    20    15    10     5     0    -5   -10   -15   -20
25    20    15    10     5     0    -5   -10   -15   -20
25    20    15    10     5     0    -5   -10   -15   -20
25    20    15    10     5     0    -5   -10   -15   -20
25    20    15    10     5     0    -5   -10   -15   -20
25    20    15    10     5     0    -5   -10   -15   -20

1     1     1     1     1     1     1     1     0     0
1     1     1     1     1     1     1     0     0     0
1     1     1     1     1     1     0     0     0     0
1     1     1     1     1     0     0     0     0     0
1     1     1     1     0     0     0     0     0     0
1     1     1     0     0     0     0     0     0     0
1     1     0     0     0     0     0     0     0     0
1     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

25    20    15     0     0     0     0     0     0     0
```

MATRIX OPERATIONS

Q1) Create a 3×4 matrix A. Use `A = [1 2 3 4; 5 6 7 8; 9 10 11 12];`

Code:

```
%1
disp("Q1");
A = [1 2 3 4; 5 6 7 8; 9 10 11 12];
disp(A);
```

Output:

Workspace		>> Experiment_1			
Name ^	Value	Q1			
A	3x4 double	1	2	3	4
		5	6	7	8
		9	10	11	12

Q2) Print only the second row of A using `A(2, :)` and print only the third column of A using `A(:, 3)`.

Code:

```
%2
disp("Q2");
disp(A(2,:));
disp(A(:,3));
```

Output:

Workspace		>> Experiment_1			
Name ^	Value	Q2			
A	3x4 double	5	6	7	8
		3			
		7			
		11			

Q3) Print only first and third column of A.

Code:

```
%3  
disp("Q3");  
disp([A(:,1) A(:,3)]);
```

Output:

Workspace		>> Experiment_1	
Name ^	Value	Q3	
A	3x4 double	1	3
		5	7
		9	11

Q4) Type help repmat. Use repmat command to create a 12×12 matrix B repeating the matrix A.

Code:

```
%4
disp("Q4");
help repmat;
B=repmat(A,[4,3]);
disp(B);
```

Output:

Workspace
Name ^ Value
A 3x4 double
B 12x12 double

```
>> Experiment_1
Q4
repmat Replicate and tile an array.
B = repmat(A,M,N) or B = repmat(A,[M,N]) creates a large matrix B
consisting of an M-by-N tiling of copies of A. If A is a matrix,
the size of B is [size(A,1)*M, size(A,2)*N].

B = repmat(A,N) creates an N-by-N tiling.

B = repmat(A,P1,P2,...,Pn) or B = repmat(A,[P1,P2,...,Pn]) tiles the array
A to produce an n-dimensional array B composed of copies of A. The size
of B is [size(A,1)*P1, size(A,2)*P2, ..., size(A,n)*Pn].
If A is m-dimensional with m > n, an m-dimensional array B is returned.
In this case, the size of B is [size(A,1)*P1, size(A,2)*P2, ...,
size(A,n)*Pn, size(A, n+1), ..., size(A, m)].

repmat(A,M,N) when A is a scalar is commonly used to produce an M-by-N
matrix filled with A's value and having A's CLASS. For certain values,
you may achieve the same results using other functions. Namely,
repmat(NAN,M,N) is the same as NAN(M,N)
repmat(SINGLE(INF),M,N) is the same as INF(M,N,'single')
repmat(INT8(0),M,N) is the same as ZEROS(M,N,'int8')
repmat(UINT32(1),M,N) is the the same as ONES(M,N,'uint32')
repmat(EPS,M,N) is the same as EPS(ONES(M,N))

Example:
repmat(magic(2), 2, 3)
repmat(uint8(5), 2, 3)

Class support for input A:
float: double, single
integer: uint8, int8, uint16, int16, uint32, int32, uint64, int64
char, logical

See also bsxfun, meshgrid, ones, zeros, nan, inf.

Documentation for repmat
Other uses of repmat
```

Workspace
Name ^ Value
A 3x4 double
B 12x12 double

```
Example:
repmat(magic(2), 2, 3)
repmat(uint8(5), 2, 3)

Class support for input A:
float: double, single
integer: uint8, int8, uint16, int16, uint32, int32, uint64, int64
char, logical

See also bsxfun, meshgrid, ones, zeros, nan, inf.

