Array Manipulation

EDU >> A = [1 2 3; 4 5 6; 7 8 9]

A =
1 2 3
4 5 6
7 8 9

EDU >> A(3,3) = 0  \% set element in 3rd row, 3rd column to zero.

A =
1 2 3
4 5 6
7 8 0

EDU >> A(2,6) = 1  \% set element in 2nd row, 6th column to one.

A =
1 2 3 0 0 0
4 5 6 0 0 1
7 8 0 0 0 0

EDU >> A(3,4) = 4

EDU >> A = [1 2 3; 4 5 6; 7 8 9]  \% restore original data

EDU >> B = A(3,: 1:1:3)  \% creates a matrix B by taking the rows of A in reverse order.

B = 7 8 9
4 5 6
1 2 3

EDU >> B = A(3,: 1:3)  \% The final single colon means take all columns, does the same as the preceding example.

B =
7 8 9
4 5 6
1 2 3
Array Manipulation

EDU >> C = [A B(C, [1 3])] % Creates C by appending
% all rows in the first and third columns of B to
% the right of A.

\[
C = \begin{bmatrix}
1 & 2 & 3 & 7 & 9 \\
4 & 5 & 6 & 4 & 6 \\
7 & 8 & 9 & 1 & 3
\end{bmatrix}
\]

EDU >> B = A(1:2, 2:3) % Creates B by extracting
% the first two rows and last two columns of A.

\[
B = \begin{bmatrix}
2 & 3 \\
5 & 6
\end{bmatrix}
\]

EDU >> C = [1 3]
>> B = A(C, C) % Uses C to index the matrix A rather than
% specifying them directly.

\[
B = \begin{bmatrix}
1 & 3 \\
7 & 9
\end{bmatrix}
\]

EDU >> B = A(:, :) % This means, A is used in its entirety.

\[
B = \begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 2 & 5 & 8 \\
3 & 6
\end{bmatrix}
\]

EDU >> B = B';
EDU >> B = A

EDU >> B(:, 2) = [] % redefines B by throwing away
% all rows in the second column of original B. When you
% set something equal to empty matrix [], it gets
% deleted.

\[
B = \begin{bmatrix}
1 & 3 \\
4 & 6 \\
7 & 9
\end{bmatrix}
\]
EDU >> B = B,'
EDU >> B(2,:) = [ ] % throws out the second row of B.
B =
    1 4 7
EDU >> A(2,:) = B % replaces the second row of A with B.
    1 4 3
    1 4 7
    7 8 9
EDU >> B = A(:,[2 2 2 2]) % duplicating rows of B.
    2 2 2 2
    4 4 4 4
    8 8 8 8
EDU >> A % shows A again.
    1 4 3
    1 4 7
    7 8 9
EDU >> A(2,2) = [ ]
??? Indexed empty matrix assignment is not allowed.
EDU >> B = A(4,:)
??? Index exceeds matrix dimensions.
EDU >> B(4,:) = A
    1 2 3
    1 4 7
    7 8 9
EDU >> B(3,2) = 2
??? Index exceeds matrix dimensions.
EDU >> B(3,:) = A
EDU >> B(4,:) = A
EDU >> B = 
    1 4 7
    0 4 7
    7 8 9
Array Manipulation

\% \text{EDU} >> G(1:6) = A(2,2:3) \% Creates a row vector \% \text{G} by extracting all rows in the second and third \% columns of \text{A}.

\text{G} = 2 4 8 3 7 9

\text{EDU} >> A(2,:) = 0 \%\text{A}(2,:) = 0 \%

\text{A} =
1 2 3
0 0 0
7 8 9

\text{EDU} >> A(2,:) = [0 0 0] \%

\text{A} =
1 2 3
0 0 0
7 8 9

In Matlab the index counts elements down the columns, starting with the first. For example.