Documentation for repmat
Other uses of repmat
```

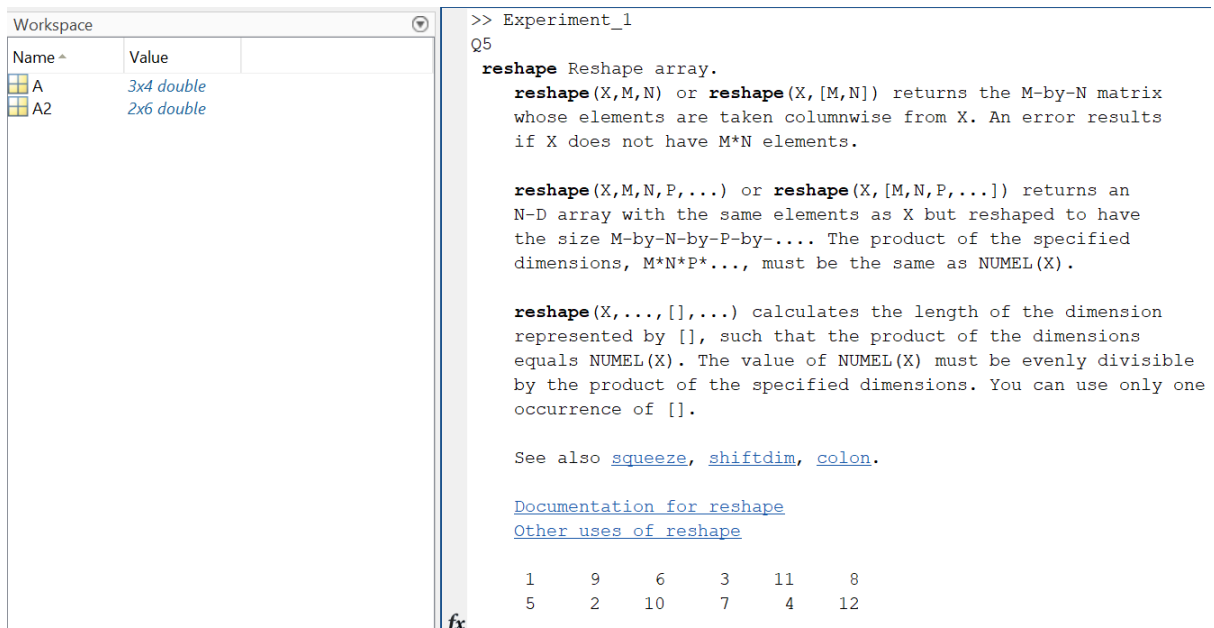
1	2	3	4	1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12	9	10	11	12
1	2	3	4	1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12	9	10	11	12
1	2	3	4	1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12	9	10	11	12
1	2	3	4	1	2	3	4	1	2	3	4
5	6	7	8	5	6	7	8	5	6	7	8
9	10	11	12	9	10	11	12	9	10	11	12

Q5) Use reshape command to change A into 2×6 matrix.

Code:

```
%5
disp("Q5");
help reshape;
A2=reshape(A,[2,6]);
disp(A2);
```

Output:



The screenshot shows the MATLAB workspace and command window. The workspace on the left contains two variables: 'A' (3x4 double) and 'A2' (2x6 double). The command window on the right shows the execution of the code, including the help text for the 'reshape' function and the resulting 2x6 matrix A2.

Workspace:

Name	Value
A	3x4 double
A2	2x6 double

Command Window:

```
>> Experiment_1
Q5
reshape Reshape array.
reshape(X,M,N) or reshape(X,[M,N]) returns the M-by-N matrix
whose elements are taken columnwise from X. An error results
if X does not have M*N elements.

reshape(X,M,N,P,...) or reshape(X,[M,N,P,...]) returns an
N-D array with the same elements as X but reshaped to have
the size M-by-N-by-P-by-.... The product of the specified
dimensions, M*N*P*..., must be the same as NUMEL(X).

reshape(X,...,[],...) calculates the length of the dimension
represented by [], such that the product of the dimensions
equals NUMEL(X). The value of NUMEL(X) must be evenly divisible
by the product of the specified dimensions. You can use only one
occurrence of [].

See also squeeze, shiftdim, colon.

Documentation for reshape
Other uses of reshape



1     9     6     3    11     8
5     2    10     7     4    12
```

Q6) Use zeros, ones commands to create matrices C and D both of order 3×4 .

Code:

```
%6  
disp("Q6");  
C=zeros(3,4);  
D=ones(3,4);  
disp(C);  
disp(D);
```

Output:

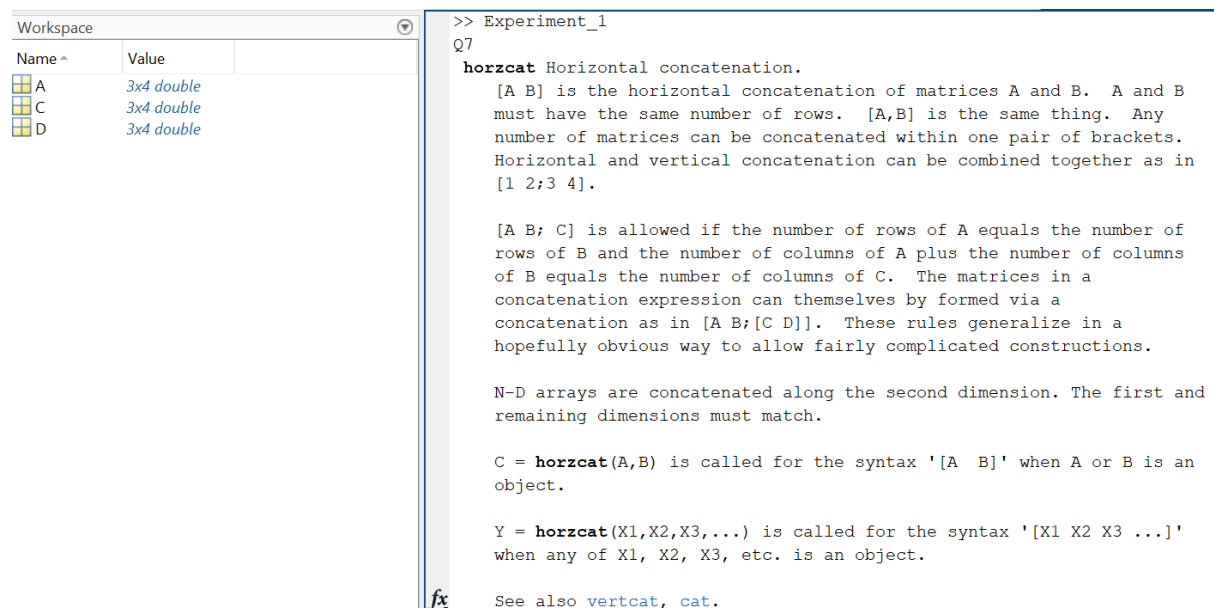
Workspace		>> Experiment_1			
Name ^	Value	Q6			
 C	3x4 double	0	0	0	0
 D	3x4 double	0	0	0	0
		0	0	0	0
		1	1	1	1
		1	1	1	1
		1	1	1	1

Q7) Use `horzcat` to concatenate A and C and use `vertcat` to concatenate A and D.

Code:

```
%7
disp("Q7");
help horzcat;
disp(horzcat(A,C));
disp(vertcat(A,D));
```

Output:



The screenshot shows the MATLAB workspace and command window. The workspace contains three variables: A (3x4 double), C (3x4 double), and D (3x4 double). The command window shows the execution of the code, displaying 'Q7' and the help text for `horzcat`.

Workspace

Name	Value
A	3x4 double
C	3x4 double
D	3x4 double

```
>> Experiment_1
Q7
horzcat Horizontal concatenation.
[A B] is the horizontal concatenation of matrices A and B. A and B
must have the same number of rows. [A,B] is the same thing. Any
number of matrices can be concatenated within one pair of brackets.
Horizontal and vertical concatenation can be combined together as in
[1 2;3 4].

[A B; C] is allowed if the number of rows of A equals the number of
rows of B and the number of columns of A plus the number of columns
of B equals the number of columns of C. The matrices in a
concatenation expression can themselves be formed via a
concatenation as in [A B;[C D]]. These rules generalize in a
hopefully obvious way to allow fairly complicated constructions.

N-D arrays are concatenated along the second dimension. The first and
remaining dimensions must match.

C = horzcat(A,B) is called for the syntax '[A B]' when A or B is an
object.

Y = horzcat(X1,X2,X3,...) is called for the syntax '[X1 X2 X3 ...]'
when any of X1, X2, X3, etc. is an object.

See also vertcat, cat.
```

[Documentation for horzcat](#)

[Other uses of horzcat](#)

1	2	3	4	0	0	0	0
5	6	7	8	0	0	0	0
9	10	11	12	0	0	0	0

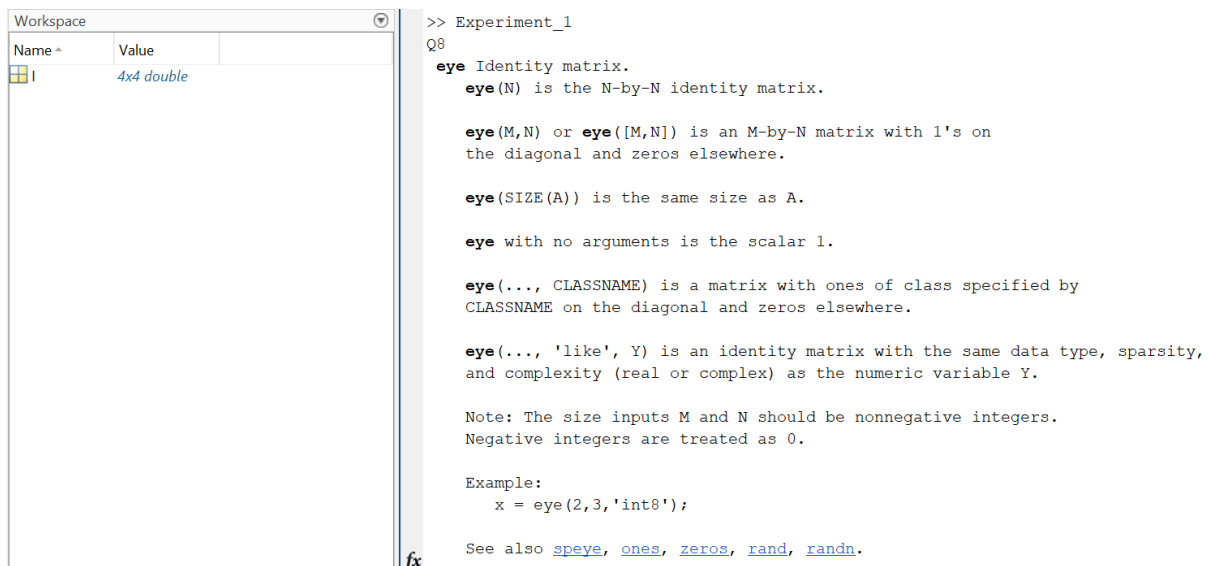
1	2	3	4
5	6	7	8
9	10	11	12
1	1	1	1
1	1	1	1
1	1	1	1

Q8) Create an identity matrix using eye command.

Code:

```
%8
disp("Q8");
help eye;
I=eye(4);
disp(I);
```

Output:



The screenshot shows the MATLAB workspace and command window. The workspace window displays a variable 'I' of type '4x4 double'. The command window shows the output of the code, including the help text for the 'eye' function.

Workspace

Name ^	Value
I	4x4 double

>> Experiment_1
Q8
eye Identity matrix.
eye(N) is the N-by-N identity matrix.

eye(M,N) or **eye**([M,N]) is an M-by-N matrix with 1's on the diagonal and zeros elsewhere.

eye(SIZE(A)) is the same size as A.

eye with no arguments is the scalar 1.

eye(..., CLASSNAME) is a matrix with ones of class specified by CLASSNAME on the diagonal and zeros elsewhere.

eye(..., 'like', Y) is an identity matrix with the same data type, sparsity, and complexity (real or complex) as the numeric variable Y.

Note: The size inputs M and N should be nonnegative integers. Negative integers are treated as 0.

Example:
x = eye(2,3,'int8');

See also [speye](#), [ones](#), [zeros](#), [rand](#), [randn](#).

[Documentation for eye](#)

```
1     0     0     0
0     1     0     0
0     0     1     0
0     0     0     1
```

Q9) Use diag command to obtain diagonal elements of B.

Code:

```
%9  
disp("Q9");  
dia=diag(B);  
disp(dia);
```

Output:

Workspace		>> Experiment_1	
Name ^	Value	Q9	
A	3x4 double		1
B	12x12 double		6
dia	12x1 double		11
			4
			5
			10
			3
			8
			9
			2
			7
			12

Q10) Use `tril` and `triu` commands on the matrix B. Apply these on matrix A also.

Code:

```
%10
disp("Q10");
help tril;
help triu;
disp(tril(A));
disp(triu(A));
```

Output:

The screenshot shows the MATLAB environment. The workspace window on the left displays a variable 'A' of type '3x4 double'. The command window on the right shows the execution of 'Experiment_1' with the following output:

```
>> Experiment_1
Q10
tril Extract lower triangular part.
    tril(X) is the lower triangular part of X.
    tril(X,K) is the elements on and below the K-th diagonal
    of X . K = 0 is the main diagonal, K > 0 is above the
    main diagonal and K < 0 is below the main diagonal.

    See also triu, diag.

    Documentation for tril
    Other uses of tril

triu Extract upper triangular part.
    triu(X) is the upper triangular part of X.
    triu(X,K) is the elements on and above the K-th diagonal of
    X. K = 0 is the main diagonal, K > 0 is above the main
    diagonal and K < 0 is below the main diagonal.

    See also tril, diag.

    Documentation for triu
    Other uses of triu
```

```
1    0    0    0
5    6    0    0
9   10   11    0
```

```
1    2    3    4
0    6    7    8
0    0   11   12
```

Q11) Using sum command find the sum of each row and the sum of each column of matrix A. Find the sum of all entries in the matrix. Use `sum(A(:))` or 'all'.

Code:

```
%11
disp("Q11");
disp(A);
disp(sum(A));
disp(sum(A'));
disp(sum(A(:)));
```

Output:

Workspace		>> Experiment_1			
Name ^	Value	Q11			
A	3x4 double	1	2	3	4
		5	6	7	8
		9	10	11	12
		15	18	21	24
		10	26	42	
		78			

Q12) Understand the difference between '*' and '.*' in matrix multiplication by choosing suitable matrices.

Code:

```
%12
disp("Q12");
E=ones(4);
disp(E);
disp(E*E);
disp(E.*E);
```

Output:

Workspace	
Name ^	Value
E	4x4 double

```
>> Experiment_1
```

```
Q12
```

```
1      1      1      1
1      1      1      1
1      1      1      1
1      1      1      1
```

```
4      4      4      4
4      4      4      4
4      4      4      4
4      4      4      4
```

```
1      1      1      1
1      1      1      1
1      1      1      1
1      1      1      1
```

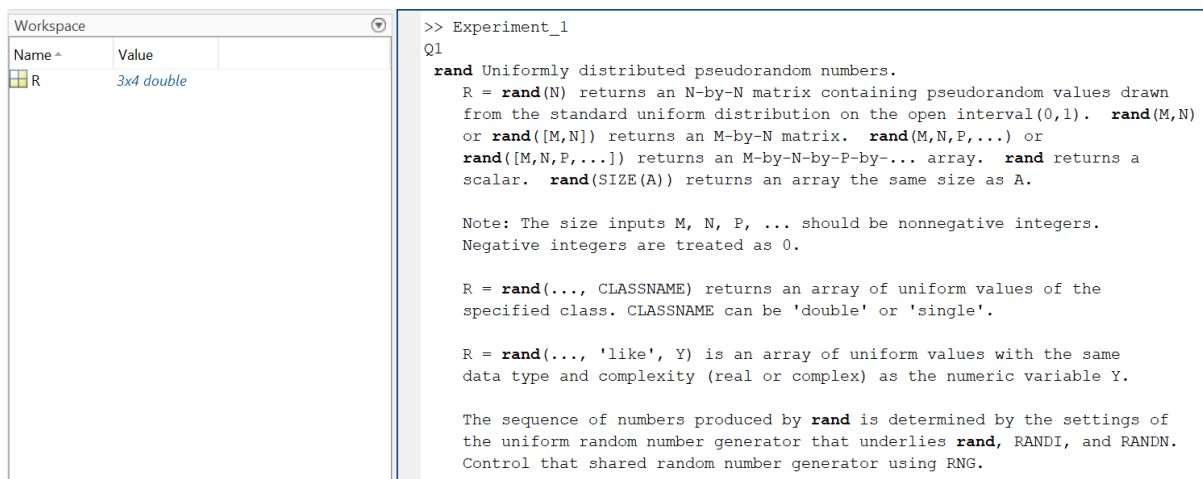
RANDOM MATRICES GENERATION

Q1) Use rand command to generate a random matrix R of order 3×4.

Code:

```
%1
disp("Q1");
help rand;
R=rand([3,4]);
disp(R);
```

Output:



The screenshot shows the MATLAB environment. On the left, the 'Workspace' window displays a variable 'R' of type '3x4 double'. On the right, the 'Command Window' shows the execution of the code from the previous block, including the help text for the 'rand' function.