\text{EDU} >> \text{D} = [1 2 3 4; 5 6 7 8; 9 10 11 12] \% New data

\text{D} =
1 2 3 4
5 6 7 8
9 10 11 12

\text{EDU} >> \text{D}(2) \% Second element

\text{ans} =
5

\text{EDU} >> \text{D}(5) \% Second element

\text{ans} =
6

\text{EDU} >> \text{D}\text{(end)} \% Second element

\text{ans} =
12

\text{EDU} >> \text{D}(4:7) \% Second element

\text{ans} =
2 6 10 3
Array Size:

```matlab
EDU >> A = [1 2 3 4 5 6 7 8];
EDU >> S = size(A)
S =
    2     4
```

```matlab
EDU >> [r, c] = size(A) % r = number of rows c = number of columns
r = 2
c = 4
```

```matlab
EDU >> r = size(A, 1) % number of rows
r = 2
```

```matlab
EDU >> c = size(A, 2) % number of columns
C = 4
```

```matlab
EDU >> length(A) % length of A
ans = 8
```

```matlab
EDU >> B = pi * 0.01 : 2 * pi
EDU >> size(B) % shows that B is row vector
ans =
    1     315
```

```matlab
EDU >> length(B)
ans =
    315
```

```matlab
EDU >> size(C) % zero size.
ans =
    0     0
```
1. If \( x \) and \( y \) are vectors, then \( x \cdot y \) is a vector
\[
[ x(y(1)), x(y(2)), \ldots, x(y(n)) ]
\]
where \( n = \text{length}(y) \).

\[
\begin{align*}
&\gg x = [3 \ 1 \ 4 \ 2 \ 7 \ 2 \ 3 \ 5 \ 4]^	ext{\textdagger} \\
&\gg y = 1:2:9 \ 3 \times (y) \\
&\gg y = [5 \ 5 \ 5] \ 3 \times (y) \\
&\gg y = [1 \ 5 \ 1 \ 7] \ 3 \times (y)
\end{align*}
\]

2. Create a row (column) vector of \( n \) uniformly spaced elements:

\[
\begin{align*}
&\gg a = -1 \ 3 \ b = 1 \ 3 \ n = 10 \ 3 \\
&\gg x1 = a : 2 / (n-1) : b \quad \% \ a \ \text{row vector} \\
&\gg y1 = (a : 2 / (n-1) : b)^	ext{\textdagger} \quad \% \ a \ \text{column vector} \\
&\gg x2 = \text{linspace}(a, b, n) \quad \% \ a \ \text{row vector} \\
&\gg y2 = \text{linspace}(a, b, n)^	ext{\textdagger} \quad \% \ a \ \text{column vector}
\end{align*}
\]

3. Shift \( k \) (\( k \) should be positive) elements of a vector:

\[
\begin{align*}
&\gg x = [3 \ -1 \ 4 \ 2 \ 7 \ 2 \ 3 \ 5 \ 4] \ 3 \\
&\gg x([\text{end} \ 1: \text{end}-1]) \quad \% \ \text{shift (right) (or down for columns)} \ 1 \ \text{element} \\
&\gg k = 5 \ 3 \\
&\gg x([\text{end}-k+1: \text{end} \ 1: \text{end}-k]) \quad \% \ \text{shift right (or down for columns)} \ k \ \text{elements} \\
&\gg x([2: \text{end} \ 1]) \quad \% \ \text{shift left (or up for columns)} \ 1 \ \text{element} \\
&\gg x([k+1: \text{end} \ 1:k]) \quad \% \ \text{shift left (or up for columns)} \ k \ \text{elements} 
\end{align*}
\]
4. Initialize a vector with a constant:
   
   ```
   >> n = 1000;
   >> X = 5 * ones(n, 1); % 1st solution
   >> Z = zeros(n, 1); Z(5) = 5; % 2nd solution - should be faster
   ```

5. Create an n x n matrix of 3's:
   
   ```
   >> n = 1000;
   >> A = 3 * ones(n, n)
   ```

6. Create a matrix consisting of a row vector duplicated m times:
   
   ```
   >> m = 10;
   >> X = 1:5;
   >> A = ones(m, 1) * X % 1st solution
   >> B = X(ones(m, 1), :); % 2nd solution - should be faster
   ```

7. Given a vector X, create a vector Y in which each element is replicated n times:
   
   ```
   >> n = 5; X = [2 1 4];
   >> Y = X(ones(1, n), :); Y = Y(:)
   >> X = [2 1 4];
   >> Y = X(:, ones(1, n)); Y = Y(:)
   ```

8. Reverse a vector:
   
   ```
   >> X = [3 -1 4 2 7 2 3 5 4];
   >> n = length(X)
   >> X(n:-1:1) % 1st solution
   >> flipr(x) % 2nd solution
   ```