```
>> Experiment_1
Q1
rand Uniformly distributed pseudorandom numbers.
R = rand(N) returns an N-by-N matrix containing pseudorandom values drawn
from the standard uniform distribution on the open interval(0,1). rand(M,N)
or rand([M,N]) returns an M-by-N matrix. rand(M,N,P,...) or
rand([M,N,P,...]) returns an M-by-N-by-P-by-... array. rand returns a
scalar. rand(SIZE(A)) returns an array the same size as A.

Note: The size inputs M, N, P, ... should be nonnegative integers.
Negative integers are treated as 0.

R = rand(..., CLASSNAME) returns an array of uniform values of the
specified class. CLASSNAME can be 'double' or 'single'.

R = rand(..., 'like', Y) is an array of uniform values with the same
data type and complexity (real or complex) as the numeric variable Y.

The sequence of numbers produced by rand is determined by the settings of
the uniform random number generator that underlies rand, RANDI, and RANDN.
Control that shared random number generator using RNG.
```

See [Replace Discouraged Syntaxes of rand and randn](#) to use RNG to replace **rand** with the 'seed', 'state', or 'twister' inputs.

See also [randi](#), [randn](#), [rng](#), [RandStream](#), [RandStream/rand](#), [sprand](#), [sprandn](#), [randperm](#).

[Documentation for rand](#)
[Other uses of rand](#)

0.8147	0.9134	0.2785	0.9649
0.9058	0.6324	0.5469	0.1576
0.1270	0.0975	0.9575	0.9706

Q2) Note that all the random values generated lie between 0 and 1. Suitably modify the code to obtain random matrices with entries between 5 and 10.

Code:

```
%2
disp("Q2");
R1=randi([5,10],[3,4]);
disp(R1);
```

Output:

Workspace		>> Experiment_1			
Name ^	Value	Q2			
R1	3x4 double	10	5	9	5
		7	7	10	10
		9	10	8	10

Q3) Use randi command to generate a matrix of uniformly distributed integers. The integers generated should belong to the interval –5 to 5.

Code:

```
%3
disp("Q3");
R2=randi([-5,5],4);
disp(R2);
```

Output:

Workspace		>> Experiment_1			
Name ^	Value	Q3			
R2	4x4 double	2	2	-2	2
		3	-4	-5	-2
		3	2	-4	5
		-1	-5	4	-5

Q4) Use randn command to generate a matrix of random values following standard normal distribution.

Code:

```
%4
disp("Q4");
R3=randn(4);
disp(R3);
```

Output:

Workspace		>> Experiment_1			
Name ^	Value	Q4			
R3	4x4 double	-0.1022	-0.8649	1.0933	-1.2141
		-0.2414	-0.0301	1.1093	-1.1135
		0.3192	-0.1649	-0.8637	-0.0068
		0.3129	0.6277	0.0774	1.5326

Q5) Modify the code to obtain entries from a normal distribution with mean 10 and variance 4.

Code:

```
%5
disp("Q5");
help randn;
R4=10+2.*randn(4);
disp(R4);
avg=mean(R4(:));
disp(avg);
```

Output:

Workspace

Name ^	Value
avg	10.2385
R4	4x4 double

```
rng(s);
z2 = randn(1,5) % z2 contains exactly the same values as z1
```

Example 5: Reinitialize the random number generator used by RAND, RANDI, and **randn** with a seed based on the current time. **randn** will return different values each time you do this. NOTE: It is usually not necessary to do this more than once per MATLAB session.

```
rng('shuffle');
randn(1,5)
```

See [Replace Discouraged Syntaxes of rand and randn](#) to use RNG to replace **randn** with the 'seed' or 'state' inputs.

See also [rand](#), [randi](#), [rng](#), [RandStream](#), [RandStream/randn](#)

[Documentation for randn](#)

[Other uses of randn](#)

8.4607	7.8219	13.0884	7.8768
10.7428	10.0651	10.1719	14.7009
9.5488	11.1051	7.0168	8.7688
12.2347	12.2012	8.5154	11.4962

10.2385

Q6) Do the same for Rayleigh and Exponential distribution with scale parameter = 2 and mean parameter = 10 respectively.

Code:

```
%6
disp("Q6");
R5=10+raylrnd(2,4);
disp(R5);
avg=mean(R5(:));
disp(avg);
R6=10+exprnd(2,4);
disp(R6);
avg=mean(R6(:));
disp(avg);
```

Output:

Workspace		>> Experiment_1			
Name ^	Value	Q6			
avg	12.0086	10.5781	13.1905	12.8434	12.3267
R5	4x4 double	12.9318	15.2624	13.9098	12.2573
R6	4x4 double	12.7588	11.3802	10.9630	11.6821
		12.8124	10.5423	14.7924	11.1932
		12.4640			
		11.0151	10.5803	10.1813	10.0077
		12.6714	11.5946	13.7628	15.0976
		10.8491	14.9581	10.3828	11.6298
		10.7444	12.9483	11.2385	14.4764
		12.0086			

SAVING AND LOADING

Q1) Create and save two variables A and B to a file savefile.mat

Code:

```
%1
disp("Q1");
varA='VARIABLE';
varB='vari_B';
save("savefile.mat",'varA');
save("savefile.mat",'varB');
```

Output:

Workspace			fx
Name ^	Value		
varA	'VARIABLE'		>> Experiment_1 Q1 >>
varB	'vari_B'		

Q2) Create a third variable C and append it to savefile.mat

Code:

```
%2
disp("Q2");
app=ones(5,5);
save("savefile.mat","app","-append");
```

Output:

Workspace			fx
Name ^	Value		
app	5x5 double		>> Experiment_1 Q2 >>

Q3) Load all three variables from savefile.mat

Code:

```
%3
disp("Q3");
var=load("savefile.m.mat","app");
disp(var);
```

Output:

Workspace		>> Experiment_1
Name ^	Value	Q3
var	1x1 struct	app: [5×5 double]

Q4) Create a string array containing the names of your three friends and their ages. Use writematrix command to write the string array to an excel file.

Code:

```
%4
disp("Q4");
name=["Alice" "Bob" "Chef"];
age=[16 17 18];
writematrix(name,"savexcel.xlsx");
writematrix(age,"savexcel.xlsx",'Writemode','append');
```

Output:

Workspace		>> Experiment_1
Name ^	Value	Q4
age	[16,17,18]	fx >>
name	1x3 string	

Q5) Use readmatrix command to read from the saved excel file and compute the average age of your three friends.

Code:

```
%5
disp("Q5");
final=readmatrix("savexcel.xlsx");
disp(final);
avrg=mean(final);
disp(avrg);
```

Output:

Workspace		>> Experiment_1		
Name ^	Value	Q5		
avrg	17	16	17	18
final	[16,17,18]	17		

Q6) Use audioread command to read an audio file.

Code:

```
%6
disp("Q6");
song=randi([1,2],[1,1]);
disp(song);
filename='yes';
if(song==1)
    filename='DBS OP1.mp3';
else
    filename='DBS OP2.mp3';
end
[y,Fs] = audioread(filename);
sound(y,Fs);
```

Output:

Workspace		>> Experiment_1	
Name ^	Value	Q6	
filename	'DBS OP2.mp3'	2	
Fs	48000		
song	2		
y	4117872x2 d...		

Q7) Use audiowrite command to write a portion of the file to a new audio file.

Code:

```
%7  
disp("Q7");  
audiowrite("DBnew.mp3",y,Fs);  
clear y Fs;
```

Output:

Workspace	
Name ^	Value
filename	'DBS OP2.mp3'
song	2

```
>> Experiment_1  
Q6  
2  
  
Q7  
fx >> |
```

Q8) Use imread to read a jpg image to a three dimensional matrix F.

Code:

```
%8  
disp("Q8");  
F=imread("goku 00.jpg");
```

Output:

Workspace	
Name ^	Value
F	563x461x3 ui...

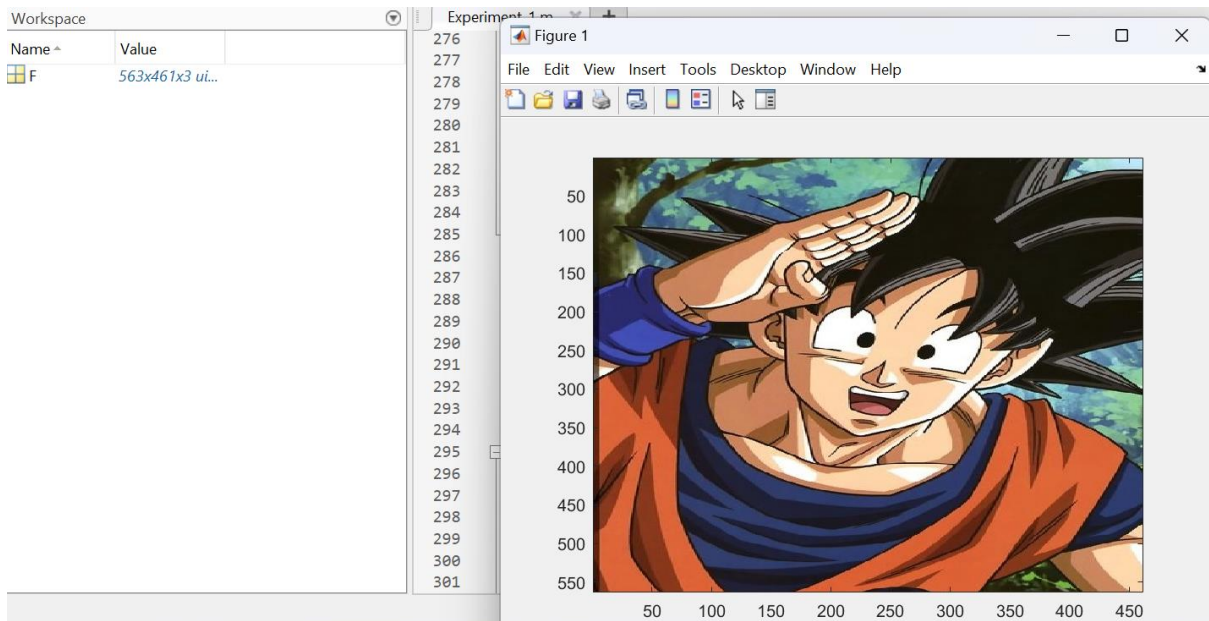
```
>> Experiment_1  
Q8  
fx >>
```

Q9) Use image function to display the image.

Code:

```
%9  
disp("Q9");  
image(F);  
|
```

Output:

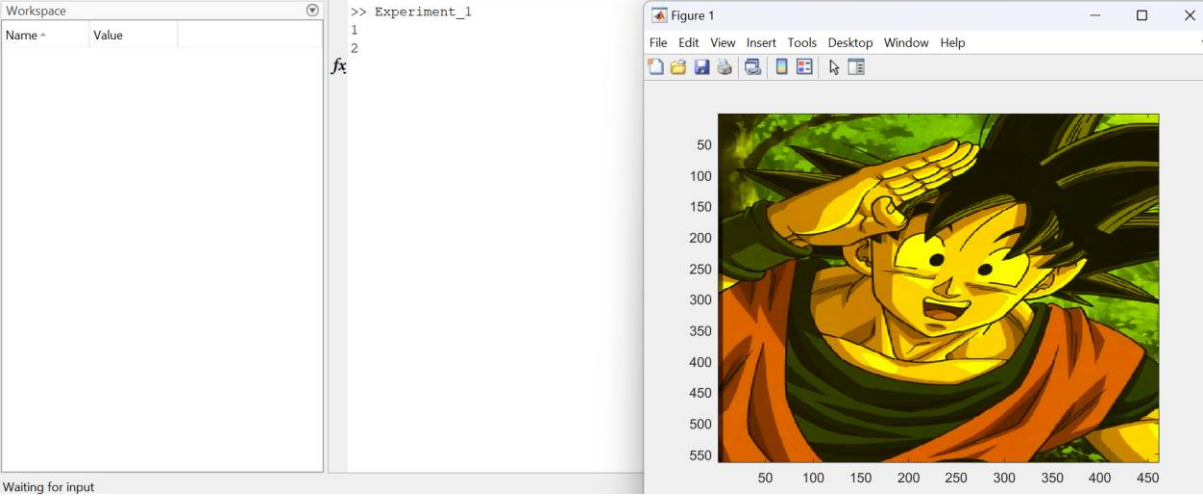
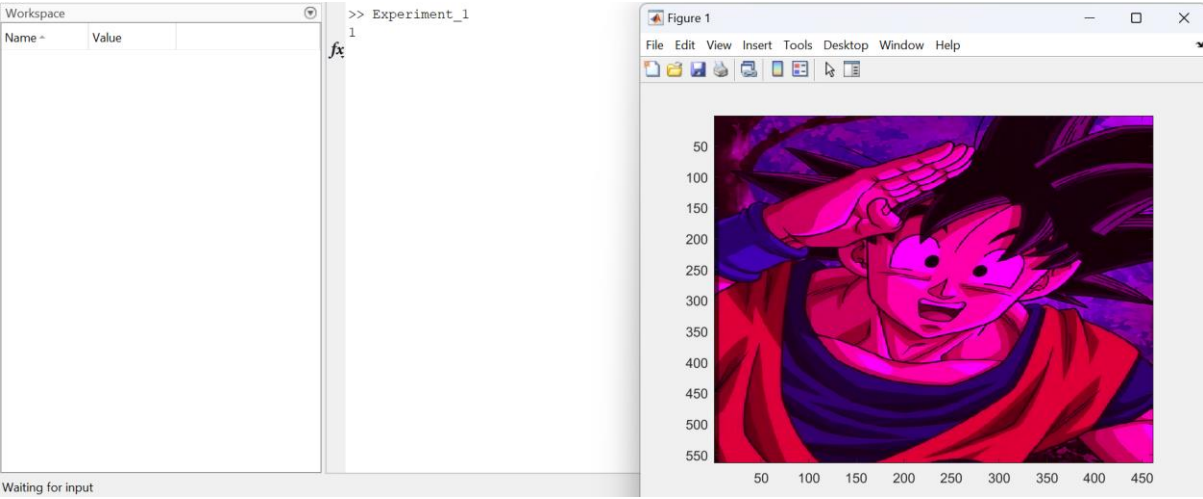
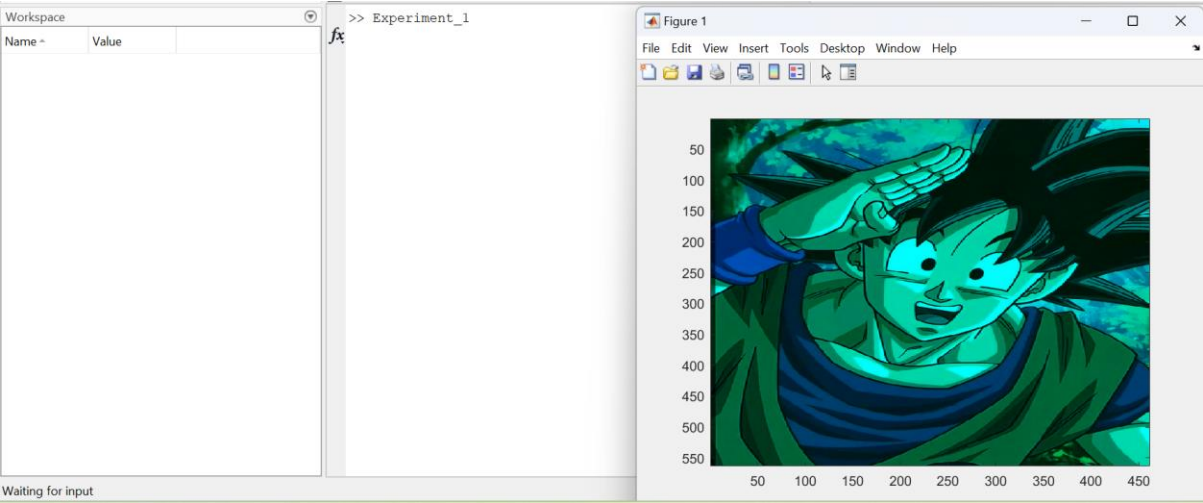


Q10) Make the entries in $F(:, :, 1)$ all zero and display the image. Restore the image and do the same for $F(:, :, 2)$ and $F(:, :, 3)$.

Code:

```
%10
disp("Q10");
G=F;
F(:, :, 1)=0;
image(F);
inp=input("");
F=G;
F(:, :, 2)=0;
image(F);
inp=input("");
F=G;
F(:, :, 3)=0;
image(F);
inp=input("");
F=G;
```

Output:



Workspace	
Name ^	Value
F	563x461x3 ui...
G	563x461x3 ui...
inp	3

```
>> Experiment_1
1
2
3
fx >> |
```


LOOPS AND FUNCTIONS

Q1) Create a 100×100 random matrix K having integer valued elements uniformly distributed in the interval 1 to 10. Use for loop to find the sum of square of the entries in the matrix K.

Code:

```
%1
disp("Q1");
K=randi([1,10],[100,100]);
sm=0;
for i=1:100
    for j=1:100
        sm=sm+K(i,j)*K(i,j);
    end
end
disp(sm);
```

Output:

Workspace		>> Experiment_1
Name ^	Value	
i	100	Q1
i	100	387426

Q2) Do part 1 without using for loop.

Code:

```
%2
disp("Q2");
K2=K.*K;
sm2=sum(K2(:));
disp(sm2);
```

Output:


Workspace		>> Experiment_1
Name ^	Value	
i	100	Q1
j	100	387426
K	100x100 dou...	Q2
K2	100x100 dou...	387426
sm	387426	
sm2	387426	fx >>

Q3) Use tic toc command to compute the time elapsed in Part 1 and Part 2. Read more about vectorization.

Code:

```
%3
disp("Q3");
tic
sm=0;
for i=1:100
    for j=1:100
        sm=sm+K(i,j)*K(i,j);
    end
end
toc
tic
K2=K.*K;
sm2=sum(K2(:));
toc
```

Output:

Workspace		fx	>>
Name ^	Value		
i	100		>> Experiment_1 Q3 Elapsed time is 0.001302 seconds. Elapsed time is 0.000278 seconds. >>
j	100		
K	100x100 dou...		
K2	100x100 dou...		
sm	380721		
sm2	380721		

Q4) Find the number of entries greater than 5 and less than 3 in the generated matrix K. Use if else statement

Code:

```
%4
disp("Q4");
cnt=0;
for i=1:100
    for j=1:100
        if K(i,j)<3 || K(i,j)>5
            cnt=cnt+1;
        end
    end
end
disp(cnt);
```

Output:

Workspace		>> Experiment_1	
Name ^	Value	Q4	
cnt	7042		7042
cnt1	7042		

Q5) Do part 3 without using if else statement.

Code:

```
%5
disp("Q5");
K3=K>5;
K4=K<3;
cnt1=sum(K3(:))+sum(K4(:));
disp(cnt1);
```

Output:

Workspace		>> Experiment_1	
Name ^	Value	Q4	
cnt	7042		7042
cnt1	7042		
i	100	Q5	
j	100		7042
K	100x100 dou...		
K3	100x100 logi...		
K4	100x100 logi...		




fx >> |

Q6) Demonstrate the use of while loop by a small program.

Code: (GCD/HCF of two integers a and b)

```
%6
disp("Q6");
ranb=randi([20,40]);
rana=randi([10,20]);
disp(ranb);
disp(rana);
while rem(ranb,rana)~=0
    temp=rem(ranb,rana);
    ranb=rana;
    rana=temp;
end
disp(rana);
```

Output:

Workspace		
Name ^	Value	
 rana	4	>> Experiment_1 Q6 32 12 4
 ranb	8	
 temp	4	

Q7) Demonstrate the use of switch statement by a small program.

Code:

```
%7
disp("Q7");
numb=input("Enter a number between 1 to 4: ");
switch(numb)
    case 1
        disp("You entered number 1");
    case 2
        disp("You entered number 2");
    case 3
        disp("You entered number 3");
    case 4
        disp("You entered number 4");
    otherwise
        disp("You entered a big number");
end
```

Output:

Workspace	
Name ^	Value
numb	3

```
>> Experiment_1
Q7
Enter a number between 1 to 4: 3
You entered number 3
```

Workspace	
Name ^	Value
numb	9

```
>> Experiment_1
Q7
Enter a number between 1 to 4: 9
You entered a big number
```

Q8) Define a function to find the factorial of a number.

Code:

```
%8
disp("Q8");
n=input("Enter the number for your factorial: ");
fact=factorial(n);
disp(fact);
function fac=factorial(n)
    if(n==0) fac=1;
    else
        fac=n*(factorial(n-1));
    end
end
```

Output:

Workspace	
Name ^	Value
fact	5040
n	7

```
>> Experiment_1
Q8
Enter the number for your factorial: 7
5040
```

